Java Beginner to Builder Bootcamp

Replay 2026 - San Francisco

20260403a

Beginner to Builder Bootcamp - Java

- **00. About this Workshop**
- 01. The Basics of Temporal
- 02. Improving Your Temporal Application Code
- 03. Using Timers in a Workflow Definition
- 04. Understanding Event History
- 05. Understanding Workflow Determinism
- 06. Signals, Queries, and Workflow Updates
- 07. Timeouts and Retry Policies
- 08. Recovering from Failure
- 09. Conclusion

During this workshop, you will

- Learn the basic architecture of the Temporal platform
- Develop and execute Workflows and Activities using the Java SDK
- Use the Web UI to gain insight into current and previous executions
- Define, send, and handle Workflow Signals and Queries
- Understand how Temporal handles crashes, timeouts, and failures
- Customize a Retry Policy for Activity Execution
- Observe how Activity Heartbeating detects failure in a long-running Activity
- Apply the Saga pattern to recover from a failure in a Workflow Execution

Logistics

- Schedule
- Asking questions
- Getting help with exercises
- Course conventions ("workflow" vs. "Workflow")
- Introductions

About the Instructor

Marcus Merrell

Greenbrae, CA, USA

Current: **Staff Developer Advocate** at Temporal Technologies Inc.

Past: Vice President of Technology Strategy, Sauce Labs

Principal SDET, RetailMeNot, inc

Software Test Architect at Peoplefluent

Beginner to Builder Bootcamp - Java

00. About this Workshop

01. The Basics of Temporal

02. Improving Your Temporal Application Code

03. Using Timers in a Workflow Definition

04. Understanding Event History

05. Understanding Workflow Determinism

06. Signals, Queries, and Workflow Updates

07. Timeouts and Retry Policies

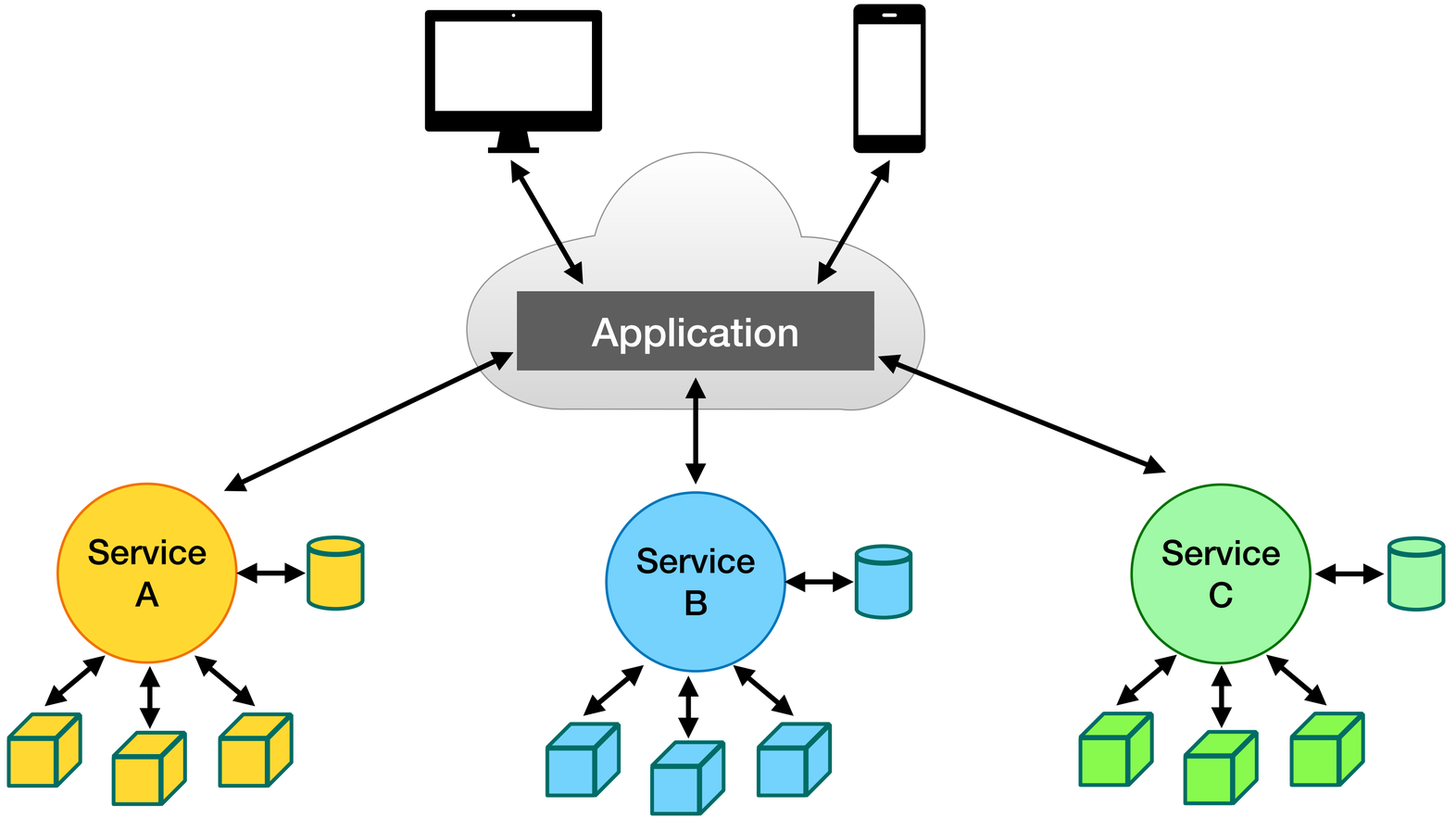
08. Recovering from Failure

09. Conclusion

What Is Temporal?

Let’s Begin with “Why Temporal Exists”

Modern Applications Are Distributed Systems



More potential for scalability, but more opportunities for failure

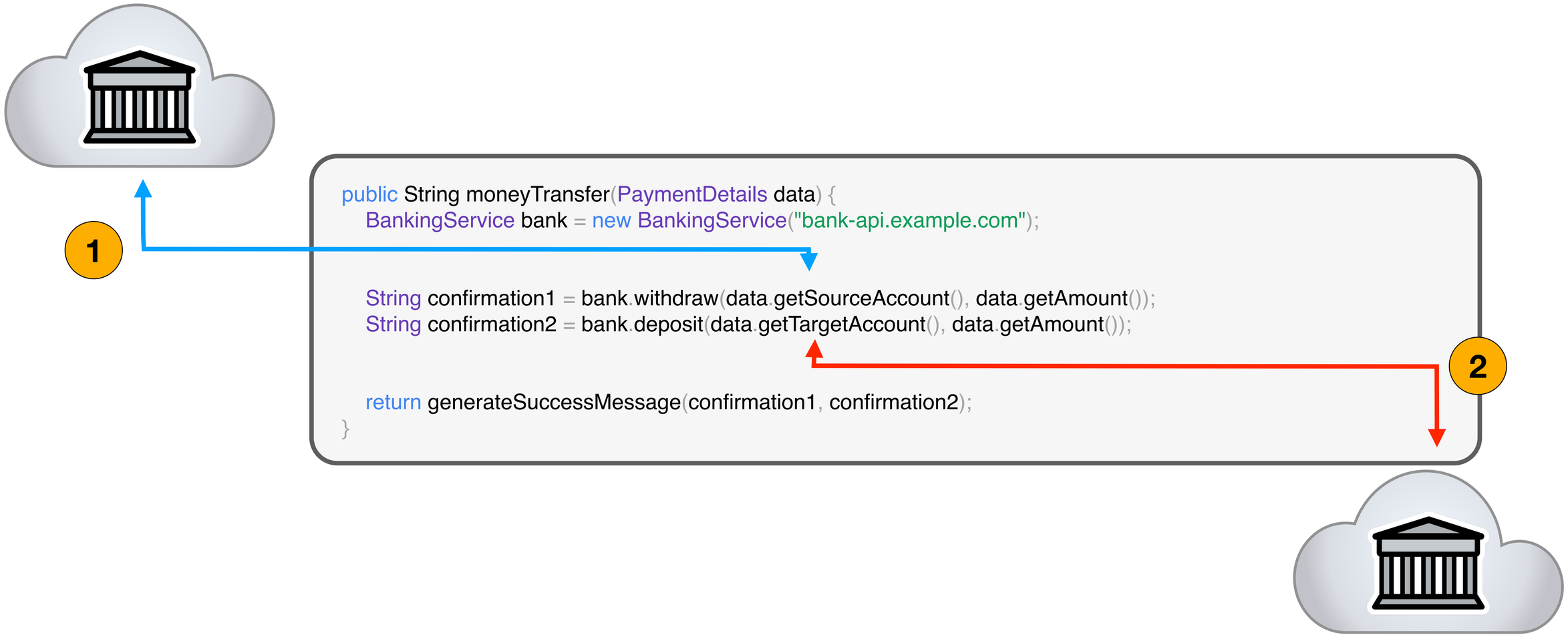
This Is a Distributed System

```
public String moneyTransfer(PaymentDetails data) {
    BankingService bank = new BankingService("bank-api.example.com");


    String confirmation1 = bank.withdraw(data.getSourceAccount(), data.getAmount());
    String confirmation2 = bank.deposit(data.getTargetAccount(), data.getAmount());


    return generateSuccessMessage(confirmation1, confirmation2);
}
```

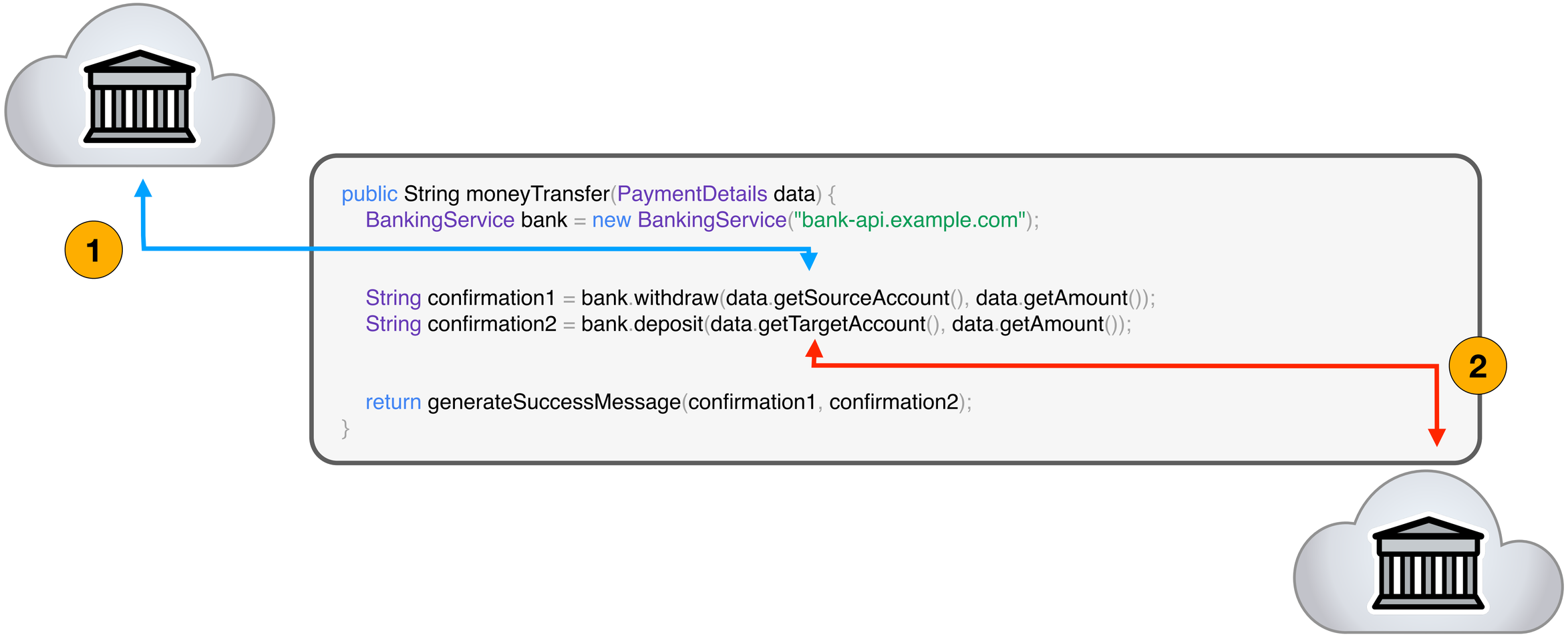
This Is a Distributed System

```
public String moneyTransfer(PaymentDetails data) {
    BankingService bank = new BankingService("bank-api.example.com");


    String confirmation1 = bank.withdraw(data.getSourceAccount(), data.getAmount());
    String confirmation2 = bank.deposit(data.getTargetAccount(), data.getAmount());


    return generateSuccessMessage(confirmation1, confirmation2);
}
```

This Is a Distributed System



There is a Big Problem Here



There is a Big Problem Here

What if it crashes between steps 1 and 2?

1

```
public String moneyTransfer(PaymentDetails data) {
    BankingService bank = new BankingService("bank-api.example.com");


    String confirmation1 = bank.withdraw(data.getSourceAccount(), data.getAmount());
    String confirmation2 = bank.deposit(data.getTargetAccount(), data.getAmount());


    return generateSuccessMessage(confirmation1, confirmation2);
}
```

2

There is a Big Problem Here

What if it crashes between steps 1 and 2?

```
public String moneyTransfer(PaymentDetails data) {
    BankingService bank = new BankingService("bank-api.example.com");


    String confirmation1 = bank.withdraw(data.getSourceAccount(), data.getAmount());
    String confirmation2 = bank.deposit(data.getTargetAccount(), data.getAmount());


    return generateSuccessMessage(confirmation1, confirmation2);
}
```

Correctness requires exactly-once execution

Durable Execution

Durable Execution

- **Crashes are inevitable, but they don’t have to be disruptive**

Durable Execution

- **Crashes are inevitable, but they don’t have to be disruptive**
  - Durable Execution is ~~*crash-proof*~~ execution

Durable Execution

- **Crashes are inevitable, but they don’t have to be disruptive**
  - Durable Execution is ~~*crash-proof*~~ execution
  - It virtualizes the execution of your application, allowing it to span a series of processes

Durable Execution

- **Crashes are inevitable, but they don’t have to be disruptive**
  - Durable Execution is ~~*crash-proof*~~ execution
  - It virtualizes the execution of your application, allowing it to span a series of processes
  - These processes can even span multiple machines over time, thus overcoming hardware failures

Durable Execution

- **Crashes are inevitable, but they don’t have to be disruptive**
  - Durable Execution is ~~*crash-proof*~~ execution
  - It virtualizes the execution of your application, allowing it to span a series of processes
  - These processes can even span multiple machines over time, thus overcoming hardware failures
- **This has profound implications**

Durable Execution

- **Crashes are inevitable, but they don’t have to be disruptive**
  - Durable Execution is *crash-proof* execution
  - It virtualizes the execution of your application, allowing it to span a series of processes
  - These processes can even span multiple machines over time, thus overcoming hardware failures
- **This has profound implications**
  - Significantly improves application reliability

Durable Execution

- **Crashes are inevitable, but they don’t have to be disruptive**
  - Durable Execution is ~~*crash-proof*~~ *fault-tolerant* execution
  - It virtualizes the execution of your application, allowing it to span a series of processes
  - These processes can even span multiple machines over time, thus overcoming hardware failures
- **This has profound implications**
  - Significantly improves application reliability
  - Lets your code focus on the *goal*, not the *potential problems*

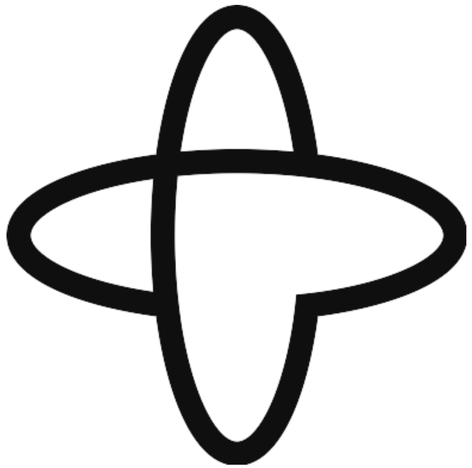
Durable Execution

- **Crashes are inevitable, but they don’t have to be disruptive**
  - Durable Execution is ~~*crash-proof*~~ execution
  - It virtualizes the execution of your application, allowing it to span a series of processes
  - These processes can even span multiple machines over time, thus overcoming hardware failures
- **This has profound implications**
  - Significantly improves application reliability
  - Lets your code focus on the *goal*, not the *potential problems*
  - Accelerates development cycles and reduces maintenance effort

Durable Execution

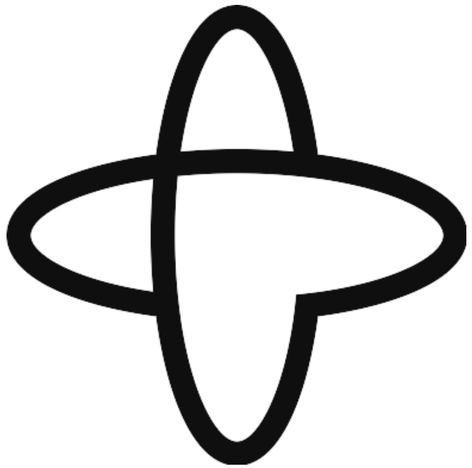
- **Crashes are inevitable, but they don’t have to be disruptive**
  - Durable Execution is *crash-proof* execution
  - It virtualizes the execution of your application, allowing it to span a series of processes
  - These processes can even span multiple machines over time, thus overcoming hardware failures
- **This has profound implications**
  - Significantly improves application reliability
  - Lets your code focus on the *goal*, not the *potential problems*
  - Accelerates development cycles and reduces maintenance effort
  - It will fundamentally change how you design software

What Is Temporal?



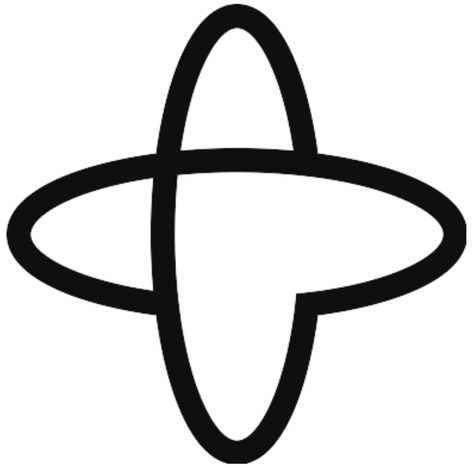
What Is Temporal?

- **Temporal is an open source Durable Execution platform**



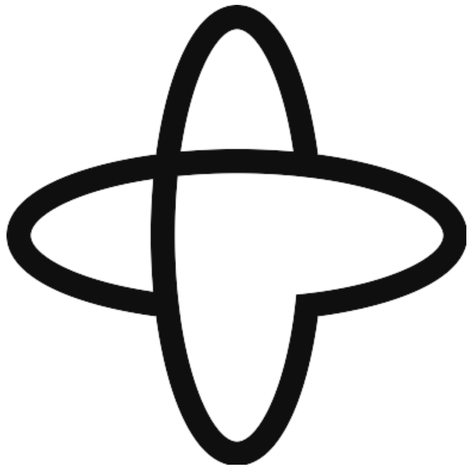
What Is Temporal?

- **Temporal is an open source Durable Execution platform**
  - Automatically maintains application state



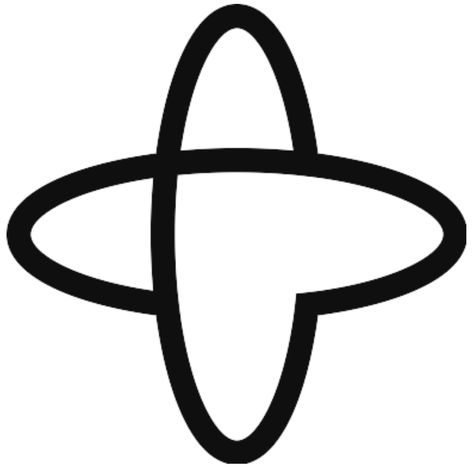
What Is Temporal?

- **Temporal is an open source Durable Execution platform**
  - Automatically maintains application state
  - Offers built-in support for managing retries and timeouts



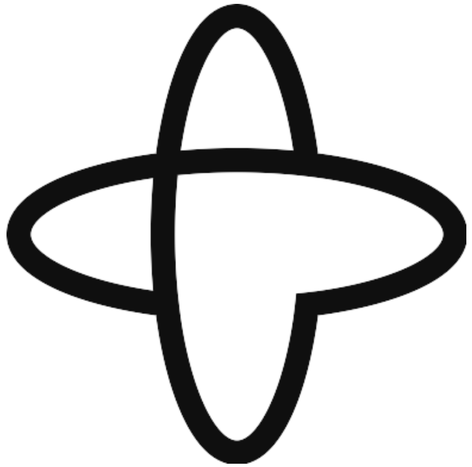
What Is Temporal?

- **Temporal is an open source Durable Execution platform**
  - Automatically maintains application state
  - Offers built-in support for managing retries and timeouts
  - Enables applications to recover from crashes and failures—as if they *never even happened*



What Is Temporal?

- **Temporal is an open source Durable Execution platform**
  - Automatically maintains application state
  - Offers built-in support for managing retries and timeouts
  - Enables applications to recover from crashes and failures—as if they *never even happened*
  - Improves developer productivity by making applications easier to develop, scale, and support



Architectural Overview

Temporal Application Overview: Workflows

Temporal Application Overview: Workflows

- **Workflows are the core abstraction in a Temporal application**

Temporal Application Overview: Workflows

- **Workflows are the core abstraction in a Temporal application**
  - They define the sequence of steps for your business logic

< / > Workflow Definition

Temporal Application Overview: Workflows

- **Workflows are the core abstraction in a Temporal application**
  - They define the sequence of steps for your business logic
  - The form the foundation of Durable Execution

< / > Workflow Definition

Temporal Application Overview: Workflows

- **Workflows are the core abstraction in a Temporal application**
  - They define the sequence of steps for your business logic
  - The form the foundation of Durable Execution
    - If a crash occurs, the state is automatically reconstructed and execution will continue

< / > Workflow Definition

Temporal Application Overview: Workflows

- **Workflows are the core abstraction in a Temporal application**
  - They define the sequence of steps for your business logic
  - The form the foundation of Durable Execution
    - If a crash occurs, the state is automatically reconstructed and execution will continue
  - Temporal requires that Workflows are *deterministic*

< / > Workflow Definition

Temporal Workflows Are Code

Temporal Workflows Are Code

- **Temporal Workflows are defined in a standard programming language**

Temporal Workflows Are Code

- **Temporal Workflows are defined in a standard programming language**
  - A *Temporal SDK* provides support for using Temporal in a specific language

Temporal Workflows Are Code

- **Temporal Workflows are defined in a standard programming language**
  - A *Temporal SDK* provides support for using Temporal in a specific language
  - Temporal currently offers seven SDKs…with an eighth (Rust) currently in development

Workflow Definition

Workflow Definition

- **With Temporal's Java SDK, you create a Workflow by writing an Interface**

Workflow Definition

- **With Temporal's Java SDK, you create a Workflow by writing an Interface**
  - The code (including the implementation) for this interface is known as a *Workflow Definition*

Workflow Definition

- **With Temporal's Java SDK, you create a Workflow by writing an Interface**
  - The code (including the implementation) for this interface is known as a *Workflow Definition*
  - Each Workflow has a name, known as its *Workflow Type*

Workflow Definition

- **With Temporal's Java SDK, you create a Workflow by writing an Interface**
  - The code (including the implementation) for this interface is known as a *Workflow Definition*
  - Each Workflow has a name, known as its *Workflow Type*
    - In the Java SDK, the Workflow Type is the name of the Interface (by default)

Workflow Definition

- **With Temporal's Java SDK, you create a Workflow by writing an Interface**
  - The code (including the implementation) for this interface is known as a *Workflow Definition*
  - Each Workflow has a name, known as its *Workflow Type*
    - In the Java SDK, the Workflow Type is the name of the Interface (by default)
- **Naming conventions for Workflow methods:**

Workflow Definition

- **With Temporal's Java SDK, you create a Workflow by writing an Interface**
  - The code (including the implementation) for this interface is known as a *Workflow Definition*
  - Each Workflow has a name, known as its *Workflow Type*
    - In the Java SDK, the Workflow Type is the name of the Interface (by default)
- **Naming conventions for Workflow methods:**
  - Temporal does not impose any rules on how you name your Workflow methods.

Workflow Definition

- **With Temporal's Java SDK, you create a Workflow by writing an Interface**
  - The code (including the implementation) for this interface is known as a *Workflow Definition*
  - Each Workflow has a name, known as its *Workflow Type*
    - In the Java SDK, the Workflow Type is the name of the Interface (by default)
- **Naming conventions for Workflow methods:**
  - Temporal does not impose any rules on how you name your Workflow methods.
  - You can use the naming convention you prefer.

Writing a Workflow Definition

Writing a Workflow Definition

- **Three steps for turning a Java Interface/Implementation into a Workflow Definition**

Interface

```
package greeting;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;

@WorkflowInterface
public interface GreetingWorkflow {

    @WorkflowMethod
    String greetSomeone(String name);
}
```

Implementation

```
package greeting;

public class GreetingWorkflowImpl implements GreetingWorkflow {

    @Override
    public String greetSomeone(String name) {
        return "Hello " + name + "!";
    }
}
```

Writing a Workflow Definition

- **Three steps for turning a Java Interface/Implementation into a Workflow Definition**

  1. Import the WorkflowInterface and WorkflowMethod annotations from the SDK

Interface

1

```
package greeting;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;

@WorkflowInterface
public interface GreetingWorkflow {

    @WorkflowMethod
    String greetSomeone(String name);
}
```

Implementation

```
package greeting;

public class GreetingWorkflowImpl implements GreetingWorkflow {

    @Override
    public String greetSomeone(String name) {
        return "Hello " + name + "!";
    }
}
```

Writing a Workflow Definition

- **Three steps for turning a Java Interface/Implementation into a Workflow Definition**

  1. Import the WorkflowInterface and WorkflowMethod annotations from the SDK
  2. Annotate the interface with @WorkflowInterface

Interface

```java
package greeting;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;

@WorkflowInterface
public interface GreetingWorkflow {

    @WorkflowMethod
    String greetSomeone(String name);
}
```

Implementation

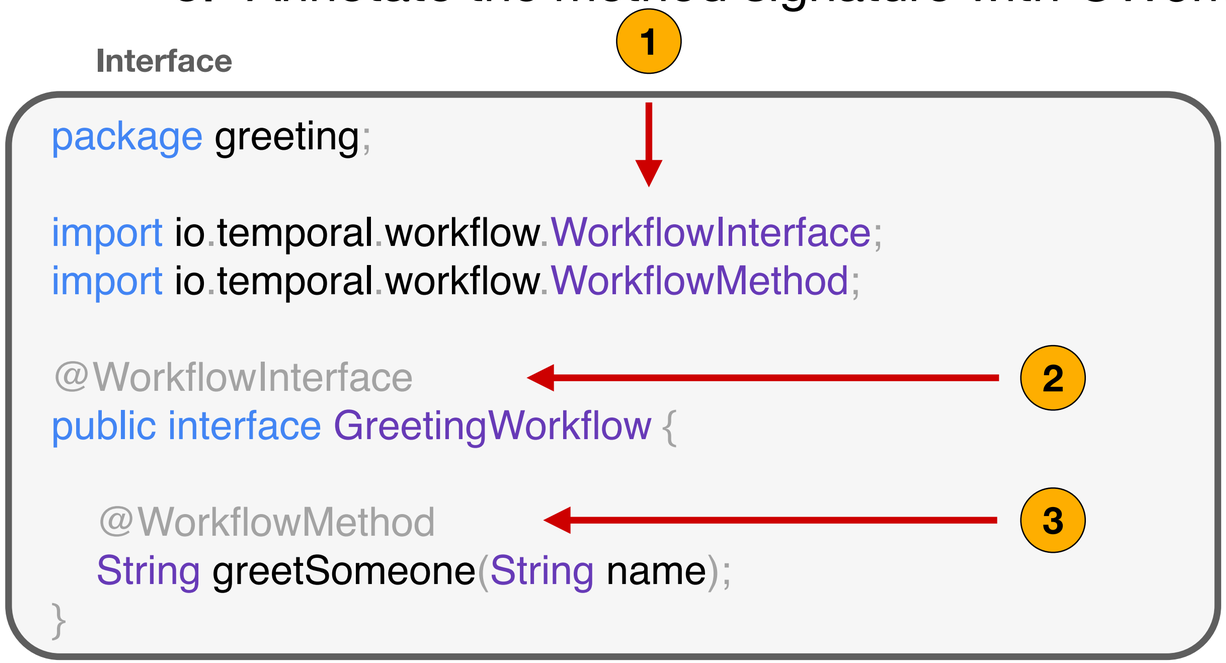
```java
package greeting;

public class GreetingWorkflowImpl implements GreetingWorkflow {

    @Override
    public String greetSomeone(String name) {
        return "Hello " + name + "!";
    }
}
```

Writing a Workflow Definition

- **Three steps for turning a Java Interface/Implementation into a Workflow Definition**

  1. Import the WorkflowInterface and WorkflowMethod annotations from the SDK
  2. Annotate the interface with @WorkflowInterface
  3. Annotate the method signature with @WorkflowMethod

Interface




```
package greeting;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;

@WorkflowInterface
public interface GreetingWorkflow {

    @WorkflowMethod
    String greetSomeone(String name);
}
```

Implementation

```
package greeting;

public class GreetingWorkflowImpl implements GreetingWorkflow {

    @Override
    public String greetSomeone(String name) {
        return "Hello " + name + "!";
    }
}
```

Input Parameters and Return Values

Input Parameters and Return Values

- **Temporal stores the history of your Workflow Executions**

Input Parameters and Return Values

- **Temporal stores the history of your Workflow Executions**
  - Allows you to view input / output of running and completed Workflows

Input Parameters and Return Values

- **Temporal stores the history of your Workflow Executions**
  - Allows you to view input / output of running and completed Workflows
  - Also affects how you will design your Workflows

Input Parameters and Return Values

- **Temporal stores the history of your Workflow Executions**
  - Allows you to view input / output of running and completed Workflows
  - Also affects how you will design your Workflows
- **Input parameters and return values must be serializable**

Input Parameters and Return Values

- **Temporal stores the history of your Workflow Executions**
  - Allows you to view input / output of running and completed Workflows
  - Also affects how you will design your Workflows
- **Input parameters and return values must be serializable**
  - Allowed: Null values, binary data, and anything serializable via the default Jackson JSON Payload Converter

Input Parameters and Return Values

- **Temporal stores the history of your Workflow Executions**
  - Allows you to view input / output of running and completed Workflows
  - Also affects how you will design your Workflows
- **Input parameters and return values must be serializable**
  - Allowed: Null values, binary data, and anything serializable via the default Jackson JSON Payload Converter
- **Avoid passing in or returning large amounts of data from your Workflow**

Input Parameters and Return Values

- **Temporal stores the history of your Workflow Executions**
  - Allows you to view input / output of running and completed Workflows
  - Also affects how you will design your Workflows
- **Input parameters and return values must be serializable**
  - Allowed: Null values, binary data, and anything serializable via the default Jackson JSON Payload Converter
- **Avoid passing in or returning large amounts of data from your Workflow**
  - May rapidly expand the size of your Temporal Service's database

Input Parameters and Return Values

- **Temporal stores the history of your Workflow Executions**
  - Allows you to view input / output of running and completed Workflows
  - Also affects how you will design your Workflows
- **Input parameters and return values must be serializable**
  - Allowed: Null values, binary data, and anything serializable via the default Jackson JSON Payload Converter
- **Avoid passing in or returning large amounts of data from your Workflow**
  - May rapidly expand the size of your Temporal Service's database
- **Data Confidentiality**

Input Parameters and Return Values

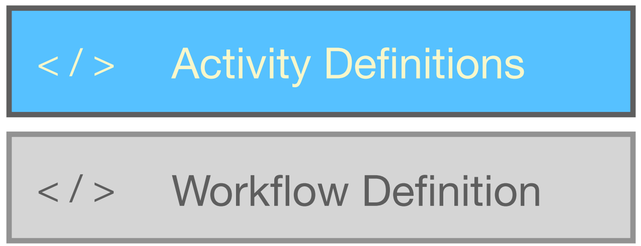
- **Temporal stores the history of your Workflow Executions**
  - Allows you to view input / output of running and completed Workflows
  - Also affects how you will design your Workflows
- **Input parameters and return values must be serializable**
  - Allowed: Null values, binary data, and anything serializable via the default Jackson JSON Payload Converter
- **Avoid passing in or returning large amounts of data from your Workflow**
  - May rapidly expand the size of your Temporal Service's database
- **Data Confidentiality**
  - All data is stored in the Event History, so encryption is required to ensure confidentiality

Temporal Activities

< / > Workflow Definition

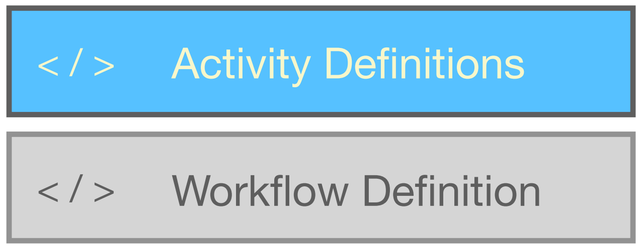
Temporal Activities

- **Activities encapsulate unreliable or non-deterministic code**



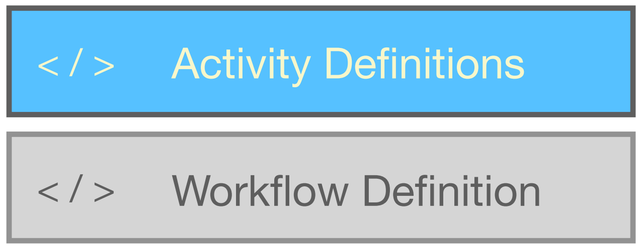
Temporal Activities

- **Activities encapsulate unreliable or non-deterministic code**
  - They are automatically retried upon failure



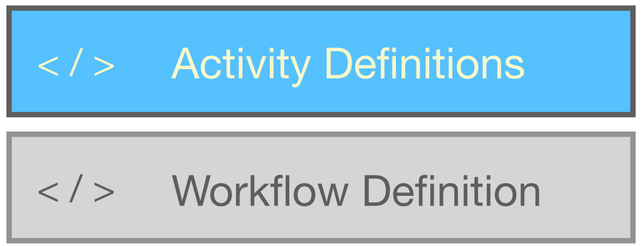
Temporal Activities

- **Activities encapsulate unreliable or non-deterministic code**
  - They are automatically retried upon failure
  - In the Java SDK, Activities are defined as an Interface and its Implementation



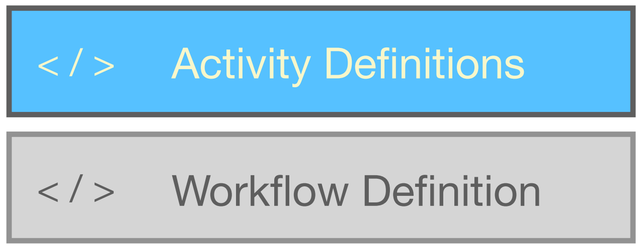
Temporal Activities

- **Activities encapsulate unreliable or non-deterministic code**
  - They are automatically retried upon failure
  - In the Java SDK, Activities are defined as an Interface and its Implementation
  - Activities should be idempotent



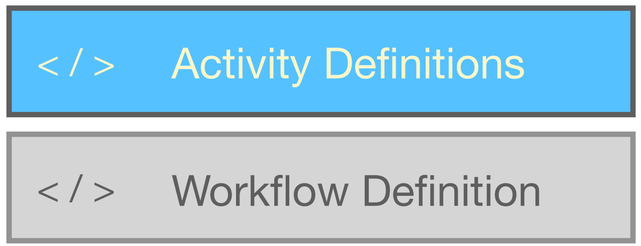
Temporal Activities

- **Activities encapsulate unreliable or non-deterministic code**
  - They are automatically retried upon failure
  - In the Java SDK, Activities are defined as an Interface and its Implementation
  - Activities should be idempotent
    - A failed Activity may be retried, which means its code will be executed again



Temporal Activities

- **Activities encapsulate unreliable or non-deterministic code**
  - They are automatically retried upon failure
  - In the Java SDK, Activities are defined as an Interface and its Implementation
  - Activities should be idempotent
    - A failed Activity may be retried, which means its code will be executed again
    - Protect against scenarios where re-running an Activity results in duplicate records or other undesirable side-effects.



Executing Activities

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

public class GreetingWorkflowImpl implements GreetingWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities = Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
```

Executing Activities

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

public class GreetingWorkflowImpl implements GreetingWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities = Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
```

Executing Activities

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

public class GreetingWorkflowImpl implements GreetingWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities = Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
```

Executing Activities

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

public class GreetingWorkflowImpl implements GreetingWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities = Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
```

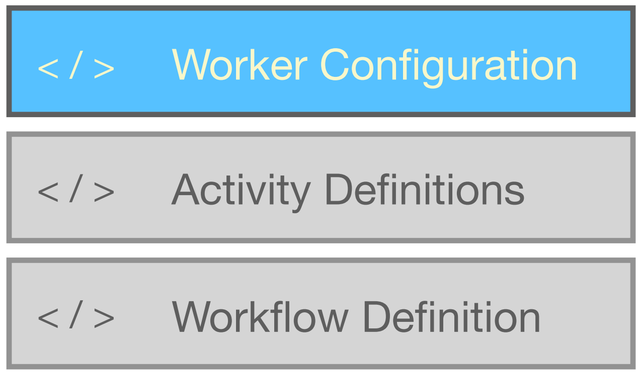
Temporal Application Overview: Workers

< / > Activity Definitions

< / > Workflow Definition

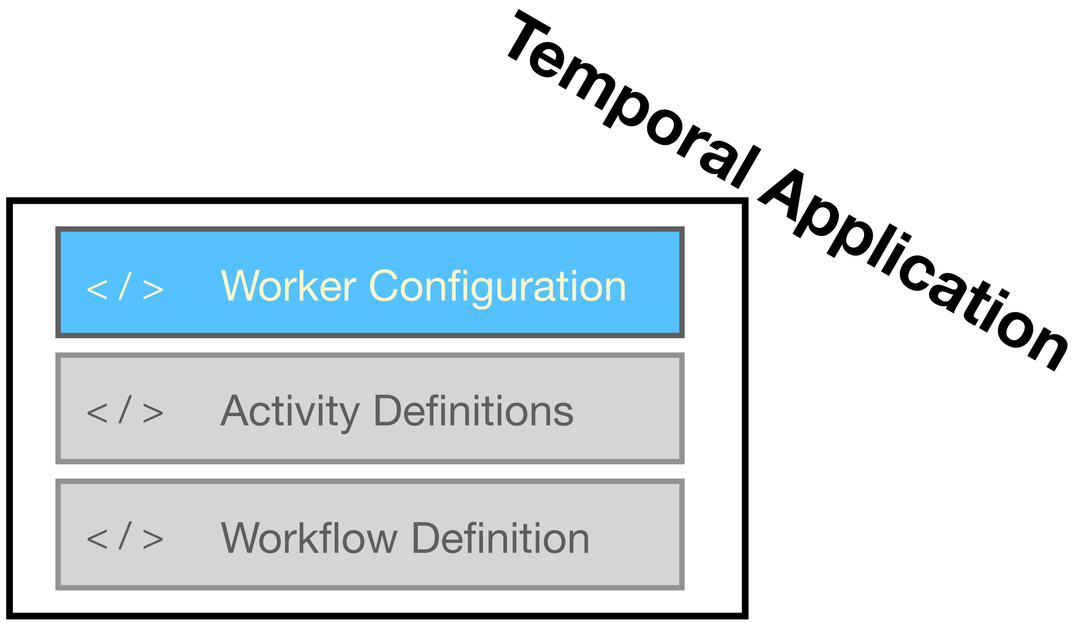
Temporal Application Overview: Workers

- **Workers are responsible for executing Workflow and Activity Definitions**



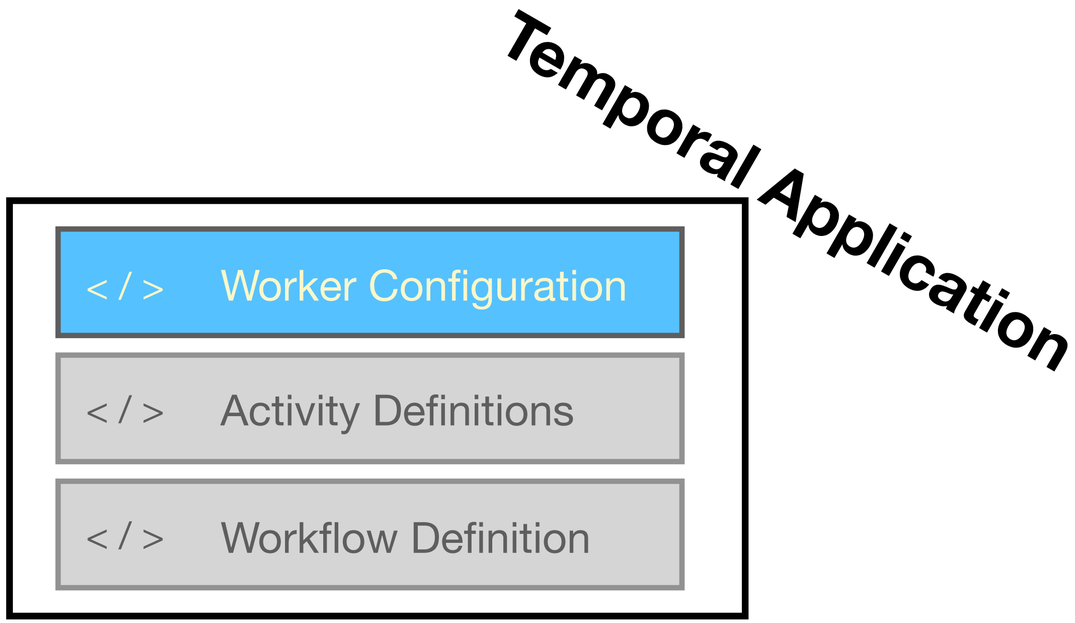
Temporal Application Overview: Workers

- **Workers are responsible for executing Workflow and Activity Definitions**



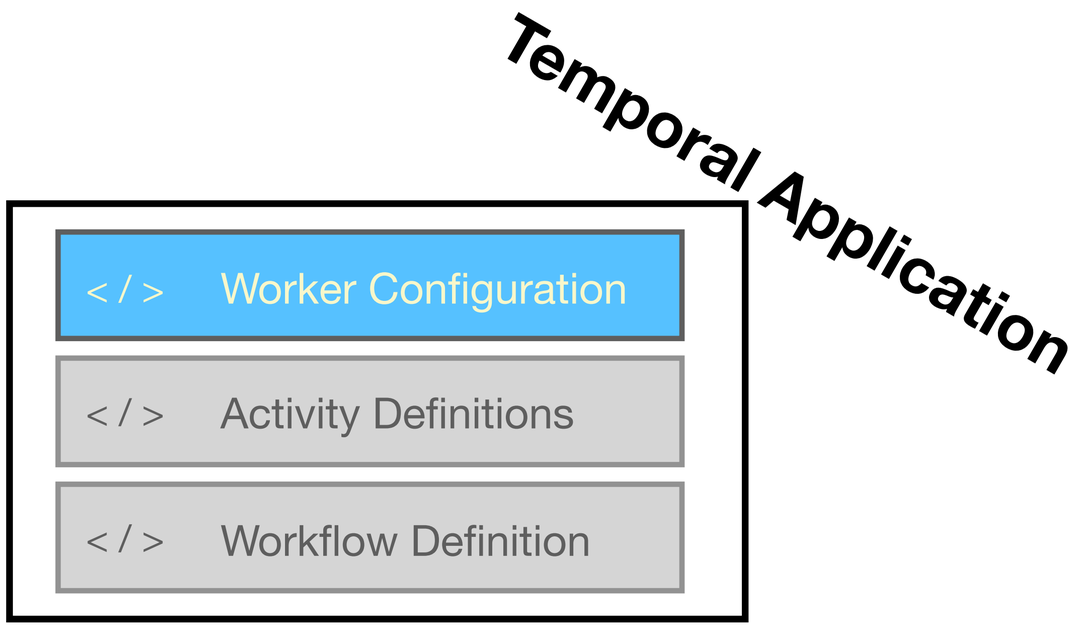
Temporal Application Overview: Workers

- **Workers are responsible for executing Workflow and Activity Definitions**
- **The Worker implementation library is provided by the Temporal SDK**



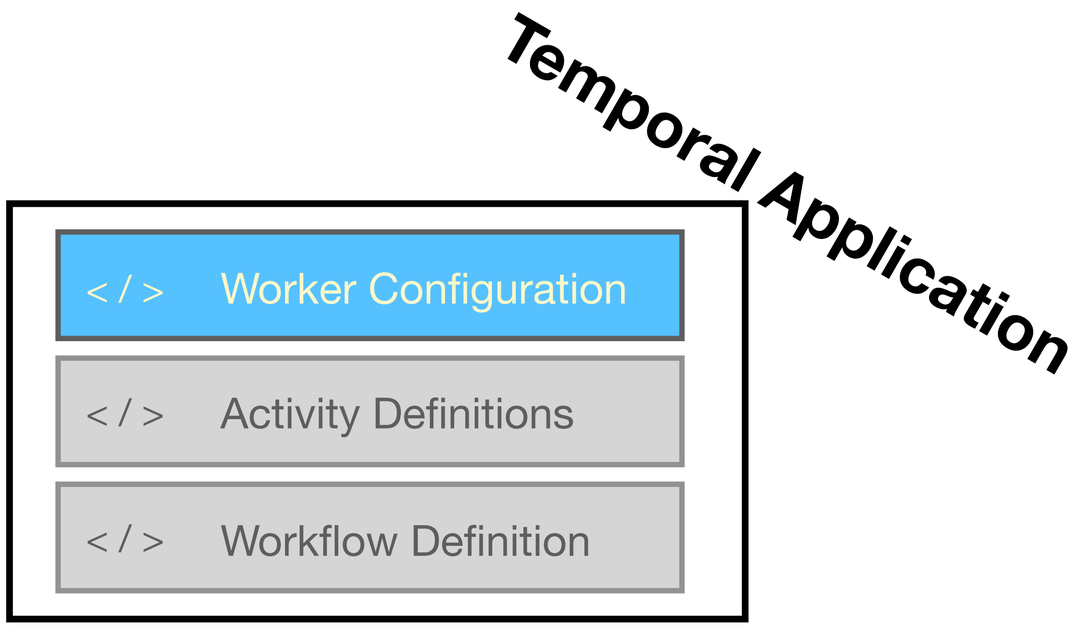
Temporal Application Overview: Workers

- **Workers are responsible for executing Workflow and Activity Definitions**
- **The Worker implementation library is provided by the Temporal SDK**
  - You must configure and start one or more Workers



Temporal Application Overview: Workers

- **Workers are responsible for executing Workflow and Activity Definitions**
- **The Worker implementation library is provided by the Temporal SDK**
  - You must configure and start one or more Workers
  - It's common to run multiple Workers: This increases the application's availability and throughput



Configuring the Worker

```java
package greeting;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

    WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
    WorkflowClient client = WorkflowClient.newInstance(service);
    WorkerFactory factory = WorkerFactory.newInstance(client);

    // Specify the name of the Task Queue that this Worker should poll
    Worker worker = factory.newWorker("greeting-tasks");

    // Specify which Workflow implementations this Worker will support
    worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

    // Begin running the Worker
    factory.start();
  }
}
```

Configuring the Worker

Configure a Temporal Client 1

```java
package greeting;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

    WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
    WorkflowClient client = WorkflowClient.newInstance(service);
    WorkerFactory factory = WorkerFactory.newInstance(client);

    // Specify the name of the Task Queue that this Worker should poll
    Worker worker = factory.newWorker("greeting-tasks");

    // Specify which Workflow implementations this Worker will support
    worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

    // Begin running the Worker
    factory.start();
  }
}
```

Configuring the Worker

Configure a Temporal Client

1

```java
package greeting;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

    WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
    WorkflowClient client = WorkflowClient.newInstance(service);
    WorkerFactory factory = WorkerFactory.newInstance(client);

    // Specify the name of the Task Queue that this Worker should poll
    Worker worker = factory.newWorker("greeting-tasks");

    // Specify which Workflow implementations this Worker will support
    worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

    // Begin running the Worker
    factory.start();
  }
}
```

2

Specify the Task Queue

Configuring the Worker

Configure a Temporal Client 1

Register the Workflow 3

```java
package greeting;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

    WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
    WorkflowClient client = WorkflowClient.newInstance(service);
    WorkerFactory factory = WorkerFactory.newInstance(client);

    // Specify the name of the Task Queue that this Worker should poll
    Worker worker = factory.newWorker("greeting-tasks");

    // Specify which Workflow implementations this Worker will support
    worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

    // Begin running the Worker
    factory.start();
  }
}
```

2 Specify the Task Queue

Configuring the Worker

Configure a Temporal Client 1

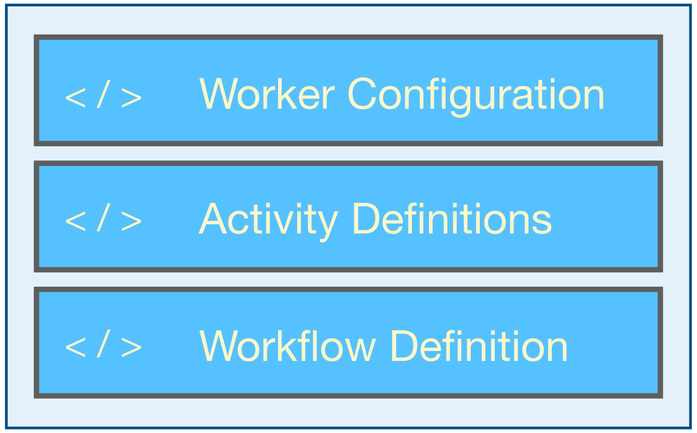
Register the Workflow 3

```java
package greeting;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

    WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
    WorkflowClient client = WorkflowClient.newInstance(service);
    WorkerFactory factory = WorkerFactory.newInstance(client);

    // Specify the name of the Task Queue that this Worker should poll
    Worker worker = factory.newWorker("greeting-tasks");

    // Specify which Workflow implementations this Worker will support
    worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

    // Begin running the Worker
    factory.start();
  }
}
```

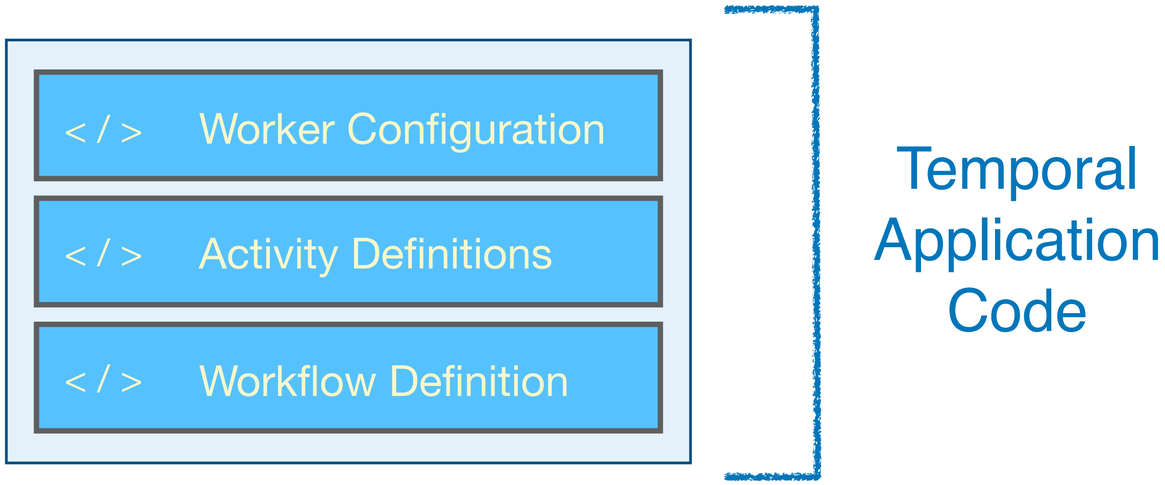
2 Specify the Task Queue

4 Begin polling the Task Queue

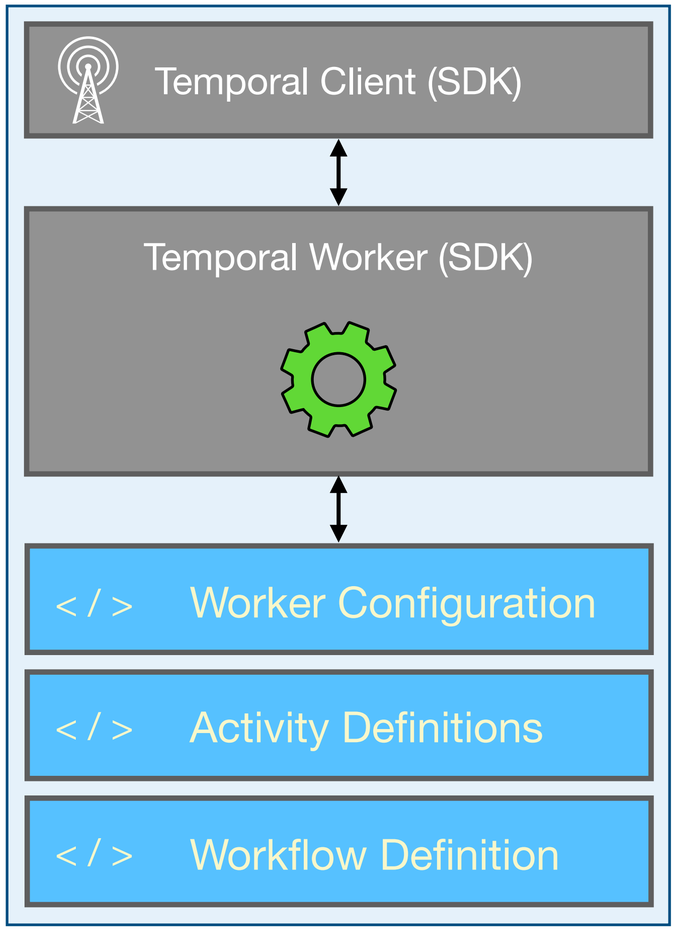
This Is the Code that You Develop



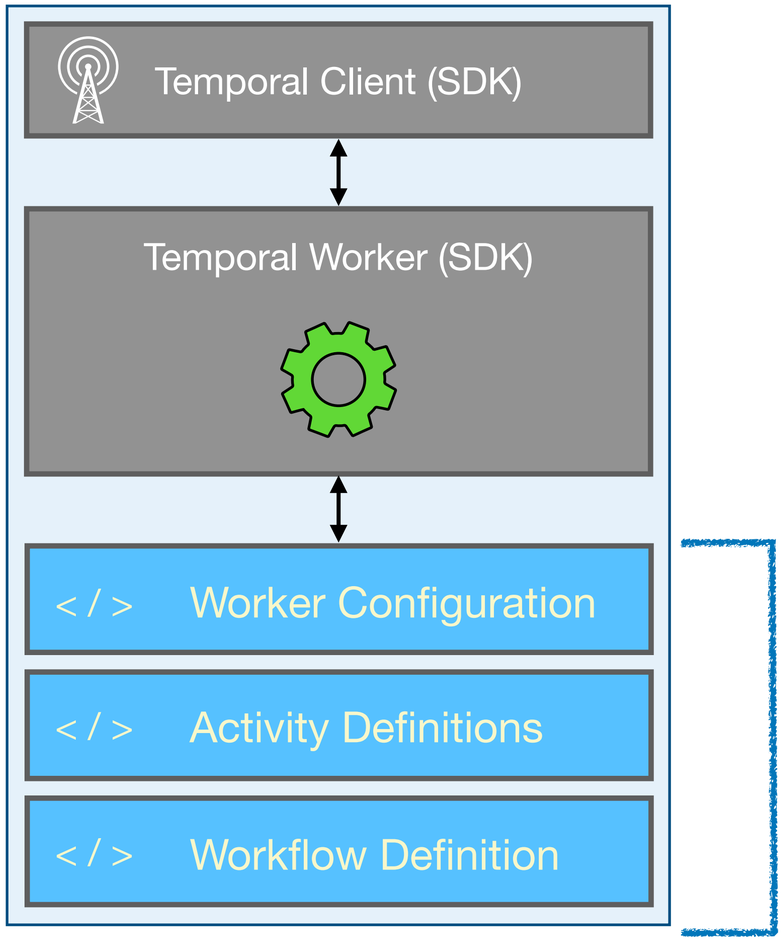
This Is the Code that You Develop



A Complete Temporal Application

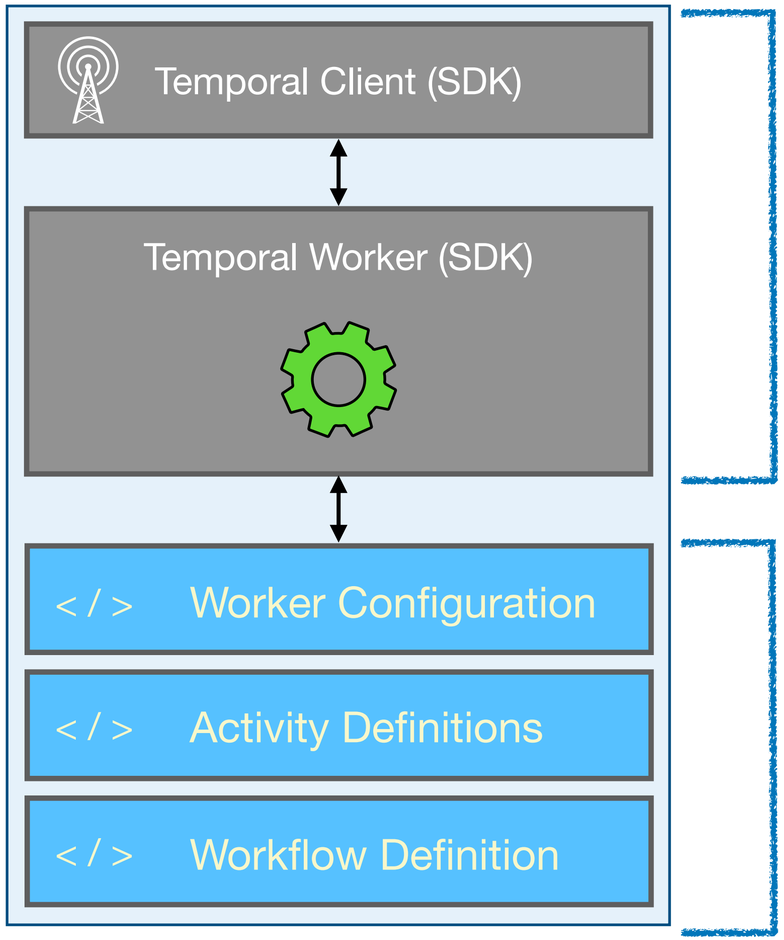


A Complete Temporal Application

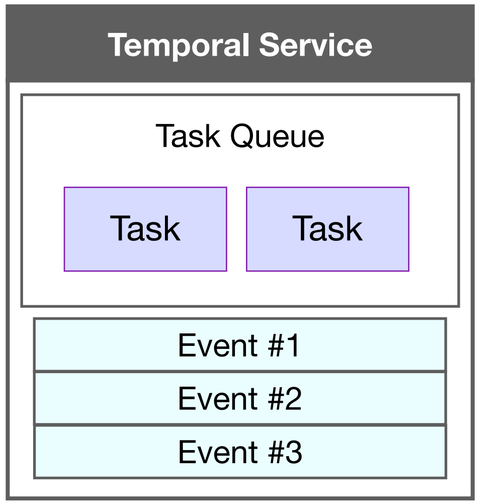


You develop

A Complete Temporal Application



The Temporal Service

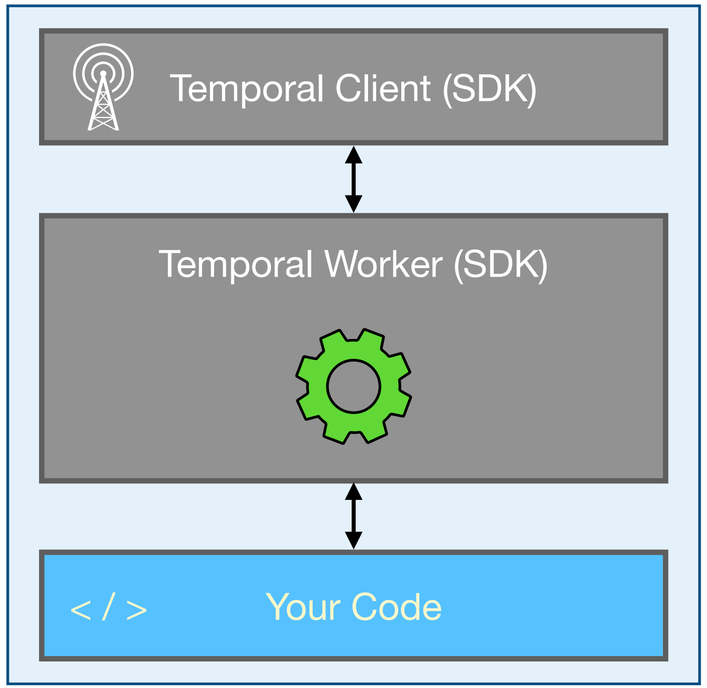


Manages tasks and history

- A backend system that *orchestrates* Workflow Executions (it does not execute your code!)
- Creates tasks that instruct Workers on when and how to run Workflow and Activity code
- Maintains Event Histories that detail progress made by a given Workflow Execution

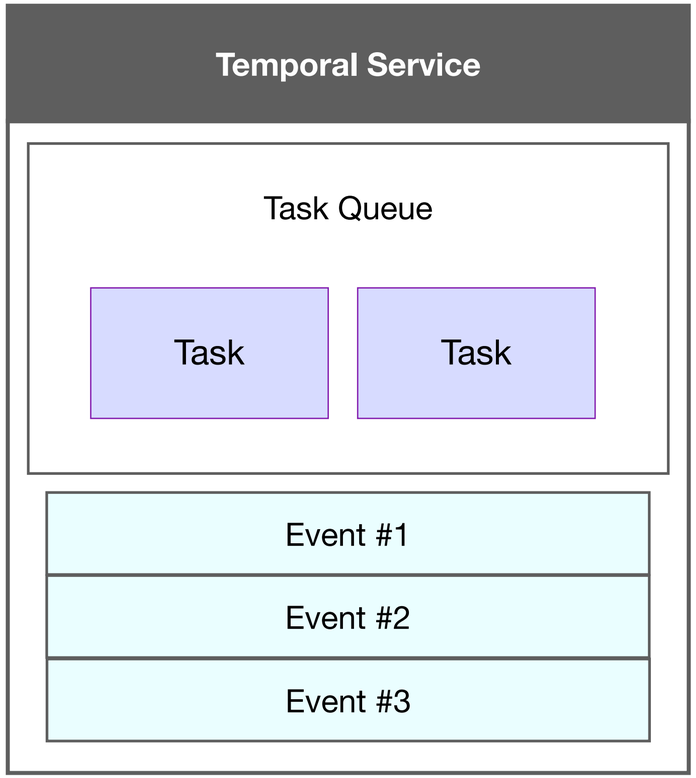
Applications Are External to the Temporal Service

Temporal Application

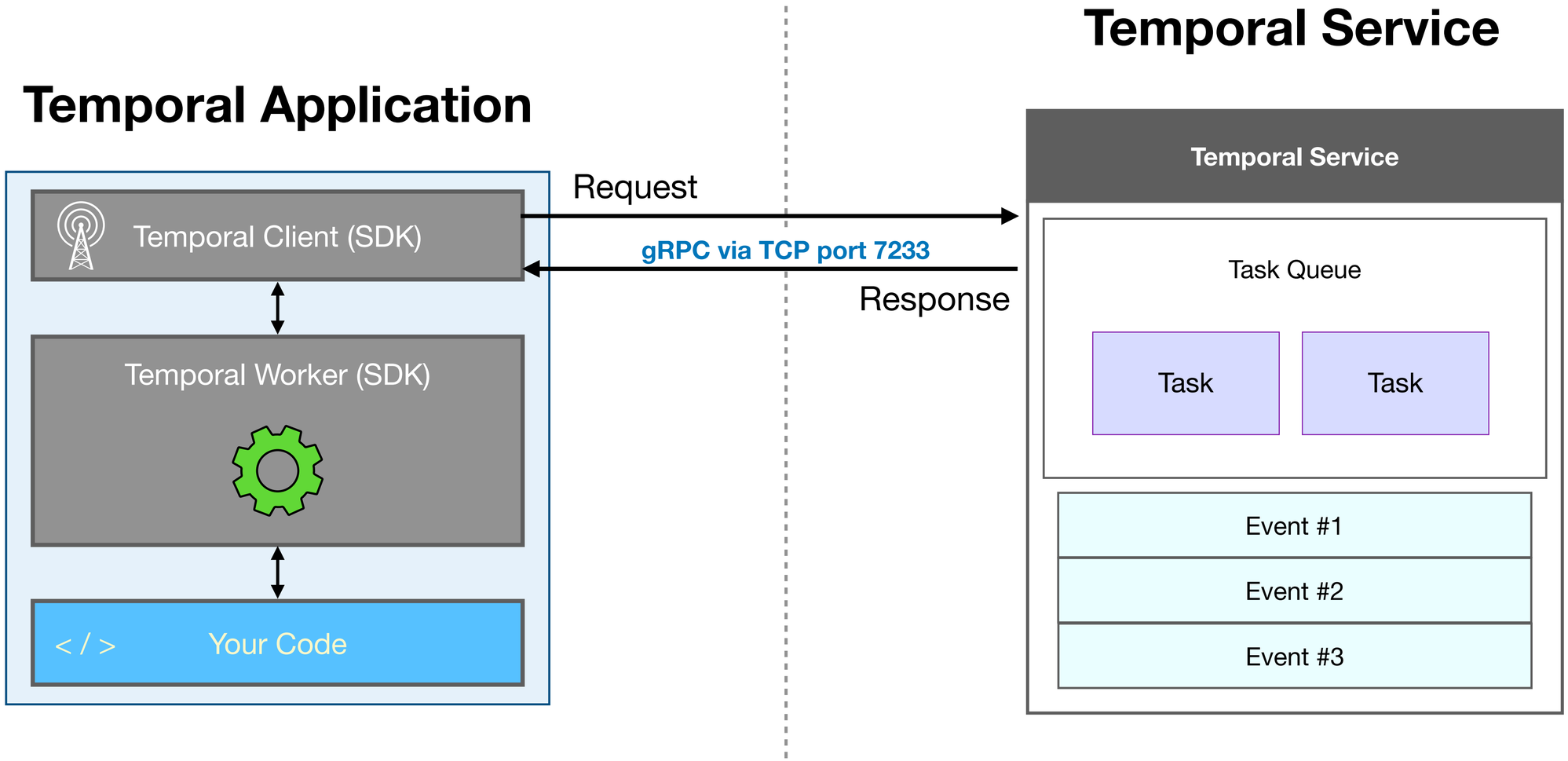


gRPC via TCP port 7233

Temporal Service



Applications Are External to the Temporal Service



Putting It All Together

Putting It All Together

- **Temporal's Durable Execution model gives Workflows long-lived, resilient state**

Putting It All Together

- **Temporal's Durable Execution model gives Workflows long-lived, resilient state**
  - Workflows orchestrate the business logic, but must behave deterministically

Putting It All Together

- **Temporal's Durable Execution model gives Workflows long-lived, resilient state**
  - Workflows orchestrate the business logic, but must behave deterministically
- **Activities are functions that encapsulate non-deterministic operations**

Putting It All Together

- **Temporal's Durable Execution model gives Workflows long-lived, resilient state**
  - Workflows orchestrate the business logic, but must behave deterministically
- **Activities are functions that encapsulate non-deterministic operations**
  - Use them to interact with the outside world (e.g., to call a service or query a database)

Putting It All Together

- **Temporal's Durable Execution model gives Workflows long-lived, resilient state**
  - Workflows orchestrate the business logic, but must behave deterministically
- **Activities are functions that encapsulate non-deterministic operations**
  - Use them to interact with the outside world (e.g., to call a service or query a database)
  - Activities are automatically retried if they fail

Putting It All Together

- **Temporal's Durable Execution model gives Workflows long-lived, resilient state**
  - Workflows orchestrate the business logic, but must behave deterministically
- **Activities are functions that encapsulate non-deterministic operations**
  - Use them to interact with the outside world (e.g., to call a service or query a database)
  - Activities are automatically retried if they fail
- **Temporal Service manages tasks and history of each Workflow Execution**

Putting It All Together

- **Temporal's Durable Execution model gives Workflows long-lived, resilient state**
  - Workflows orchestrate the business logic, but must behave deterministically
- **Activities are functions that encapsulate non-deterministic operations**
  - Use them to interact with the outside world (e.g., to call a service or query a database)
  - Activities are automatically retried if they fail
- **Temporal Service manages tasks and history of each Workflow Execution**
  - Workers poll for tasks, execute code, and report the result back to the Temporal Service

Putting It All Together

- **Temporal's Durable Execution model gives Workflows long-lived, resilient state**
  - Workflows orchestrate the business logic, but must behave deterministically
- **Activities are functions that encapsulate non-deterministic operations**
  - Use them to interact with the outside world (e.g., to call a service or query a database)
  - Activities are automatically retried if they fail
- **Temporal Service manages tasks and history of each Workflow Execution**
  - Workers poll for tasks, execute code, and report the result back to the Temporal Service
  - The Worker and Temporal Service are separate, typically deployed to different machines

Tools

Temporal Command-Line Interface

```
temporal --help
a local Temporal Service, start Workflow Executions,
pass messages to running Workflows, inspect state,
and more.

* Start a local development service:
      temporal server start-dev
* View help: pass --help to any command:
      temporal activity complete --help

Usage:
  temporal [command]

Available Commands:
  activity    Complete or fail an Activity
  batch       Manage running batch jobs
  completion  Generate the autocompletion script for
              the specified shell
...
```

Temporal Command-Line Interface

- **temporal is a CLI for interacting with the Temporal Service**

```
temporal --help
a local Temporal Service, start Workflow Executions,
pass messages to running Workflows, inspect state,
and more.

* Start a local development service:
      temporal server start-dev
* View help: pass --help to any command:
      temporal activity complete --help

Usage:
  temporal [command]

Available Commands:
  activity    Complete or fail an Activity
  batch       Manage running batch jobs
  completion  Generate the autocompletion script for
              the specified shell
...
```

Temporal Command-Line Interface

- **temporal is a CLI for interacting with the Temporal Service**
  - You can use it to start, terminate, and view details of Workflow Executions

```
temporal --help
a local Temporal Service, start Workflow Executions,
pass messages to running Workflows, inspect state,
and more.

* Start a local development service:
      temporal server start-dev
* View help: pass --help to any command:
      temporal activity complete --help

Usage:
  temporal [command]

Available Commands:
  activity    Complete or fail an Activity
  batch       Manage running batch jobs
  completion  Generate the autocompletion script for
              the specified shell
...
```

Temporal Command-Line Interface

- **temporal is a CLI for interacting with the Temporal Service**
  - You can use it to start, terminate, and view details of Workflow Executions
  - You can also use it to run a local Temporal Service for development

```
temporal --help
a local Temporal Service, start Workflow Executions,
pass messages to running Workflows, inspect state,
and more.

* Start a local development service:
      temporal server start-dev
* View help: pass --help to any command:
      temporal activity complete --help

Usage:
  temporal [command]

Available Commands:
  activity    Complete or fail an Activity
  batch       Manage running batch jobs
  completion  Generate the autocompletion script for
              the specified shell
...
```

Temporal Command-Line Interface

- **temporal is a CLI for interacting with the Temporal Service**
  - You can use it to start, terminate, and view details of Workflow Executions
  - You can also use it to run a local Temporal Service for development
  - Append --help to any command or subcommand to see usage info

```
temporal --help
a local Temporal Service, start Workflow Executions,
pass messages to running Workflows, inspect state,
and more.

* Start a local development service:
      temporal server start-dev
* View help: pass --help to any command:
      temporal activity complete --help

Usage:
  temporal [command]

Available Commands:
  activity    Complete or fail an Activity
  batch       Manage running batch jobs
  completion  Generate the autocompletion script for
              the specified shell
...
```

Temporal Command-Line Interface

- **temporal is a CLI for interacting with the Temporal Service**
  - You can use it to start, terminate, and view details of Workflow Executions
  - You can also use it to run a local Temporal Service for development
  - Append --help to any command or subcommand to see usage info
  - See installation instructions in the Temporal documentation

```
temporal --help
a local Temporal Service, start Workflow Executions,
pass messages to running Workflows, inspect state,
and more.

* Start a local development service:
      temporal server start-dev
* View help: pass --help to any command:
      temporal activity complete --help

Usage:
  temporal [command]

Available Commands:
  activity    Complete or fail an Activity
  batch       Manage running batch jobs
  completion  Generate the autocompletion script for
              the specified shell
...
```

Temporal Service for Development

Temporal Service for Development

- **The temporal CLI provides a Temporal Service for local development**

Temporal Service for Development

- **The temporal CLI provides a Temporal Service for local development**
  - Install this CLI tool (using homebrew)

```
$ brew install temporal
```

Temporal Service for Development

- **The temporal CLI provides a Temporal Service for local development**
  - Install this CLI tool (using homebrew)

```
$ brew install temporal
```

  - Start a development cluster (using default settings)

```
$ temporal server start-dev
```

Temporal Service for Development

- **The temporal CLI provides a Temporal Service for local development**
  - Install this CLI tool (using homebrew)

```
$ brew install temporal
```

  - Start a development cluster (using default settings)

```
$ temporal server start-dev
```

  - Start a development cluster (specifying path for durable storage and a custom Web UI port)

```
$ temporal server start-dev \
    --db-filename /Users/mmerrell/dev/mycluster.db \
    --ui-port 8080
```

Viewing History from Web UI

Viewing History from Web UI

- **Temporal's Web UI is a powerful tool for Workflow Execution visibility**

Viewing History from Web UI

- **Temporal's Web UI is a powerful tool for Workflow Execution visibility**
  - Let's you explore the details of current and past execution

Viewing History from Web UI

- **Temporal's Web UI is a powerful tool for Workflow Execution visibility**
  - Let's you explore the details of current and past execution
  - Helps you to identify problems, right down to the line of code that caused them

Viewing History from Web UI

- **Temporal's Web UI is a powerful tool for Workflow Execution visibility**
  - Let's you explore the details of current and past execution
  - Helps you to identify problems, right down to the line of code that caused them
- **The port number used to access it may vary based on how it's deployed**

Viewing History from Web UI

- **Temporal's Web UI is a powerful tool for Workflow Execution visibility**
  - Let's you explore the details of current and past execution
  - Helps you to identify problems, right down to the line of code that caused them
- **The port number used to access it may vary based on how it's deployed**
  - The temporal server start-dev command runs it on port 8233 by default

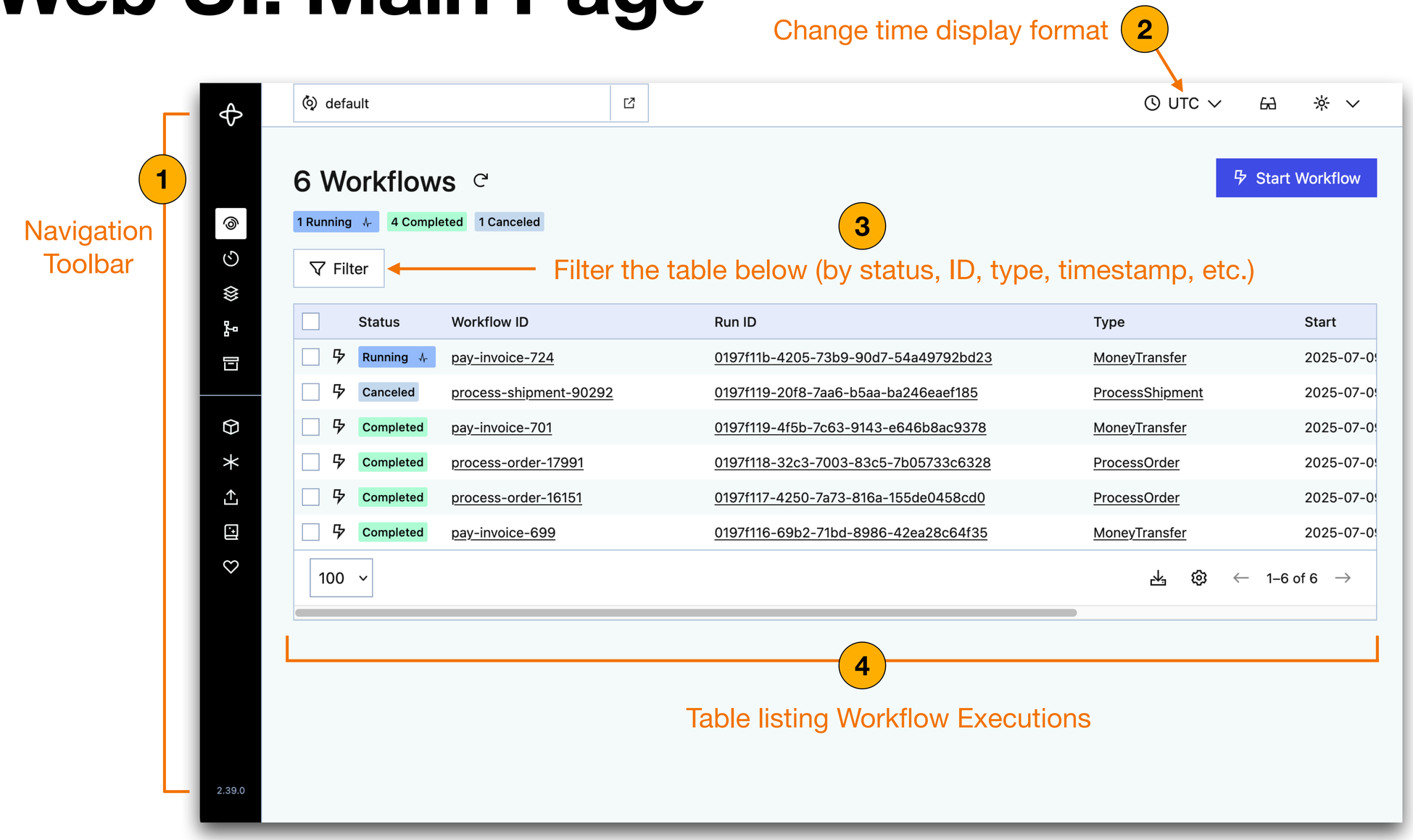
Viewing History from Web UI

- **Temporal's Web UI is a powerful tool for Workflow Execution visibility**
  - Let's you explore the details of current and past execution
  - Helps you to identify problems, right down to the line of code that caused them
- **The port number used to access it may vary based on how it's deployed**
  - The temporal server start-dev command runs it on port 8233 by default
    - You would therefore access it with the URL http://localhost:8233/

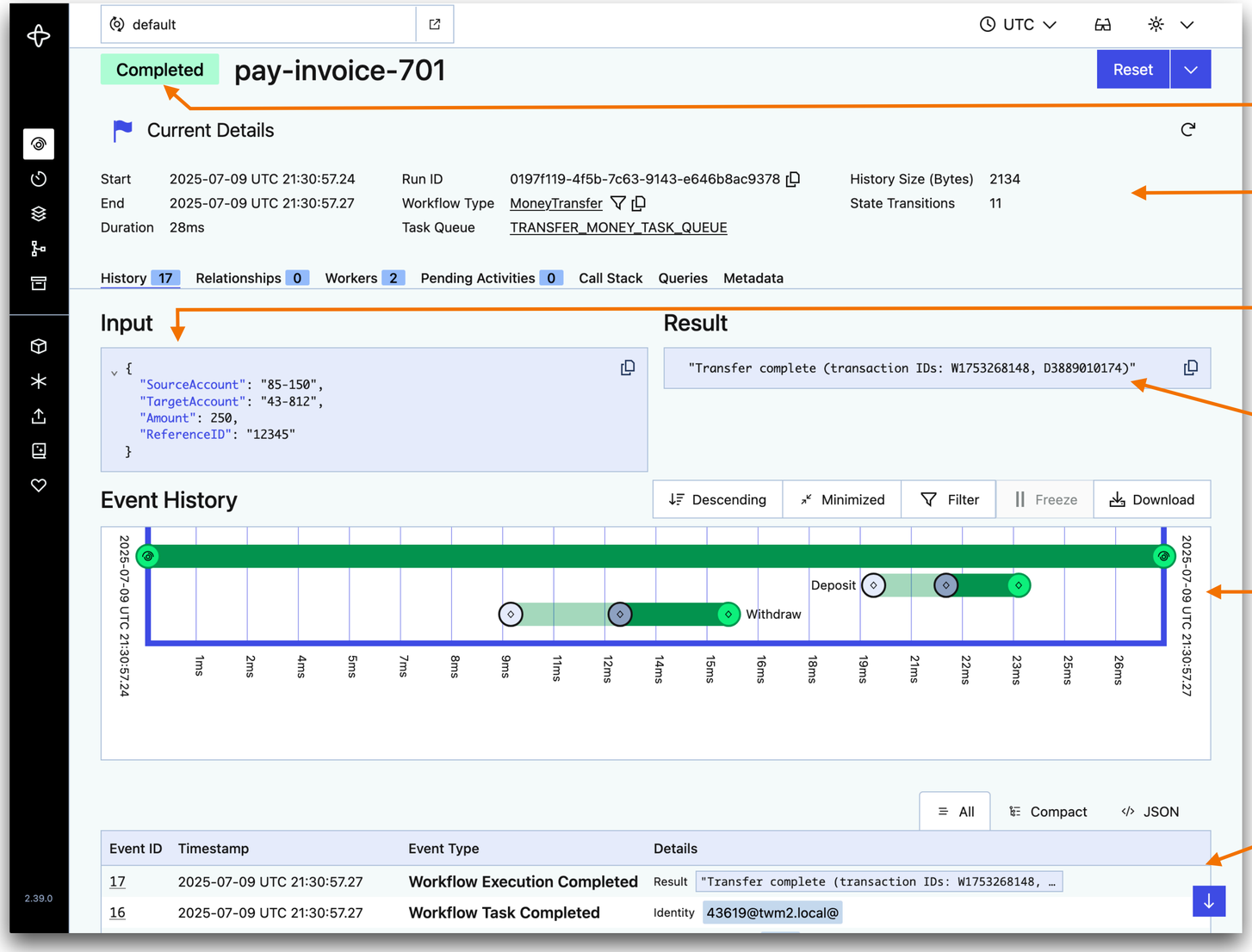
Viewing History from Web UI

- **Temporal's Web UI is a powerful tool for Workflow Execution visibility**
  - Let's you explore the details of current and past execution
  - Helps you to identify problems, right down to the line of code that caused them
- **The port number used to access it may vary based on how it's deployed**
  - The temporal server start-dev command runs it on port 8233 by default
    - You would therefore access it with the URL http://localhost:8233/
    - You can override the port number with a command-line option (example: --ui-port 8080)

Web UI: Main Page



Web UI: Detail Page



1. Status
2. Workflow Execution details
3. Input data passed to Workflow Execution
4. Output data returned by Workflow Executio
5. Timeline display
6. Event History table

Executing a Basic Workflow

Executing a Workflow from the Command Line

Executing a Workflow from the Command Line

- One way to start a Workflow is with `temporal workflow start`

```
$ temporal workflow start \
    --type GreetSomeone \
    --task-queue greeting-tasks \
    --workflow-id my-first-workflow \
    --input '"Mason"'

Running execution:
  WorkflowId  my-first-workflow
  RunId       ab62c808-44d7-4b3e-97ef-777493c4da09
  Type        GreetSomeone
  Namespace   default
  TaskQueue   greeting-tasks
```

Executing a Workflow from the Command Line

- **One way to start a Workflow is with `temporal workflow start`**
  - The type value is the name of your Workflow, by default the name of the class

```
$ temporal workflow start \
    --type GreetSomeone \
    --task-queue greeting-tasks \
    --workflow-id my-first-workflow \
    --input '"Mason"'

Running execution:
  WorkflowId  my-first-workflow
  RunId       ab62c808-44d7-4b3e-97ef-777493c4da09
  Type        GreetSomeone
  Namespace   default
  TaskQueue   greeting-tasks
```

Executing a Workflow from the Command Line

- **One way to start a Workflow is with `temporal workflow start`**
  - The type value is the name of your Workflow, by default the name of the class
  - The task-queue value must match the value specified in your Worker initialization code

```
$ temporal workflow start \
    --type GreetSomeone \
    --task-queue greeting-tasks \
    --workflow-id my-first-workflow \
    --input '"Mason"'

Running execution:
  WorkflowId  my-first-workflow
  RunId       ab62c808-44d7-4b3e-97ef-777493c4da09
  Type        GreetSomeone
  Namespace   default
  TaskQueue   greeting-tasks
```

Executing a Workflow from the Command Line

- **One way to start a Workflow is with `temporal workflow start`**
  - The type value is the name of your Workflow, by default the name of the class
  - The task-queue value must match the value specified in your Worker initialization code
  - The workflow-id is a user-defined identifier, which typically has some business meaning

```
$ temporal workflow start \
    --type GreetSomeone \
    --task-queue greeting-tasks \
    --workflow-id my-first-workflow \
    --input '"Mason"'

Running execution:
  WorkflowId  my-first-workflow
  RunId       ab62c808-44d7-4b3e-97ef-777493c4da09
  Type        GreetSomeone
  Namespace   default
  TaskQueue   greeting-tasks
```

Executing a Workflow from the Command Line

- **One way to start a Workflow is with `temporal workflow start`**
  - The type value is the name of your Workflow, by default the name of the class
  - The task-queue value must match the value specified in your Worker initialization code
  - The workflow-id is a user-defined identifier, which typically has some business meaning
  - The input argument's value must be JSON and passed as Workflow method parameter

```
$ temporal workflow start \
    --type GreetSomeone \
    --task-queue greeting-tasks \
    --workflow-id my-first-workflow \
    --input '"Mason"'

Running execution:
  WorkflowId  my-first-workflow
  RunId       ab62c808-44d7-4b3e-97ef-777493c4da09
  Type        GreetSomeone
  Namespace   default
  TaskQueue   greeting-tasks
```

Executing a Workflow from Application Code (1)

```
package greeting;

import io.temporal.client.WorkflowClient;
import io.temporal.client.WorkflowOptions;
import io.temporal.client.WorkflowStub;
import io.temporal.serviceclient.WorkflowServiceStubs;

public class Starter {
  public static void main(String[] args) throws Exception {

    WorkflowServiceStubs service =
          WorkflowServiceStubs.newLocalServiceStubs();

    WorkflowClient client = WorkflowClient.newInstance(service);

    WorkflowOptions options = WorkflowOptions.newBuilder()
              .setWorkflowId("my-first-workflow")
              .setTaskQueue("greeting-tasks")
              .build();

    Greeting workflow = client.newWorkflowStub(GreetingWorkflow.class,
                                               options);

    // code continued on next slide
```

Executing a Workflow from Application Code (1)

- **An alternative to using temporal is to execute the Workflow from code**

```
package greeting;

import io.temporal.client.WorkflowClient;
import io.temporal.client.WorkflowOptions;
import io.temporal.client.WorkflowStub;
import io.temporal.serviceclient.WorkflowServiceStubs;

public class Starter {
  public static void main(String[] args) throws Exception {

    WorkflowServiceStubs service =
            WorkflowServiceStubs.newLocalServiceStubs();

    WorkflowClient client = WorkflowClient.newInstance(service);

    WorkflowOptions options = WorkflowOptions.newBuilder()
                .setWorkflowId("my-first-workflow")
                .setTaskQueue("greeting-tasks")
                .build();

    Greeting workflow = client.newWorkflowStub(GreetingWorkflow.class,
                                                    options);

    // code continued on next slide
```

Executing a Workflow from Application Code (1)

- **An alternative to using temporal is to execute the Workflow from code**
  - This provides a way of integrating Temporal into your own applications

```
package greeting;

import io.temporal.client.WorkflowClient;
import io.temporal.client.WorkflowOptions;
import io.temporal.client.WorkflowStub;
import io.temporal.serviceclient.WorkflowServiceStubs;

public class Starter {
  public static void main(String[] args) throws Exception {

    WorkflowServiceStubs service =
            WorkflowServiceStubs.newLocalServiceStubs();

    WorkflowClient client = WorkflowClient.newInstance(service);

    WorkflowOptions options = WorkflowOptions.newBuilder()
                .setWorkflowId("my-first-workflow")
                .setTaskQueue("greeting-tasks")
                .build();

    Greeting workflow = client.newWorkflowStub(GreetingWorkflow.class,
                                                    options);

    // code continued on next slide
```

Executing a Workflow from Application Code (1)

- **An alternative to using temporal is to execute the Workflow from code**
  - This provides a way of integrating Temporal into your own applications
  - You can do this in three steps

```
package greeting;

import io.temporal.client.WorkflowClient;
import io.temporal.client.WorkflowOptions;
import io.temporal.client.WorkflowStub;
import io.temporal.serviceclient.WorkflowServiceStubs;

public class Starter {
  public static void main(String[] args) throws Exception {

    WorkflowServiceStubs service =
          WorkflowServiceStubs.newLocalServiceStubs();

    WorkflowClient client = WorkflowClient.newInstance(service);

    WorkflowOptions options = WorkflowOptions.newBuilder()
              .setWorkflowId("my-first-workflow")
              .setTaskQueue("greeting-tasks")
              .build();

    Greeting workflow = client.newWorkflowStub(GreetingWorkflow.class,
                                               options);

    // code continued on next slide
```

Executing a Workflow from Application Code (1)

- **An alternative to using temporal is to execute the Workflow from code**
  - This provides a way of integrating Temporal into your own applications
  - You can do this in three steps
    - Import Client APIs from the SDK

```
package greeting;

import io.temporal.client.WorkflowClient;
import io.temporal.client.WorkflowOptions;
import io.temporal.client.WorkflowStub;
import io.temporal.serviceclient.WorkflowServiceStubs;

public class Starter {
  public static void main(String[] args) throws Exception {

    WorkflowServiceStubs service =
            WorkflowServiceStubs.newLocalServiceStubs();

    WorkflowClient client = WorkflowClient.newInstance(service);

    WorkflowOptions options = WorkflowOptions.newBuilder()
                .setWorkflowId("my-first-workflow")
                .setTaskQueue("greeting-tasks")
                .build();

    Greeting workflow = client.newWorkflowStub(GreetingWorkflow.class,
                                                options);

    // code continued on next slide
```

Executing a Workflow from Application Code (1)

- **An alternative to using temporal is to execute the Workflow from code**
  - This provides a way of integrating Temporal into your own applications
  - You can do this in three steps
    - Import Client APIs from the SDK
    - Create and configure a Client

```
package greeting;

import io.temporal.client.WorkflowClient;
import io.temporal.client.WorkflowOptions;
import io.temporal.client.WorkflowStub;
import io.temporal.serviceclient.WorkflowServiceStubs;

public class Starter {
  public static void main(String[] args) throws Exception {

    WorkflowServiceStubs service =
            WorkflowServiceStubs.newLocalServiceStubs();

    WorkflowClient client = WorkflowClient.newInstance(service);

    WorkflowOptions options = WorkflowOptions.newBuilder()
                .setWorkflowId("my-first-workflow")
                .setTaskQueue("greeting-tasks")
                .build();

    Greeting workflow = client.newWorkflowStub(GreetingWorkflow.class,
                                                options);

    // code continued on next slide
```

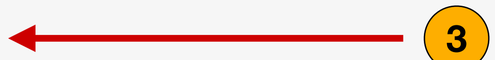
Executing a Workflow from Application Code (2)

```
        WorkflowOptions options = WorkflowOptions.newBuilder()
                .setWorkflowId("my-first-workflow")
                .setTaskQueue("greeting-tasks")
                .build();

        Greeting workflow = client.newWorkflowStub(GreetingWorkflow.class, options);

        String greeting = workflow.greetSomeone(args[0]);

        String workflowId = WorkflowStub.fromTyped(workflow).getExecution().getWorkflowId();

        System.out.println(workflowId + " " + greeting);

    }
}
```

Executing a Workflow from Application Code (2)

- Use the API to request execution

```
        WorkflowOptions options = WorkflowOptions.newBuilder()
                .setWorkflowId("my-first-workflow")
                .setTaskQueue("greeting-tasks")
                .build();

        Greeting workflow = client.newWorkflowStub(GreetingWorkflow.class, options);

        String greeting = workflow.greetSomeone(args[0]);

        String workflowId = WorkflowStub.fromTyped(workflow).getExecution().getWorkflowId();

        System.out.println(workflowId + " " + greeting);

    }
}
```



Executing a Workflow from Application Code (2)

- Use the API to request execution
- We will use similar code to run Workflows in later exercises

```
        WorkflowOptions options = WorkflowOptions.newBuilder()
                .setWorkflowId("my-first-workflow")
                .setTaskQueue("greeting-tasks")
                .build();

        Greeting workflow = client.newWorkflowStub(GreetingWorkflow.class, options);

        String greeting = workflow.greetSomeone(args[0]);

        String workflowId = WorkflowStub.fromTyped(workflow).getExecution().getWorkflowId();

        System.out.println(workflowId + " " + greeting);

    }
}
```

Making Changes to a Workflow

Making Changes to a Workflow

- **Backwards compatibility is an important consideration in Temporal**

Making Changes to a Workflow

- **Backwards compatibility is an important consideration in Temporal**
- **Avoid changing the number or types of input parameters**

Making Changes to a Workflow

- **Backwards compatibility is an important consideration in Temporal**
- **Avoid changing the number or types of input parameters**
  - We recommend that your Workflow uses a class as the only input parameter

Making Changes to a Workflow

- **Backwards compatibility is an important consideration in Temporal**
- **Avoid changing the number or types of input parameters**
  - We recommend that your Workflow uses a class as the only input parameter
  - Changing the fields used to create the class does not change its type

Making Changes to a Workflow

- **Backwards compatibility is an important consideration in Temporal**
- **Avoid changing the number or types of input parameters**
  - We recommend that your Workflow uses a class as the only input parameter
  - Changing the fields used to create the class does not change its type
- **You must also ensure that your Workflow is *deterministic***

Making Changes to a Workflow

- **Backwards compatibility is an important consideration in Temporal**
- **Avoid changing the number or types of input parameters**
  - We recommend that your Workflow uses a class as the only input parameter
  - Changing the fields used to create the class does not change its type
- **You must also ensure that your Workflow is *deterministic***
  - Each execution of a given Workflow must produce the same output, given the same input

Making Changes to a Workflow

- **Backwards compatibility is an important consideration in Temporal**
- **Avoid changing the number or types of input parameters**
  - We recommend that your Workflow uses a class as the only input parameter
  - Changing the fields used to create the class does not change its type
- **You must also ensure that your Workflow is *deterministic***
  - Each execution of a given Workflow must produce the same output, given the same input
  - Handle non-deterministic operations using SDK-provided alternatives and Temporal features

Making Changes to a Workflow

- **Backwards compatibility is an important consideration in Temporal**
- **Avoid changing the number or types of input parameters**
  - We recommend that your Workflow uses a class as the only input parameter
  - Changing the fields used to create the class does not change its type
- **You must also ensure that your Workflow is *deterministic***
  - Each execution of a given Workflow must produce the same output, given the same input
  - Handle non-deterministic operations using SDK-provided alternatives and Temporal features
  - Restart Workers after making changes for consistency in Workflow Executions.

Exercise Environment Orientation

- You’re just about to begin the first hands-on exercise
- Follow the link below to access the Instruqt environment
- We will explain and demonstrate the exercise environment

https://t.mp/replay26-workshop-java

Exercise #1: Hello Workflow

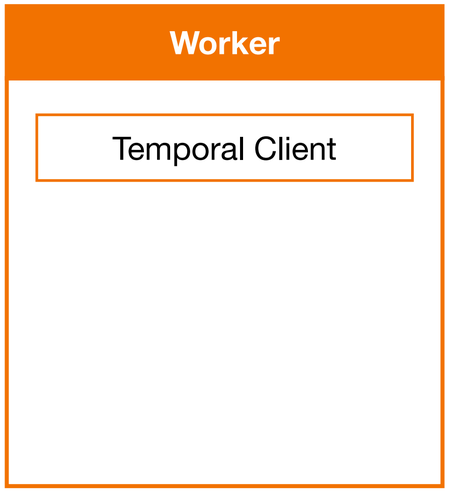
https://t.mp/replay26-workshop-java

- **During this exercise, you will**
  - Review the business logic of the provided Workflow Definition to understand its behavior
  - Modify the Worker initialization code to specify a Task Queue name (**greeting-tasks**)
  - Run the Worker initialization code to start the Worker process
  - Use **temporal** to execute the Workflow from the command line, specifying your name as input
- **Refer to the README.md file in the exercise environment for details**
  - The code is below the **exercises/hello-workflow** directory
    - Make your changes to the code in the **practice** subdirectory (look for TODO comments)
    - If you need a hint or want to verify your changes, look at the complete version in the **solution** subdirectory

Understanding Workflow Execution

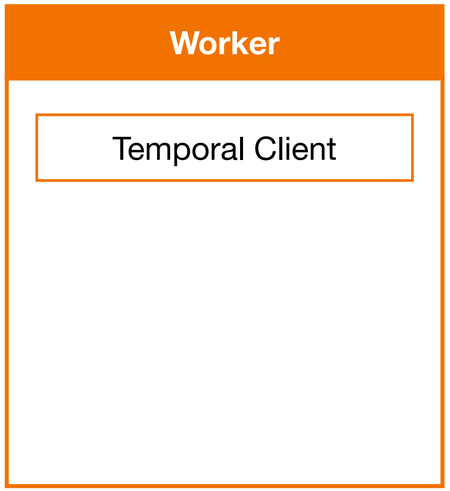
Actors in this Workflow Execution Scenario

Actors in this Workflow Execution Scenario

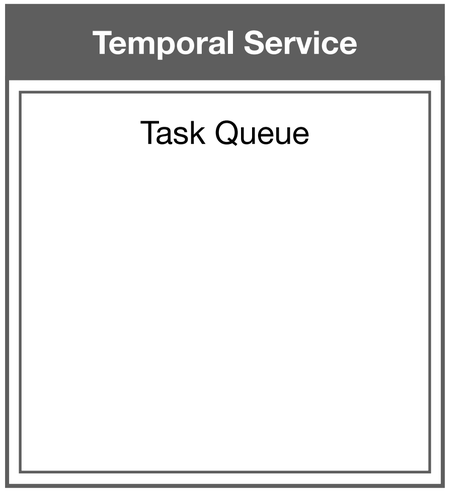


Executes the code

Actors in this Workflow Execution Scenario

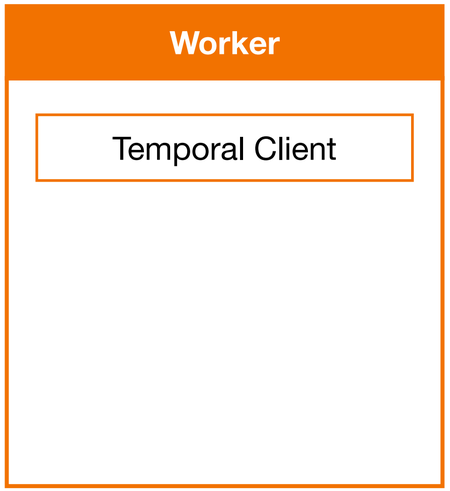


Executes the code

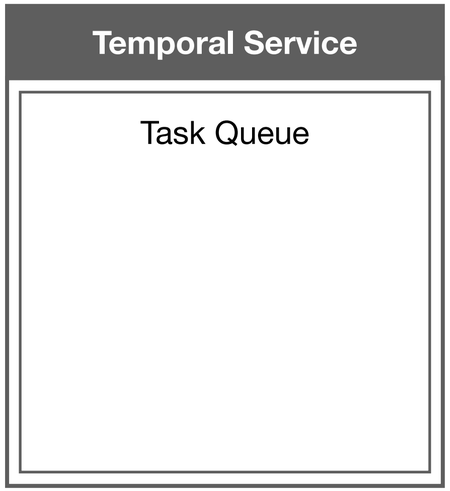


Orchestrates code execution

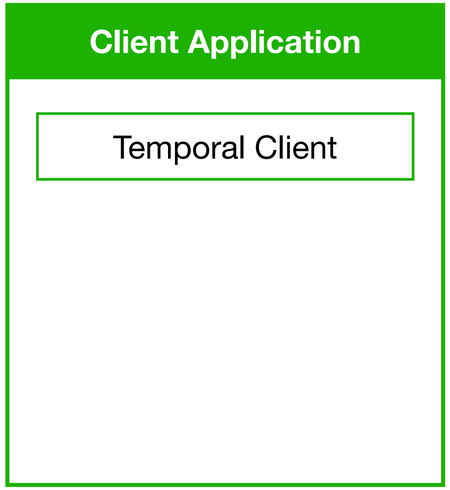
Actors in this Workflow Execution Scenario



Executes the code



Orchestrates code execution



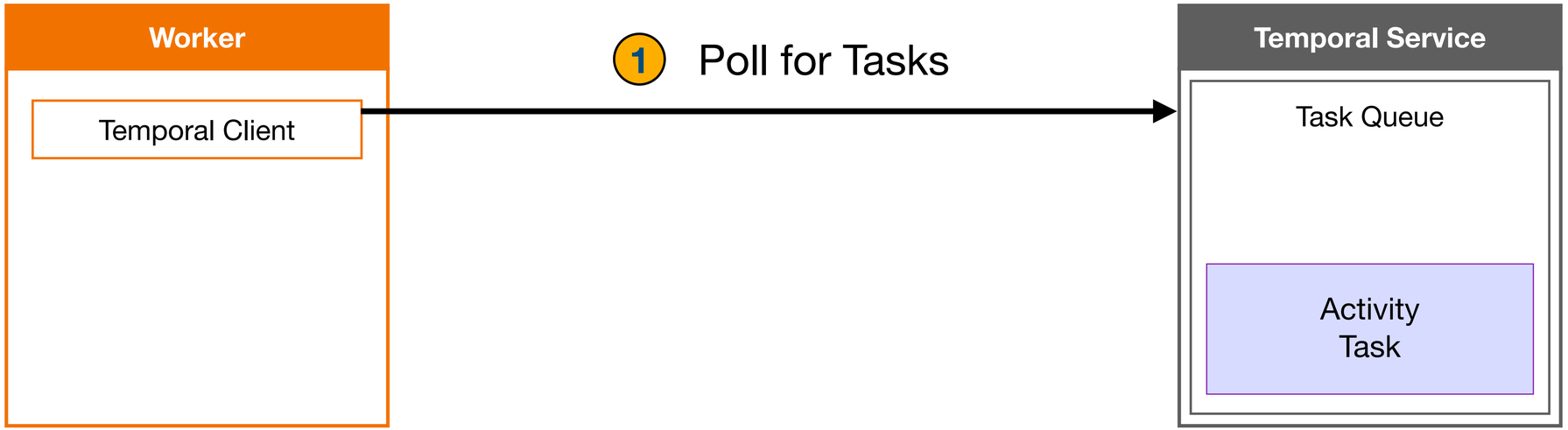
Requests code execution
and retrieves the result

Workers and Tasks

Workers and Tasks

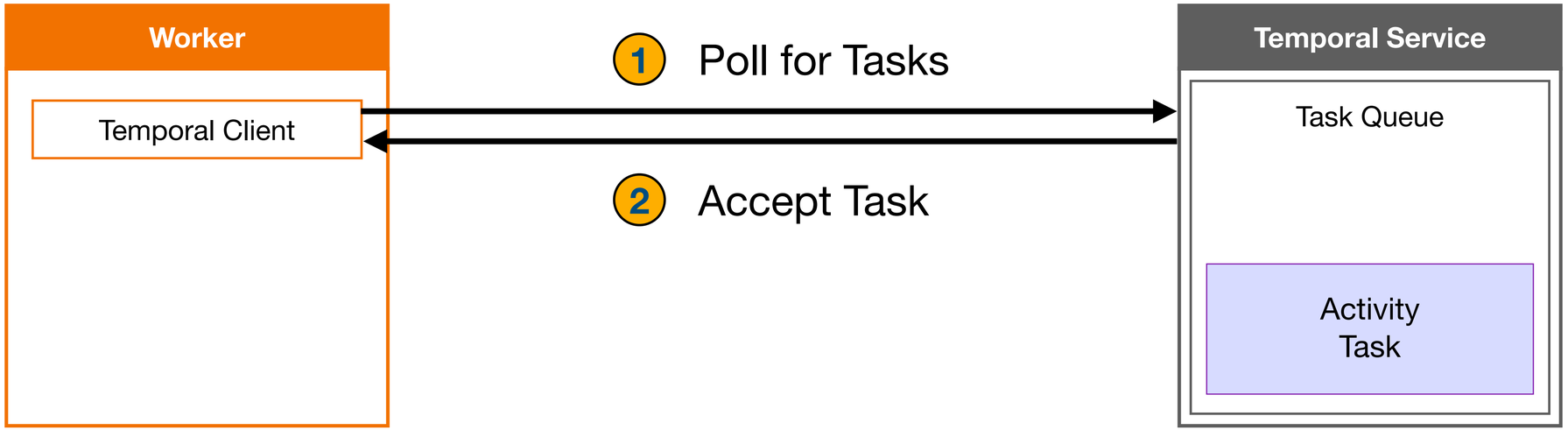
- Temporal does not assign tasks to Workers

Workers and Tasks



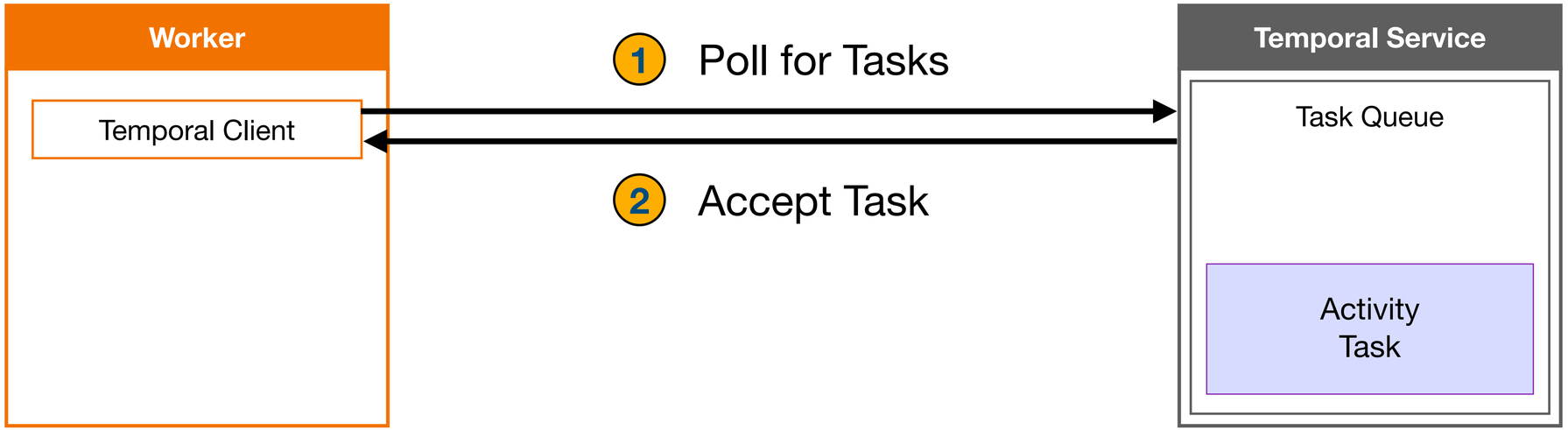
- Temporal does not assign tasks to Workers
- Workers continuously poll, accepting tasks when they have spare capacity

Workers and Tasks



- Temporal does not assign tasks to Workers
- Workers continuously poll, accepting tasks when they have spare capacity

Workers and Tasks



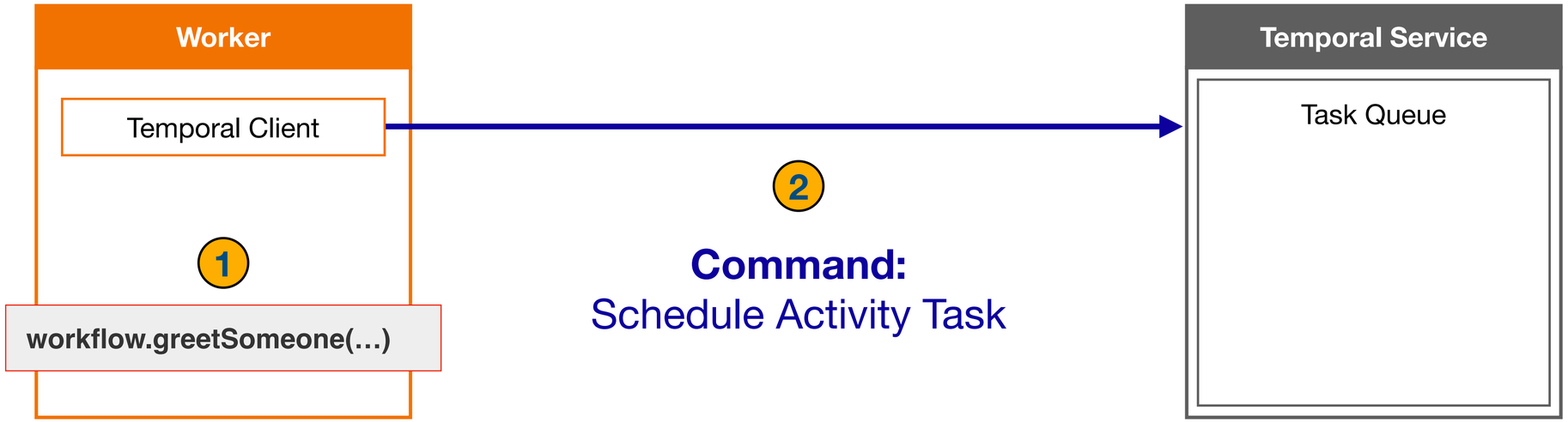
- Temporal does not assign tasks to Workers
- Workers continuously poll, accepting tasks when they have spare capacity
- You can increase throughput and scalability by adding Workers

Commands

Commands

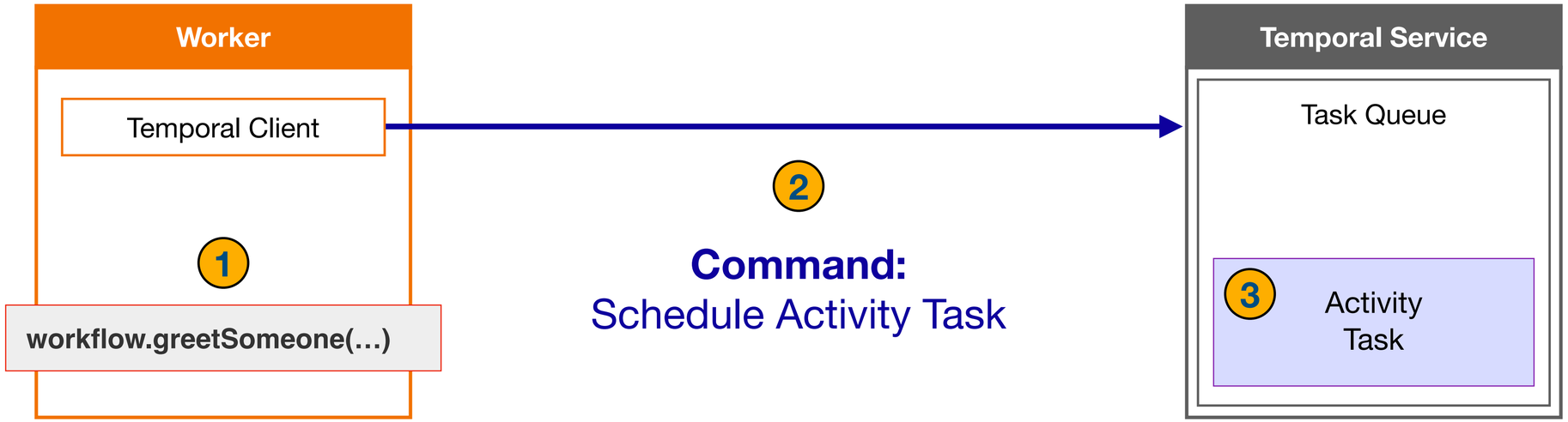
Worker
Temporal Client
1
workflow.greetSomeone(…)
Temporal Service
Task Queue

Commands



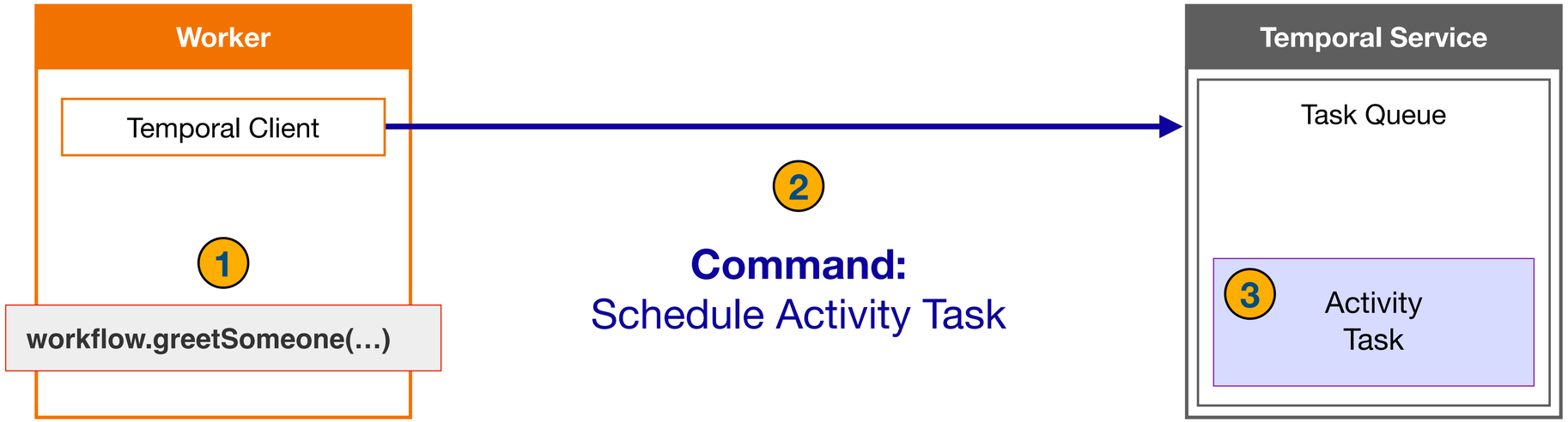
- Certain API calls result in the Worker issuing a Command to the Temporal Service

Commands



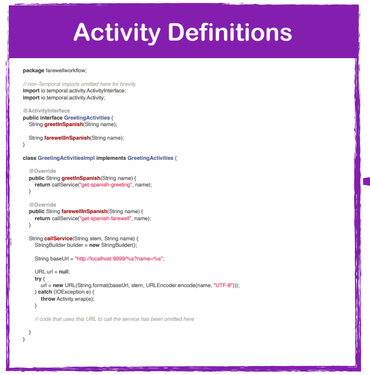
- Certain API calls result in the Worker issuing a Command to the Temporal Service
- The Service acts on these Commands, but also stores them

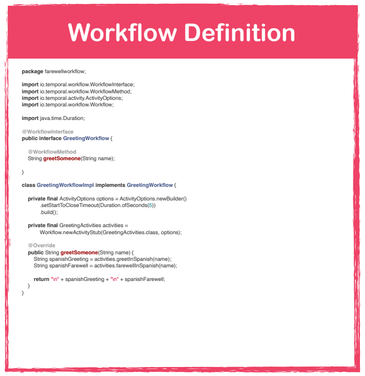
Commands

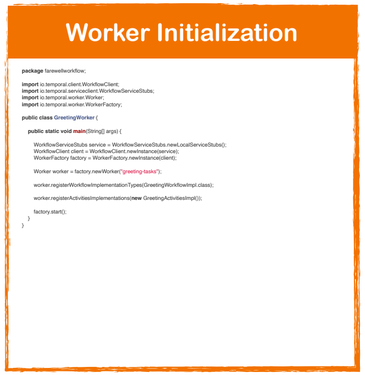


- Certain API calls result in the Worker issuing a Command to the Temporal Service
- The Service acts on these Commands, but also stores them
- This allows the Worker to recreate the state of a Workflow Execution following a crash

Activity Definitions








```java
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
    String greetInSpanish(String name);

    String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        try {
            url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
        } catch (IOException e) {
            throw Activity.wrap(e);
        }

        // code that uses this URL to call the service has been omitted here

    }
}
```

This is just a utility method for calling the microservice and is not specific to Temporal

Workflow Definition

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ru/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ei) {
          throw Activity.wrap(ei);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
      return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ru?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ex) {
          throw Activity.wrap(ex);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

    public static void main(String[] args) {

        WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
        WorkflowClient client = WorkflowClient.newInstance(service);
        WorkerFactory factory = WorkerFactory.newInstance(client);

        Worker worker = factory.newWorker("greeting-tasks");

        worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

        worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

        factory.start();
    }
}
```

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ei) {
          throw Activity.wrap(ei);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

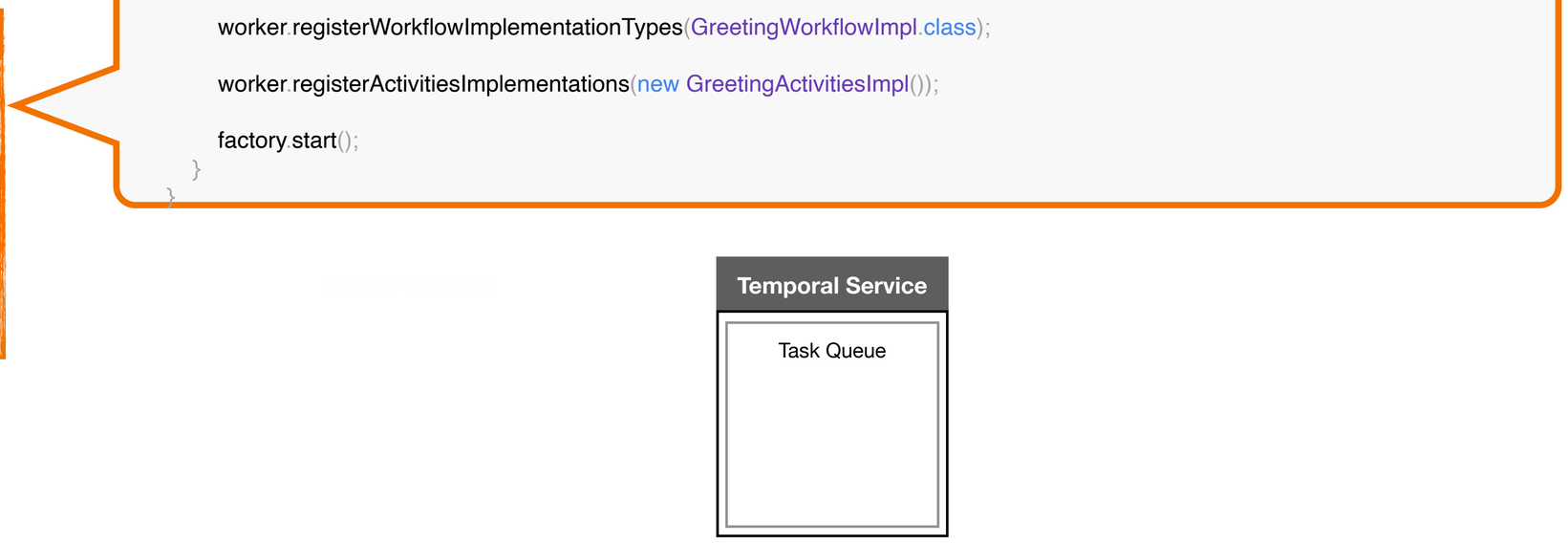
Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ei) {
          throw Activity.wrap(ei);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Launch

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Worker Process

Temporal Service

Task Queue

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ei) {
          throw Activity.wrap(ei);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

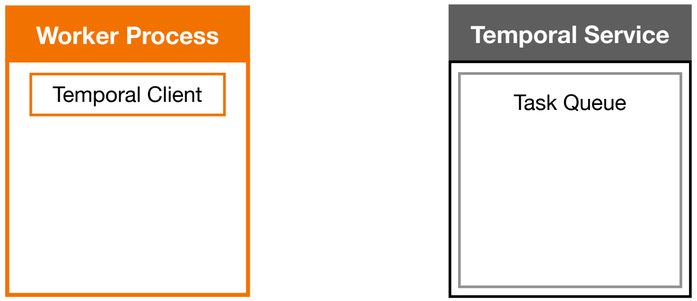
Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

    public static void main(String[] args) {

        WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
        WorkflowClient client = WorkflowClient.newInstance(service);
        WorkerFactory factory = WorkerFactory.newInstance(client);

        Worker worker = factory.newWorker("greeting-tasks");

        worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

        worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

        factory.start();
    }
}
```



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ei) {
          throw Activity.wrap(ei);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

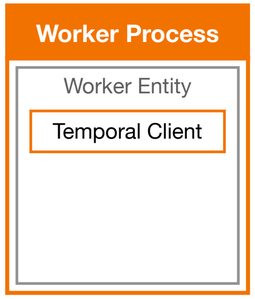
Workflow Definition

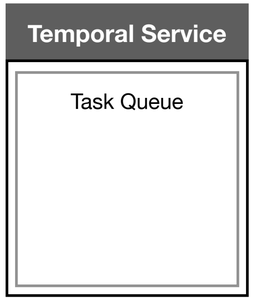
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

    public static void main(String[] args) {

        WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
        WorkflowClient client = WorkflowClient.newInstance(service);
        WorkerFactory factory = WorkerFactory.newInstance(client);

        Worker worker = factory.newWorker("greeting-tasks");

        worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

        worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

        factory.start();
    }
}
```





Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ei) {
          throw Activity.wrap(ei);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

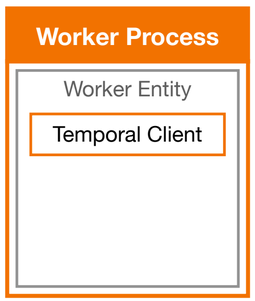
Workflow Definition

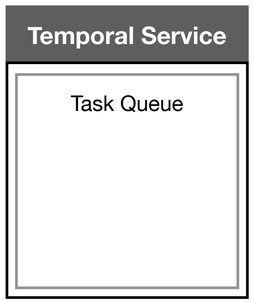
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

    public static void main(String[] args) {

        WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
        WorkflowClient client = WorkflowClient.newInstance(service);
        WorkerFactory factory = WorkerFactory.newInstance(client);

        Worker worker = factory.newWorker("greeting-tasks");

        worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

        worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

        factory.start();
    }
}
```





Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/"+s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ei) {
          throw Activity.wrap(ei);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

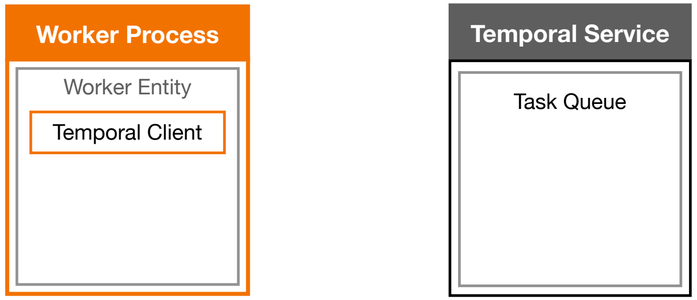
Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

    public static void main(String[] args) {

        WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
        WorkflowClient client = WorkflowClient.newInstance(service);
        WorkerFactory factory = WorkerFactory.newInstance(client);

        Worker worker = factory.newWorker("greeting-tasks");

        worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

        worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

        factory.start();
    }
}
```



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ei) {
          throw Activity.wrap(ei);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

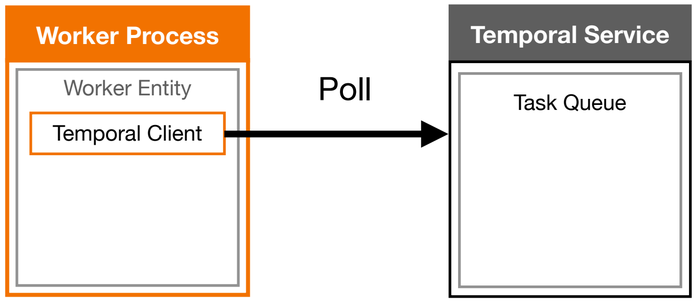
Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

    public static void main(String[] args) {

        WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
        WorkflowClient client = WorkflowClient.newInstance(service);
        WorkerFactory factory = WorkerFactory.newInstance(client);

        Worker worker = factory.newWorker("greeting-tasks");

        worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

        worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

        factory.start();
    }
}
```



Launching from Command Line

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
    StringBuilder builder = new StringBuilder();

    String baseUrl = "http://localhost:9999/"+a?name=%a";

    URL url = null;
    try {
      url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
    } catch (IOException ex) {
      throw Activity.wrap(ex);
    }

    // code that uses this URL to call the service has been omitted here

  }
}
```

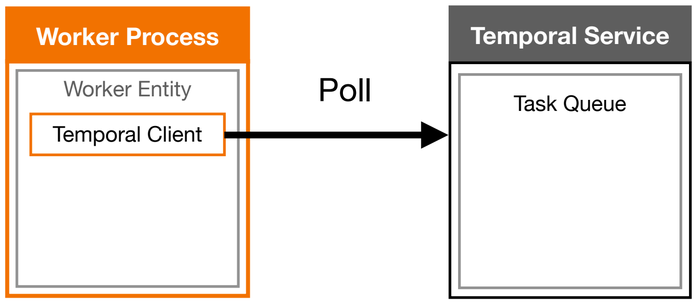
Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final GreetingActivities activities =
      Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
    String spanishFarewell = activities.farewellInSpanish(name);

    return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

    WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
    WorkflowClient client = WorkflowClient.newInstance(service);
    WorkerFactory factory = WorkerFactory.newInstance(client);

    Worker worker = factory.newWorker("greeting-tasks");

    worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

    worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

    factory.start();
  }
}
```

```
$ temporal workflow start \
    --task-queue greeting-tasks \
    --workflow-id greeting-workflow \
    --workflow-type GreetingWorkflow \
    --input '"Mason"'
```



Launching from Command Line

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
    StringBuilder builder = new StringBuilder();

    String baseUrl = "http://localhost:9999/%s?name=%s";

    URL url = null;
    try {
      url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
    } catch (IOException ex) {
      throw Activity.wrap(ex);
    }

    // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

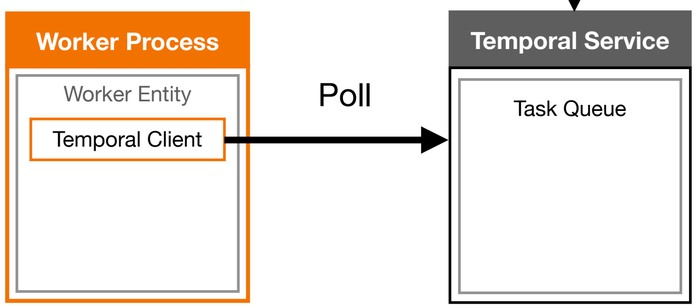
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final GreetingActivities activities =
      Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
    String spanishFarewell = activities.farewellInSpanish(name);

    return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

    WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
    WorkflowClient client = WorkflowClient.newInstance(service);
    WorkerFactory factory = WorkerFactory.newInstance(client);

    Worker worker = factory.newWorker("greeting-tasks");

    worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

    worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

    factory.start();
  }
}
```

```
$ temporal workflow start \
    --task-queue greeting-tasks \
    --workflow-id greeting-workflow \
    --workflow-type GreetingWorkflow \
    --input '"Mason"'
```

Workflow Execution Request



Launching from Application Code

Activity Definitions

```
package tarewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
    StringBuilder builder = new StringBuilder();

    String baseUrl = "http://localhost:9999/"+stem+"/";

    URL url = null;
    try {
      url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
    } catch (IOException ex) {
        throw Activity.wrap(ex);
    }

    // code that uses this URL to call the service has been omitted here

  }
}
```

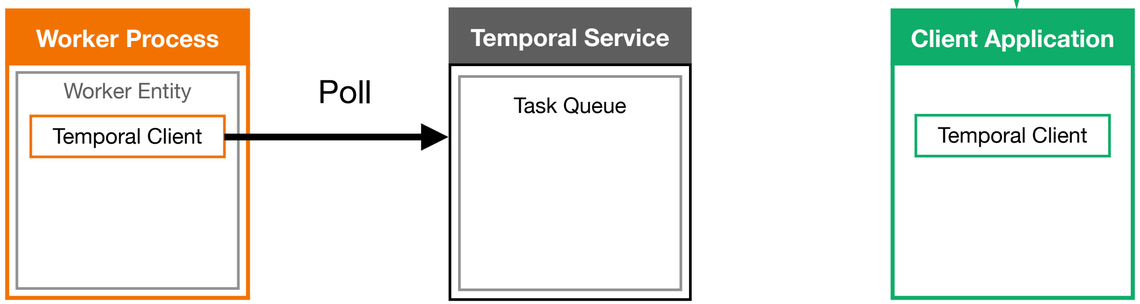
Workflow Definition

```
package tarewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
    String spanishFarewell = activities.farewellInSpanish(name);

    return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package tarewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

    WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
    WorkflowClient client = WorkflowClient.newInstance(service);
    WorkerFactory factory = WorkerFactory.newInstance(client);

    Worker worker = factory.newWorker("greeting-tasks");

    worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

    worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

    factory.start();
  }
}
```

```
// ... this is code within your own application (for example a web or mobile app)

WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
WorkflowClient client = WorkflowClient.newInstance(service);

WorkflowOptions options = WorkflowOptions.newBuilder()
        .setWorkflowId("greeting-workflow")
        .setTaskQueue("greeting-tasks")
        .build();

GreetingWorkflow workflow = client.newWorkflowStub(GreetingWorkflow.class, options);

String greeting = workflow.greetSomeone(args[0]);

String workflowId = WorkflowStub.fromTyped(workflow).getExecution().getWorkflowId();

System.out.println(workflowId + " " + greeting);

// ... other application-specific code might follow
```



Launching from Application Code

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
    StringBuilder builder = new StringBuilder();

    String baseUrl = "http://localhost:9999/rs/";name=%s";

    URL url = null;
    try {
      url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
    } catch (IOException ei) {
        throw Activity.wrap(ei);
    }

    // code that uses this URL to call the service has been omitted here

  }
}
```

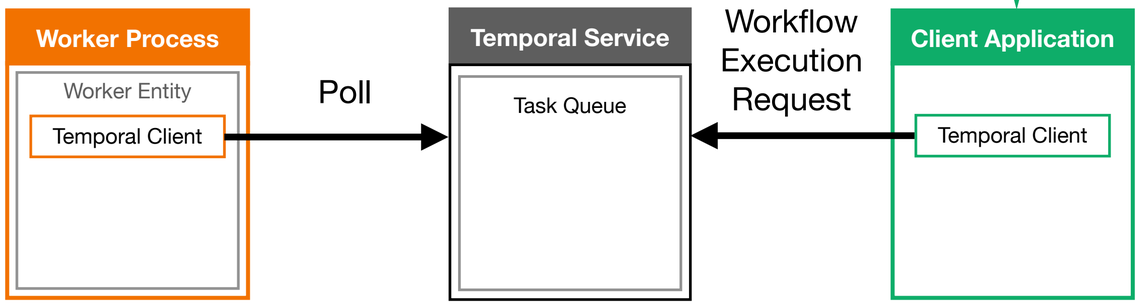
Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
    String spanishFarewell = activities.farewellInSpanish(name);

    return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

    WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
    WorkflowClient client = WorkflowClient.newInstance(service);
    WorkerFactory factory = WorkerFactory.newInstance(client);

    Worker worker = factory.newWorker("greeting-tasks");

    worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

    worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

    factory.start();
  }
}
```

```
// ... this is code within your own application (for example a web or mobile app)

WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
WorkflowClient client = WorkflowClient.newInstance(service);

WorkflowOptions options = WorkflowOptions.newBuilder()
        .setWorkflowId("greeting-workflow")
        .setTaskQueue("greeting-tasks")
        .build();

GreetingWorkflow workflow = client.newWorkflowStub(GreetingWorkflow.class, options);

String greeting = workflow.greetSomeone(args[0]);

String workflowId = WorkflowStub.fromTyped(workflow).getExecution().getWorkflowId();

System.out.println(workflowId + " " + greeting);

// ... other application-specific code might follow
```



Event History

Activity Definitions

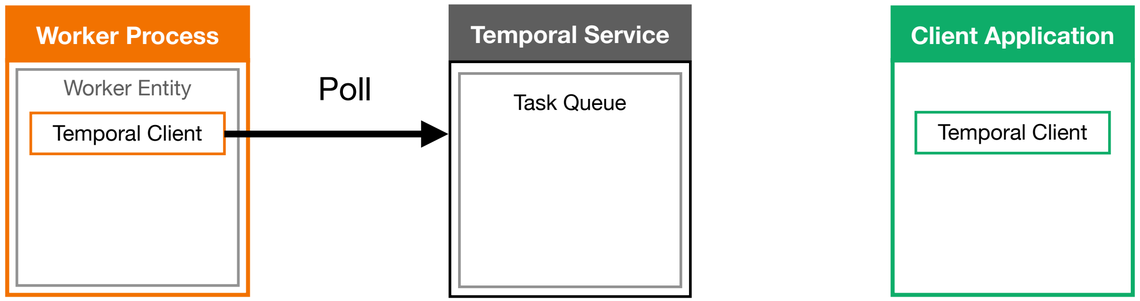
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/"+s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ex) {
          throw Activity.wrap(ex);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "hi" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

    public static void main(String[] args) {

        WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
        WorkflowClient client = WorkflowClient.newInstance(service);
        WorkerFactory factory = WorkerFactory.newInstance(client);

        Worker worker = factory.newWorker("greeting-tasks");

        worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

        worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

        factory.start();
    }
}
```



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
    StringBuilder builder = new StringBuilder();

    String baseUrl = "http://localhost:9999/%s?name=%s";

    URL url = null;
    try {
      url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
    } catch (IOException ex) {
      throw Activity.wrap(ex);
    }

    // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

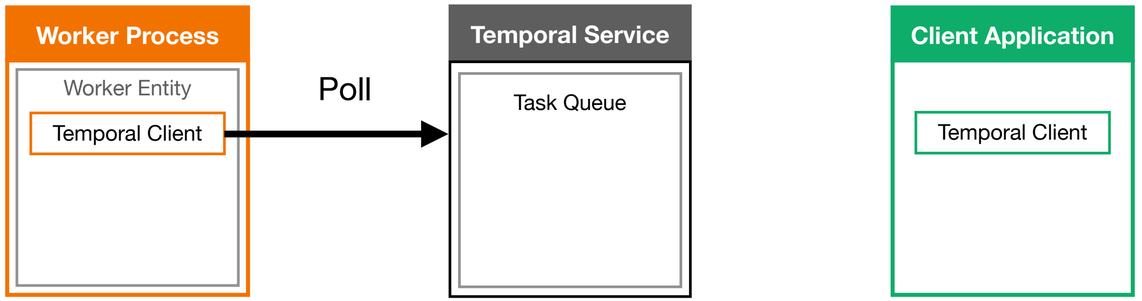
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final GreetingActivities activities =
      Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
    String spanishFarewell = activities.farewellInSpanish(name);

    return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

    WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
    WorkflowClient client = WorkflowClient.newInstance(service);
    WorkerFactory factory = WorkerFactory.newInstance(client);

    Worker worker = factory.newWorker("greeting-tasks");

    worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

    worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

    factory.start();
  }
}
```

Event History

WorkflowExecutionStarted



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/?ui?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ex) {
          throw Activity.wrap(ex);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

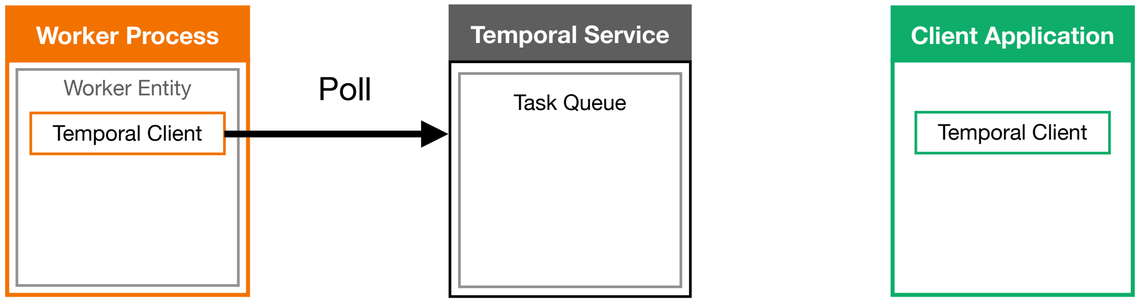
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

    public static void main(String[] args) {

        WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
        WorkflowClient client = WorkflowClient.newInstance(service);
        WorkerFactory factory = WorkerFactory.newInstance(client);

        Worker worker = factory.newWorker("greeting-tasks");

        worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

        worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

        factory.start();
    }
}
```

Event History

WorkflowExecutionStarted



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
    StringBuilder builder = new StringBuilder();

    String baseUrl = "http://localhost:9999/%s?name=%s";

    URL url = null;
    try {
      url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
    } catch (IOException ex) {
      throw Activity.wrap(ex);
    }

    // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

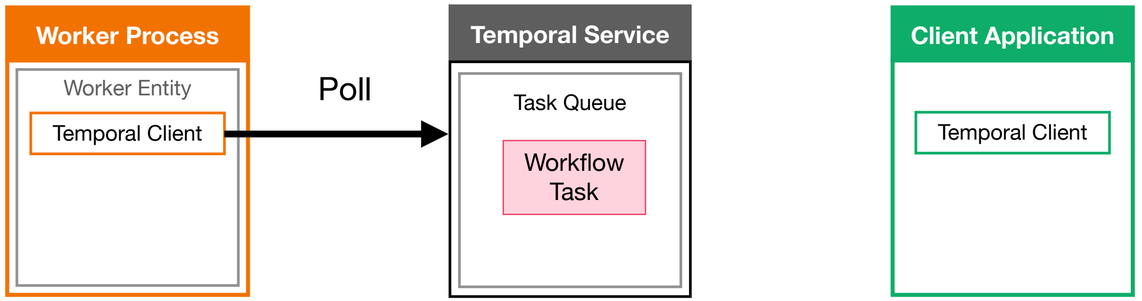
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final GreetingActivities activities =
      Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
    String spanishFarewell = activities.farewellInSpanish(name);

    return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

    WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
    WorkflowClient client = WorkflowClient.newInstance(service);
    WorkerFactory factory = WorkerFactory.newInstance(client);

    Worker worker = factory.newWorker("greeting-tasks");

    worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

    worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

    factory.start();
  }
}
```

Event History

WorkflowExecutionStarted



Activity Definitions

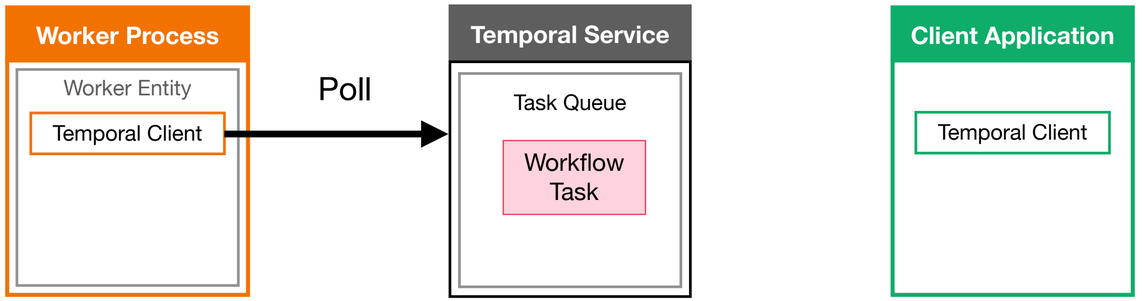
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ex) {
          throw Activity.wrap(ex);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted
WorkflowTaskScheduled

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
    String greetInSpanish(String name);

    String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        try {
            url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
        } catch (IOException ex) {
            throw Activity.wrap(ex);
        }

        // code that uses this URL to call the service has been omitted here

    }
}
```

Workflow Definition

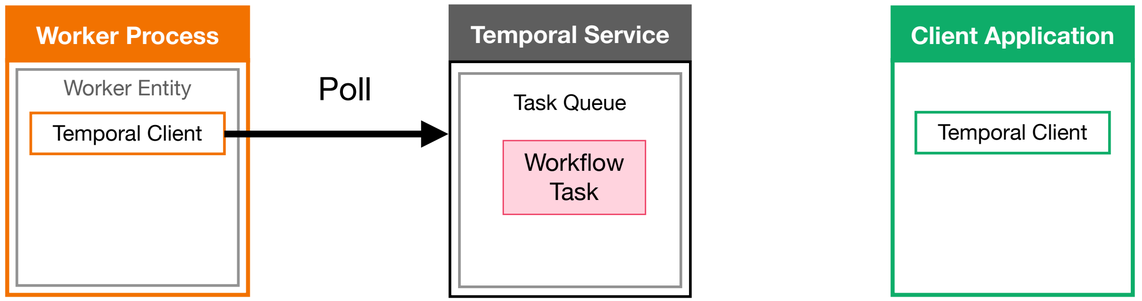
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

    @WorkflowMethod
    String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

    private final ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofSeconds(5))
            .build();

    private final GreetingActivities activities =
            Workflow.newActivityStub(GreetingActivities.class, options);

    @Override
    public String greetSomeone(String name) {
        String spanishGreeting = activities.greetInSpanish(name);
        String spanishFarewell = activities.farewellInSpanish(name);

        return "\n" + spanishGreeting + "\n" + spanishFarewell;
    }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

    public static void main(String[] args) {

        WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
        WorkflowClient client = WorkflowClient.newInstance(service);
        WorkerFactory factory = WorkerFactory.newInstance(client);

        Worker worker = factory.newWorker("greeting-tasks");

        worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

        worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

        factory.start();
    }
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ex) {
          throw Activity.wrap(ex);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

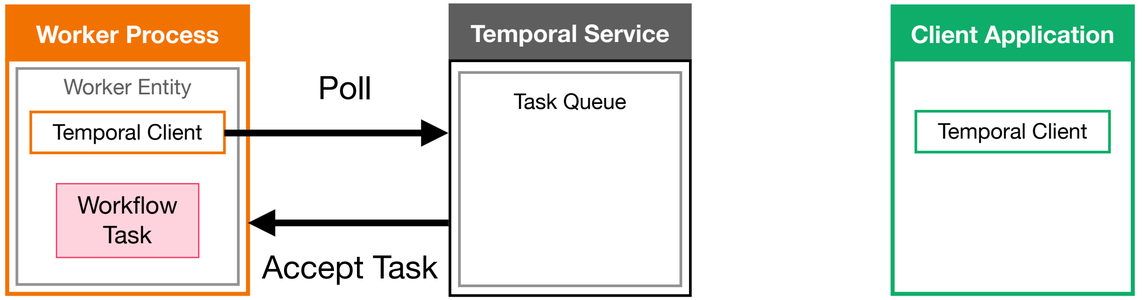
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled



Activity Definitions

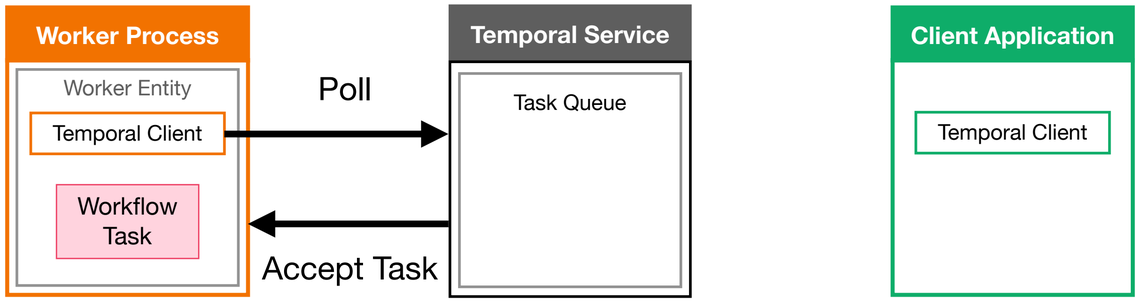
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/?ui?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ex) {
          throw Activity.wrap(ex);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

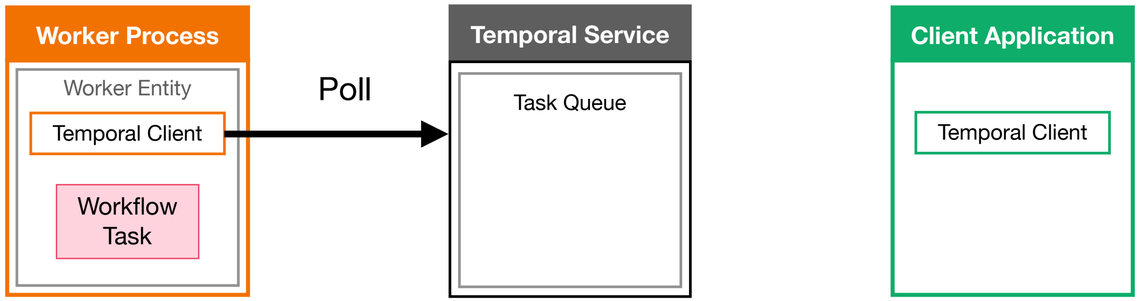
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingWorkflowImpl implements GreetingWorkflow {

    private final ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofSeconds(5))
            .build();

    private final GreetingActivities activities =
            Workflow.newActivityStub(GreetingActivities.class, options);

    @Override
    public String greetSomeone(String name) {
        String spanishGreeting = activities.greetInSpanish(name);
        String spanishFarewell = activities.farewellInSpanish(name);

        return "\n" + spanishGreeting + "\n" + spanishFarewell;
    }
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ex) {
          throw Activity.wrap(ex);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

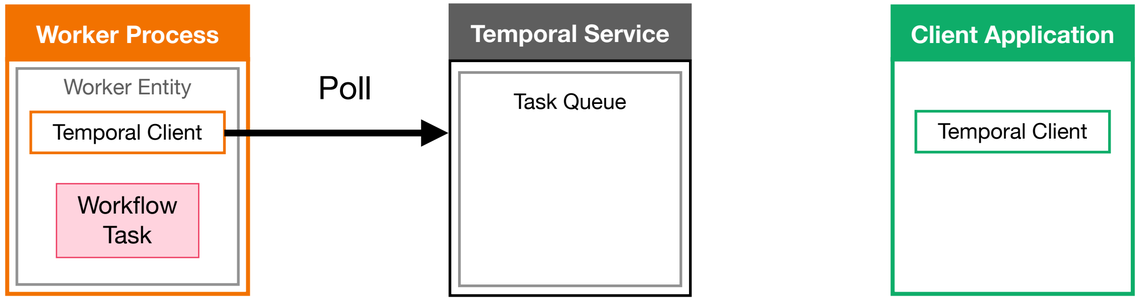
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ei) {
          throw Activity.wrap(ei);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

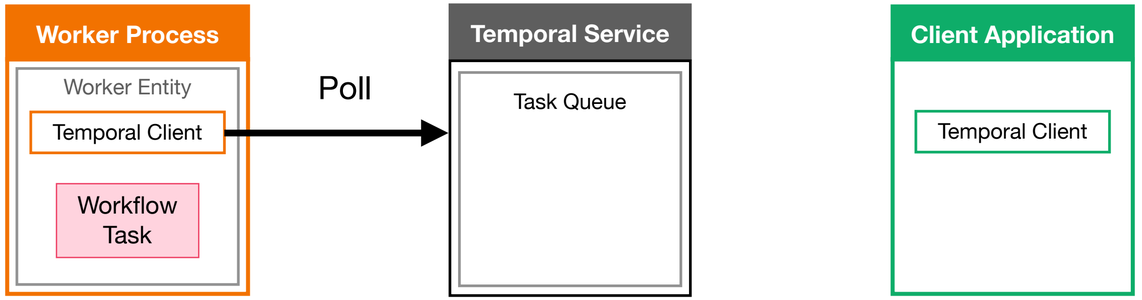
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ei) {
          throw Activity.wrap(ei);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

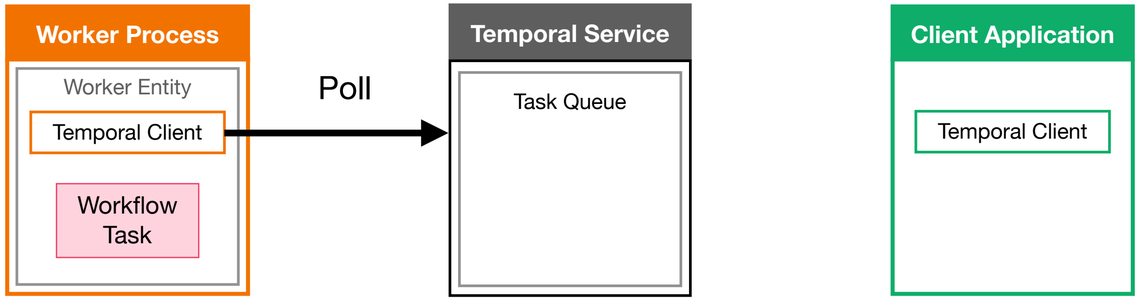
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingWorkflowImpl implements GreetingWorkflow {

    private final ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofSeconds(5))
            .build();

    private final GreetingActivities activities =
            Workflow.newActivityStub(GreetingActivities.class, options);

    @Override
    public String greetSomeone(String name) {
        String spanishGreeting = activities.greetInSpanish(name);
        String spanishFarewell = activities.farewellInSpanish(name);

        return "\n" + spanishGreeting + "\n" + spanishFarewell;
    }
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ex) {
          throw Activity.wrap(ex);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

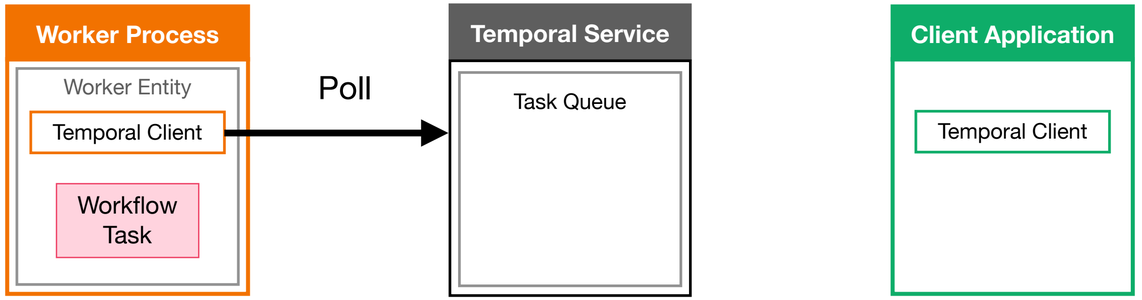
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted



Activity Definitions

```
package tareweliworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package tareweliworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

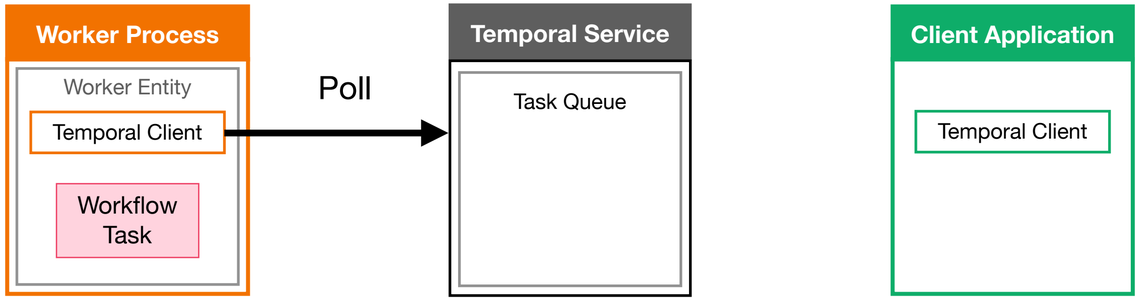
Worker Initialization

```
package tareweliworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

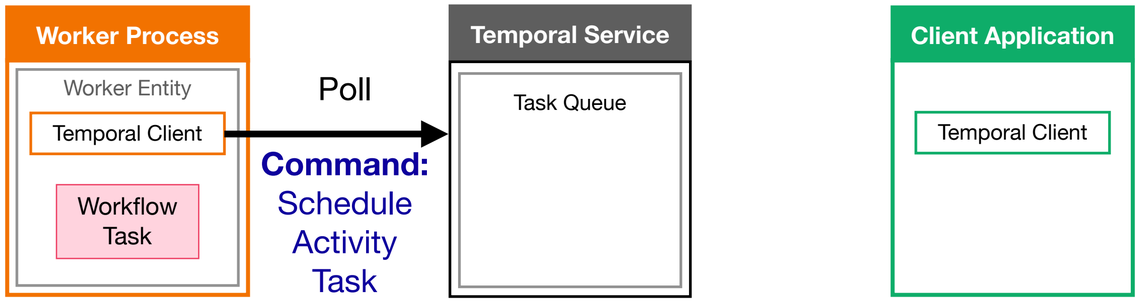
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingWorkflowImpl implements GreetingWorkflow {

    private final ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofSeconds(5))
            .build();

    private final GreetingActivities activities =
            Workflow.newActivityStub(GreetingActivities.class, options);

    @Override
    public String greetSomeone(String name) {
        String spanishGreeting = activities.greetInSpanish(name);
        String spanishFarewell = activities.farewellInSpanish(name);

        return "\n" + spanishGreeting + "\n" + spanishFarewell;
    }
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

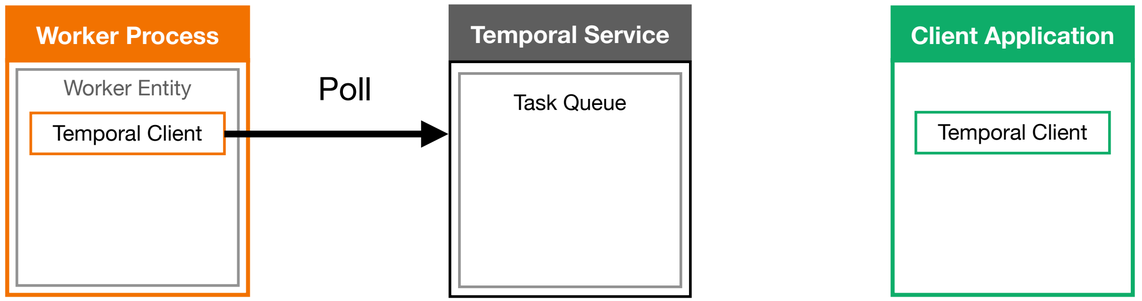
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ru/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

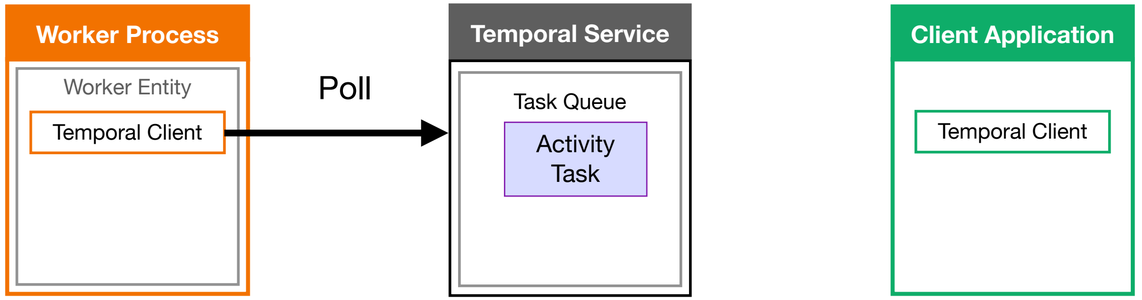
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ha?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

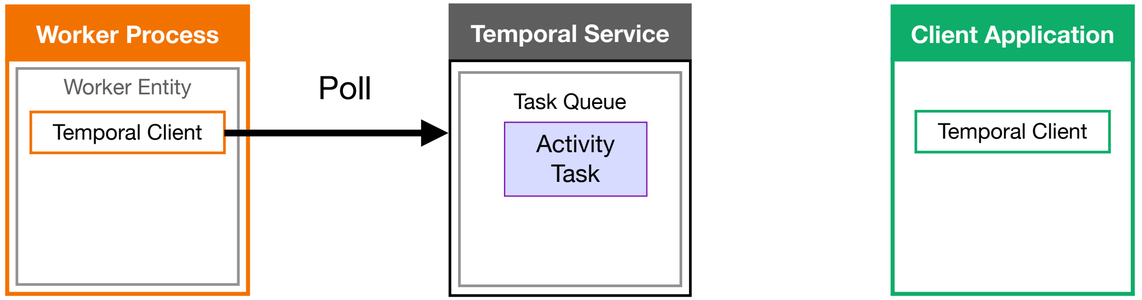
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Event History

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled (Greeting)



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ha?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

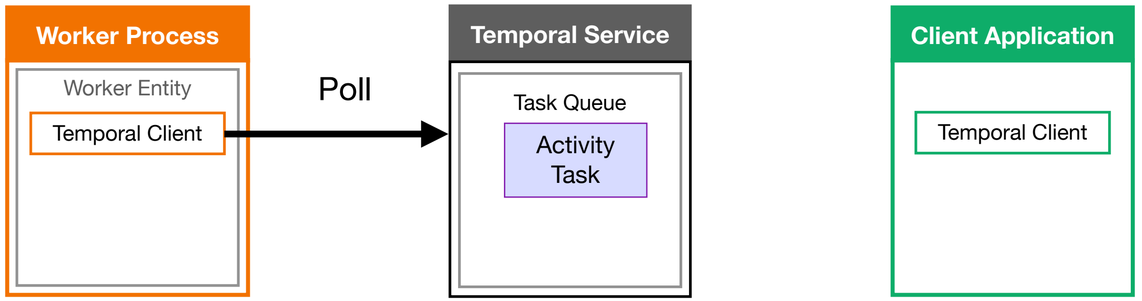
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

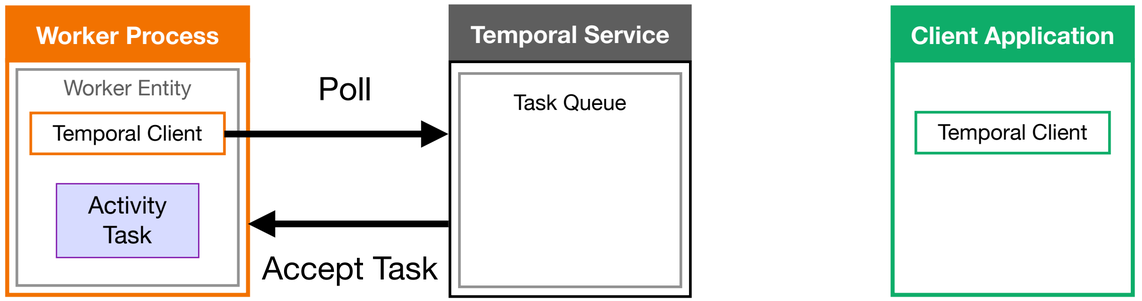
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)



Activity Definitions

```
package tareweliworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

       // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package tareweliworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

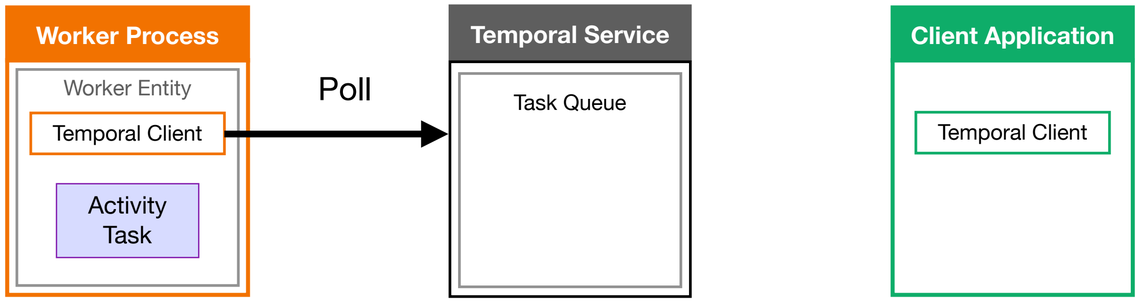
Worker Initialization

```
package tareweliworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
      return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      // code that initializes this URL has been omitted here
       try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
          String line;
          while ((line = in.readLine()) != null) {
              builder.append(line).append("\n");
          }
      } catch (IOException e) {
          throw Activity.wrap(e);
      }
      return builder.toString();
  }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)



Activity Definitions

```
package tarewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package tarewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

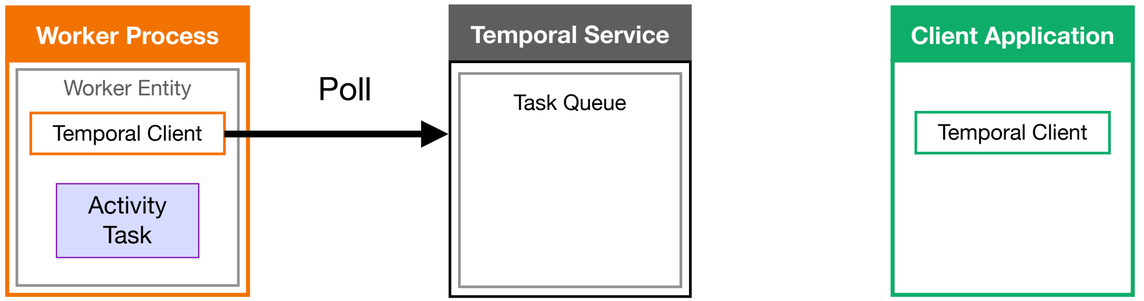
```
package tarewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Translation

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

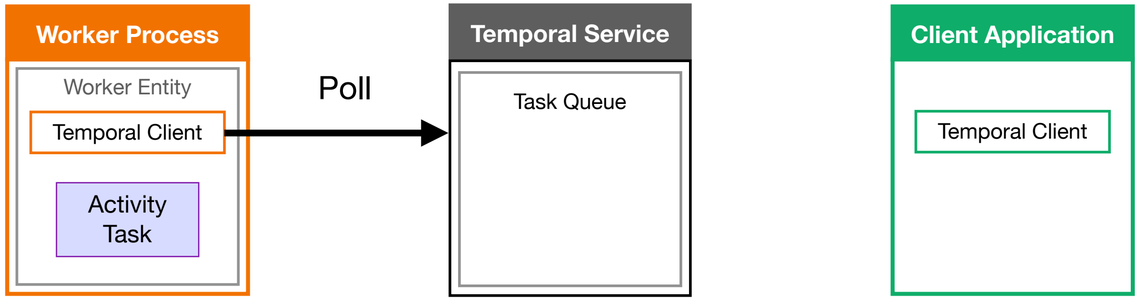
```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Translation

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

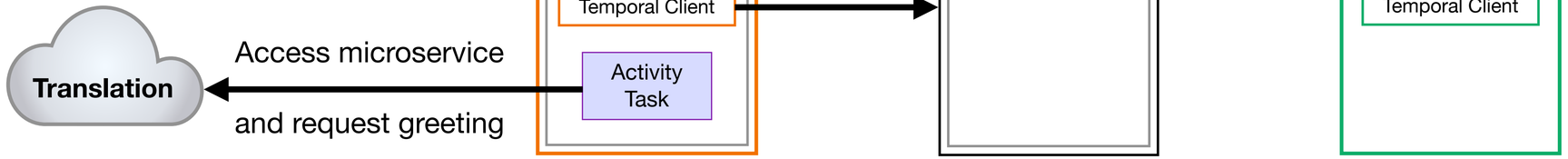
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)



Activity Definitions

```
package tarewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package tarewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
    String spanishFarewell = activities.farewellInSpanish(name);

    return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

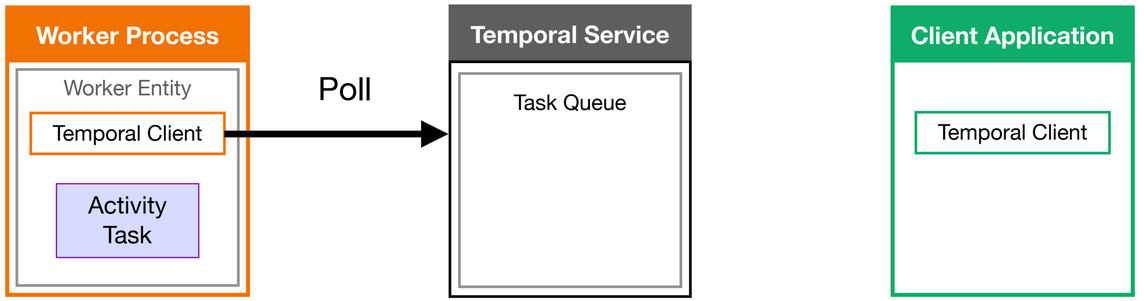
```
package tarewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Translation

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)



Activity Definitions

```
package tarewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package tarewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

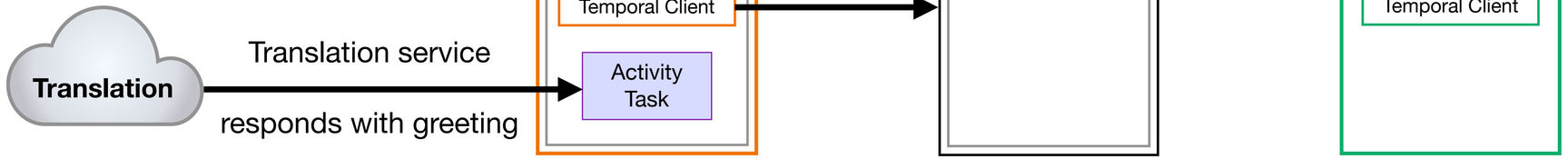
Worker Initialization

```
package tarewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)



Activity Definitions

```
package tarewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package tarewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

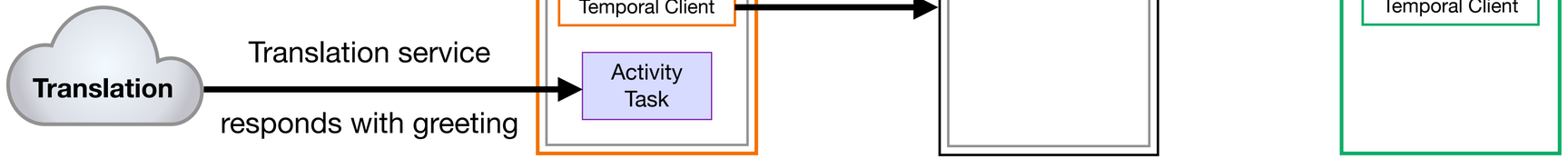
Worker Initialization

```
package tarewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)



Activity Definitions

```
package tarewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package tarewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

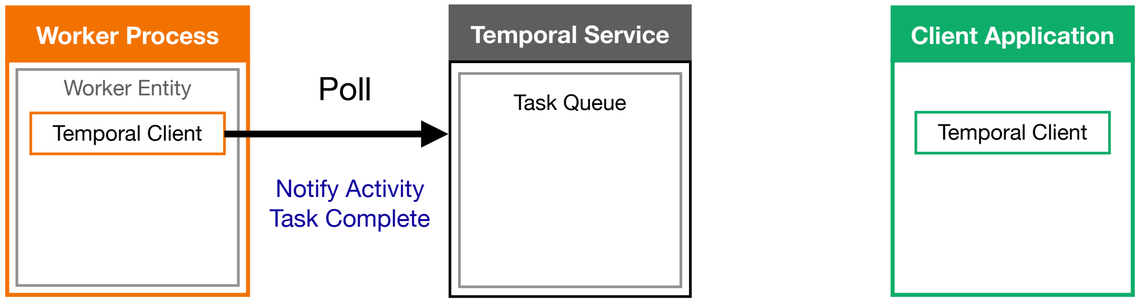
Worker Initialization

```
package tarewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)



Activity Definitions

```
package tareweliworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package tareweliworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

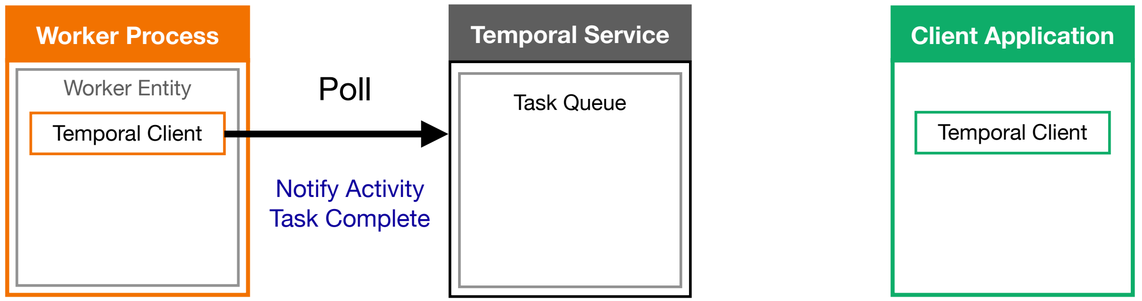
Worker Initialization

```
package tareweliworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)



Activity Definitions

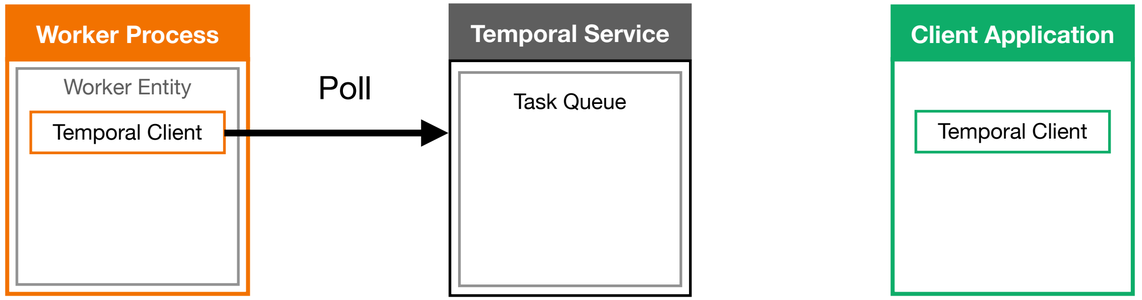
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ha?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ex) {
          throw Activity.wrap(ex);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ha?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ex) {
          throw Activity.wrap(ex);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

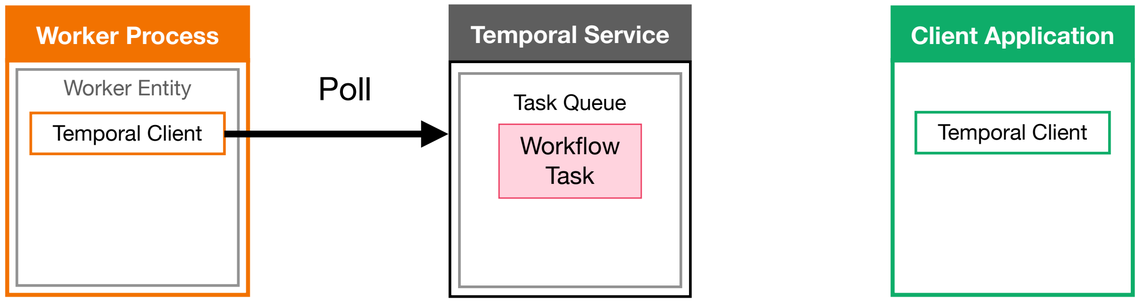
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)



Activity Definitions

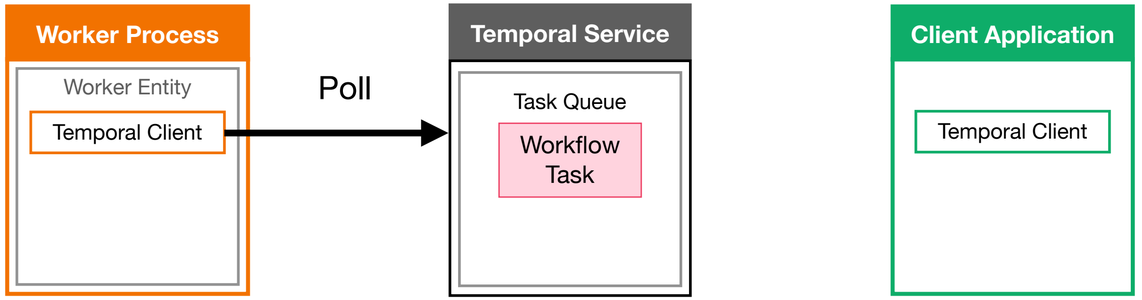
```
package tareweliworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ha?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ex) {
          throw Activity.wrap(ex);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package tareweliworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package tareweliworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	

Activity Definitions

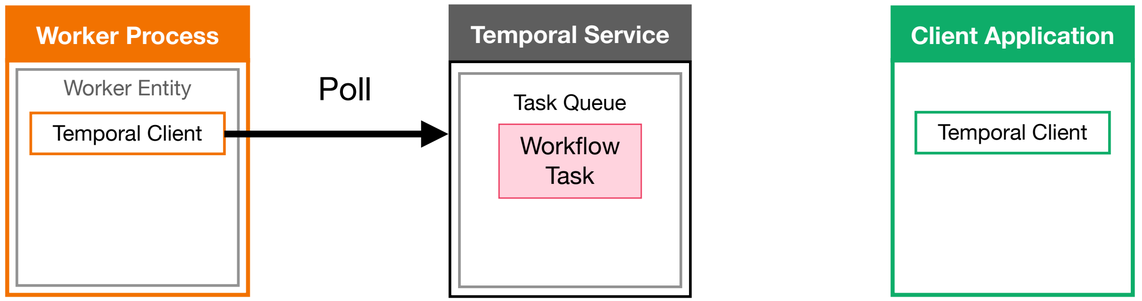
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ha?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ex) {
          throw Activity.wrap(ex);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	

Activity Definitions

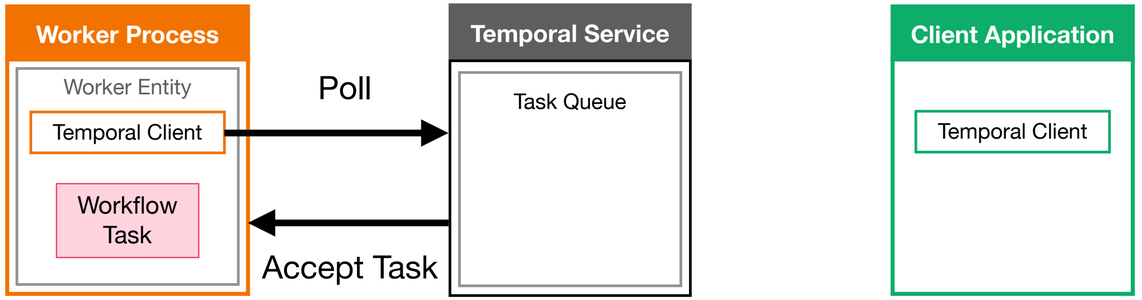
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ha?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	

Activity Definitions

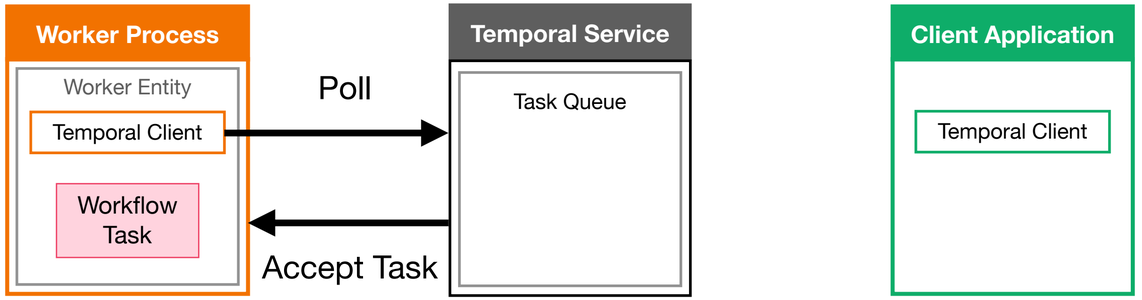
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ha?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	

Activity Definitions

```
package tareweliworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/";name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ei) {
          throw Activity.wrap(ei);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package tareweliworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

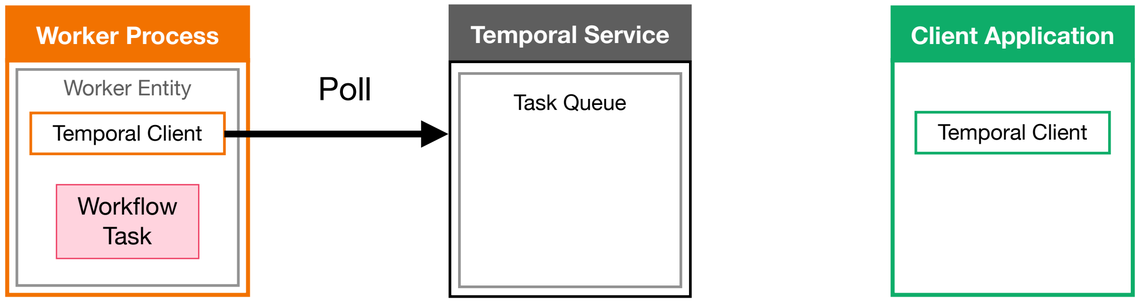
Worker Initialization

```
package tareweliworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	



Activity Definitions

```
package tarewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ei) {
          throw Activity.wrap(ei);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package tarewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

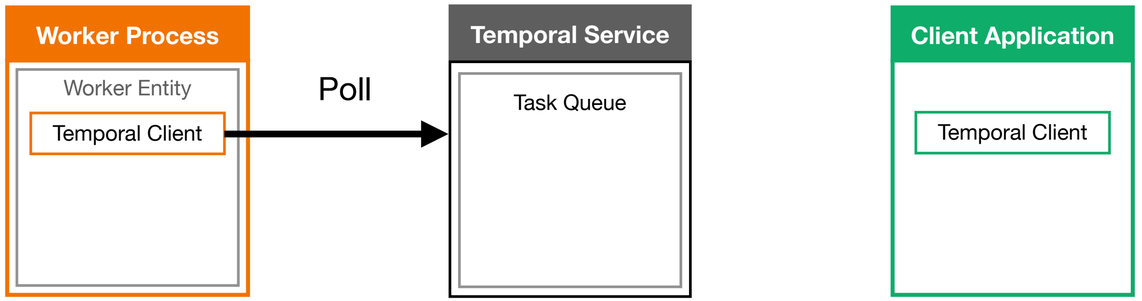
Worker Initialization

```
package tarewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	



Activity Definitions

```
package tarewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package tarewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

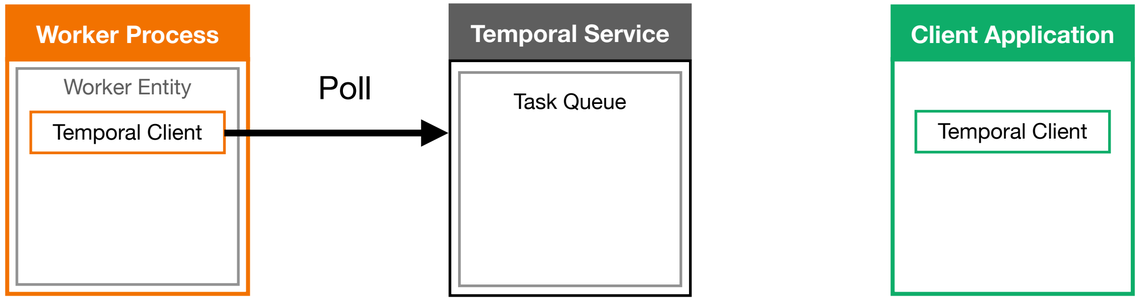
Worker Initialization

```
package tarewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/";name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ei) {
          throw Activity.wrap(ei);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

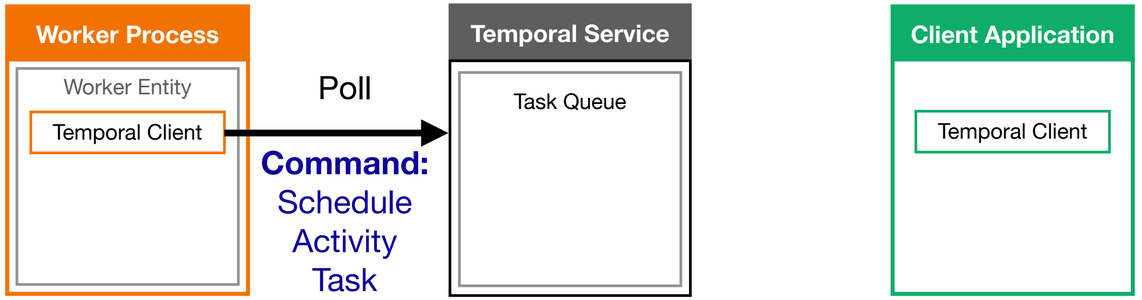
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	



Activity Definitions

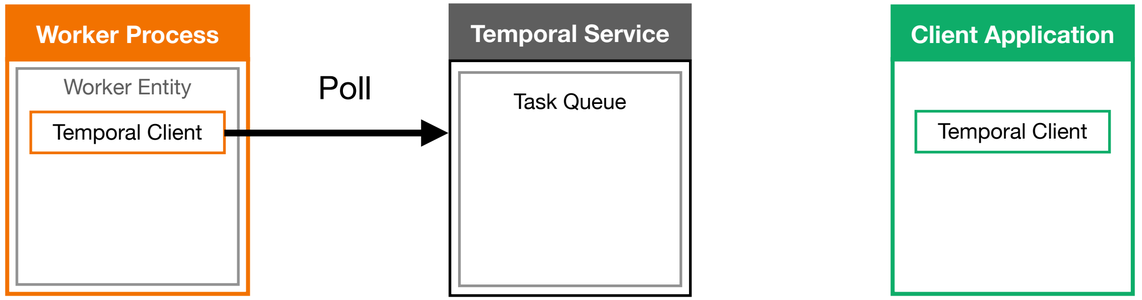
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ua?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

Activity Definitions

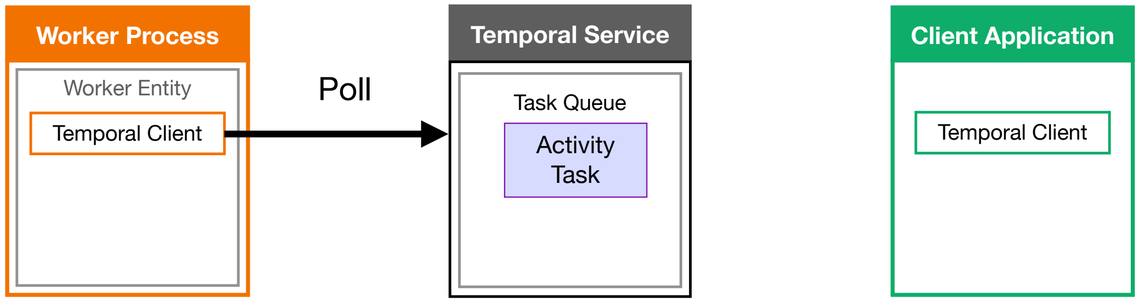
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ha?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);

}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

Activity Definitions

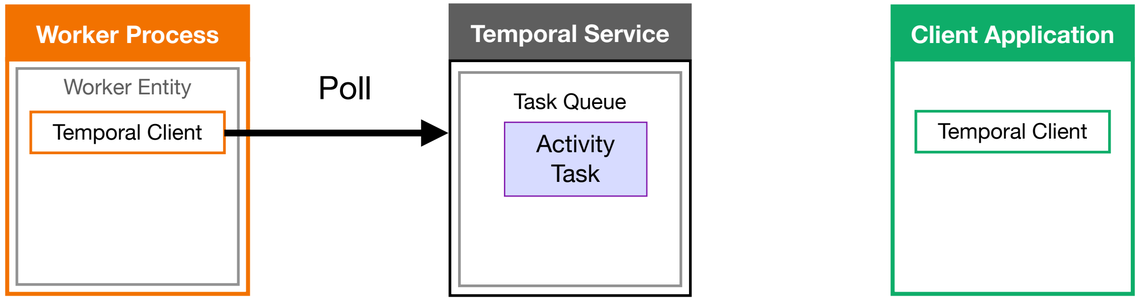
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/"+a?name=%a";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ex) {
          throw Activity.wrap(ex);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

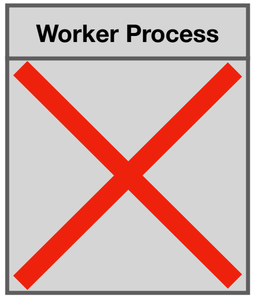
Workflow Definition

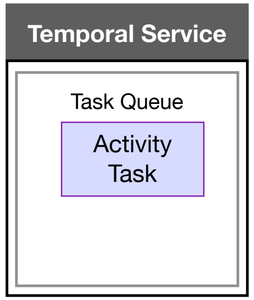
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

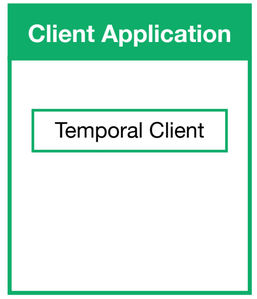
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

What happens if the Worker crashes?







Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)

Activity Definitions

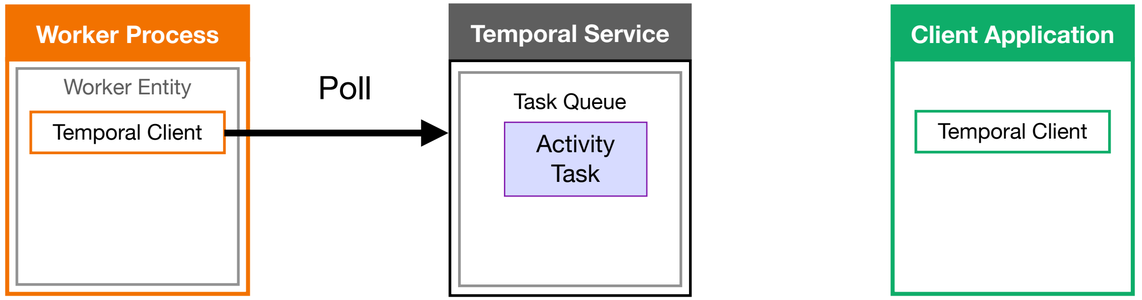
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ua?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ex) {
          throw Activity.wrap(ex);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)

Activity Definitions

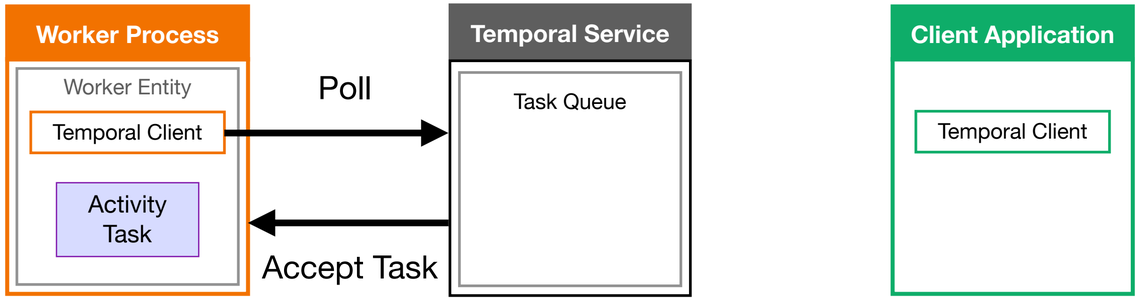
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ru?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)

Activity Definitions

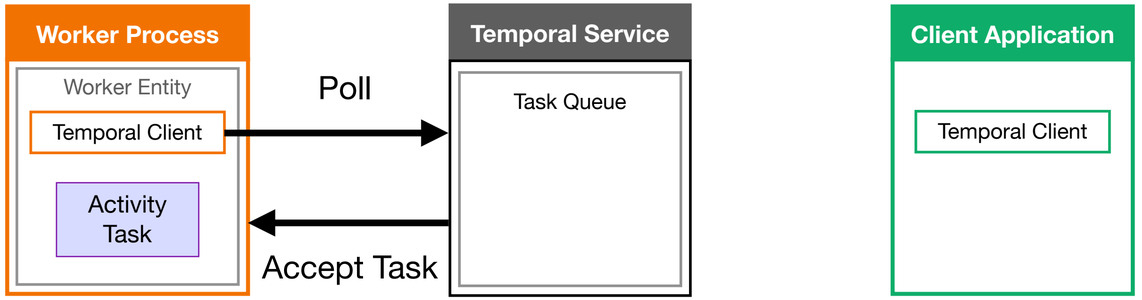
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

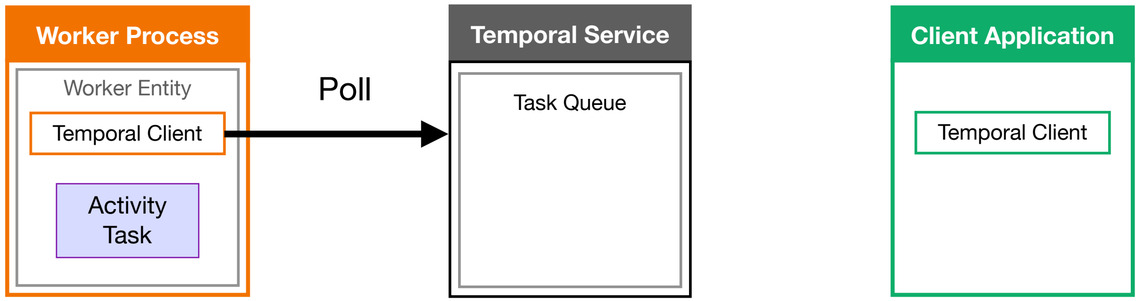
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
    String spanishFarewell = activities.farewellInSpanish(name);

    return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

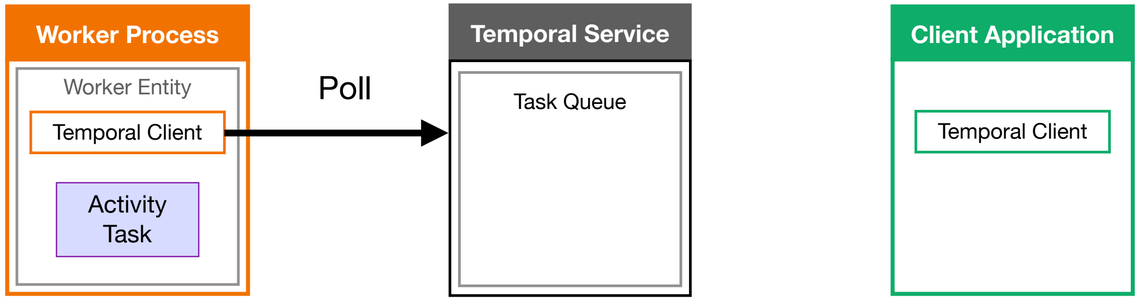
```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Translation

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package tarewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

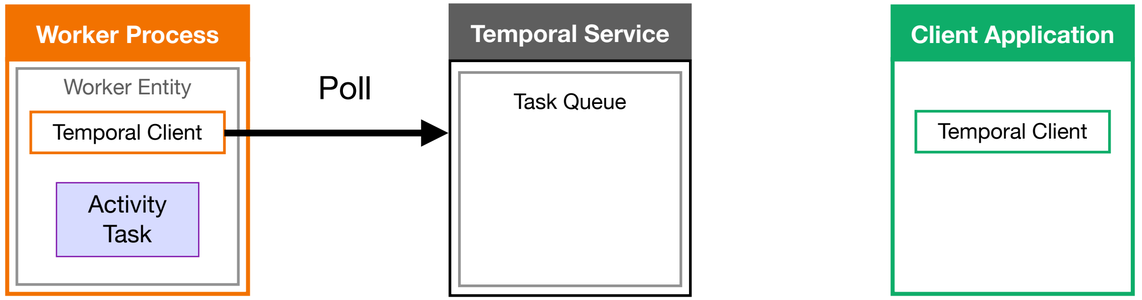
```
package tarewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package tarewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Translation

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

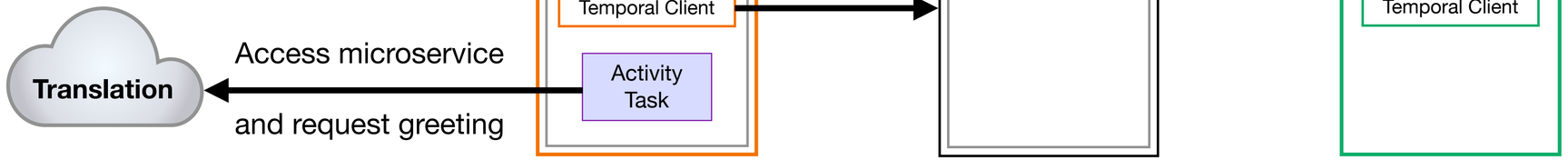
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

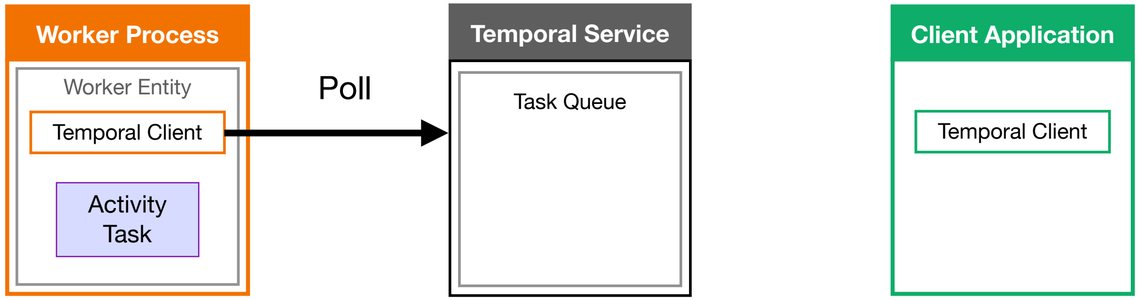
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
    String spanishFarewell = activities.farewellInSpanish(name);

    return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Error

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
    String spanishFarewell = activities.farewellInSpanish(name);

    return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

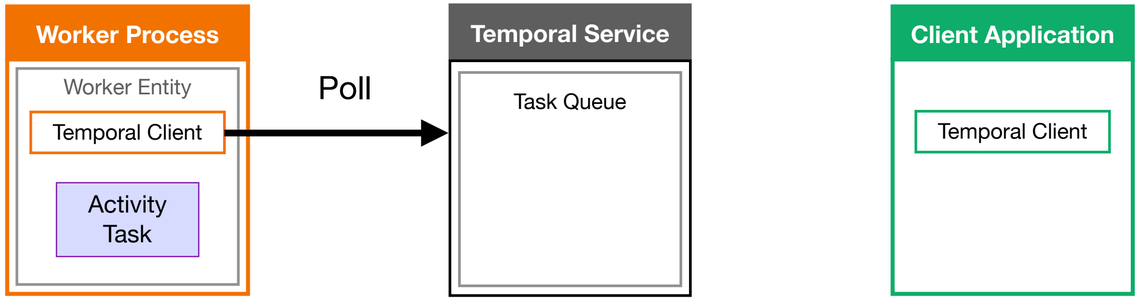
```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Translation

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

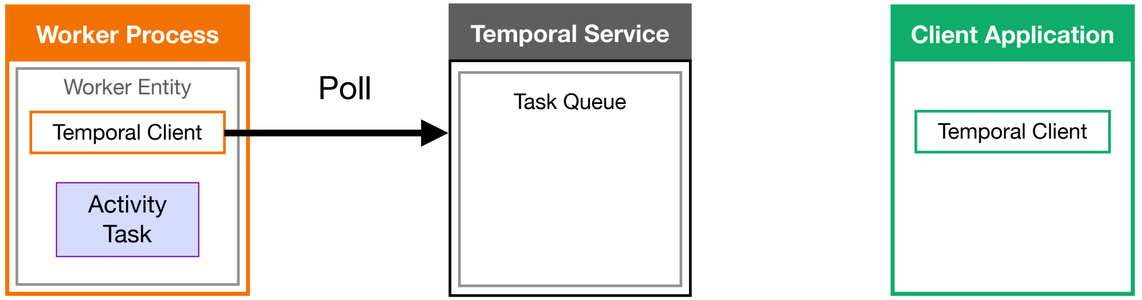
Translation

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Activity is invoked again during retry

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

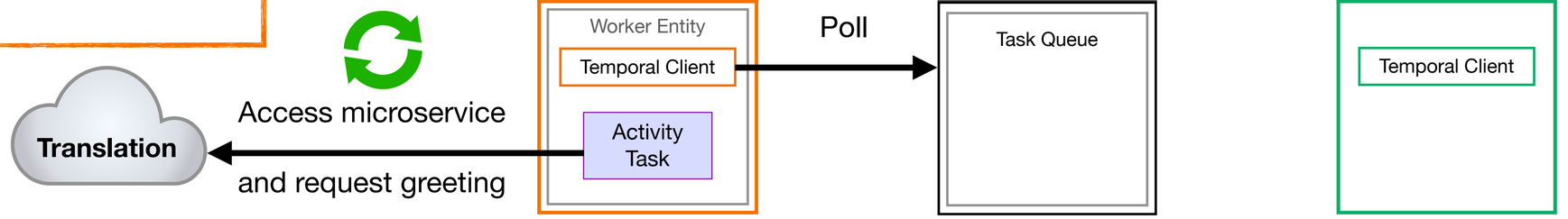
```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Activity is invoked again during retry

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

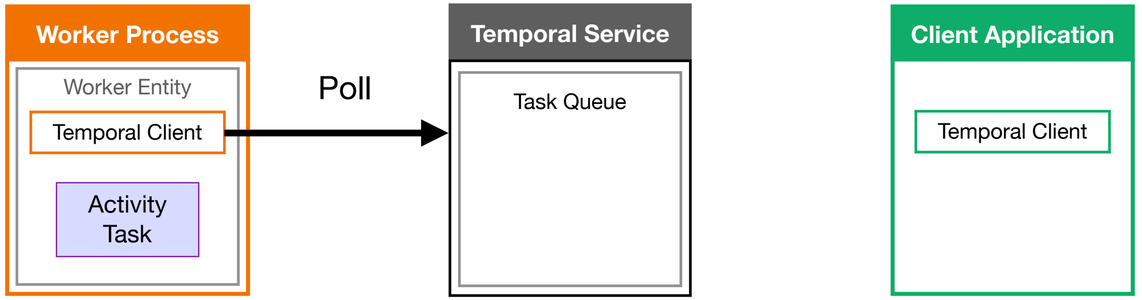
```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

Translation

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

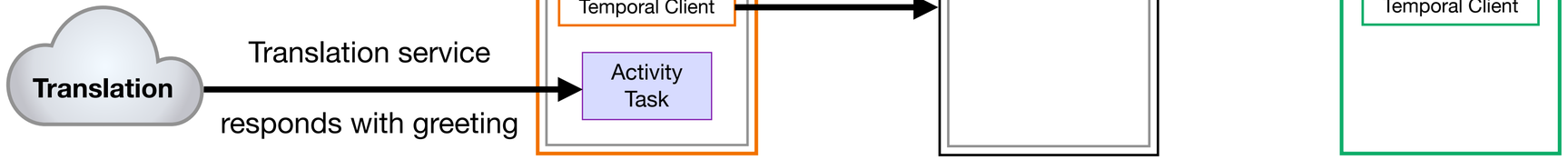
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

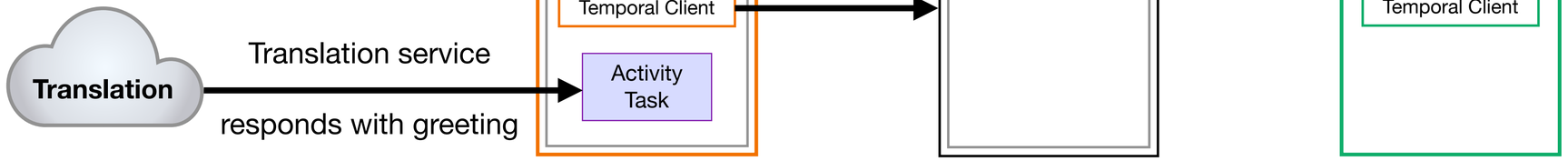
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
    String spanishFarewell = activities.farewellInSpanish(name);

    return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

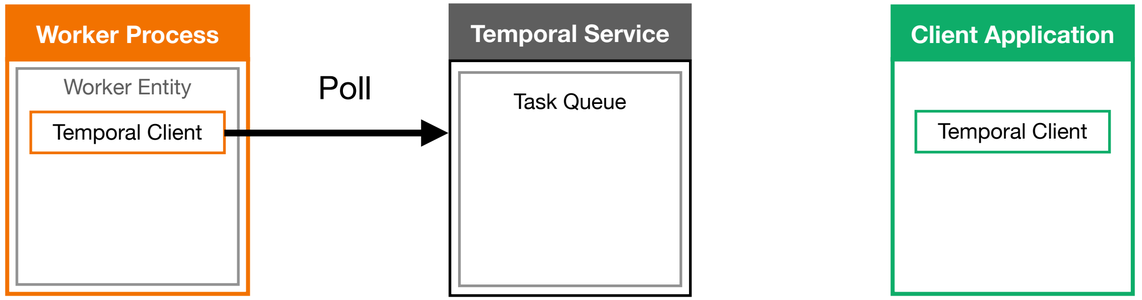
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          setStartToCloseTimeout(Duration.ofSeconds(5))
          build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
    String spanishGreeting = activities.greetInSpanish(name);
    String spanishFarewell = activities.farewellInSpanish(name);

    return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

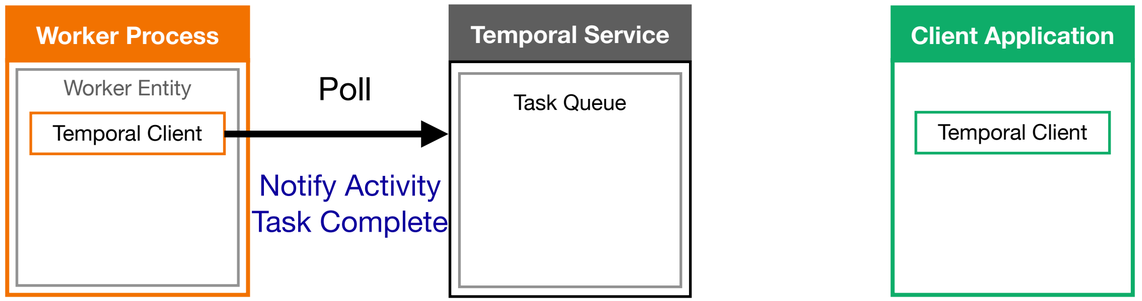
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

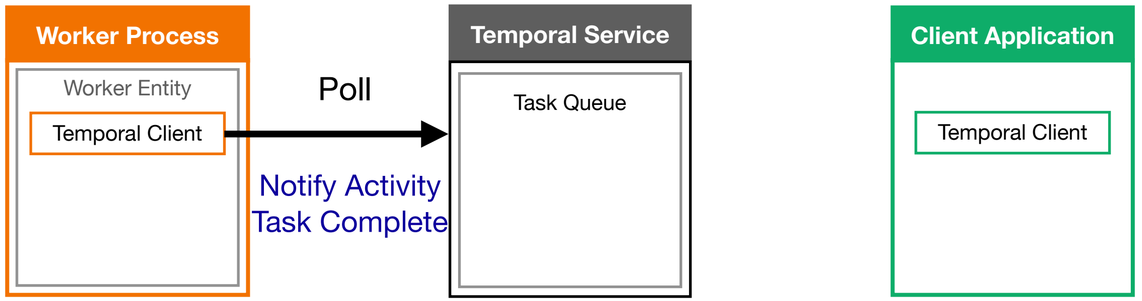
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingActivitiesImpl implements GreetingActivities {

    @Override
    public String greetInSpanish(String name) {
        return callService("get-spanish-greeting", name);
    }

    @Override
    public String farewellInSpanish(String name) {
        return callService("get-spanish-farewell", name);
    }

    String callService(String stem, String name) {
        StringBuilder builder = new StringBuilder();

        String baseUrl = "http://localhost:9999/%s?name=%s";

        URL url = null;
        // code that initializes this URL has been omitted here
        try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
            String line;
            while ((line = in.readLine()) != null) {
                builder.append(line).append("\n");
            }
        } catch (IOException e) {
            throw Activity.wrap(e);
        }
        return builder.toString();
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)



Activity Definitions

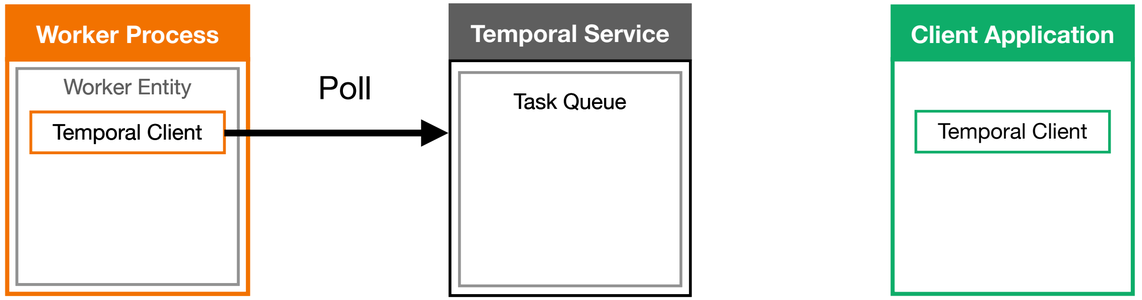
```
package tarewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ua?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package tarewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package tarewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)

Activity Definitions

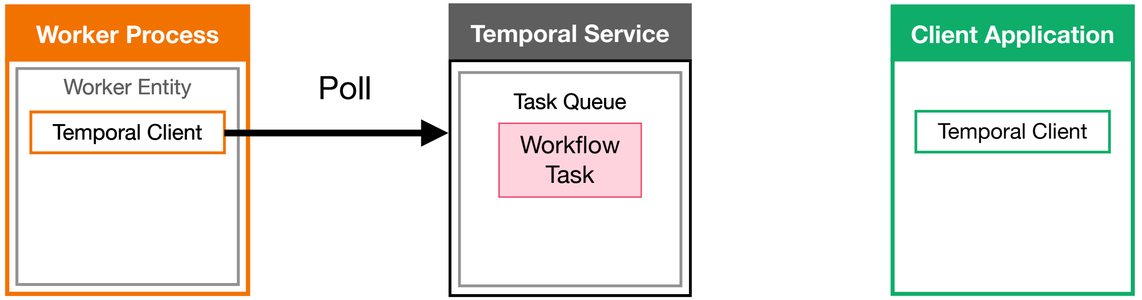
```
package tareweliworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ta?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package tareweliworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package tareweliworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	

Activity Definitions

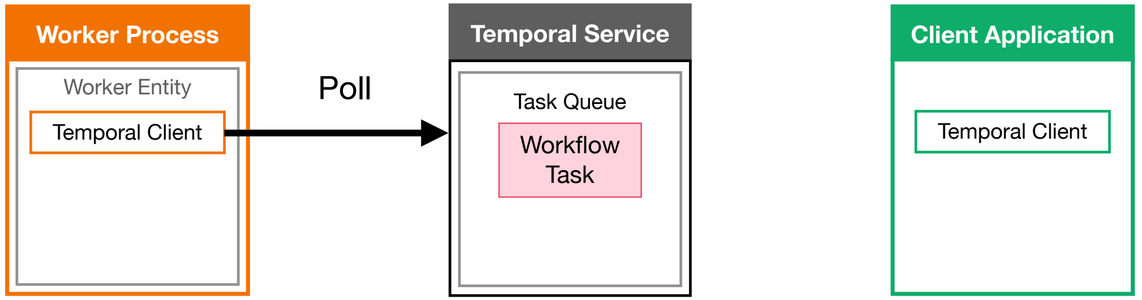
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException ex) {
          throw Activity.wrap(ex);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	

Activity Definitions

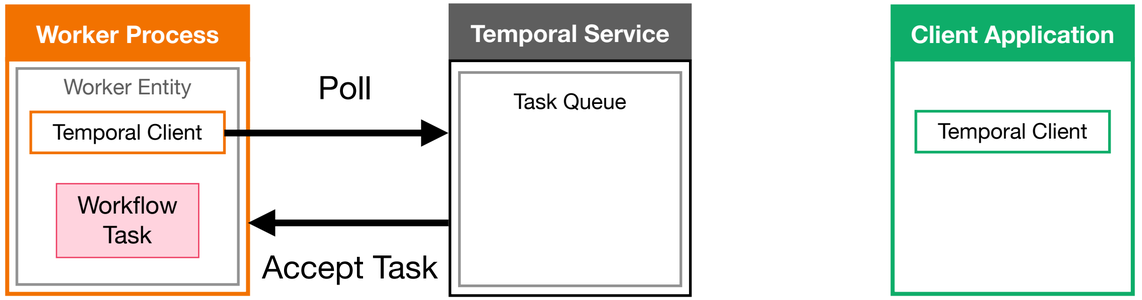
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/ru?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	

Activity Definitions

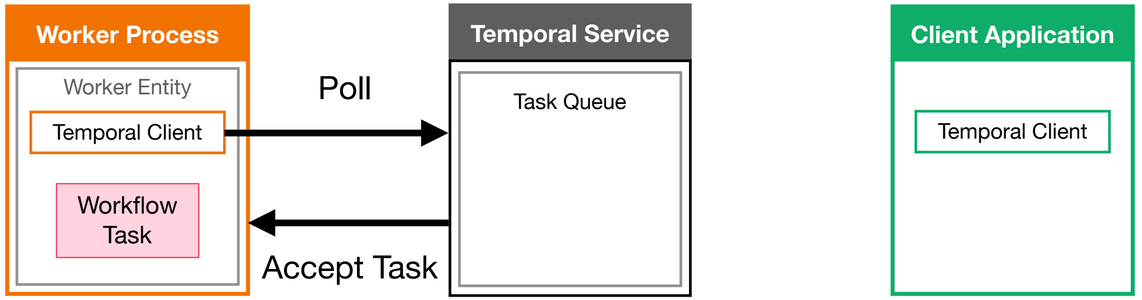
```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	
WorkflowTaskStarted	

Activity Definitions

```
package tareweltworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package tareweltworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

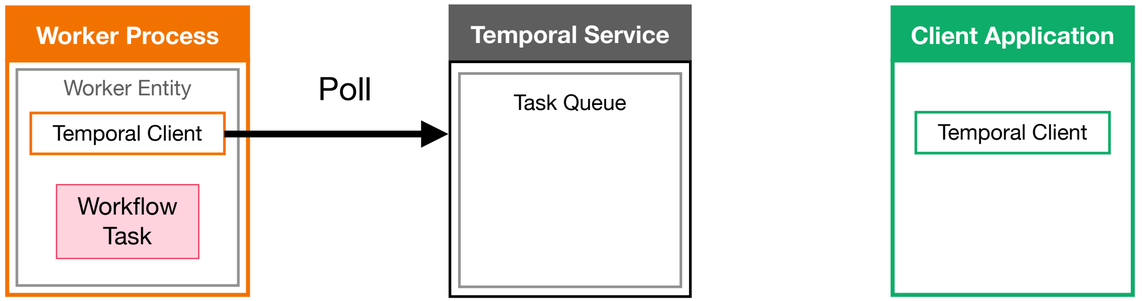
Worker Initialization

```
package tareweltworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingWorkflowImpl implements GreetingWorkflow {

    private final ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofSeconds(5))
            .build();

    private final GreetingActivities activities =
            Workflow.newActivityStub(GreetingActivities.class, options);

    @Override
    public String greetSomeone(String name) {
        String spanishGreeting = activities.greetInSpanish(name);
        String spanishFarewell = activities.farewellInSpanish(name);

        return "\n" + spanishGreeting + "\n" + spanishFarewell;
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	
WorkflowTaskStarted	



Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/%s?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

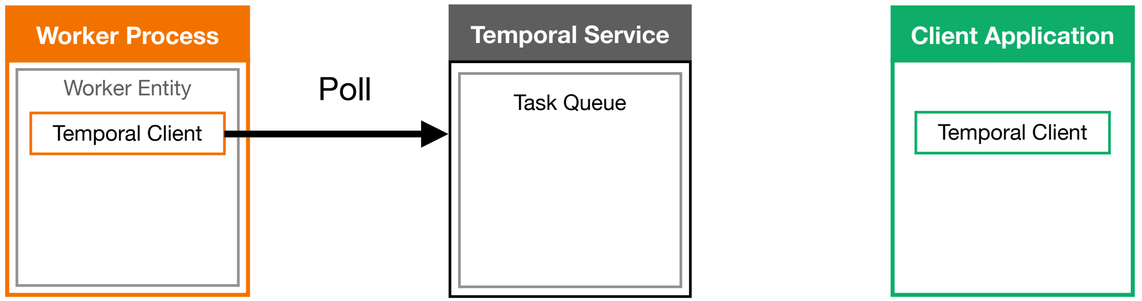
Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingWorkflowImpl implements GreetingWorkflow {

    private final ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofSeconds(5))
            .build();

    private final GreetingActivities activities =
            Workflow.newActivityStub(GreetingActivities.class, options);

    @Override
    public String greetSomeone(String name) {
        String spanishGreeting = activities.greetInSpanish(name);
        String spanishFarewell = activities.farewellInSpanish(name);

        return "\n" + spanishGreeting + "\n" + spanishFarewell;
    }
}
```

Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	
WorkflowTaskStarted	



Activity Definitions

```
package tareweltworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

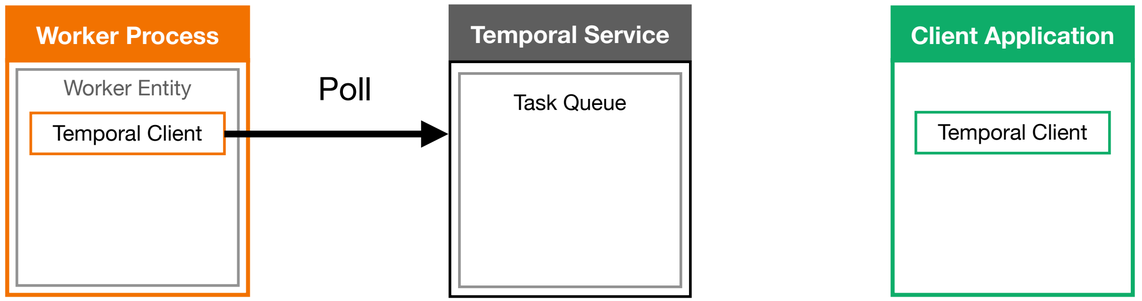
Workflow Definition

```
package tareweltworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package tareweltworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... code above has been omitted from this excerpt

class GreetingWorkflowImpl implements GreetingWorkflow {

    private final ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofSeconds(5))
            .build();

    private final GreetingActivities activities =
            Workflow.newActivityStub(GreetingActivities.class, options);

    @Override
    public String greetSomeone(String name) {
        String spanishGreeting = activities.greetInSpanish(name);
        String spanishFarewell = activities.farewellInSpanish(name);

        return "\n" + spanishGreeting + "\n" + spanishFarewell;
    }
}
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

Activity Definitions

```
package tarweliworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

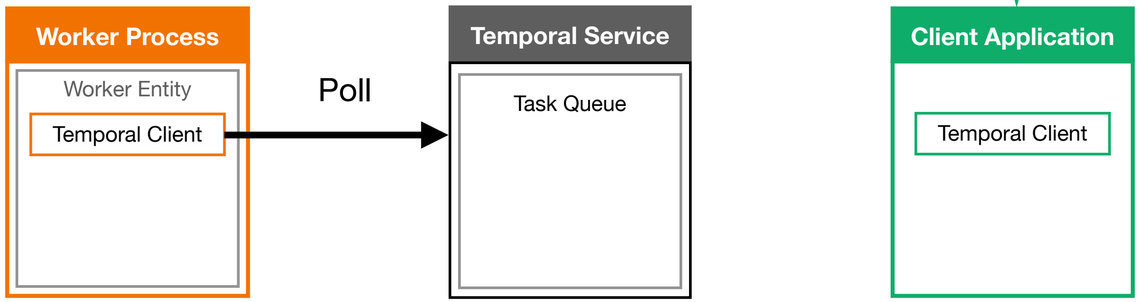
Workflow Definition

```
package tarweliworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package tarweliworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... this is code within your own application (for example a web or mobile app)

WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
WorkflowClient client = WorkflowClient.newInstance(service);

WorkflowOptions options = WorkflowOptions.newBuilder()
        .setWorkflowId("greeting-workflow")
        .setTaskQueue("greeting-tasks")
        .build();

GreetingWorkflow workflow = client.newWorkflowStub(GreetingWorkflow.class, options);

String greeting = workflow.greetSomeone(args[0]);

String workflowId = WorkflowStub.fromTyped(workflow).getExecution().getWorkflowId();

System.out.println(workflowId + " " + greeting);

// ... other application-specific code might follow
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

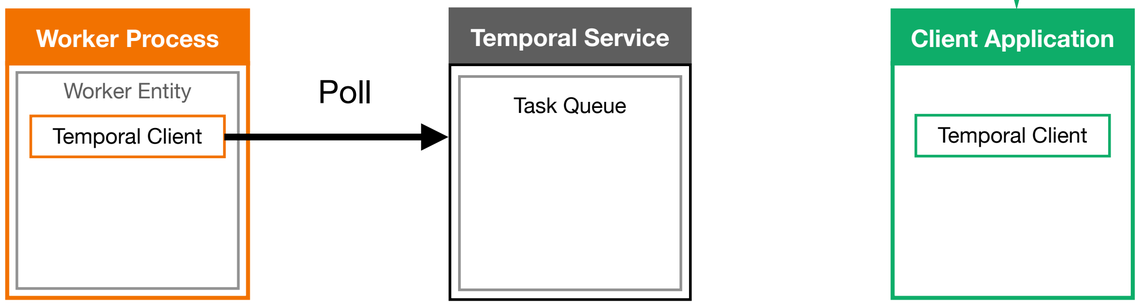
Workflow Definition

```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "hi" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... this is code within your own application (for example a web or mobile app)

WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
WorkflowClient client = WorkflowClient.newInstance(service);

WorkflowOptions options = WorkflowOptions.newBuilder()
        .setWorkflowId("greeting-workflow")
        .setTaskQueue("greeting-tasks")
        .build();

GreetingWorkflow workflow = client.newWorkflowStub(GreetingWorkflow.class, options);

String greeting = workflow.greetSomeone(args[0]);

String workflowId = WorkflowStub.fromTyped(workflow).getExecution().getWorkflowId();

System.out.println(workflowId + " " + greeting);

// ... other application-specific code might follow
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
WorkflowExecutionCompleted	

Activity Definitions

```
package tarewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

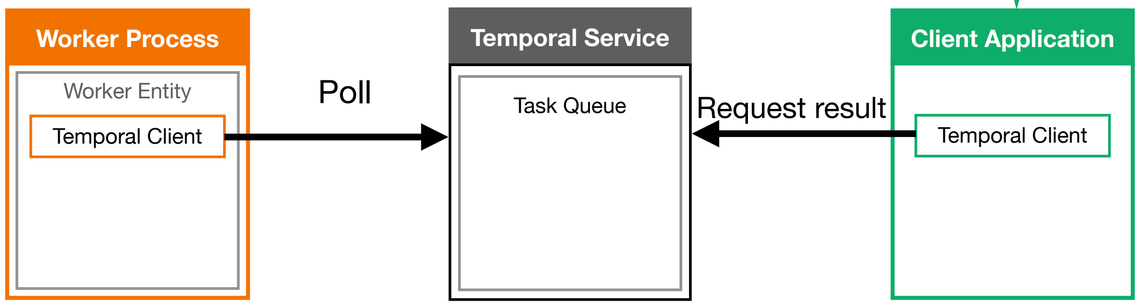
Workflow Definition

```
package tarewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package tarewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... this is code within your own application (for example a web or mobile app)

WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
WorkflowClient client = WorkflowClient.newInstance(service);

WorkflowOptions options = WorkflowOptions.newBuilder()
        .setWorkflowId("greeting-workflow")
        .setTaskQueue("greeting-tasks")
        .build();

GreetingWorkflow workflow = client.newWorkflowStub(GreetingWorkflow.class, options);

String greeting = workflow.greetSomeone(args[0]);

String workflowId = WorkflowStub.fromTyped(workflow).getExecution().getWorkflowId();

System.out.println(workflowId + " " + greeting);

// ... other application-specific code might follow
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
WorkflowExecutionCompleted	

The End

Activity Definitions

```
package tarewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/?name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

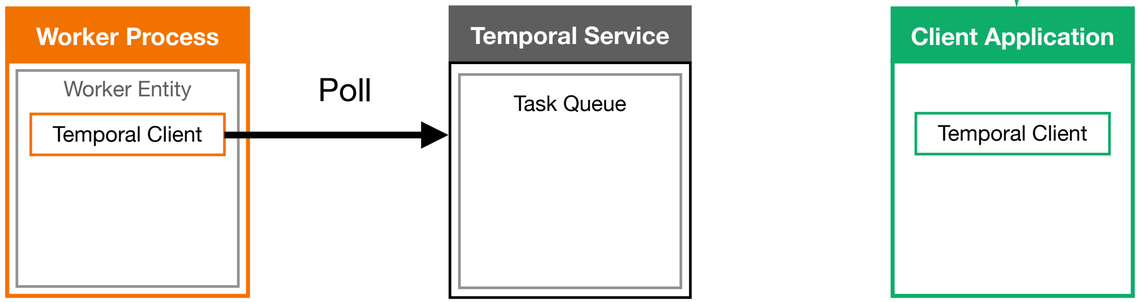
Workflow Definition

```
package tarewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package tarewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

```
// ... this is code within your own application (for example a web or mobile app)

WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
WorkflowClient client = WorkflowClient.newInstance(service);

WorkflowOptions options = WorkflowOptions.newBuilder()
        .setWorkflowId("greeting-workflow")
        .setTaskQueue("greeting-tasks")
        .build();

GreetingWorkflow workflow = client.newWorkflowStub(GreetingWorkflow.class, options);

String greeting = workflow.greetSomeone(args[0]);

String workflowId = WorkflowStub.fromTyped(workflow).getExecution().getWorkflowId();

System.out.println(workflowId + " " + greeting);

// ... other application-specific code might follow
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
WorkflowExecutionCompleted	

Activity Definitions

```
package farewellworkflow;

// non-Temporal imports omitted here for brevity
import io.temporal.activity.ActivityInterface;
import io.temporal.activity.Activity;

@ActivityInterface
public interface GreetingActivities {
  String greetInSpanish(String name);

  String farewellInSpanish(String name);
}

class GreetingActivitiesImpl implements GreetingActivities {

  @Override
  public String greetInSpanish(String name) {
      return callService("get-spanish-greeting", name);
  }

  @Override
  public String farewellInSpanish(String name) {
    return callService("get-spanish-farewell", name);
  }

  String callService(String stem, String name) {
      StringBuilder builder = new StringBuilder();

      String baseUrl = "http://localhost:9999/rs/";name=%s";

      URL url = null;
      try {
          url = new URL(String.format(baseUrl, stem, URLEncoder.encode(name, "UTF-8")));
      } catch (IOException e) {
          throw Activity.wrap(e);
      }

      // code that uses this URL to call the service has been omitted here

  }
}
```

Workflow Definition

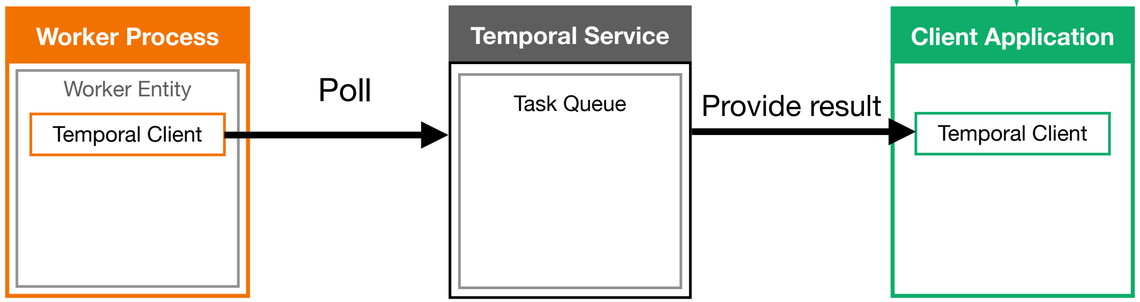
```
package farewellworkflow;

import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

import java.time.Duration;

@WorkflowInterface
public interface GreetingWorkflow {

  @WorkflowMethod
  String greetSomeone(String name);
}

class GreetingWorkflowImpl implements GreetingWorkflow {

  private final ActivityOptions options = ActivityOptions.newBuilder()
          .setStartToCloseTimeout(Duration.ofSeconds(5))
          .build();

  private final GreetingActivities activities =
          Workflow.newActivityStub(GreetingActivities.class, options);

  @Override
  public String greetSomeone(String name) {
      String spanishGreeting = activities.greetInSpanish(name);
      String spanishFarewell = activities.farewellInSpanish(name);

      return "\n" + spanishGreeting + "\n" + spanishFarewell;
  }
}
```

Worker Initialization

```
package farewellworkflow;

import io.temporal.client.WorkflowClient;
import io.temporal.serviceclient.WorkflowServiceStubs;
import io.temporal.worker.Worker;
import io.temporal.worker.WorkerFactory;

public class GreetingWorker {

  public static void main(String[] args) {

      WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
      WorkflowClient client = WorkflowClient.newInstance(service);
      WorkerFactory factory = WorkerFactory.newInstance(client);

      Worker worker = factory.newWorker("greeting-tasks");

      worker.registerWorkflowImplementationTypes(GreetingWorkflowImpl.class);

      worker.registerActivitiesImplementations(new GreetingActivitiesImpl());

      factory.start();
  }
}
```

The End

```
// ... this is code within your own application (for example a web or mobile app)

WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();
WorkflowClient client = WorkflowClient.newInstance(service);

WorkflowOptions options = WorkflowOptions.newBuilder()
        .setWorkflowId("greeting-workflow")
        .setTaskQueue("greeting-tasks")
        .build();

GreetingWorkflow workflow = client.newWorkflowStub(GreetingWorkflow.class, options);

String greeting = workflow.greetSomeone(args[0]);

String workflowId = WorkflowStub.fromTyped(workflow).getExecution().getWorkflowId();

System.out.println(workflowId + " " + greeting);

// ... other application-specific code might follow
```



Event History

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Greeting)
ActivityTaskStarted	(Greeting)
ActivityTaskCompleted	(Greeting)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	(Farewell)
ActivityTaskStarted	(Farewell)
ActivityTaskCompleted	(Farewell)
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
WorkflowExecutionCompleted	

Beginner to Builder Bootcamp - Java

00. About this Workshop

01. The Basics of Temporal

02. Improving Your Temporal Application Code

03. Using Timers in a Workflow Definition

04. Understanding Event History

05. Understanding Workflow Determinism

06. Signals, Queries, and Workflow Updates

07. Timeouts and Retry Policies

08. Recovering from Failure

09. Conclusion

Changing your inputs can break running Workflows

Changing your inputs can break running Workflows

- **Workflows and Activities can take any number of parameters as input**

Changing your inputs can break running Workflows

- **Workflows and Activities can take any number of parameters as input**
  - If you add, remove, or reorder parameters on a Workflow or Activity, you can break compatibility with workflows that are already in flight

Changing your inputs can break running Workflows

- **Workflows and Activities can take any number of parameters as input**
  - If you add, remove, or reorder parameters on a Workflow or Activity, you can break compatibility with workflows that are already in flight
- **Wrap your inputs in a single data class instead of using individual parameters**

Changing your inputs can break running Workflows

- **Workflows and Activities can take any number of parameters as input**
  - If you add, remove, or reorder parameters on a Workflow or Activity, you can break compatibility with workflows that are already in flight
- **Wrap your inputs in a single data class instead of using individual parameters**
  - Define a class (e.g. `GreetingInput`) and pass that as the sole argument

Changing your inputs can break running Workflows

- **Workflows and Activities can take any number of parameters as input**
  - If you add, remove, or reorder parameters on a Workflow or Activity, you can break compatibility with workflows that are already in flight
- **Wrap your inputs in a single data class instead of using individual parameters**
  - Define a class (e.g. `GreetingInput`) and pass that as the sole argument
  - Now you can add or rename fields freely without touching the method signature

Changing your inputs can break running Workflows

- **Workflows and Activities can take any number of parameters as input**
  - If you add, remove, or reorder parameters on a Workflow or Activity, you can break compatibility with workflows that are already in flight
- **Wrap your inputs in a single data class instead of using individual parameters**
  - Define a class (e.g. `GreetingInput`) and pass that as the sole argument
  - Now you can add or rename fields freely without touching the method signature
- **Do the same for return values**

Changing your inputs can break running Workflows

- **Workflows and Activities can take any number of parameters as input**
  - If you add, remove, or reorder parameters on a Workflow or Activity, you can break compatibility with workflows that are already in flight
- **Wrap your inputs in a single data class instead of using individual parameters**
  - Define a class (e.g. `GreetingInput`) and pass that as the sole argument
  - Now you can add or rename fields freely without touching the method signature
- **Do the same for return values**
  - A dedicated output class gives you the same flexibility on the way out. You can evolve return values without breaking callers

Example: Using a class in an Activity (1)

Example: Using a class in an Activity (1)

- **Imagine that you have the following Activity**

```
// This Activity returns a customized greeting in English, using the provided name
String createGreeting(String name){
    // implementation omitted for brevity
```

Example: Using a class in an Activity (1)

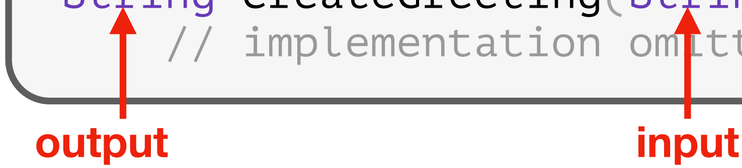
- **Imagine that you have the following Activity**

```
// This Activity returns a customized greeting in English, using the provided name
String createGreeting(String name){
    // implementation omitted for brevity
```

Example: Using a class in an Activity (1)

- **Imagine that you have the following Activity**

```
// This Activity returns a customized greeting in English, using the provided name
String createGreeting(String name){
    // implementation omitted for brevity
```



Example: Using a class in an Activity (1)

- **Imagine that you have the following Activity**

```
// This Activity returns a customized greeting in English, using the provided name
String createGreeting(String name){
    // implementation omitted for brevity
```

output

input

- **You later need to update it to support other languages, such as Spanish**

Example: Using a class in an Activity (1)

- **Imagine that you have the following Activity**

```
// This Activity returns a customized greeting in English, using the provided name
String createGreeting(String name){
    // implementation omitted for brevity
```

output

input

- **You later need to update it to support other languages, such as Spanish**
  - Changing what is passed into or returned from the method changes its signature

Example: Using a class in an Activity (1)

- **Imagine that you have the following Activity**

```
// This Activity returns a customized greeting in English, using the provided name
String createGreeting(String name){
    // implementation omitted for brevity
```

output

input

- **You later need to update it to support other languages, such as Spanish**
  - Changing what is passed into or returned from the method changes its signature
  - Changes to the class composition don't affect the signature of the methods that use it

Example: Using a class in an Activity (2)

Example: Using a class in an Activity (2)

- **The following code sample illustrates how you could support this**

```
// Define a class to encapsulate all data passed as input for this Activity
class GreetingInput {
    private String name;
    private String languageCode;

    // Constructors and Getters/Setters omitted for brevity
}

// Define a class to encapsulate all data returned by this Activity
class GreetingOutput {
    private String greeting;

    // Constructors and Getters/Setters omitted for brevity
}

// Specify these types for the input parameter and return type of the Activity
GreetingOutput createGreeting(GreetingInput input) {

    // An example to show how to access input parameters and create the return value
    if (input.getLanguageCode().equals("fr")) {
        String bonjour = "Bonjour, " + input.getName();
        return new GreetingOutput(bonjour)
    }
    // support for additional languages would follow...
```

Example: Using a class in an Activity (2)

- **The following code sample illustrates how you could support this**

```
// Define a class to encapsulate all data passed as input for this Activity
class GreetingInput {
    private String name;
    private String languageCode;

    // Constructors and Getters/Setters omitted for brevity
}

// Define a class to encapsulate all data returned by this Activity
class GreetingOutput {
    private String greeting;

    // Constructors and Getters/Setters omitted for brevity
}

// Specify these types for the input parameter and return type of the Activity
GreetingOutput createGreeting(GreetingInput input) {

    // An example to show how to access input parameters and create the return value
    if (input.getLanguageCode().equals("fr")) {
        String bonjour = "Bonjour, " + input.getName();
        return new GreetingOutput(bonjour)
    }
    // support for additional languages would follow...
```

Example: Using a class in an Activity (2)

- **The following code sample illustrates how you could support this**

```
// Define a class to encapsulate all data passed as input for this Activity
class GreetingInput {
    private String name;
    private String languageCode;

    // Constructors and Getters/Setters omitted for brevity
}

// Define a class to encapsulate all data returned by this Activity
class GreetingOutput {
    private String greeting;

    // Constructors and Getters/Setters omitted for brevity
}

// Specify these types for the input parameter and return type of the Activity
GreetingOutput createGreeting(GreetingInput input) {

    // An example to show how to access input parameters and create the return value
    if (input.getLanguageCode().equals("fr")) {
        String bonjour = "Bonjour, " + input.getName();
        return new GreetingOutput(bonjour)
    }
    // support for additional languages would follow...
```

Example: Using a class in an Activity (2)

- **The following code sample illustrates how you could support this**

```
// Define a class to encapsulate all data passed as input for this Activity
class GreetingInput {
    private String name;
    private String languageCode;

    // Constructors and Getters/Setters omitted for brevity
}

// Define a class to encapsulate all data returned by this Activity
class GreetingOutput {
    private String greeting;

    // Constructors and Getters/Setters omitted for brevity
}

// Specify these types for the input parameter and return type of the Activity
GreetingOutput createGreeting(GreetingInput input) {

    // An example to show how to access input parameters and create the return value
    if (input.getLanguageCode().equals("fr")) {
        String bonjour = "Bonjour, " + input.getName();
        return new GreetingOutput(bonjour)
    }
    // support for additional languages would follow...
```

Example: Using a class in an Activity (2)

- **The following code sample illustrates how you could support this**

```
// Define a class to encapsulate all data passed as input for this Activity
class GreetingInput {
    private String name;
    private String languageCode;

    // Constructors and Getters/Setters omitted for brevity
}

// Define a class to encapsulate all data returned by this Activity
class GreetingOutput {
    private String greeting;

    // Constructors and Getters/Setters omitted for brevity
}

// Specify these types for the input parameter and return type of the Activity
GreetingOutput createGreeting(GreetingInput input) {

    // An example to show how to access input parameters and create the return value
    if (input.getLanguageCode().equals("fr")) {
        String bonjour = "Bonjour, " + input.getName();
        return new GreetingOutput(bonjour)
    }
    // support for additional languages would follow...
```

input

output

Task Queues

Task Queues

- **Temporal Services coordinate with Workers through named Task Queues**

Task Queues

- **Temporal Services coordinate with Workers through named Task Queues**
  - The name of this Task Queue is specified in the Worker configuration

Task Queues

- **Temporal Services coordinate with Workers through named Task Queues**
  - The name of this Task Queue is specified in the Worker configuration
  - The Task Queue name is also specified by a Client when starting a Workflow

Task Queues

- **Temporal Services coordinate with Workers through named Task Queues**
  - The name of this Task Queue is specified in the Worker configuration
  - The Task Queue name is also specified by a Client when starting a Workflow
  - Task Queues are dynamically created, so a mismatch in names does not result in an error

Task Queues

- **Temporal Services coordinate with Workers through named Task Queues**
  - The name of this Task Queue is specified in the Worker configuration
  - The Task Queue name is also specified by a Client when starting a Workflow
  - Task Queues are dynamically created, so a mismatch in names does not result in an error
- **Recommendations for naming Task Queues**

Task Queues

- **Temporal Services coordinate with Workers through named Task Queues**
  - The name of this Task Queue is specified in the Worker configuration
  - The Task Queue name is also specified by a Client when starting a Workflow
  - Task Queues are dynamically created, so a mismatch in names does not result in an error
- **Recommendations for naming Task Queues**
  - Do not hardcode the name in multiple places: Use a shared constant if possible

Task Queues

- **Temporal Services coordinate with Workers through named Task Queues**
  - The name of this Task Queue is specified in the Worker configuration
  - The Task Queue name is also specified by a Client when starting a Workflow
  - Task Queues are dynamically created, so a mismatch in names does not result in an error
- **Recommendations for naming Task Queues**
  - Do not hardcode the name in multiple places: Use a shared constant if possible
  - Avoid mixed case: Task Queue names are case sensitive

Task Queues

- **Temporal Services coordinate with Workers through named Task Queues**
  - The name of this Task Queue is specified in the Worker configuration
  - The Task Queue name is also specified by a Client when starting a Workflow
  - Task Queues are dynamically created, so a mismatch in names does not result in an error
- **Recommendations for naming Task Queues**
  - Do not hardcode the name in multiple places: Use a shared constant if possible
  - Avoid mixed case: Task Queue names are case sensitive
  - Use descriptive names, but make them as short and simple as practical

Task Queues

- **Temporal Services coordinate with Workers through named Task Queues**
  - The name of this Task Queue is specified in the Worker configuration
  - The Task Queue name is also specified by a Client when starting a Workflow
  - Task Queues are dynamically created, so a mismatch in names does not result in an error
- **Recommendations for naming Task Queues**
  - Do not hardcode the name in multiple places: Use a shared constant if possible
  - Avoid mixed case: Task Queue names are case sensitive
  - Use descriptive names, but make them as short and simple as practical
- **Plan to run *at least* two Worker Processes per Task Queue**

Workflow IDs

Workflow IDs

- **You specify a Workflow ID when starting a Workflow Execution**

```
// Example: An order processing Workflow might include order number in the Workflow ID
WorkflowOptions options = WorkflowOptions.newBuilder()
        .setWorkflowId("translation-workflow-" + input.getOrderNumber())
        .setTaskQueue("translation-tasks").build();

OrderProcessingWorkflow workflow = client.newWorkflowStub(OrderProcessingWorkflow.class, options);
```

Workflow IDs

- **You specify a Workflow ID when starting a Workflow Execution**
  - This should be a value that is meaningful to your business logic

```
// Example: An order processing Workflow might include order number in the Workflow ID
WorkflowOptions options = WorkflowOptions.newBuilder()
        .setWorkflowId("translation-workflow-" + input.getOrderNumber())
        .setTaskQueue("translation-tasks").build();

OrderProcessingWorkflow workflow = client.newWorkflowStub(OrderProcessingWorkflow.class, options);
```

Workflow IDs

- **You specify a Workflow ID when starting a Workflow Execution**
  - This should be a value that is meaningful to your business logic

```
// Example: An order processing Workflow might include order number in the Workflow ID
WorkflowOptions options = WorkflowOptions.newBuilder()
        .setWorkflowId("translation-workflow-" + input.getOrderNumber())
        .setTaskQueue("translation-tasks").build();

OrderProcessingWorkflow workflow = client.newWorkflowStub(OrderProcessingWorkflow.class, options);
```

- **Must be unique among all *running* Workflow Executions in the namespace**

Workflow IDs

- **You specify a Workflow ID when starting a Workflow Execution**
  - This should be a value that is meaningful to your business logic

```
// Example: An order processing Workflow might include order number in the Workflow ID
WorkflowOptions options = WorkflowOptions.newBuilder()
        .setWorkflowId("translation-workflow-" + input.getOrderNumber())
        .setTaskQueue("translation-tasks").build();

OrderProcessingWorkflow workflow = client.newWorkflowStub(OrderProcessingWorkflow.class, options);
```

- **Must be unique among all *running* Workflow Executions in the namespace**
  - This constraint applies across *all* Workflow Types, not just those of the *same Type*

Workflow IDs

- **You specify a Workflow ID when starting a Workflow Execution**
  - This should be a value that is meaningful to your business logic

```
// Example: An order processing Workflow might include order number in the Workflow ID
WorkflowOptions options = WorkflowOptions.newBuilder()
        .setWorkflowId("translation-workflow-" + input.getOrderNumber())
        .setTaskQueue("translation-tasks").build();

OrderProcessingWorkflow workflow = client.newWorkflowStub(OrderProcessingWorkflow.class, options);
```

- **Must be unique among all *running* Workflow Executions in the namespace**
  - This constraint applies across *all* Workflow Types, not just those of the *same Type*
  - This is an important consideration for choosing a Workflow ID

How Exceptions Affect Activity Execution

How Exceptions Affect Activity Execution

- **An Activity that raises an exception is considered as failed**

How Exceptions Affect Activity Execution

- **An Activity that raises an exception is considered as failed**
  - It may or may not retried, based on the Retry Policy associated with its execution

How Exceptions Affect Activity Execution

- **An Activity that raises an exception is considered as failed**
  - It may or may not retried, based on the Retry Policy associated with its execution
  - By default, Activity Execution is associated with a Retry Policy

How Exceptions Affect Activity Execution

- **An Activity that raises an exception is considered as failed**
  - It may or may not retried, based on the Retry Policy associated with its execution
  - By default, Activity Execution is associated with a Retry Policy
    - The default policy results in retrying until execution succeeds or is canceled

How to Raise Exceptions in Activity Code

How to Raise Exceptions in Activity Code

- **You can throw exceptions as necessary in Activities**
  - Wrapping an exception allows you to throw a checked exception without specifying it in the method signature. These failures will be retried:

```
try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
  String line;
  while ((line = in.readLine()) != null) {
    builder.append(line);
  }
} catch (IOException e) {
  throw Activity.wrap(e);
}
```

How to Raise Exceptions in Activity Code

- **You can throw exceptions as necessary in Activities**
  - Wrapping an exception allows you to throw a checked exception without specifying it in the method signature. These failures will be retried:

```
try (BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()))) {
  String line;
  while ((line = in.readLine()) != null) {
    builder.append(line);
  }
} catch (IOException e) {
  throw Activity.wrap(e);
}
```

  - If you don't want an Activity to retry, you can specify it as a Non-Retryable Failure

```
RetryOptions retryOptions = RetryOptions.newBuilder()
    .setInitialInterval(Duration.ofSeconds(15))
    .setBackoffCoefficient(2.0)
    .setMaximumInterval(Duration.ofSeconds(60))
    .setMaximumAttempts(100)
    .setDoNotRetry("IOException")
    .build();
```

How Exceptions Affect Workflow Execution

How Exceptions Affect Workflow Execution

- **A Workflow that throws an exception *may* be considered failed, depending on the type of failure that is encountered**

How Exceptions Affect Workflow Execution

- **A Workflow that throws an exception *may* be considered failed, depending on the type of failure that is encountered**
  - By default, Workflow Execution is *not* associated with a Retry Policy

How Exceptions Affect Workflow Execution

- **A Workflow that throws an exception *may* be considered failed, depending on the type of failure that is encountered**
  - By default, Workflow Execution is *not* associated with a Retry Policy
  - Raising an exception in a Workflow causes a Workflow Task Failure

How Exceptions Affect Workflow Execution

- **A Workflow that throws an exception *may* be considered failed, depending on the type of failure that is encountered**
  - By default, Workflow Execution is *not* associated with a Retry Policy
  - Raising an exception in a Workflow causes a Workflow Task Failure
    - This failure will be retried

How Exceptions Affect Workflow Execution

- **A Workflow that throws an exception *may* be considered failed, depending on the type of failure that is encountered**
  - By default, Workflow Execution is *not* associated with a Retry Policy
  - Raising an exception in a Workflow causes a Workflow Task Failure
    - This failure will be retried
  - Raising an exception that extends TemporalFailure causes a Workflow Execution Failure

How Exceptions Affect Workflow Execution

- **A Workflow that throws an exception *may* be considered failed, depending on the type of failure that is encountered**
  - By default, Workflow Execution is *not* associated with a Retry Policy
  - Raising an exception in a Workflow causes a Workflow Task Failure
    - This failure will be retried
  - Raising an exception that extends TemporalFailure causes a Workflow Execution Failure
    - The Workflow will be marked as Failed and not retried

How to Raise Exceptions in Workflow Code

How to Raise Exceptions in Workflow Code

- **You can throw exceptions as necessary in Workflows**

```
if (isDelivery && (distance.getKilometers() > 25)) {
      logger.error("Customer lives outside the service area");
      throw ApplicationFailure.newFailure("Customer lives outside the service area",
          OutOfServiceAreaException.class.getName());
    }
```

How to Raise Exceptions in Workflow Code

- **You can throw exceptions as necessary in Workflows**
  - If this exception does not extend TemporalFailure or its child classes, this will fail the Workflow Task and will be retried

```
if (isDelivery && (distance.getKilometers() > 25)) {
      logger.error("Customer lives outside the service area");
      throw ApplicationFailure.newFailure("Customer lives outside the service area",
          OutOfServiceAreaException.class.getName());
    }
```

How to Raise Exceptions in Workflow Code

- **You can throw exceptions as necessary in Workflows**
  - If this exception does not extend TemporalFailure or its child classes, this will fail the Workflow Task and will be retried
    - This is known as a WorkflowTaskFailure

```
if (isDelivery && (distance.getKilometers() > 25)) {
      logger.error("Customer lives outside the service area");
      throw ApplicationFailure.newFailure("Customer lives outside the service area",
          OutOfServiceAreaException.class.getName());
    }
```

How to Raise Exceptions in Workflow Code

- **You can throw exceptions as necessary in Workflows**
  - If this exception does not extend TemporalFailure or its child classes, this will fail the Workflow Task and will be retried
    - This is known as a WorkflowTaskFailure
  - If this exception extends TemporalFailure or its child classes, this will fail the Workflow Execution and the failure will not be retried.

```
if (isDelivery && (distance.getKilometers() > 25)) {
      logger.error("Customer lives outside the service area");
      throw ApplicationFailure.newFailure("Customer lives outside the service area",
          OutOfServiceAreaException.class.getName());
    }
```

How to Raise Exceptions in Workflow Code

- **You can throw exceptions as necessary in Workflows**
  - If this exception does not extend TemporalFailure or its child classes, this will fail the Workflow Task and will be retried
    - This is known as a WorkflowTaskFailure
  - If this exception extends TemporalFailure or its child classes, this will fail the Workflow Execution and the failure will not be retried.
    - This is known as a WorkflowExecutionFailure

```
if (isDelivery && (distance.getKilometers() > 25)) {
      logger.error("Customer lives outside the service area");
      throw ApplicationFailure.newFailure("Customer lives outside the service area",
          OutOfServiceAreaException.class.getName());
    }
```

How to Raise Exceptions in Workflow Code

- **You can throw exceptions as necessary in Workflows**
  - If this exception does not extend TemporalFailure or its child classes, this will fail the Workflow Task and will be retried
    - This is known as a WorkflowTaskFailure
  - If this exception extends TemporalFailure or its child classes, this will fail the Workflow Execution and the failure will not be retried.
    - This is known as a WorkflowExecutionFailure
      - ApplicationFailure is a child of TemporalFailure

```
if (isDelivery && (distance.getKilometers() > 25)) {
      logger.error("Customer lives outside the service area");
      throw ApplicationFailure.newFailure("Customer lives outside the service area",
          OutOfServiceAreaException.class.getName());
    }
```

Logging in Temporal Applications

Logging in Temporal Applications

- **The recommended way of logging in Workflows is via an slf4j implementation provide by the Temporal Java SDK**

Logging in Temporal Applications

- **The recommended way of logging in Workflows is via an slf4j implementation provide by the Temporal Java SDK**
  - Is replay aware

Logging in Temporal Applications

- **The recommended way of logging in Workflows is via an slf4j implementation provide by the Temporal Java SDK**
  - Is replay aware
- **The standard log levels are present, in increasing order of importance**

Logging in Temporal Applications

- **The recommended way of logging in Workflows is via an slf4j implementation provide by the Temporal Java SDK**
  - Is replay aware
- **The standard log levels are present, in increasing order of importance**
  - debug
  - info
  - warn
  - error

Using the WorkflowLogger

Using the WorkflowLogger

- **Accessing and using the Workflow logger using Workflow.getLogger**

```
import org.slf4j.Logger;
import io.temporal.workflow.Workflow;

// instance variable
public static final Logger logger = Workflow.getLogger(TranslationWorkflowImpl.class);

// code within a method
logger.debug("Preparing to execute an Activity")
logger.info("Calculated cost of order. Tax {}, Total {}", tax, total)
```

Logging in Activities

- **Activity logging needs no special Temporal Logger and can be done normally**

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

// instance variable
private static final Logger logger = LoggerFactory.getLogger(TranslationActivitiesImpl.class);

// code within a method
logger.info("getDistance invoked; determining distance to customer address");
logger.error("Database connection failed");
```

Long-Running Executions

Long-Running Executions

- **Temporal Workflows may have executions that span several years**

Long-Running Executions

- **Temporal Workflows may have executions that span several years**
  - Activities may also run for long periods of time

Long-Running Executions

- **Temporal Workflows may have executions that span several years**
  - Activities may also run for long periods of time
- **Workflow and Activity Executions can be synchronous or asynchronous**

Long-Running Executions

- **Temporal Workflows may have executions that span several years**
  - Activities may also run for long periods of time
- **Workflow and Activity Executions can be synchronous or asynchronous**
  - Synchronous calls block, awaiting the Workflow or Activity result before continuing

Long-Running Executions

- **Temporal Workflows may have executions that span several years**
  - Activities may also run for long periods of time
- **Workflow and Activity Executions can be synchronous or asynchronous**
  - Synchronous calls block, awaiting the Workflow or Activity result before continuing
  - Asynchronous calls do not block, so the result must be retrieved later

Long-Running Executions

- **Temporal Workflows may have executions that span several years**
  - Activities may also run for long periods of time
- **Workflow and Activity Executions can be synchronous or asynchronous**
  - Synchronous calls block, awaiting the Workflow or Activity result before continuing
  - Asynchronous calls do not block, so the result must be retrieved later
- **Workflows run until all Tasks yield and resume when new ones appear**

Long-Running Executions

- **Temporal Workflows may have executions that span several years**
  - Activities may also run for long periods of time
- **Workflow and Activity Executions can be synchronous or asynchronous**
  - Synchronous calls block, awaiting the Workflow or Activity result before continuing
  - Asynchronous calls do not block, so the result must be retrieved later
- **Workflows run until all Tasks yield and resume when new ones appear**
  - Example: If a Worker encounters a synchronous Activity call, it halts the current Workflow Task and requests Activity Execution. Once that completes, the Temporal Service adds a new Workflow Task to the queue.

Activity Execution

Activity Execution

- **Synchronous Execution**

```
private final GreetingActivities activities =
  Workflow.newActivityStub(GreetingActivities.class, options);

// Synchronous Activity Method call
String greeting = activities.createGreeting(bill);
```

Activity Execution

- **Synchronous Execution**

```
private final GreetingActivities activities =
  Workflow.newActivityStub(GreetingActivities.class, options);

// Synchronous Activity Method call
String greeting = activities.createGreeting(bill);
```

- **Asynchronous Execution**

```
private final GreetingActivities activities =
  Workflow.newActivityStub(GreetingActivities.class, options);

// Asynchronous Activity Method call
Promise<String> hello = Async.function(activities::createGreeting, name);

// Later in the program
String result = hello.get();
```

Workflow Execution

Workflow Execution

- **Synchronous Execution**

```
// Use a client to request Workflow Execution.
GreetingWorkflow workflow = client.newWorkflowStub(GreetingWorkflow.class, options);
String greeting = workflow.greetSomeone(name);
```

Workflow Execution

- **Synchronous Execution**

```
// Use a client to request Workflow Execution.
GreetingWorkflow workflow = client.newWorkflowStub(GreetingWorkflow.class, options);
String greeting = workflow.greetSomeone(name);
```

- **Asynchronous Execution**

```
import java.util.concurrent.CompletableFuture;
import io.temporal.client.WorkflowClient;

// Options defining code omitted for brevity
GreetingWorkflow workflow = client.newWorkflowStub(GreetingWorkflow.class, options);

// Workflow will be started at this point but the call doesn't block.
CompletableFuture<String> greeting = WorkflowClient.execute(workflow::greetSomeone, "World");

// This line will block, waiting on the result from the Workflow.
String result = greeting.get();
```

Deferring Access to Execution Results

Deferring Access to Execution Results

- **Deferring access to results *may* reduce overall execution time**

```
Promise<String> hello = Async.function(activities::greetInSpanish, name);
Promise<String> goodbye = Async.function(activities::farewellInSpanish, name);
Promise<String> thanks = Async.function(activities::thankInSpanish, name);

// The following lines block until their respective executions have finished

String hello_result = hello.get();
String goodbye_result = goodbye.get();
String thanks_result = thanks.get();
// You could also collect them all at once
```

Deferring Access to Execution Results

- **Deferring access to results *may* reduce overall execution time**
  - This is a good strategy when a Workflow needs to call unrelated Activities

```
Promise<String> hello = Async.function(activities::greetInSpanish, name);
Promise<String> goodbye = Async.function(activities::farewellInSpanish, name);
Promise<String> thanks = Async.function(activities::thankInSpanish, name);

// The following lines block until their respective executions have finished

String hello_result = hello.get();
String goodbye_result = goodbye.get();
String thanks_result = thanks.get();
// You could also collect them all at once
```

Deferring Access to Execution Results

- **Deferring access to results *may* reduce overall execution time**
  - This is a good strategy when a Workflow needs to call unrelated Activities
  - It allows these Activities to execute in parallel, blocking only while accessing their results

```
Promise<String> hello = Async.function(activities::greetInSpanish, name);
Promise<String> goodbye = Async.function(activities::farewellInSpanish, name);
Promise<String> thanks = Async.function(activities::thankInSpanish, name);

// The following lines block until their respective executions have finished

String hello_result = hello.get();
String goodbye_result = goodbye.get();
String thanks_result = thanks.get();
// You could also collect them all at once
```

Idempotence

Idempotence

- **An operation is idempotent if subsequent invocations do not adversely change state beyond that of the initial invocation**

Idempotence

- An operation is idempotent if subsequent invocations do not adversely change state beyond that of the initial invocation
- Consider the idempotence of buttons used to control device power

Idempotence

- An operation is idempotent if subsequent invocations do not adversely change state beyond that of the initial invocation

- Consider the idempotence of buttons used to control device power

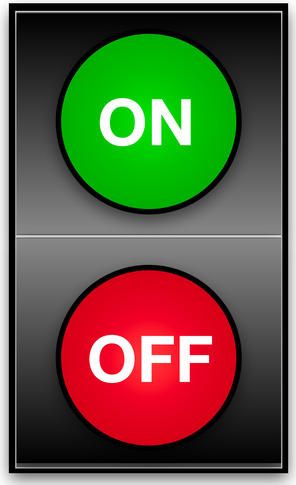
Toggle Button

Idempotence

- An operation is idempotent if subsequent invocations do not adversely change state beyond that of the initial invocation

- Consider the idempotence of buttons used to control device power

Toggle Button



Separate On/Off Buttons

Activity Idempotence

Activity Idempotence

- **It is strongly recommended that you make your Activities idempotent**

Activity Idempotence

- **It is strongly recommended that you make your Activities idempotent**
  - A non-idempotent Activity could adversely affect the state of the system

Activity Idempotence

- **It is strongly recommended that you make your Activities idempotent**
  - A non-idempotent Activity could adversely affect the state of the system
- **For example, consider an Activity that performs the following steps**

Activity Idempotence

- **It is strongly recommended that you make your Activities idempotent**
  - A non-idempotent Activity could adversely affect the state of the system
- **For example, consider an Activity that performs the following steps**
  1. Queries a database

Activity Idempotence

- **It is strongly recommended that you make your Activities idempotent**
  - A non-idempotent Activity could adversely affect the state of the system
- **For example, consider an Activity that performs the following steps**
  1. Queries a database
  2. Calls a microservice using data returned by the query

Activity Idempotence

- **It is strongly recommended that you make your Activities idempotent**
  - A non-idempotent Activity could adversely affect the state of the system
- **For example, consider an Activity that performs the following steps**
  1. Queries a database
  2. Calls a microservice using data returned by the query
  3. Writes the result of the microservice call to the filesystem

Activity Idempotence

- **It is strongly recommended that you make your Activities idempotent**
  - A non-idempotent Activity could adversely affect the state of the system
- **For example, consider an Activity that performs the following steps**
  1. Queries a database
  2. Calls a microservice using data returned by the query
  3. Writes the result of the microservice call to the filesystem
- **This will be retried if any one of those steps fails**

Activity Idempotence

- **It is strongly recommended that you make your Activities idempotent**
  - A non-idempotent Activity could adversely affect the state of the system
- **For example, consider an Activity that performs the following steps**
  1. Queries a database
  2. Calls a microservice using data returned by the query
  3. Writes the result of the microservice call to the filesystem
- **This will be retried if any one of those steps fails**
  - You should balance the granularity of your Activities with the need to keep Event History small

Idempotence and At-Least-Once Execution

Idempotence and At-Least-Once Execution

- Idempotence is also important due to an edge case in distributed systems

Idempotence and At-Least-Once Execution

- Idempotence is also important due to an edge case in distributed systems
- Consider the following scenario

Idempotence and At-Least-Once Execution

- **Idempotence is also important due to an edge case in distributed systems**
- **Consider the following scenario**
  - Worker polls the Temporal Service and accepts an Activity Task

Idempotence and At-Least-Once Execution

- **Idempotence is also important due to an edge case in distributed systems**
- **Consider the following scenario**
  - Worker polls the Temporal Service and accepts an Activity Task
  - Worker begins executing the Activity

Idempotence and At-Least-Once Execution

- **Idempotence is also important due to an edge case in distributed systems**
- **Consider the following scenario**
  - Worker polls the Temporal Service and accepts an Activity Task
  - Worker begins executing the Activity
  - Worker finishes executing the Activity

Idempotence and At-Least-Once Execution

- **Idempotence is also important due to an edge case in distributed systems**
- **Consider the following scenario**
  - Worker polls the Temporal Service and accepts an Activity Task
  - Worker begins executing the Activity
  - Worker finishes executing the Activity
  - Worker crashes just before reporting the result to the Temporal Service

Idempotence and At-Least-Once Execution

- **Idempotence is also important due to an edge case in distributed systems**
- **Consider the following scenario**
  - Worker polls the Temporal Service and accepts an Activity Task
  - Worker begins executing the Activity
  - Worker finishes executing the Activity
  - Worker crashes just before reporting the result to the Temporal Service
- **Activity will be retried since Event History does not indicate completion**

Idempotence and At-Least-Once Execution

- **Idempotence is also important due to an edge case in distributed systems**
- **Consider the following scenario**
  - Worker polls the Temporal Service and accepts an Activity Task
  - Worker begins executing the Activity
  - Worker finishes executing the Activity
  - Worker crashes just before reporting the result to the Temporal Service
- **Activity will be retried since Event History does not indicate completion**
  - Therefore, idempotence is essential for preventing unwanted changes in application state

Idempotency Keys

Idempotency Keys

- You can achieve idempotency by ignoring duplicate requests

Idempotency Keys

- **You can achieve idempotency by ignoring duplicate requests**
  - This raises a question: How can one distinguish a *duplicate* request from one that looks similar?

Idempotency Keys

- **You can achieve idempotency by ignoring duplicate requests**
  - This raises a question: How can one distinguish a *duplicate* request from one that looks similar?
- **Idempotency keys are unique identifiers associated with a request**

Idempotency Keys

- **You can achieve idempotency by ignoring duplicate requests**
  - This raises a question: How can one distinguish a *duplicate* request from one that looks similar?
- **Idempotency keys are unique identifiers associated with a request**
  - They are interpreted by the system receiving the request (e.g., a payment processor)

Idempotency Keys

- **You can achieve idempotency by ignoring duplicate requests**
  - This raises a question: How can one distinguish a *duplicate* request from one that looks similar?
- **Idempotency keys are unique identifiers associated with a request**
  - They are interpreted by the system receiving the request (e.g., a payment processor)
  - In a Temporal Activity, you can compose one from a Workflow Run ID and Activity ID

```
import io.temporal.activity.Activity;
import io.temporal.activity.ActivityExecutionContext;

ActivityExecutionContext context = Activity.getExecutionContext();
String idempotencyKey = context.getInfo().getRunId() + "-" context.getInfo().getActivityId();
```

Idempotency Keys

- **You can achieve idempotency by ignoring duplicate requests**
  - This raises a question: How can one distinguish a *duplicate* request from one that looks similar?
- **Idempotency keys are unique identifiers associated with a request**
  - They are interpreted by the system receiving the request (e.g., a payment processor)
  - In a Temporal Activity, you can compose one from a Workflow Run ID and Activity ID
  - Guaranteed to be consistent across retry attempts, but unique among Workflow Executions

```
import io.temporal.activity.Activity;
import io.temporal.activity.ActivityExecutionContext;

ActivityExecutionContext context = Activity.getExecutionContext();
String idempotencyKey = context.getInfo().getRunId() + "-" context.getInfo().getActivityId();
```

Beginner to Builder Bootcamp - Java

00. About this Workshop
01. The Basics of Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
04. Understanding Event History
05. Understanding Workflow Determinism
06. Signals, Queries, and Workflow Updates
07. Timeouts and Retry Policies
08. Recovering from Failure
09. Conclusion

What is a Timer?

What is a Timer?

- **Timers are used to introduce delays into a Workflow Execution**

What is a Timer?

- **Timers are used to introduce delays into a Workflow Execution**
  - Code that awaits the Timer pauses execution for the specified duration

What is a Timer?

- **Timers are used to introduce delays into a Workflow Execution**
  - Code that awaits the Timer pauses execution for the specified duration
  - The duration is fixed and may range from seconds to years

What is a Timer?

- **Timers are used to introduce delays into a Workflow Execution**
  - Code that awaits the Timer pauses execution for the specified duration
  - The duration is fixed and may range from seconds to years
  - Once the time has elapsed, the Timer fires, and execution continues

What is a Timer?

- **Timers are used to introduce delays into a Workflow Execution**
  - Code that awaits the Timer pauses execution for the specified duration
  - The duration is fixed and may range from seconds to years
  - Once the time has elapsed, the Timer fires, and execution continues
- **Workflow code must not use Java's built-in timers or sleep (non-deterministic)**

Use Cases for Timers

Use Cases for Timers

- **Execute an Activity multiple times at predefined intervals**

Use Cases for Timers

- **Execute an Activity multiple times at predefined intervals**
  - Send reminder e-mails to a new customer after 1, 7, and 30 days

Use Cases for Timers

- **Execute an Activity multiple times at predefined intervals**
  - Send reminder e-mails to a new customer after 1, 7, and 30 days
- **Execute an Activity multiple times at dynamically-calculated intervals**

Use Cases for Timers

- **Execute an Activity multiple times at predefined intervals**
  - Send reminder e-mails to a new customer after 1, 7, and 30 days
- **Execute an Activity multiple times at dynamically-calculated intervals**
  - Delay calling the next Activity based on a value returned by a previous one

Use Cases for Timers

- **Execute an Activity multiple times at predefined intervals**
  - Send reminder e-mails to a new customer after 1, 7, and 30 days
- **Execute an Activity multiple times at dynamically-calculated intervals**
  - Delay calling the next Activity based on a value returned by a previous one
- **Allow time for offline steps to complete**

Use Cases for Timers

- **Execute an Activity multiple times at predefined intervals**
  - Send reminder e-mails to a new customer after 1, 7, and 30 days
- **Execute an Activity multiple times at dynamically-calculated intervals**
  - Delay calling the next Activity based on a value returned by a previous one
- **Allow time for offline steps to complete**
  - Wait five business days for a check to clear before proceeding

Timer APIs Provided by the Java SDK

Timer APIs Provided by the Java SDK

- **Java SDK offers two Workflow-safe, replay-aware ways to start a Timer**

Timer APIs Provided by the Java SDK

- **Java SDK offers two Workflow-safe, replay-aware ways to start a Timer**
  - There are synchronous and asynchronous versions

Timer APIs Provided by the Java SDK

- **Java SDK offers two Workflow-safe, replay-aware ways to start a Timer**
  - There are synchronous and asynchronous versions
  - A Workflow-safe replacement for Thread.sleep and java.util.Timer are available

Timer APIs Provided by the Java SDK

- **Java SDK offers two Workflow-safe, replay-aware ways to start a Timer**
  - There are synchronous and asynchronous versions
  - A Workflow-safe replacement for Thread.sleep and java.util.Timer are available
  - Workflow code must not use Java's methods for timers (non-deterministic)

Pausing Workflow Execution for a Specified Duration

Pausing Workflow Execution for a Specified Duration

- **Use the Workflow.sleep method for this**

```
import java.time.Duration;
import io.temporal.workflow.Workflow;

// This will pause Workflow Execution for 10 seconds
Workflow.sleep(Duration.ofSeconds(10));
```

Pausing Workflow Execution for a Specified Duration

- **Use the Workflow.sleep method for this**
  - This is an alternative to Java's Thread.sleep method

```
import java.time.Duration;
import io.temporal.workflow.Workflow;

// This will pause Workflow Execution for 10 seconds
Workflow.sleep(Duration.ofSeconds(10));
```

Pausing Workflow Execution for a Specified Duration

- **Use the Workflow.sleep method for this**
  - This is an alternative to Java's Thread.sleep method
  - It blocks until the Timer is fired (or is canceled)

```
import java.time.Duration;
import io.temporal.workflow.Workflow;

// This will pause Workflow Execution for 10 seconds
Workflow.sleep(Duration.ofSeconds(10));
```

Running Code a Specific Point in the Future

Running Code a Specific Point in the Future

- **Use the Workflow.newTimer method for this**

```
import java.time.Duration;
import io.temporal.workflow.Workflow;

// Workflow.newTimer is a Workflow-safe counterpart to java.util.Timer
Promise timerPromise = Workflow.newTimer(Duration.ofSeconds(30))
logger.info("The timer was set")

// Unlike Workflow.sleep, waiting for the timer is a separate operation
timerPromise.get()
logger.Info("The timer has fired")
```

Running Code a Specific Point in the Future

- **Use the Workflow.newTimer method for this**
  - This is an alternative to Java's `java.util.NewTimer` method

```
import java.time.Duration;
import io.temporal.workflow.Workflow;

// Workflow.newTimer is a Workflow-safe counterpart to java.util.Timer
Promise timerPromise = Workflow.newTimer(Duration.ofSeconds(30))
logger.info("The timer was set")

// Unlike Workflow.sleep, waiting for the timer is a separate operation
timerPromise.get()
logger.Info("The timer has fired")
```

Running Code a Specific Point in the Future

- **Use the Workflow.newTimer method for this**
  - This is an alternative to Java's `java.util.NewTimer` method
  - This returns a Promise, which becomes ready when the Timer fires (or is canceled)

```
import java.time.Duration;
import io.temporal.workflow.Workflow;

// Workflow.newTimer is a Workflow-safe counterpart to java.util.Timer
Promise timerPromise = Workflow.newTimer(Duration.ofSeconds(30))
logger.info("The timer was set")

// Unlike Workflow.sleep, waiting for the timer is a separate operation
timerPromise.get()
logger.Info("The timer has fired")
```

What Happens to a Timer if the Worker Crashes?

What Happens to a Timer if the Worker Crashes?

- **Timers are maintained by the Temporal Service**

What Happens to a Timer if the Worker Crashes?

- **Timers are maintained by the Temporal Service**
  - Once set, they fire regardless of whether any Workers are running

What Happens to a Timer if the Worker Crashes?

- **Timers are maintained by the Temporal Service**
  - Once set, they fire regardless of whether any Workers are running
- **Scenario: Timer set for 10 seconds and Worker crashes 3 seconds later**

What Happens to a Timer if the Worker Crashes?

- **Timers are maintained by the Temporal Service**
  - Once set, they fire regardless of whether any Workers are running
- **Scenario: Timer set for 10 seconds and Worker crashes 3 seconds later**
  - If Worker is restarted within 7 seconds, it will be running when the Timer fires

What Happens to a Timer if the Worker Crashes?

- **Timers are maintained by the Temporal Service**
  - Once set, they fire regardless of whether any Workers are running
- **Scenario: Timer set for 10 seconds and Worker crashes 3 seconds later**
  - If Worker is restarted within 7 seconds, it will be running when the Timer fires
    - It will be as if the Worker had never crashed at all

What Happens to a Timer if the Worker Crashes?

- **Timers are maintained by the Temporal Service**
  - Once set, they fire regardless of whether any Workers are running
- **Scenario: Timer set for 10 seconds and Worker crashes 3 seconds later**
  - If Worker is restarted within 7 seconds, it will be running when the Timer fires
    - It will be as if the Worker had never crashed at all
  - If Worker is restarted 5 *minutes* later, the Timer will have already fired

What Happens to a Timer if the Worker Crashes?

- **Timers are maintained by the Temporal Service**
  - Once set, they fire regardless of whether any Workers are running
- **Scenario: Timer set for 10 seconds and Worker crashes 3 seconds later**
  - If Worker is restarted within 7 seconds, it will be running when the Timer fires
    - It will be as if the Worker had never crashed at all
  - If Worker is restarted 5 *minutes* later, the Timer will have already fired
    - In this case, the Worker will resume executing the Workflow code without delay

Exercise #2: Observing Durable Execution

https://t.mp/replay26-workshop-java

- **During this exercise, you will**
  - Create Workflow and Activity loggers
  - Add logging statements to the code
  - Add a Timer to the Workflow Definition
  - Launch two Workers, run the Workflow, and kill one of the Workers, observing that the remaining Worker completes the execution
- **Refer to this exercise's README.md file for details**
  - Don't forget to make your changes in the practice subdirectory

Beginner to Builder Bootcamp - Java

00. About this Workshop
01. The Basics of Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
04. Understanding Event History
05. Understanding Workflow Determinism
06. Signals, Queries, and Workflow Updates
07. Timeouts and Retry Policies
08. Recovering from Failure
09. Conclusion

Workflow Definition

```java
import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;

@WorkflowInterface
public interface HelloWorkflow {

    @WorkflowMethod
    String greetSomeone(String name);
}

public class HelloWorkflowImpl implements HelloWorkflow {

    @Override
    public MyWorkflowOutput greetSomeone(MyWorkflowInput name) {
        return new WorkflowOutput("Hello " + name + "!");
    }
}
```

Workflow Definition

combined with

Execution Request

```java
import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;

@WorkflowInterface
public interface HelloWorkflow {

    @WorkflowMethod
    String greetSomeone(String name);
}

public class HelloWorkflowImpl implements HelloWorkflow {

    @Override
    public MyWorkflowOutput greetSomeone(MyWorkflowInput name) {
        return new WorkflowOutput("Hello " + name + "!");
    }
}
```

+

```java
MyWorkflowOutput greeting = workflow.greetSomeone("Brian");
```

Workflow Definition

```java
import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;

@WorkflowInterface
public interface HelloWorkflow {

    @WorkflowMethod
    String greetSomeone(String name);
}

public class HelloWorkflowImpl implements HelloWorkflow {

    @Override
    public MyWorkflowOutput greetSomeone(MyWorkflowInput name) {
        return new WorkflowOutput("Hello " + name + "!");
    }
}
```

combined with

+

Execution Request

```java
MyWorkflowOutput greeting = workflow.greetSomeone("Brian");
```

results in

=

Workflow Execution

Running Workflow

1 Workflow Definition

```java
import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;

@WorkflowInterface
public interface HelloWorkflow {

    @WorkflowMethod
    String greetSomeone(String name);
}

public class HelloWorkflowImpl implements HelloWorkflow {

    @Override
    public MyWorkflowOutput greetSomeone(MyWorkflowInput name) {
        return new WorkflowOutput("Hello " + name + "!");
    }
}
```

combined with

+ +

n Execution Requests

```java
workflow.greetSomeone("Brian");
```

```java
workflow.greetSomeone("Tom");
```

results in

= =

n Workflow Executions

Workflow Execution 1

Workflow Execution 2

1 Workflow Definition

```java
import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;

@WorkflowInterface
public interface HelloWorkflow {

    @WorkflowMethod
    String greetSomeone(String name);
}

public class HelloWorkflowImpl implements HelloWorkflow {

    @Override
    public MyWorkflowOutput greetSomeone(MyWorkflowInput name) {
        return new WorkflowOutput("Hello " + name + "!");
    }
}
```

combined with

\+ \+

n Execution Requests

```java
workflow.greetSomeone("Brian");
```

```java
workflow.greetSomeone("Tom");
```

results in

= =

n Workflow Executions

Workflow Execution 1

Workflow Execution 2

1 Workflow Definition

```java
import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;

@WorkflowInterface
public interface HelloWorkflow {

    @WorkflowMethod
    String greetSomeone(String name);
}

public class HelloWorkflowImpl implements HelloWorkflow {

    @Override
    public MyWorkflowOutput greetSomeone(MyWorkflowInput name) {
        return new WorkflowOutput("Hello " + name + "!");
    }
}
```

combined with

\+ \+

n Execution Requests

```java
workflow.greetSomeone("Brian");
```

```java
workflow.greetSomeone("Tom");
```

results in

= =

n Workflow Executions

Workflow Execution 1

Workflow Execution 2

1 Workflow Definition

```java
import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;

@WorkflowInterface
public interface HelloWorkflow {

    @WorkflowMethod
    String greetSomeone(String name);
}

public class HelloWorkflowImpl implements HelloWorkflow {

    @Override
    public MyWorkflowOutput greetSomeone(MyWorkflowInput name) {
        return new WorkflowOutput("Hello " + name + "!");
    }
}
```

combined with

\+ \+

n Execution Requests

```java
workflow.greetSomeone("Brian");
```

```java
workflow.greetSomeone("Tom");
```

results in

= =

n Workflow Executions

Workflow Execution 1

Workflow Execution 2

Workflow Execution States

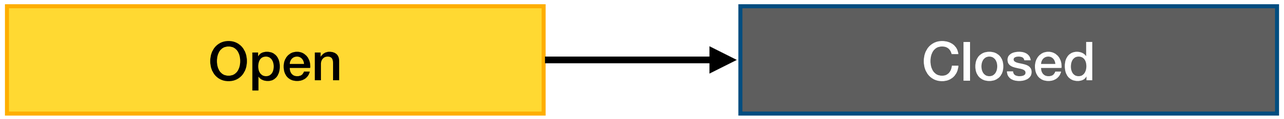
Workflow Execution States

Workflow Execution States

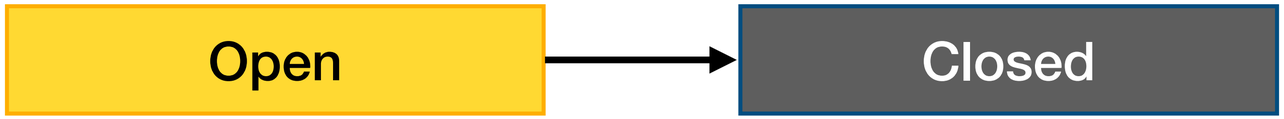
Workflow Execution States

Workflow Execution States

Workflow Execution States

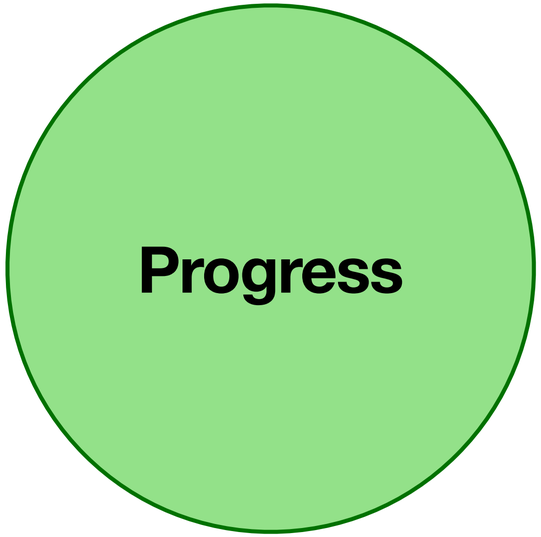


Workflow Execution States



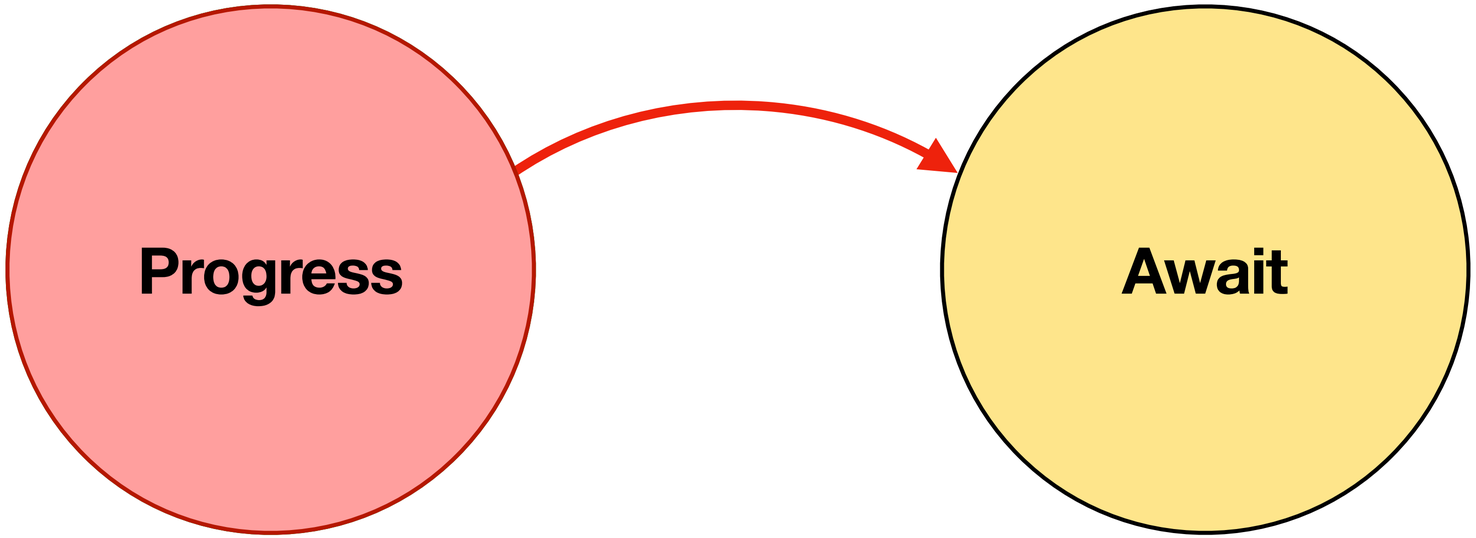
This is a one-way transition

What Happens During Workflow Execution



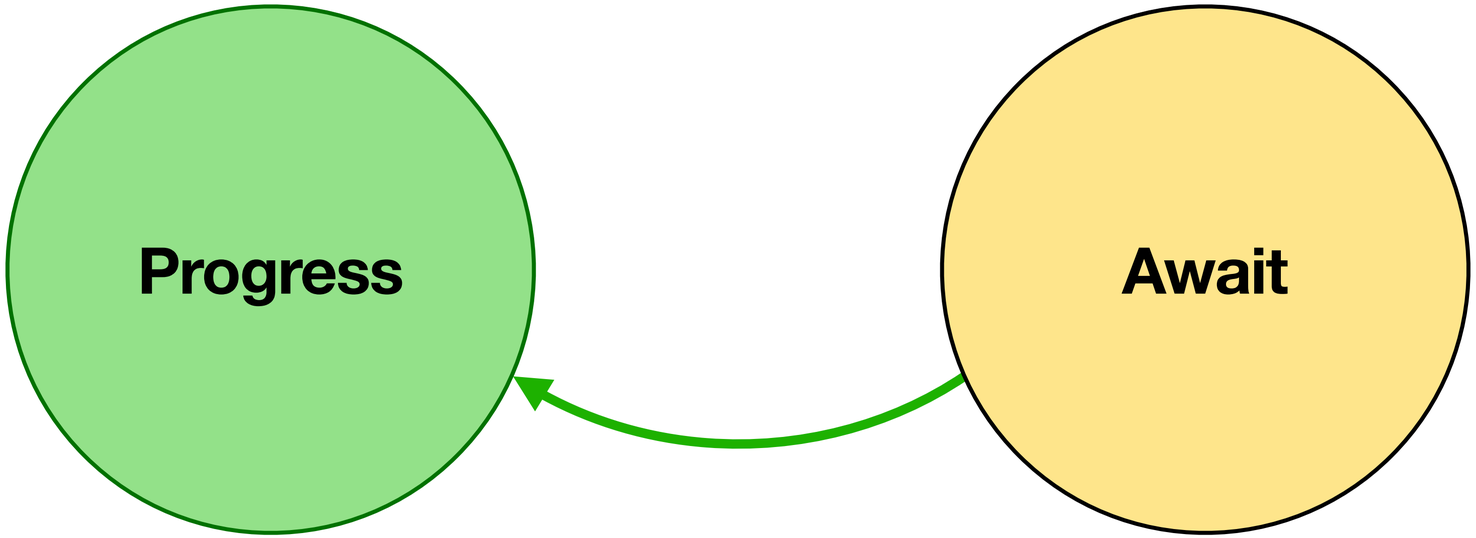
This cycle continues throughout Workflow Execution

What Happens During Workflow Execution



This cycle continues throughout Workflow Execution

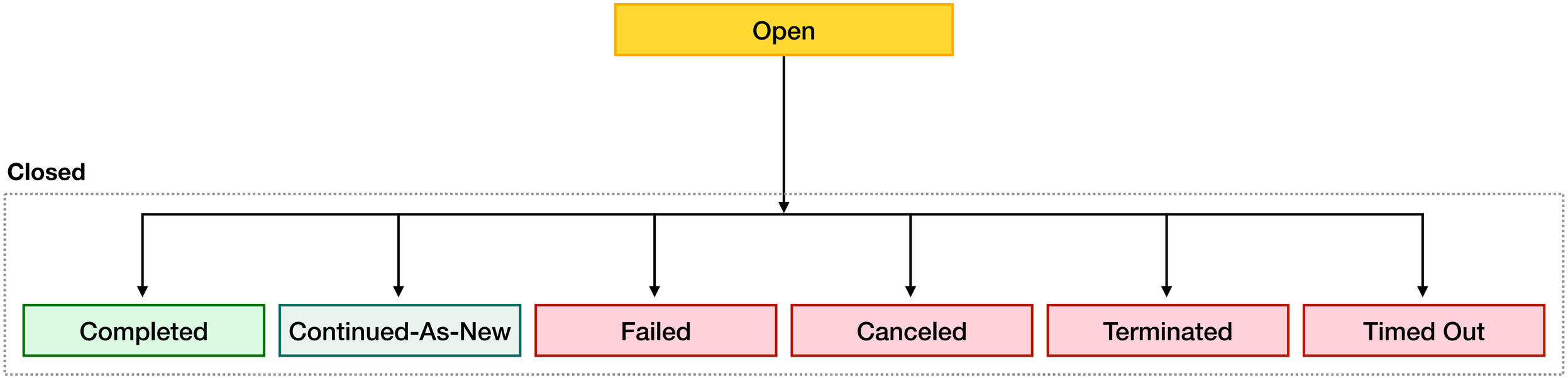
What Happens During Workflow Execution



This cycle continues throughout Workflow Execution

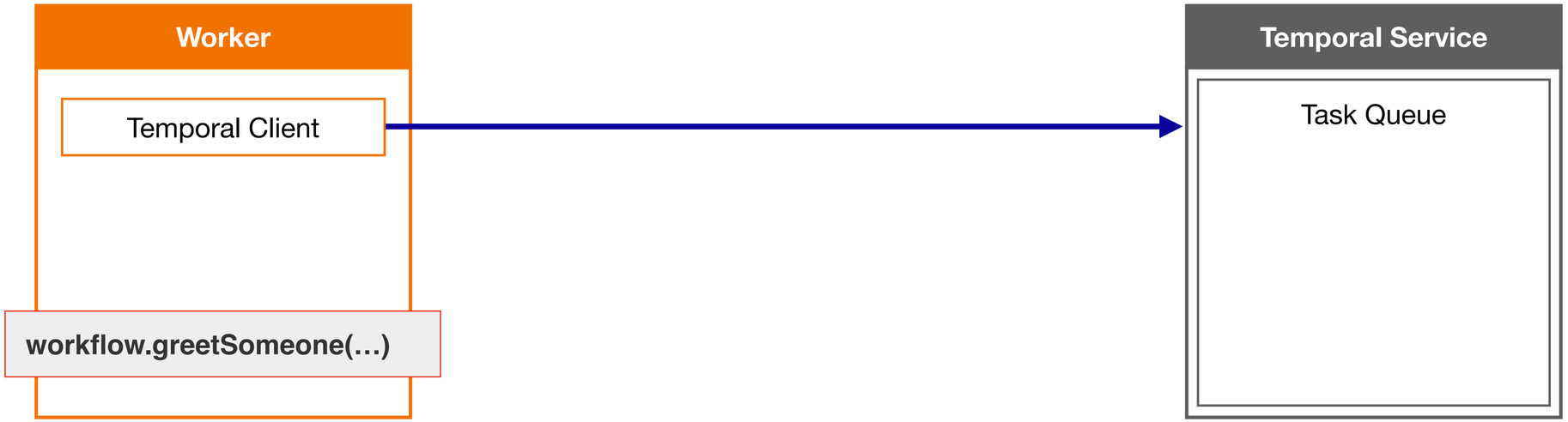
Summary of Workflow Execution States

Summary of Workflow Execution States

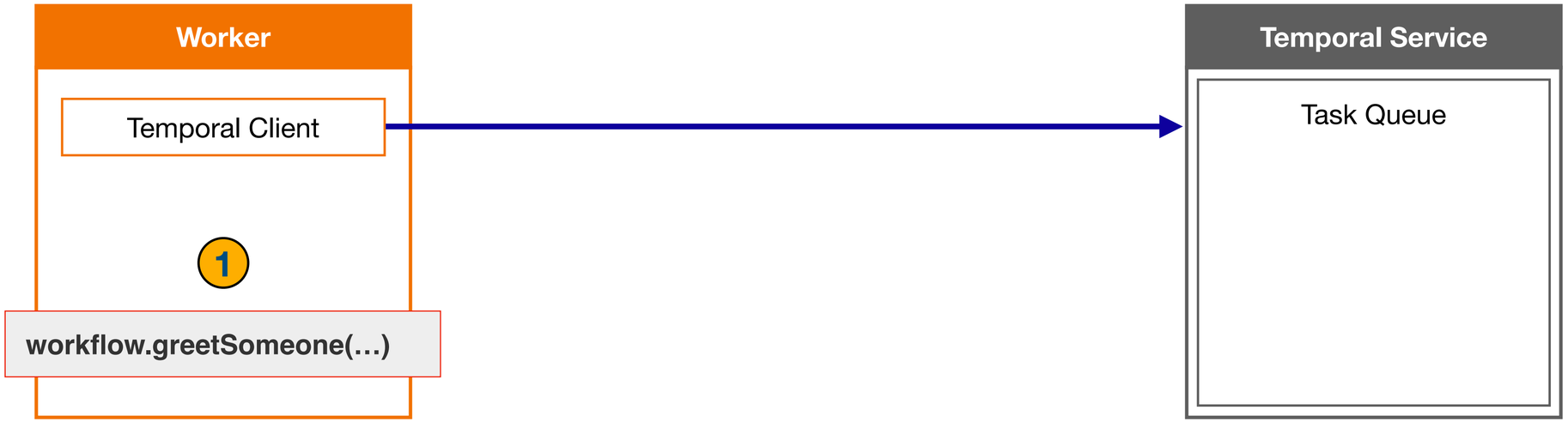


How Workflow Code Maps to Commands

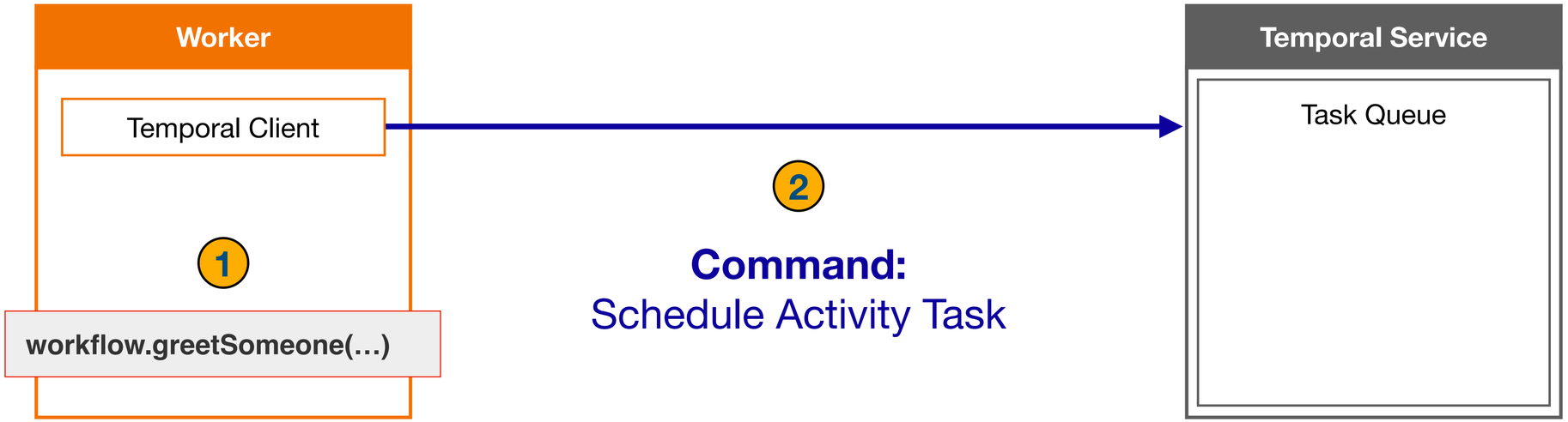
Commands


Worker
Temporal Client
workflow.greetSomeone(…)
Temporal Service
Task Queue

Commands

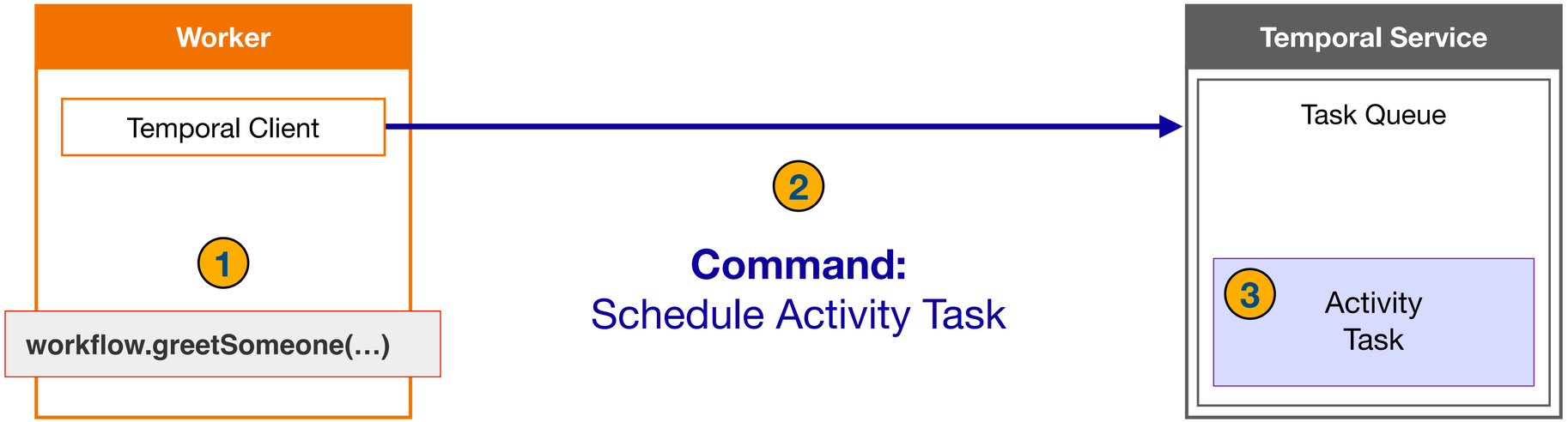

Worker
Temporal Client
1
workflow.greetSomeone(…)
Temporal Service
Task Queue

Commands



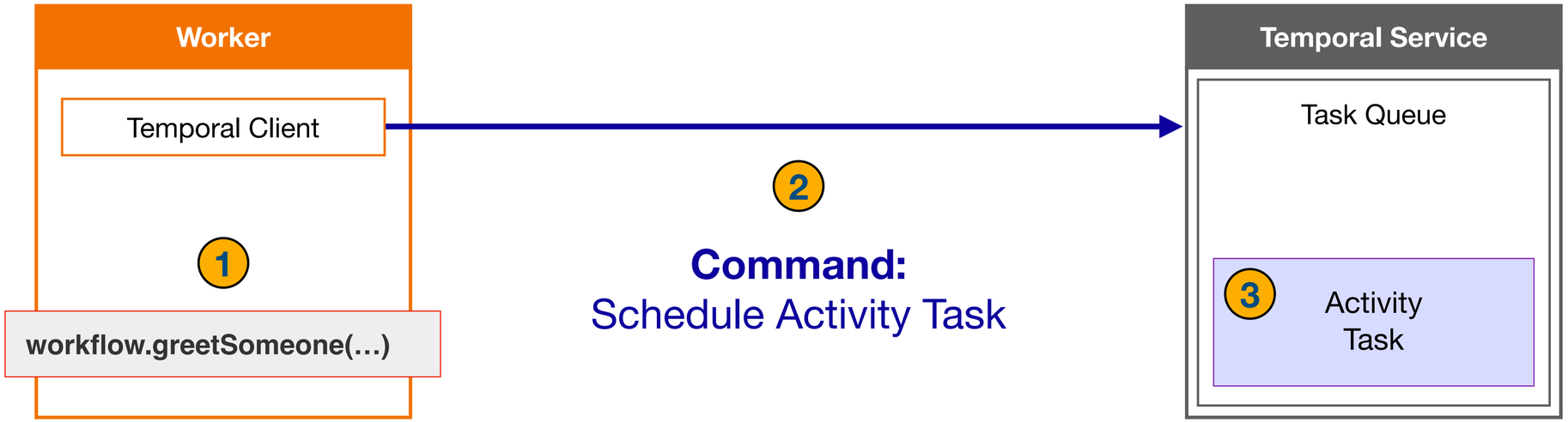
- Certain API calls result in the Worker issuing a Command to the Temporal Service

Commands



- Certain API calls result in the Worker issuing a Command to the Temporal Service
- The Service acts on these commands, but also stores them

Commands



- Certain API calls result in the Worker issuing a Command to the Temporal Service
- The Service acts on these commands, but also stores them
- This allows the Worker to recreate the state of a Workflow Execution following a crash

Basic Temporal Workflow Definition

pseudocode

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

pseudocode

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Defines a Start-to-Close Timeout

pseudocode

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Defines a Start-to-Close Timeout
- Calculates total price of the pizzas

pseudocode

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Defines a Start-to-Close Timeout
- Calculates total price of the pizzas
- Determines distance to customer

pseudocode

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Defines a Start-to-Close Timeout
- Calculates total price of the pizzas
- Determines distance to customer
- Fails if customer is too far away for delivery

pseudocode

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Defines a Start-to-Close Timeout
- Calculates total price of the pizzas
- Determines distance to customer
- Fails if customer is too far away for delivery
- Sleeps for 30 minutes

pseudocode

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Defines a Start-to-Close Timeout
- Calculates total price of the pizzas
- Determines distance to customer
- Fails if customer is too far away for delivery
- Sleeps for 30 minutes
- Populates a class with billing information

pseudocode

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Defines a Start-to-Close Timeout
- Calculates total price of the pizzas
- Determines distance to customer
- Fails if customer is too far away for delivery
- Sleeps for 30 minutes
- Populates a class with billing information
- Sends a bill to the customer

Basic Temporal Workflow Definition

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- A Workflow is a sequence of steps

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- A Workflow is a sequence of steps
- Some steps are *internal to the Workflow*

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- A Workflow is a sequence of steps
- Some steps are *internal to the Workflow*
  - Do not involve interaction with the Temporal Service

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- A Workflow is a sequence of steps
- Some steps are *internal to the Workflow*
  - Do not involve interaction with the Temporal Service
- Examples include

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- A Workflow is a sequence of steps
- Some steps are *internal to the Workflow*
  - Do not involve interaction with the Temporal Service
- Examples include
  - Setting configuration parameters

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- A Workflow is a sequence of steps
- Some steps are *internal to the Workflow*
  - Do not involve interaction with the Temporal Service
- Examples include
  - Setting configuration parameters
  - Evaluating variables or expressions

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- A Workflow is a sequence of steps
- Some steps are *internal to the Workflow*
  - Do not involve interaction with the Temporal Service
- Examples include
  - Setting configuration parameters
  - Evaluating variables or expressions
  - Performing calculations

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- A Workflow is a sequence of steps
- Some steps are *internal to the Workflow*
  - Do not involve interaction with the Temporal Service
- Examples include
  - Setting configuration parameters
  - Evaluating variables or expressions
  - Performing calculations
  - Populating data structures

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- A Workflow is a sequence of steps
- Some steps are *internal to the Workflow*
  - Do not involve interaction with the Temporal Service
- Examples include
  - Setting configuration parameters
  - Evaluating variables or expressions
  - Performing calculations
  - Populating data structures
- These internal steps are highlighted in white

Basic Temporal Workflow Definition

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Other steps *do* involve interaction with the cluster

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Other steps *do* involve interaction with the cluster
- Examples include

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Other steps *do* involve interaction with the cluster
- Examples include
  - Executing an Activity

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Other steps *do* involve interaction with the cluster
- Examples include
  - Executing an Activity
  - Throwing an exception from the Workflow

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Other steps *do* involve interaction with the cluster
- Examples include
  - Executing an Activity
  - Throwing an exception from the Workflow
  - Setting a Timer

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Other steps *do* involve interaction with the cluster
- Examples include
  - Executing an Activity
  - Throwing an exception from the Workflow
  - Setting a Timer
  - Returning a value from the Workflow

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Basic Temporal Workflow Definition

- Other steps *do* involve interaction with the cluster
- Examples include
  - Executing an Activity
  - Throwing an exception from the Workflow
  - Setting a Timer
  - Returning a value from the Workflow
- These external steps are highlighted in yellow

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Command

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Command

ScheduleActivityTask

("pizza-tasks", GetDistance, { Line1: "123 Oak St.", Line2: "", ... })

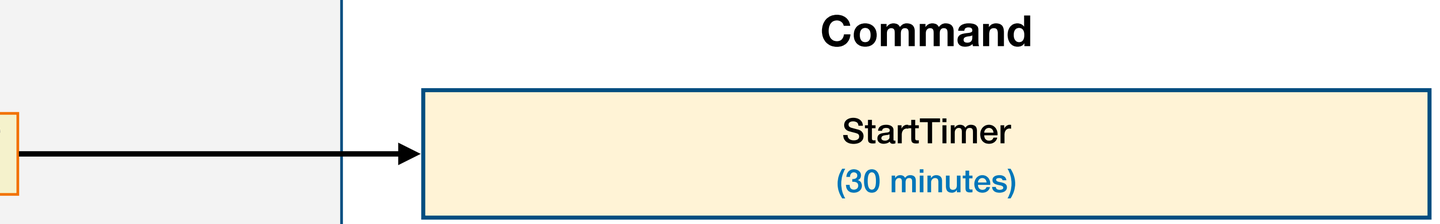
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Command

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Command

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Command

ScheduleActivityTask

("pizza-tasks", SendBill, { Amount: 2750, Description: "Pizzas", ... }

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Command

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Command

CompleteWorkflowExecution

({ConfirmationNumber: "TPD-26074139"})

Workflow Execution Event History

Workflow Execution Event History

- **Each Workflow Execution is associated with an Event History**

Workflow Execution Event History

- **Each Workflow Execution is associated with an Event History**
- **Represents the source of truth for what transpired during execution**

Workflow Execution Event History

- **Each Workflow Execution is associated with an Event History**
- **Represents the source of truth for what transpired during execution**
  - As viewed from the Temporal Service's perspective

Workflow Execution Event History

- **Each Workflow Execution is associated with an Event History**
- **Represents the source of truth for what transpired during execution**
  - As viewed from the Temporal Service's perspective
  - Durably persisted by the Temporal Service

Workflow Execution Event History

- **Each Workflow Execution is associated with an Event History**
- **Represents the source of truth for what transpired during execution**
  - As viewed from the Temporal Service's perspective
  - Durably persisted by the Temporal Service
- **Event Histories serve two key purposes in Temporal**

Workflow Execution Event History

- **Each Workflow Execution is associated with an Event History**
- **Represents the source of truth for what transpired during execution**
  - As viewed from the Temporal Service's perspective
  - Durably persisted by the Temporal Service
- **Event Histories serve two key purposes in Temporal**
  - Allow reconstruction of Workflow state following a crash

Workflow Execution Event History

- **Each Workflow Execution is associated with an Event History**
- **Represents the source of truth for what transpired during execution**
  - As viewed from the Temporal Service's perspective
  - Durably persisted by the Temporal Service
- **Event Histories serve two key purposes in Temporal**
  - Allow reconstruction of Workflow state following a crash
  - Enable developers to investigate both current and past executions

Workflow Execution Event History

- **Each Workflow Execution is associated with an Event History**
- **Represents the source of truth for what transpired during execution**
  - As viewed from the Temporal Service's perspective
  - Durably persisted by the Temporal Service
- **Event Histories serve two key purposes in Temporal**
  - Allow reconstruction of Workflow state following a crash
  - Enable developers to investigate both current and past executions
- **You can access them from code, command line, and Web UI**

Event History Content

Event History Content

- **An Event History acts as an ordered append-only record of Events**

Event History Content

- **An Event History acts as an ordered append-only record of Events**
  - Begins with the WorkflowExecutionStarted Event

Event History Content

- **An Event History acts as an ordered append-only record of Events**
  - Begins with the WorkflowExecutionStarted Event
  - New Events are appended as Workflow Execution progresses

Event History Content

- **An Event History acts as an ordered append-only record of Events**
  - Begins with the WorkflowExecutionStarted Event
  - New Events are appended as Workflow Execution progresses
  - Ends when the Workflow Execution closes

Event History Limits

Event History Limits

- **Temporal places limits on a Workflow Execution's Event History**

Event History Limits

- Temporal places limits on a Workflow Execution's Event History
- Warnings begin after 10K (10,240) Events

Event History Limits

- **Temporal places limits on a Workflow Execution's Event History**
- **Warnings begin after 10K (10,240) Events**
  - These say "history size exceeds warn limit" and will appear the Temporal Service logs

Event History Limits

- **Temporal places limits on a Workflow Execution's Event History**
- **Warnings begin after 10K (10,240) Events**
  - These say "history size exceeds warn limit" and will appear the Temporal Service logs
  - They identify the Workflow ID, Run ID, and namespace for the Workflow Execution

Event History Limits

- **Temporal places limits on a Workflow Execution's Event History**
- **Warnings begin after 10K (10,240) Events**
  - These say "history size exceeds warn limit" and will appear the Temporal Service logs
  - They identify the Workflow ID, Run ID, and namespace for the Workflow Execution
- **Workflow Execution will be *terminated* after exceeding additional limits**

Event History Limits

- **Temporal places limits on a Workflow Execution's Event History**
- **Warnings begin after 10K (10,240) Events**
  - These say "history size exceeds warn limit" and will appear the Temporal Service logs
  - They identify the Workflow ID, Run ID, and namespace for the Workflow Execution
- **Workflow Execution will be *terminated* after exceeding additional limits**
  - If its Event History exceeds 50K (51,200) Events

Event History Limits

- **Temporal places limits on a Workflow Execution's Event History**
- **Warnings begin after 10K (10,240) Events**
  - These say "history size exceeds warn limit" and will appear the Temporal Service logs
  - They identify the Workflow ID, Run ID, and namespace for the Workflow Execution
- **Workflow Execution will be *terminated* after exceeding additional limits**
  - If its Event History exceeds 50K (51,200) Events
  - If its Event History exceeds 50 MB of storage

Event Structure and Characteristics

Event Structure and Characteristics

- **Every Event always contains the following three attributes**

Event Structure and Characteristics

- **Every Event always contains the following three attributes**
  - ID (uniquely identifies this Event within the History)

Event Structure and Characteristics

- **Every Event always contains the following three attributes**
  - ID (uniquely identifies this Event within the History)
  - Time (timestamp representing when the Event occurred)

Event Structure and Characteristics

- **Every Event always contains the following three attributes**
  - ID (uniquely identifies this Event within the History)
  - Time (timestamp representing when the Event occurred)
  - Type (the kind of Event it is)

Attributes Vary by Event Type

Attributes Vary by Event Type

- **Additionally, each Event contains attributes specific to its type**

Attributes Vary by Event Type

- **Additionally, each Event contains attributes specific to its type**
  - **WorkflowExecutionStarted** contains the Workflow Type and input parameters

Attributes Vary by Event Type

- **Additionally, each Event contains attributes specific to its type**
  - **WorkflowExecutionStarted** contains the Workflow Type and input parameters
  - **WorkflowExecutionCompleted** contains the result returned by the Workflow method

Attributes Vary by Event Type

- **Additionally, each Event contains attributes specific to its type**
  - **WorkflowExecutionStarted** contains the Workflow Type and input parameters
  - **WorkflowExecutionCompleted** contains the result returned by the Workflow method
  - **WorkflowExecutionFailed** contains the exception thrown by the Workflow method

Attributes Vary by Event Type

- **Additionally, each Event contains attributes specific to its type**
  - **WorkflowExecutionStarted** contains the Workflow Type and input parameters
  - **WorkflowExecutionCompleted** contains the result returned by the Workflow method
  - **WorkflowExecutionFailed** contains the exception thrown by the Workflow method
  - **ActivityTaskScheduled** contains the Activity Type and input parameters

Attributes Vary by Event Type

- **Additionally, each Event contains attributes specific to its type**
  - **WorkflowExecutionStarted** contains the Workflow Type and input parameters
  - **WorkflowExecutionCompleted** contains the result returned by the Workflow method
  - **WorkflowExecutionFailed** contains the exception thrown by the Workflow method
  - **ActivityTaskScheduled** contains the Activity Type and input parameters
  - **ActivityTaskCompleted** contains the result returned by the Activity method

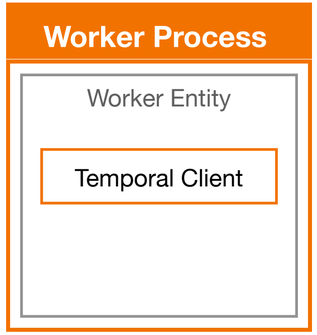
How Commands Map to Events

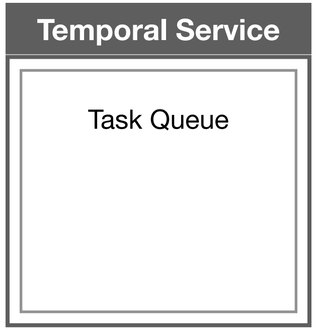
pseudocode

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

pseudocode

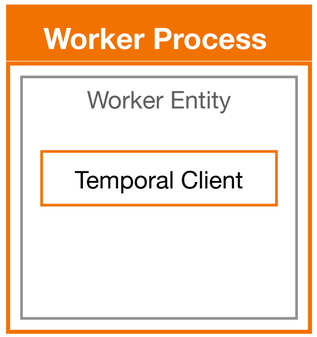
```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

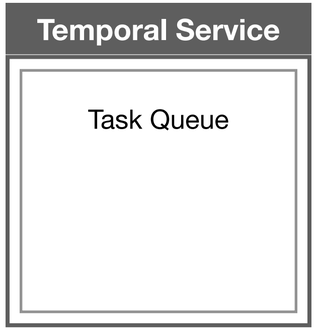




pseudocode

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

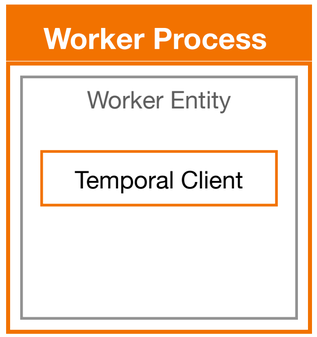


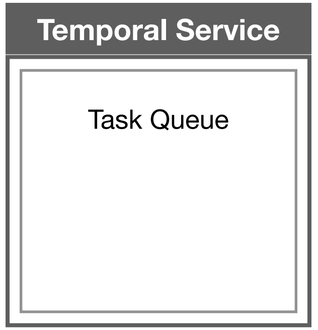


Commands

pseudocode

```
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

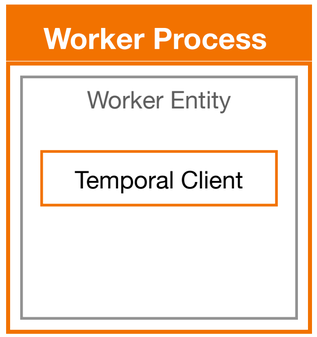


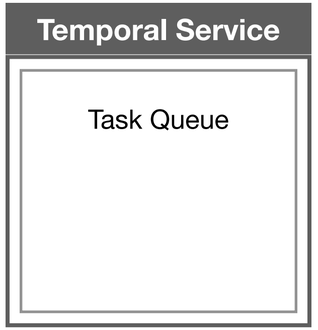


Commands

Events

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

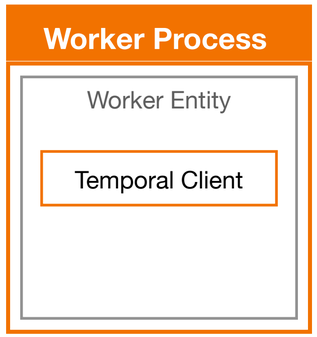


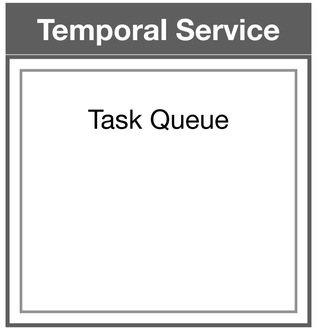


Commands

Events

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

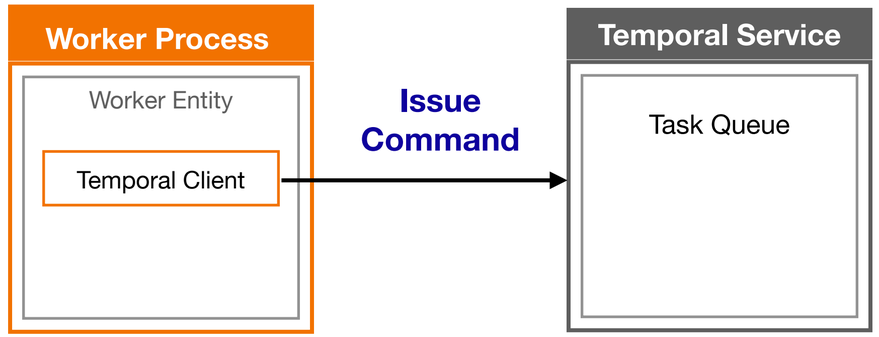




Commands

Events

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

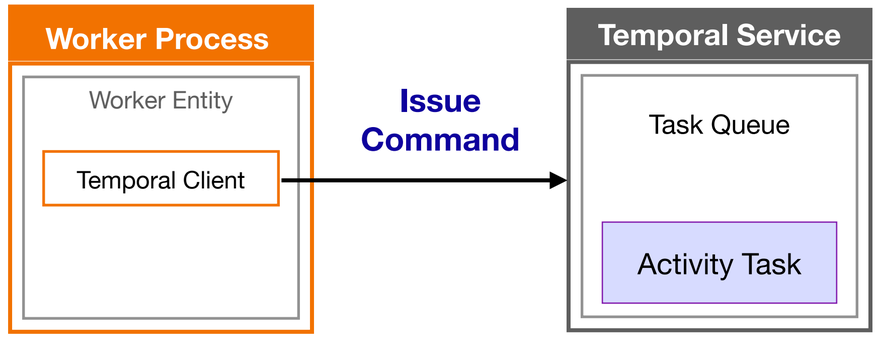


Commands

ScheduleActivityTask
(GetDistance)

Events

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

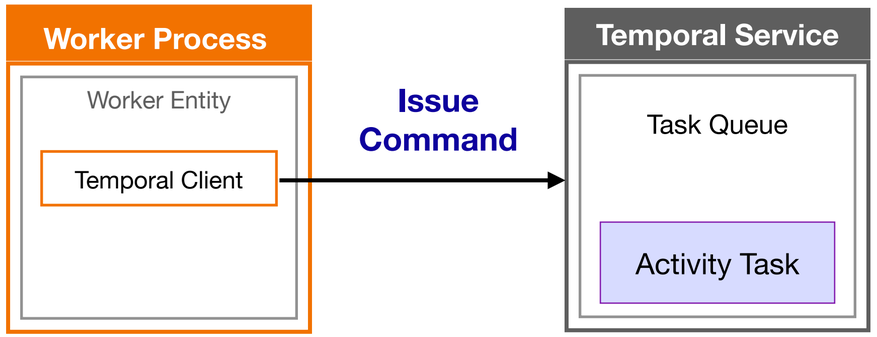


Commands

ScheduleActivityTask
(GetDistance)

Events

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



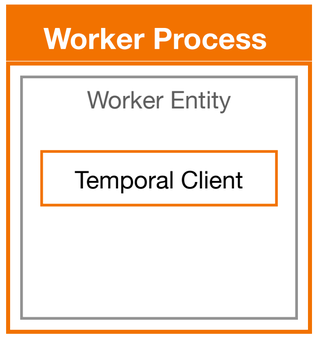
Commands

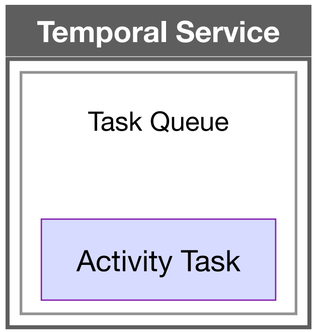
ScheduleActivityTask
(GetDistance)

Events

ActivityTaskScheduled

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





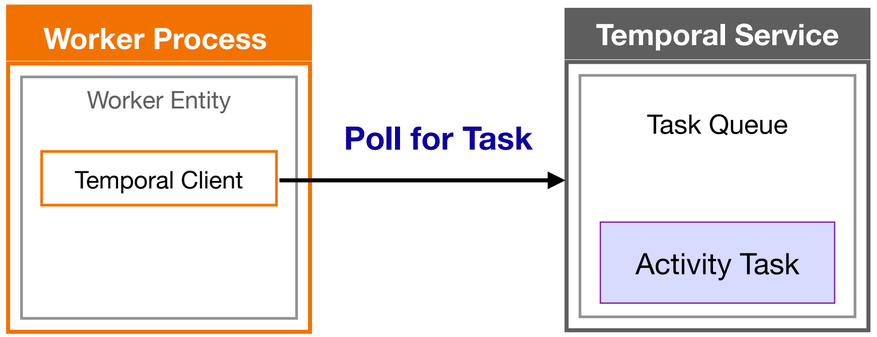
Commands

ScheduleActivityTask
(GetDistance)

Events

ActivityTaskScheduled

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



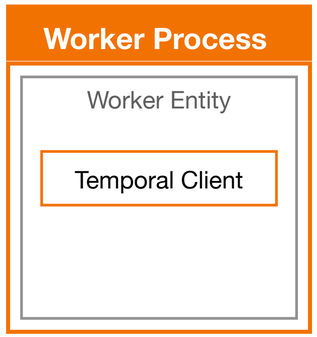
Commands

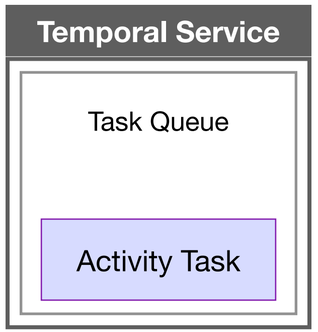
ScheduleActivityTask
(GetDistance)

Events

ActivityTaskScheduled

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





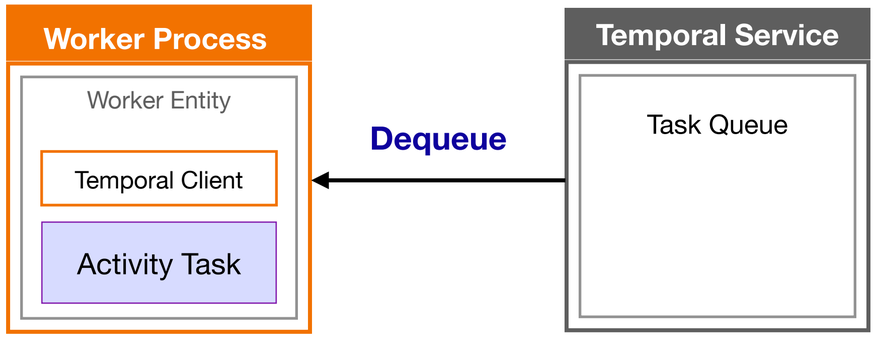
Commands

ScheduleActivityTask
(GetDistance)

Events

ActivityTaskScheduled

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



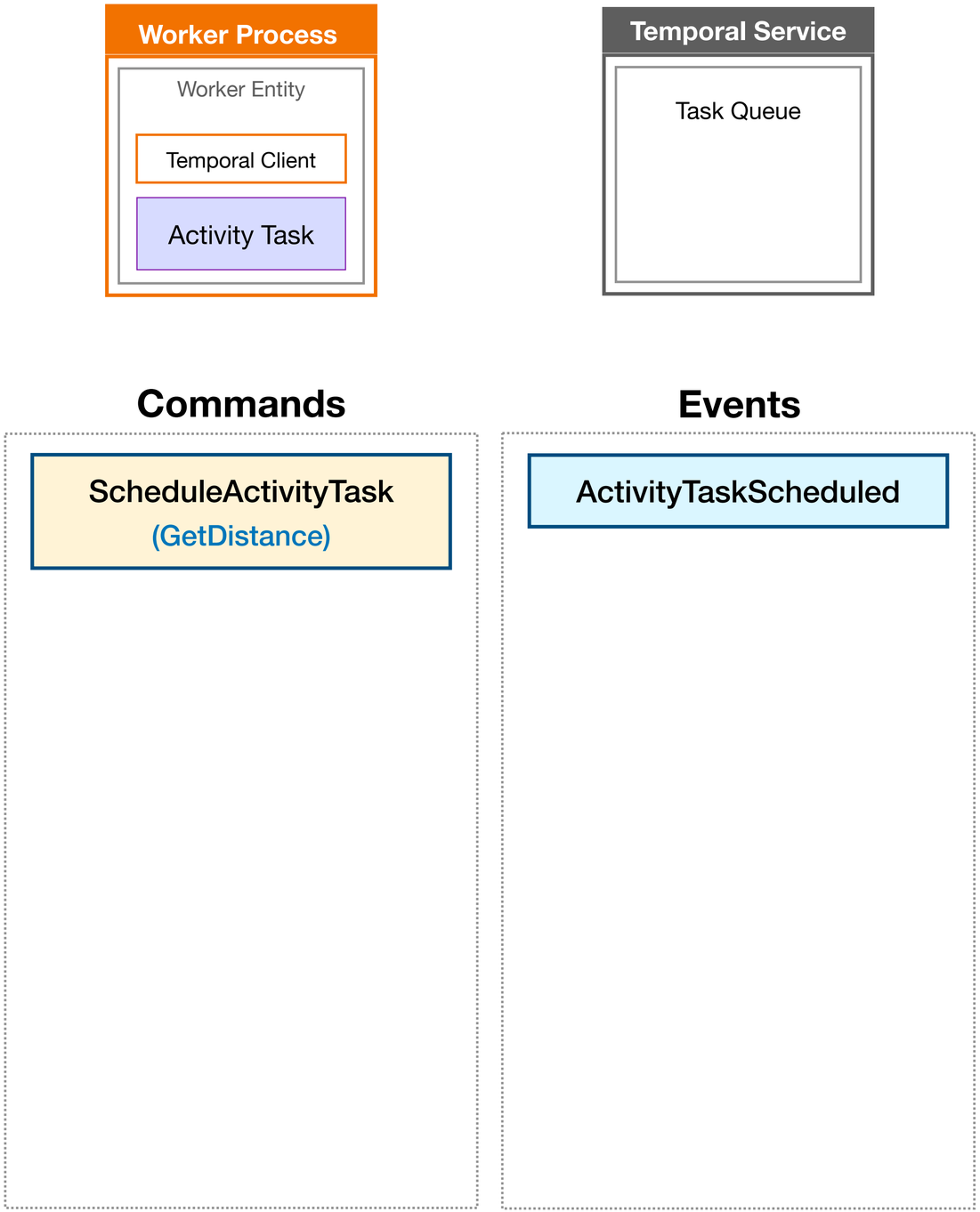
Commands

ScheduleActivityTask
(GetDistance)

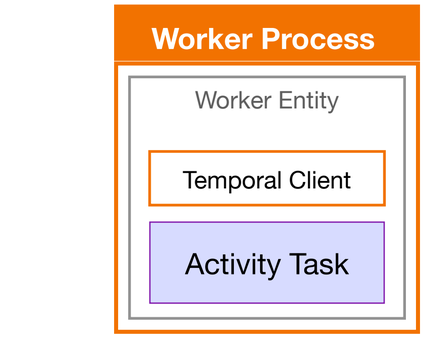
Events

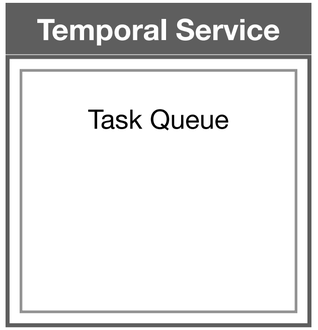
ActivityTaskScheduled

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

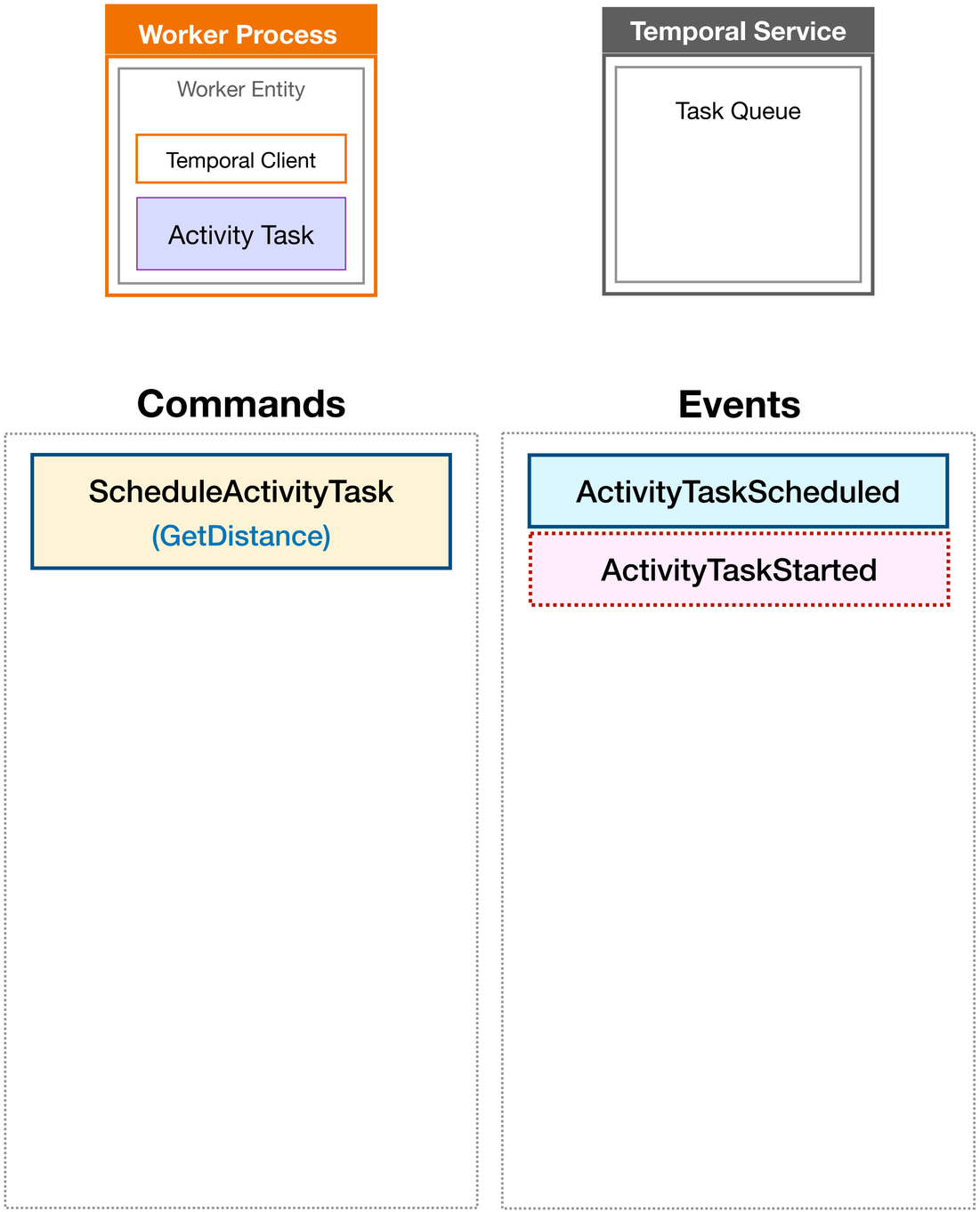
ScheduleActivityTask
(GetDistance)

Events

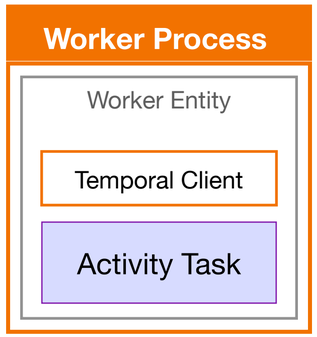
ActivityTaskScheduled

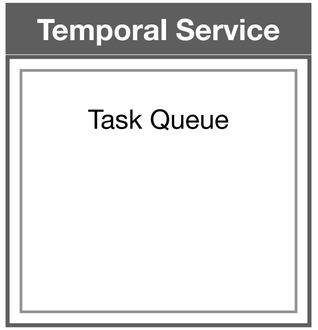
ActivityTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

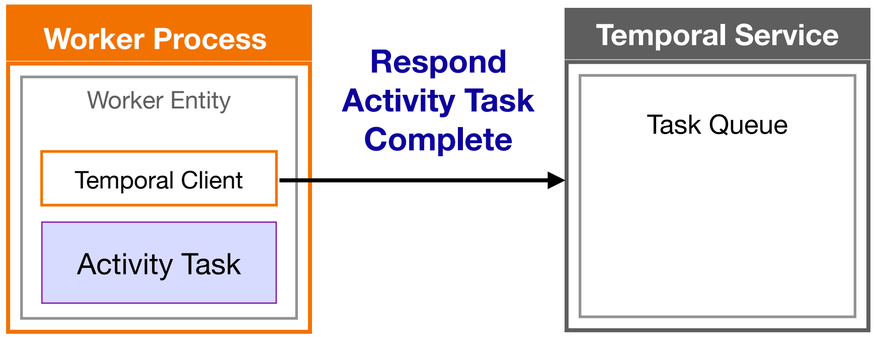
ScheduleActivityTask
(GetDistance)

Events

ActivityTaskScheduled

ActivityTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

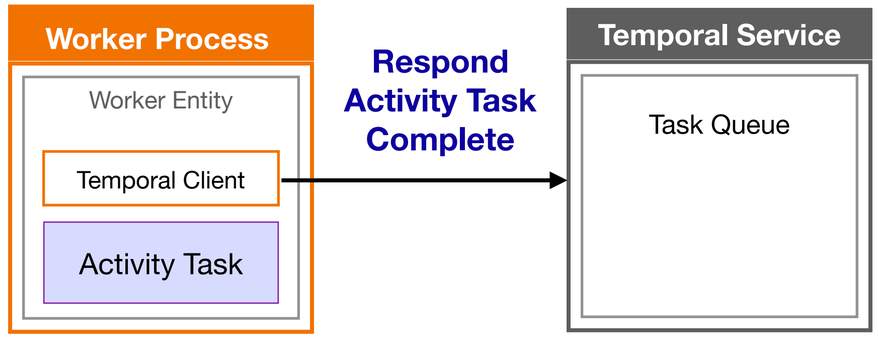
ScheduleActivityTask
(GetDistance)

Events

ActivityTaskScheduled

ActivityTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask
(GetDistance)

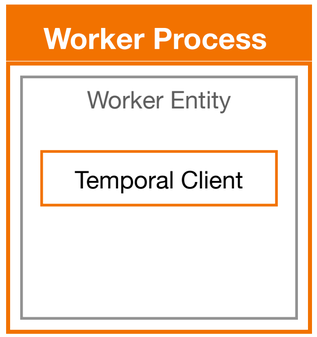
Events

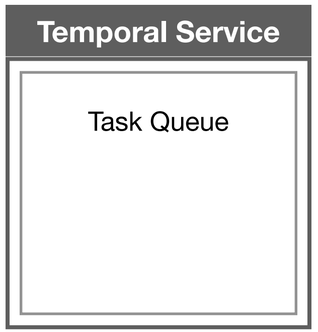
ActivityTaskScheduled

ActivityTaskStarted

ActivityTaskCompleted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask
(GetDistance)

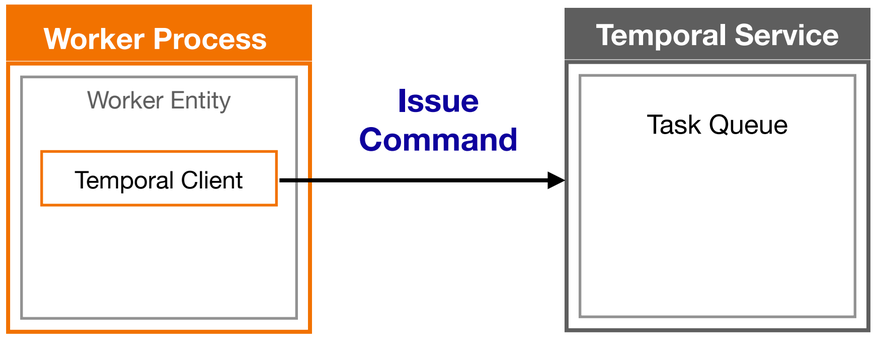
Events

ActivityTaskScheduled

ActivityTaskStarted

ActivityTaskCompleted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask
(GetDistance)

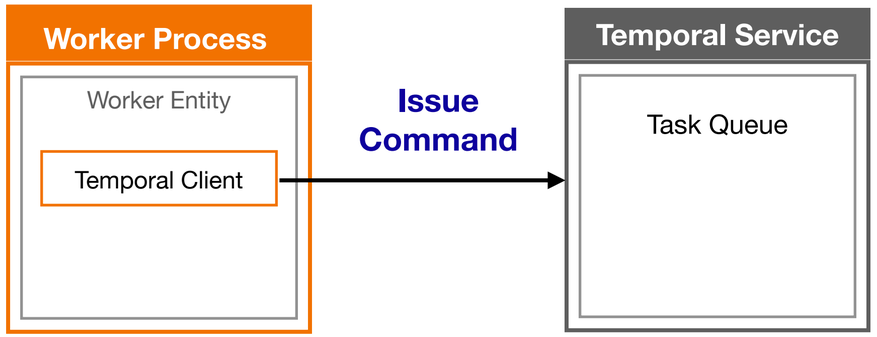
Events

ActivityTaskScheduled

ActivityTaskStarted

ActivityTaskCompleted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask
(GetDistance)

StartTimer
(30 Minutes)

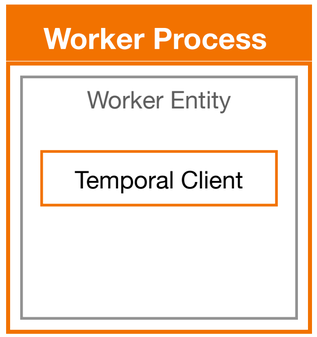
Events

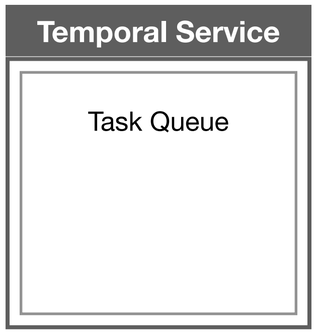
ActivityTaskScheduled

ActivityTaskStarted

ActivityTaskCompleted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask
(GetDistance)

StartTimer
(30 Minutes)

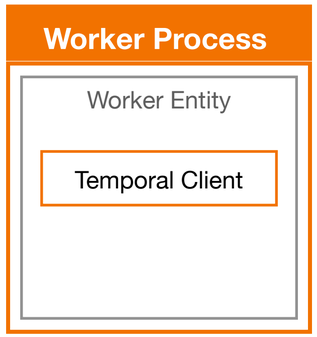
Events

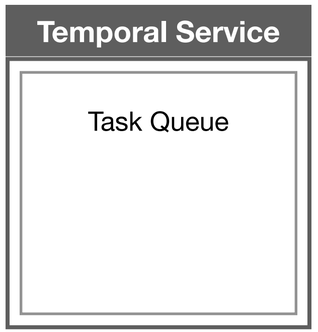
ActivityTaskScheduled

ActivityTaskStarted

ActivityTaskCompleted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask
(GetDistance)

StartTimer
(30 Minutes)

Events

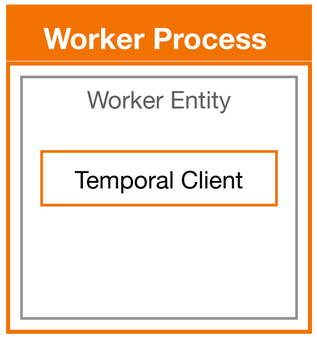
ActivityTaskScheduled

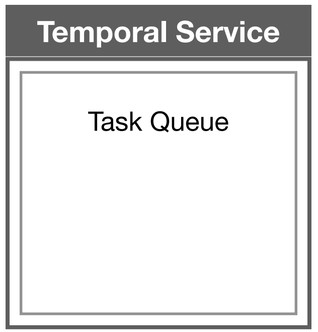
ActivityTaskStarted

ActivityTaskCompleted

TimerStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask
(GetDistance)

StartTimer
(30 Minutes)

Events

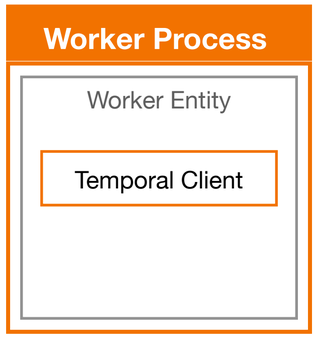
ActivityTaskScheduled

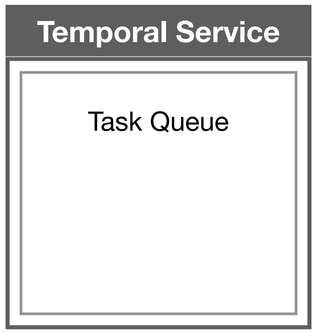
ActivityTaskStarted

ActivityTaskCompleted

TimerStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask
(GetDistance)

StartTimer
(30 Minutes)

Events

ActivityTaskScheduled

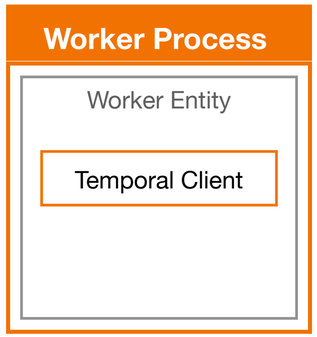
ActivityTaskStarted

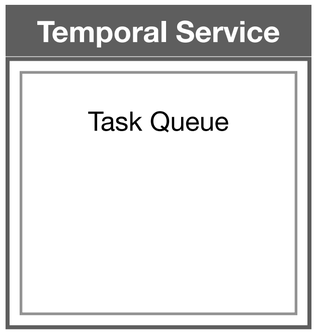
ActivityTaskCompleted

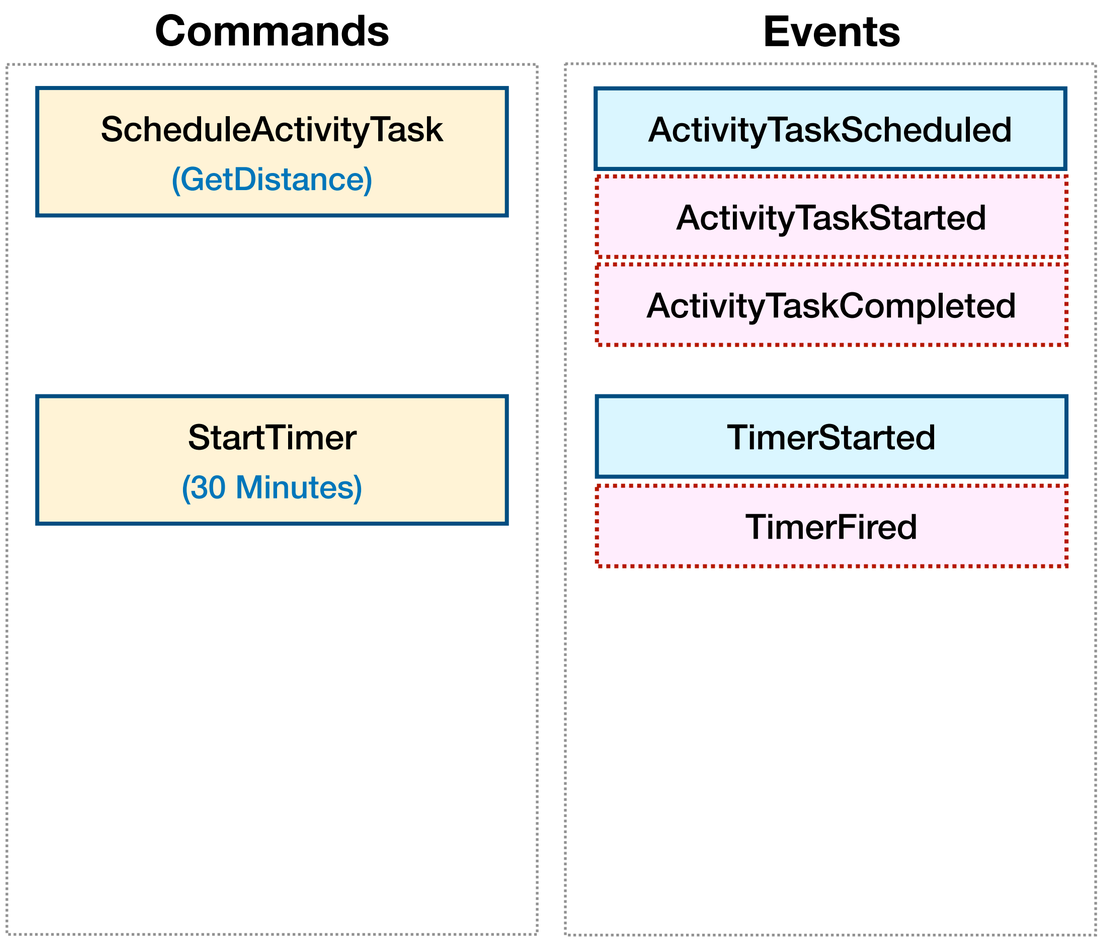
TimerStarted

TimerFired

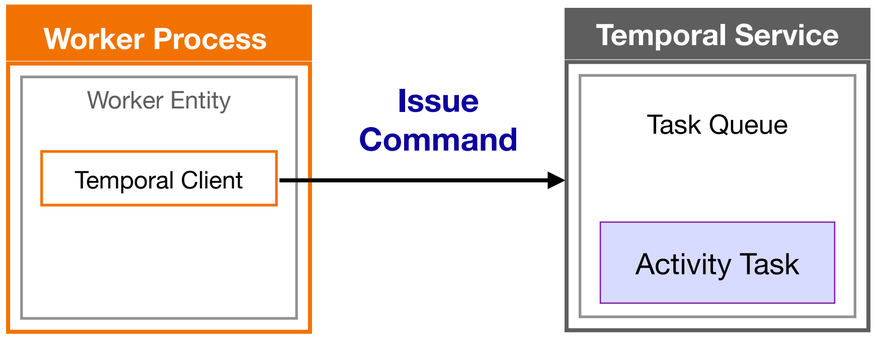
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

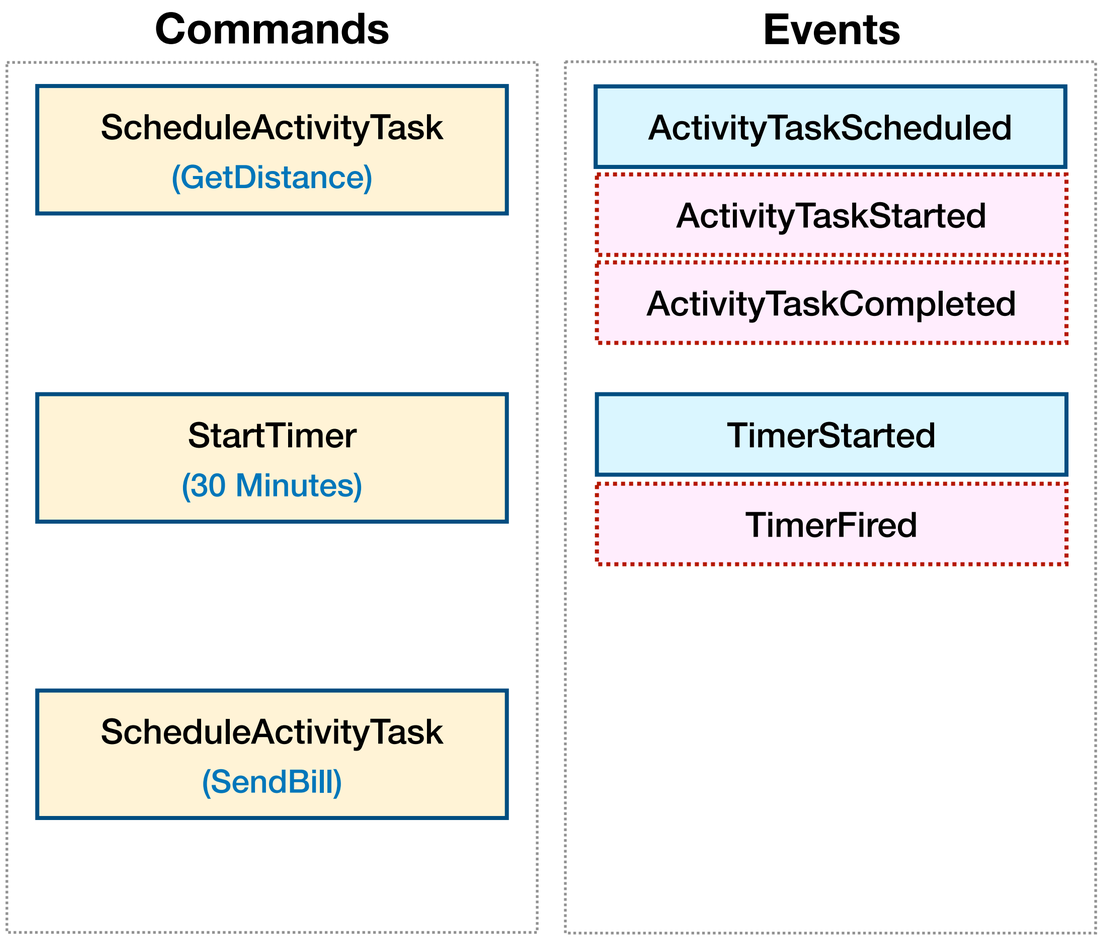




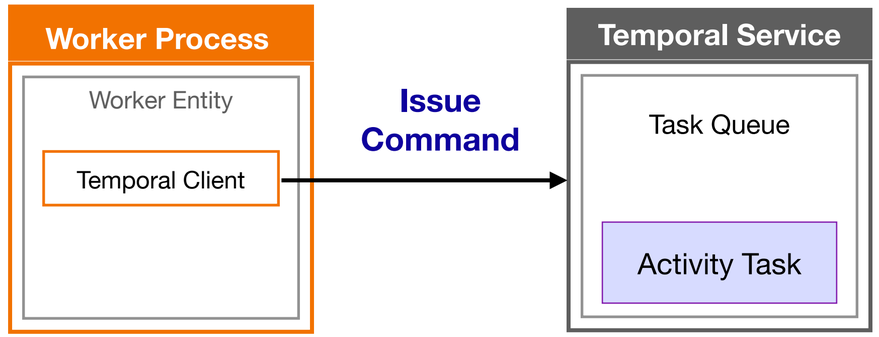


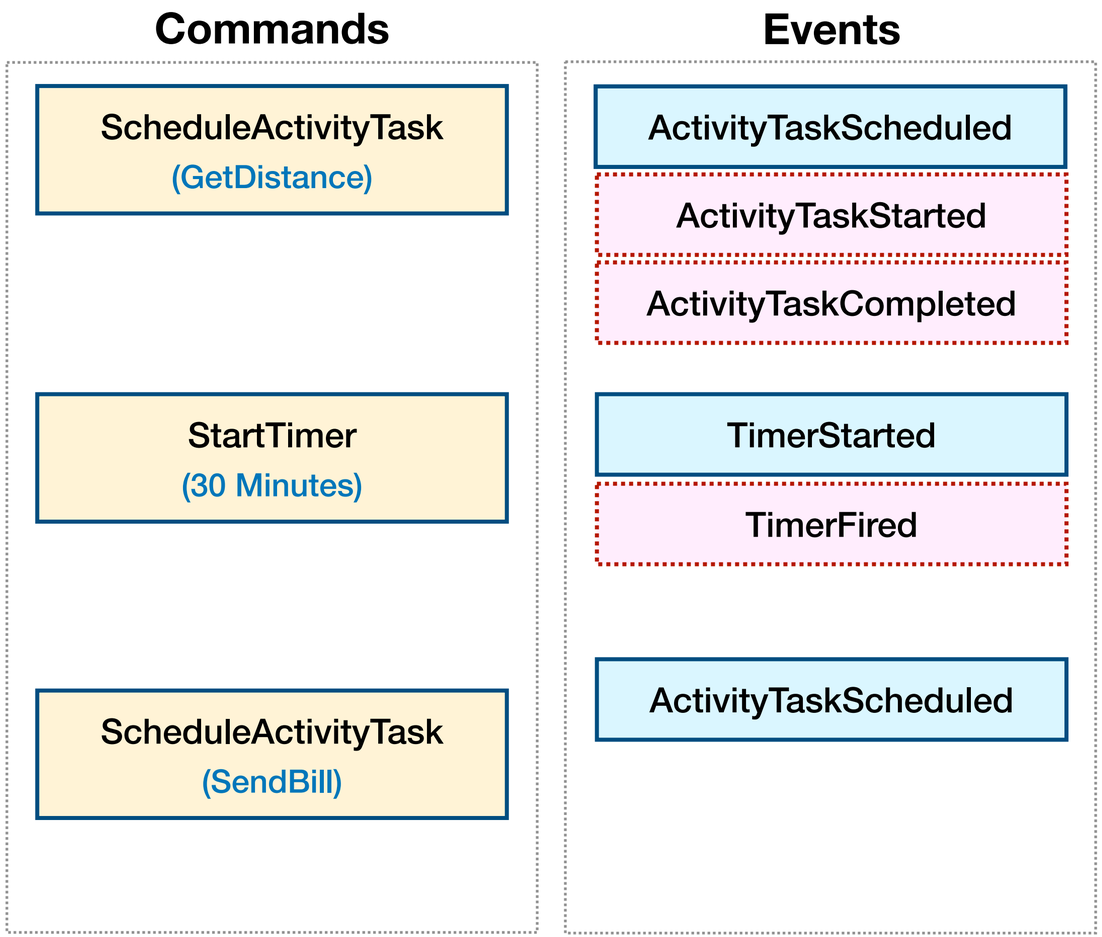
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



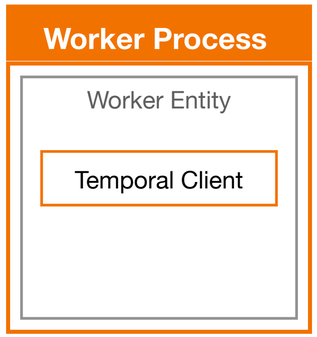


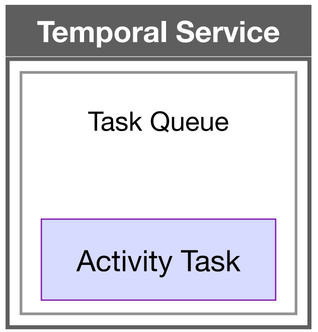
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

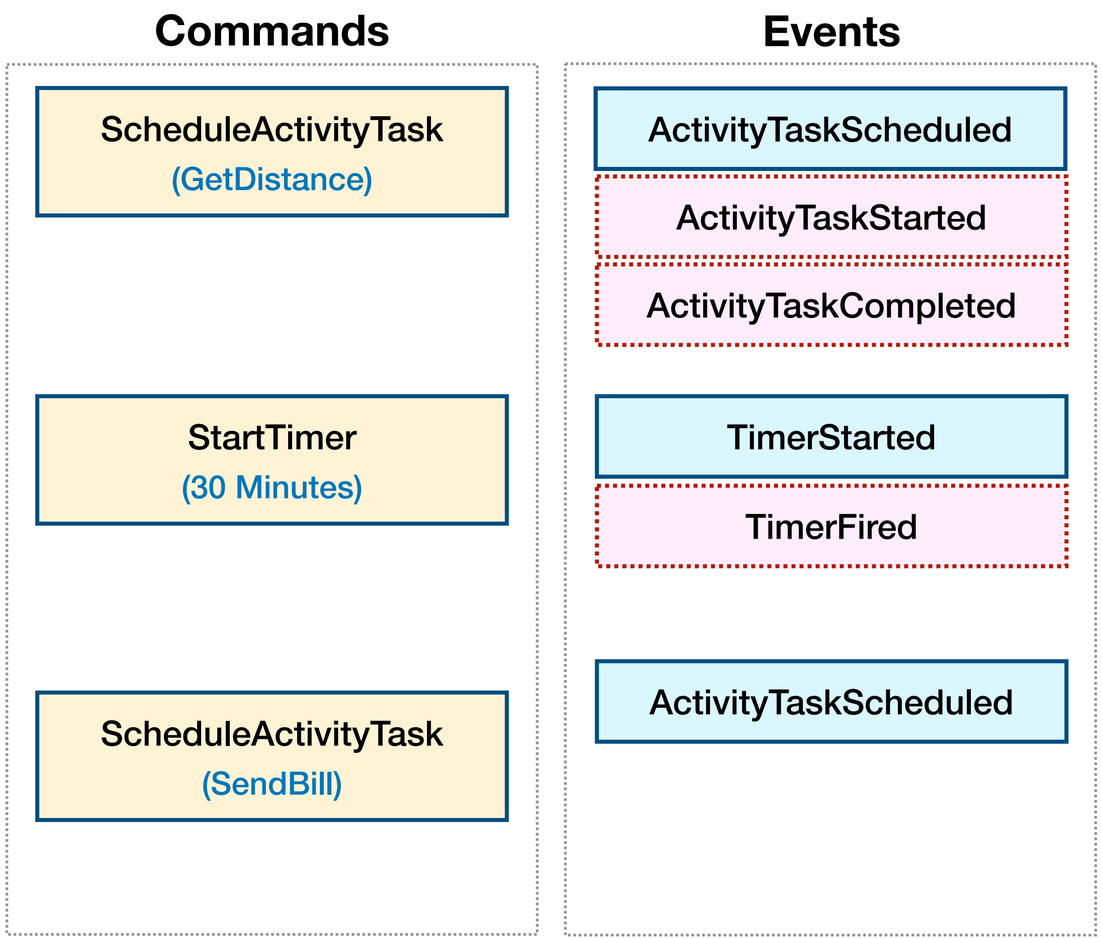




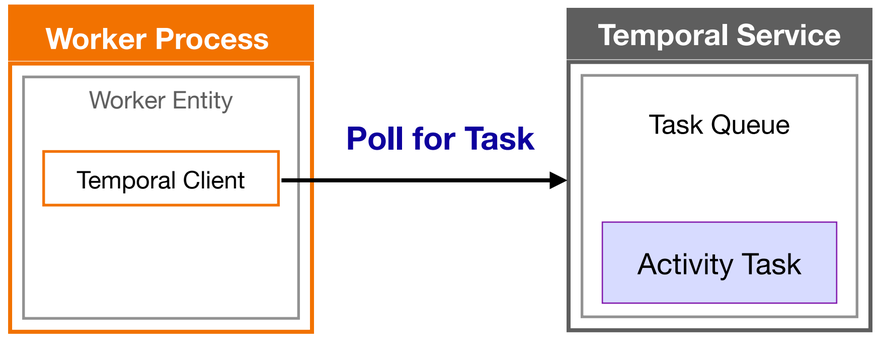
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

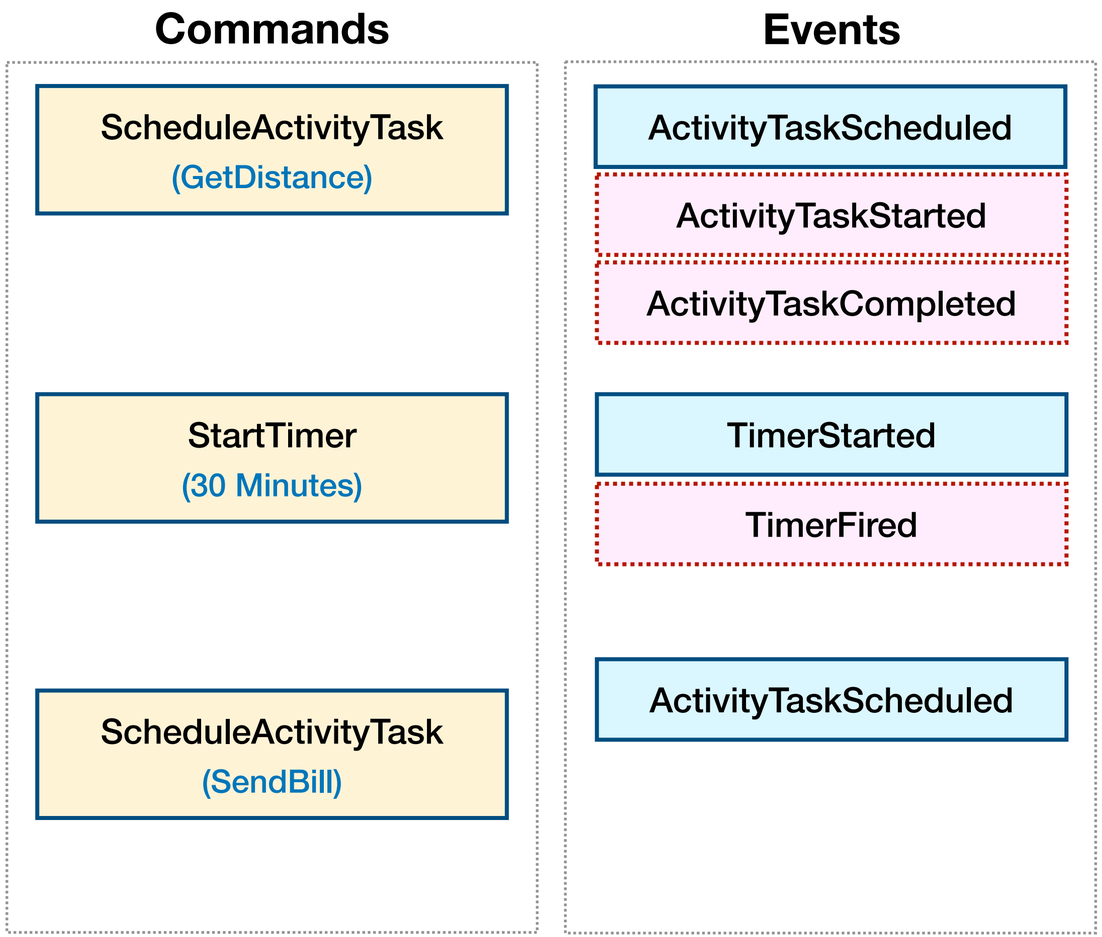




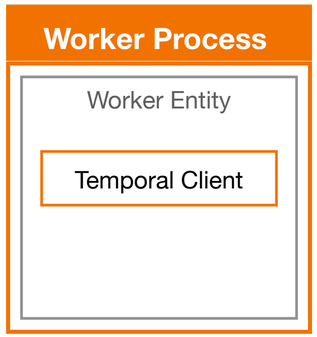


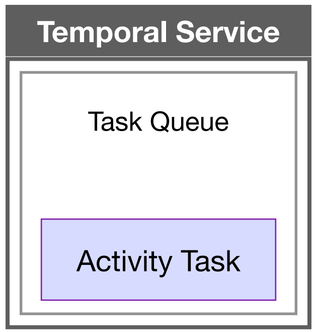
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

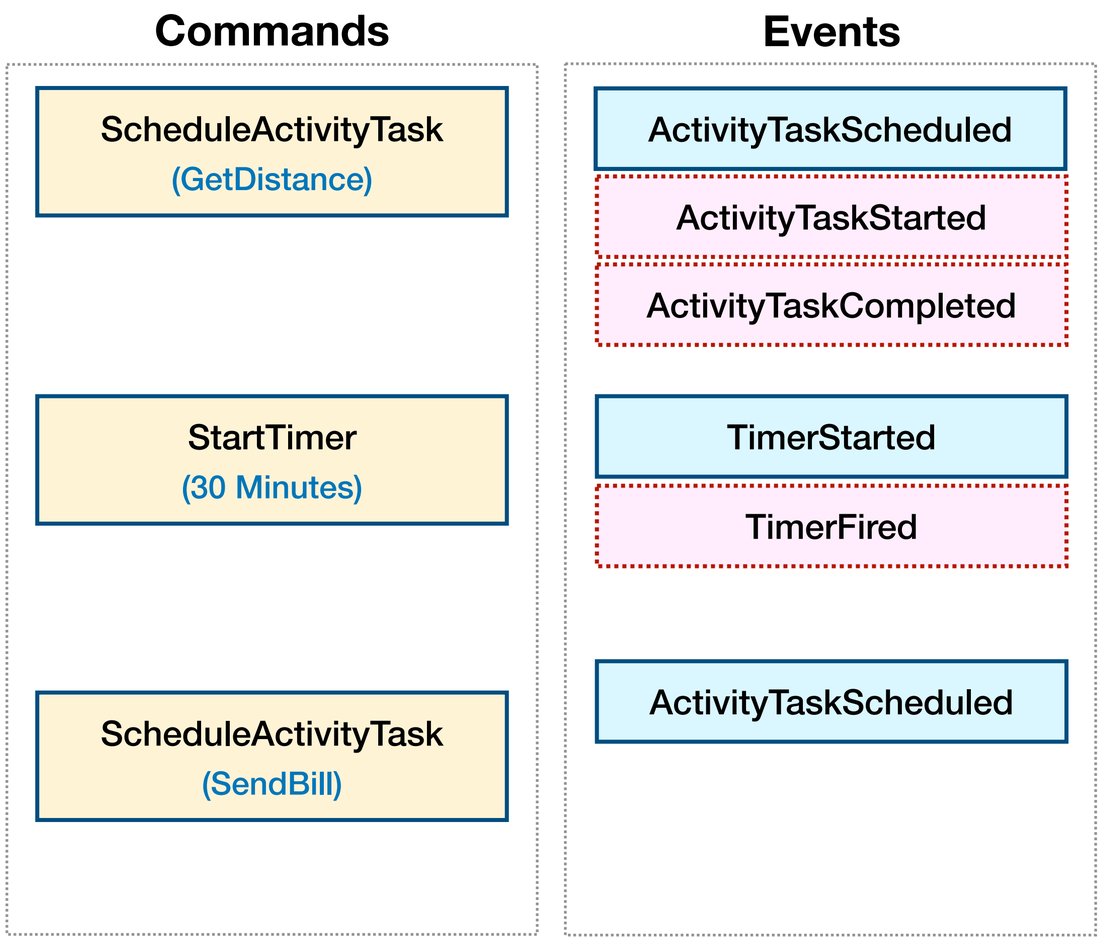




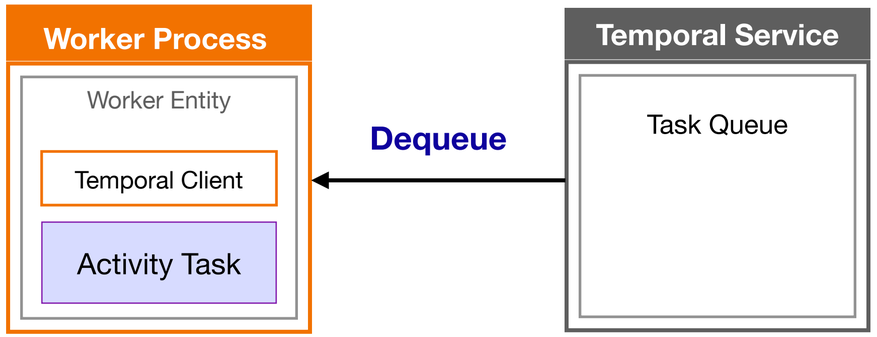
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

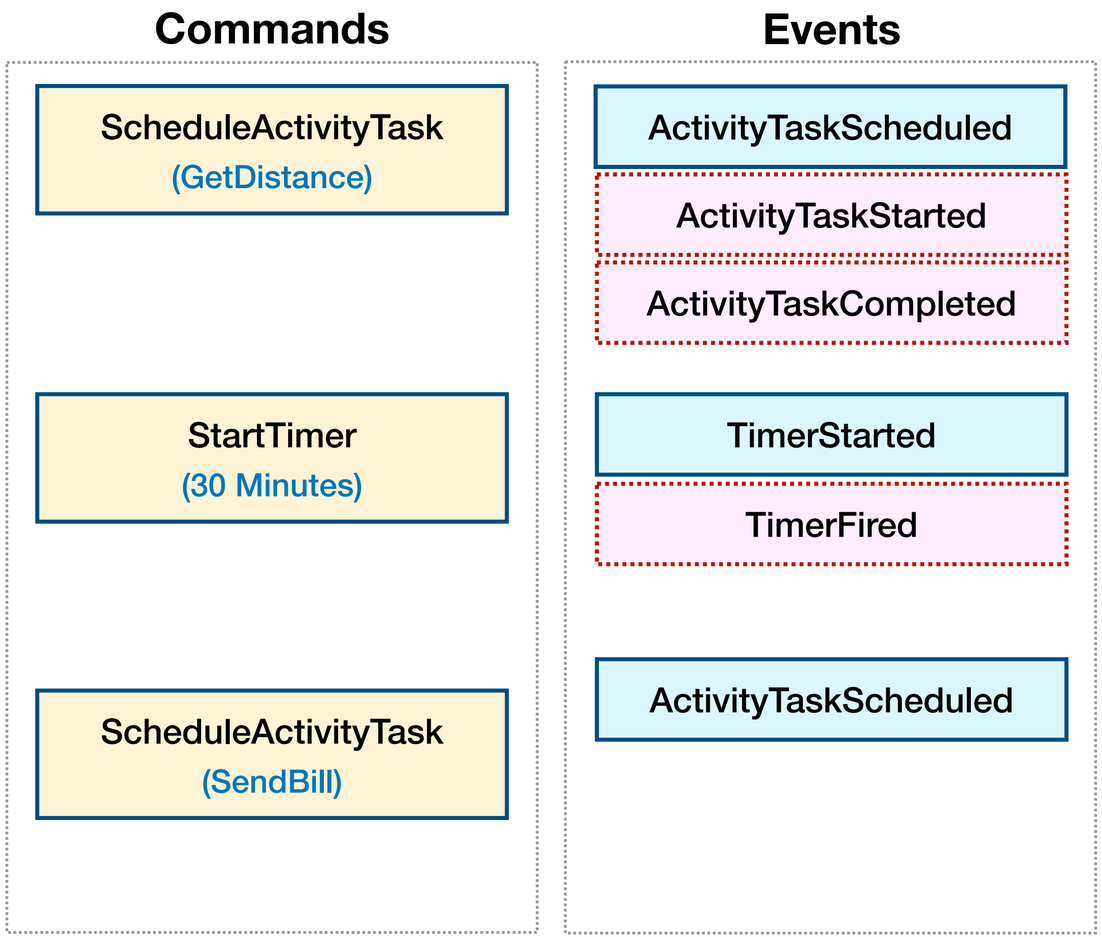




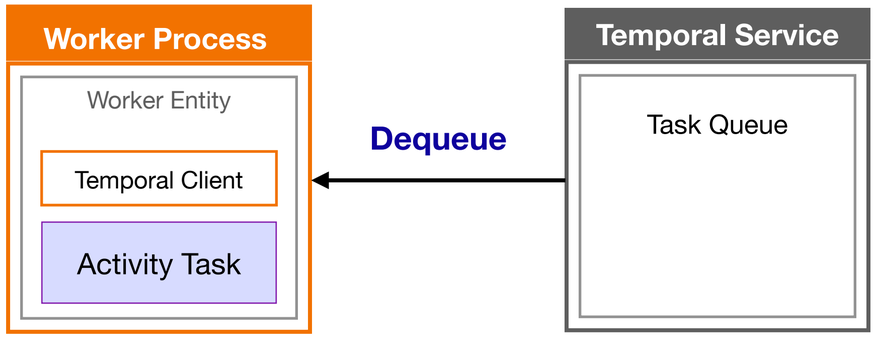


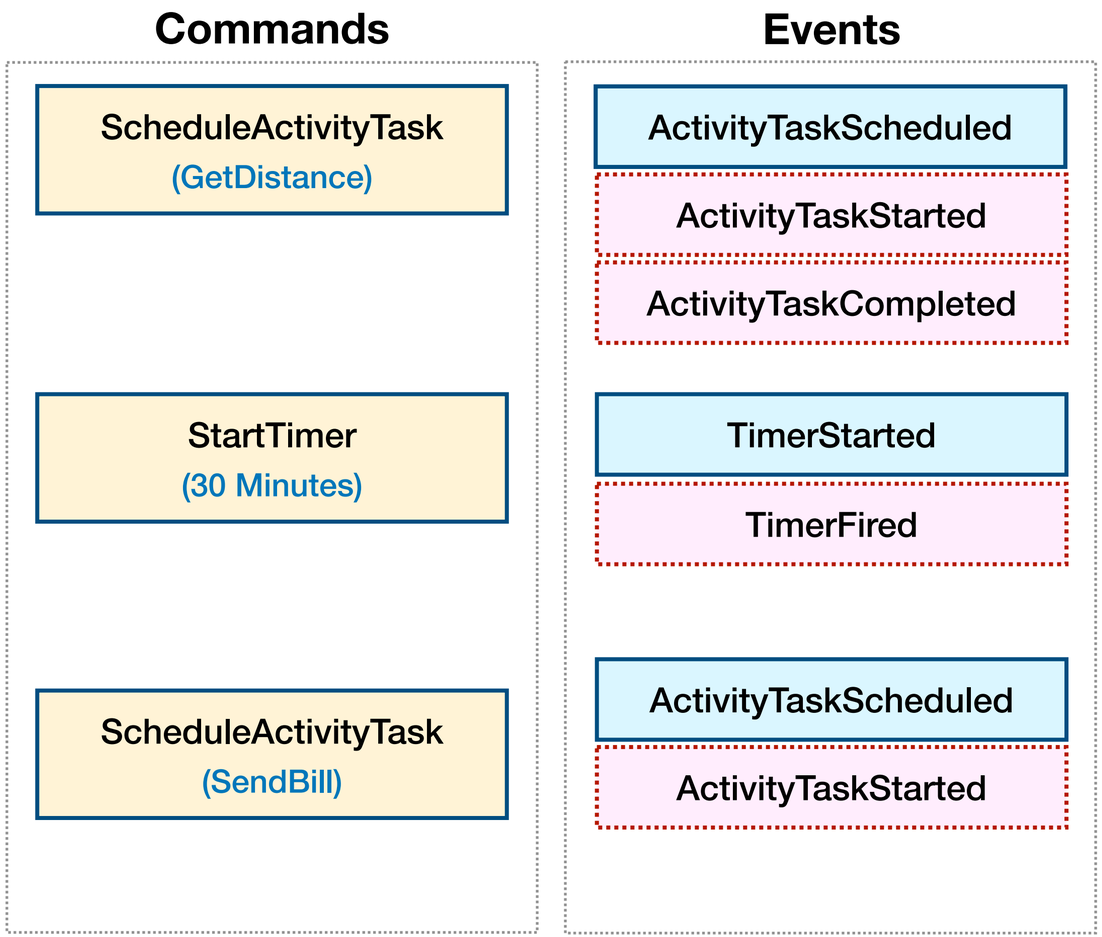
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



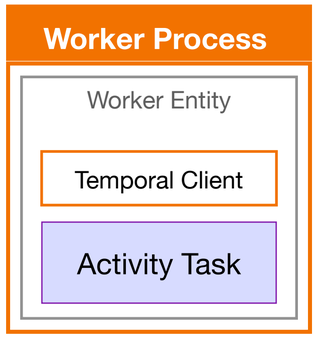


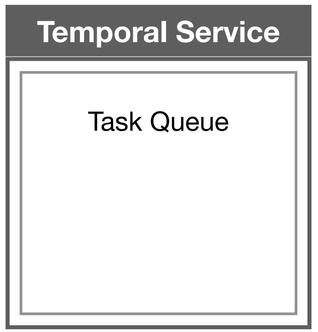
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

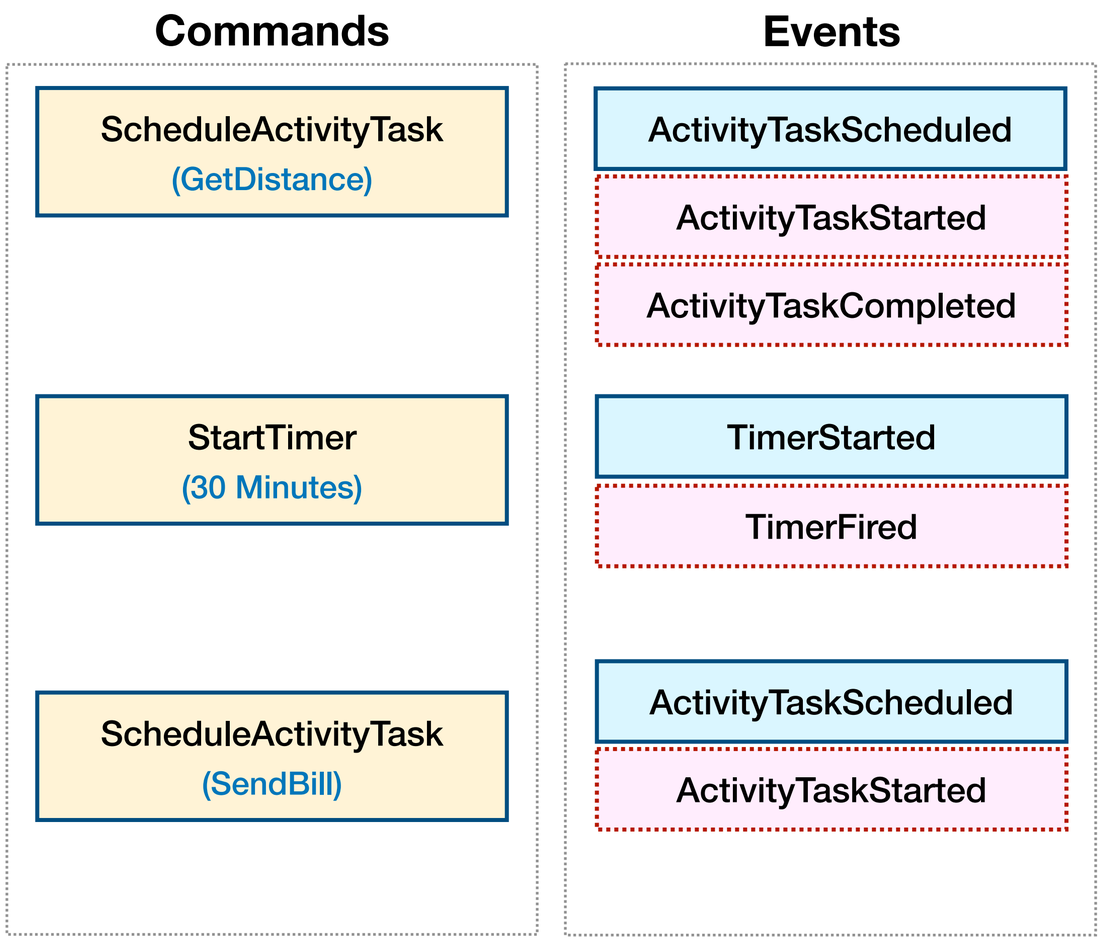




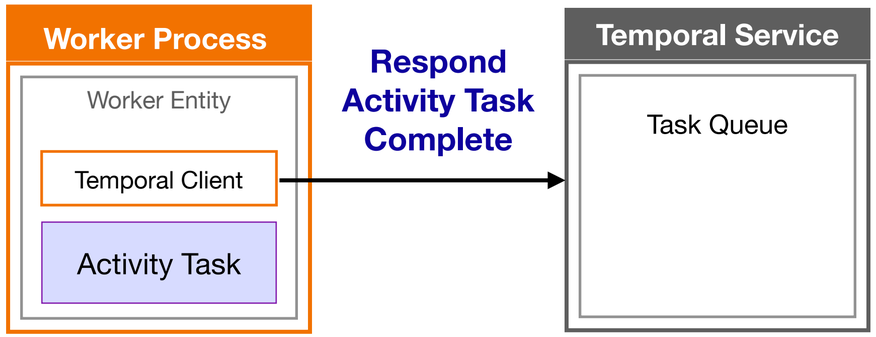
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

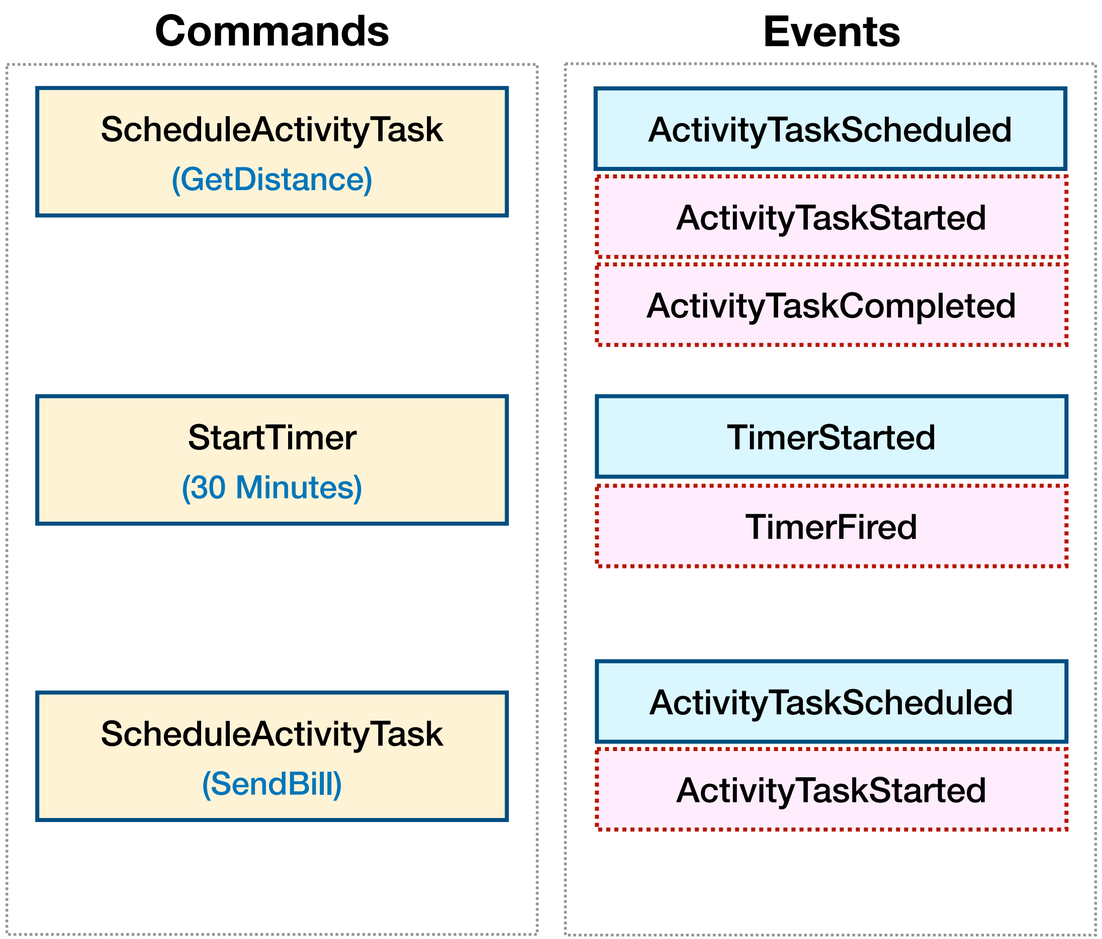




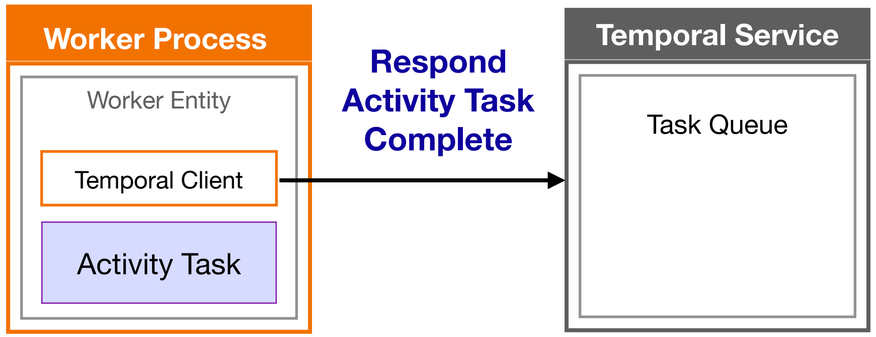


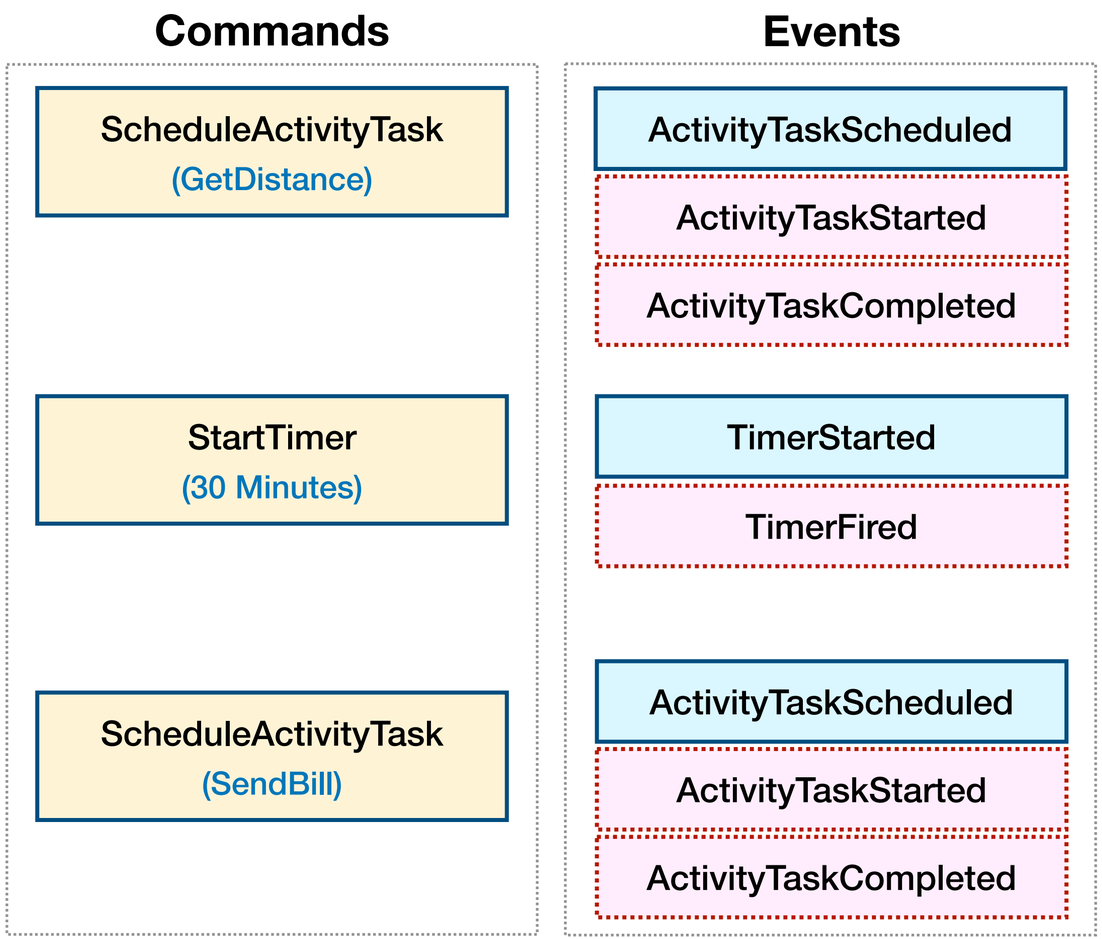
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



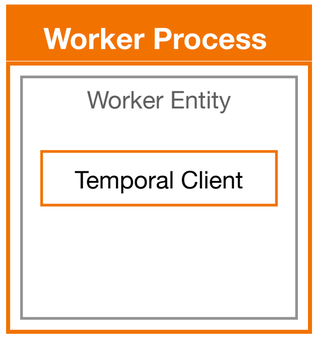


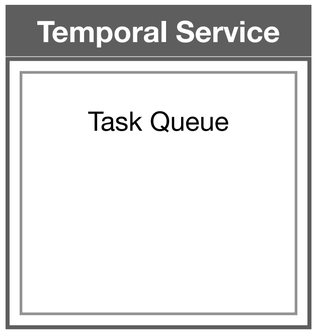
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

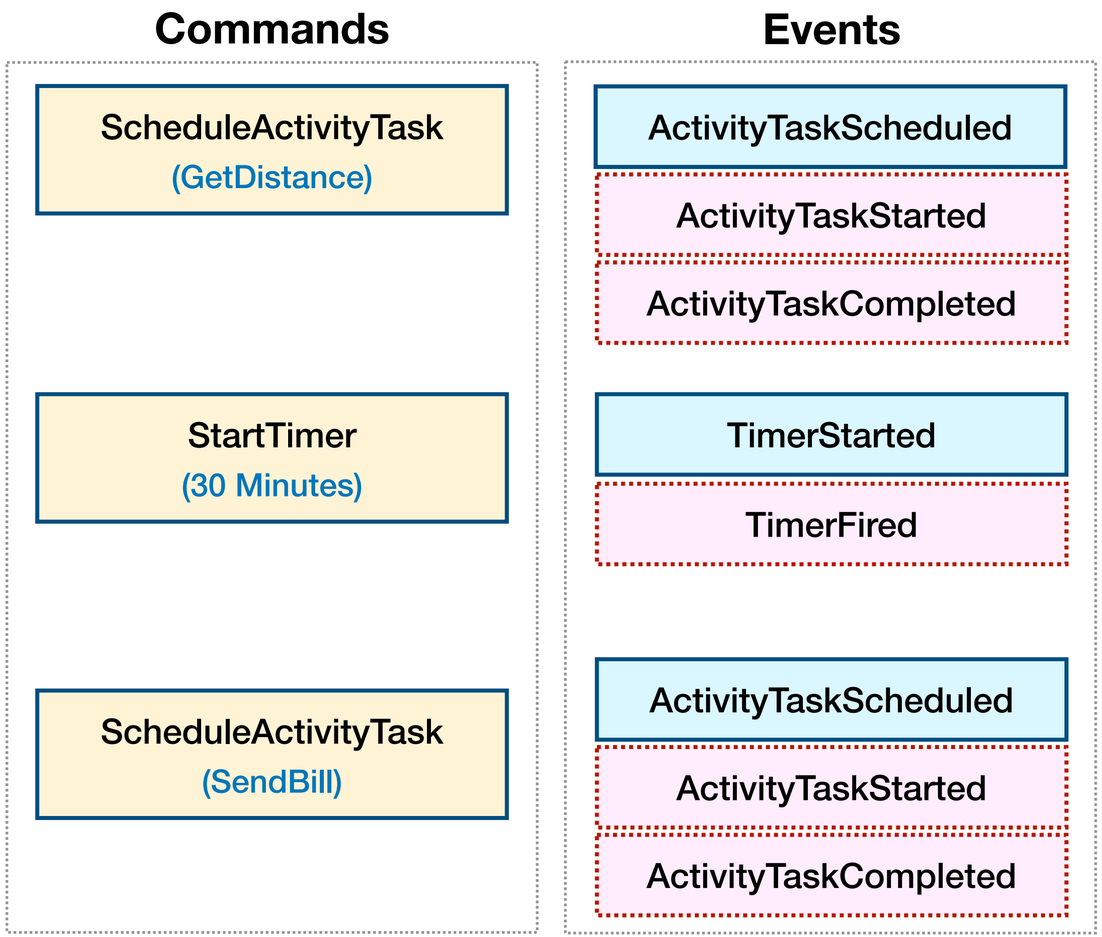




```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  @Override
  public String pizzaWorkflow(Order order) {

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Iterate over the items and calculate the cost of the order
    int totalPrice = 0;
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

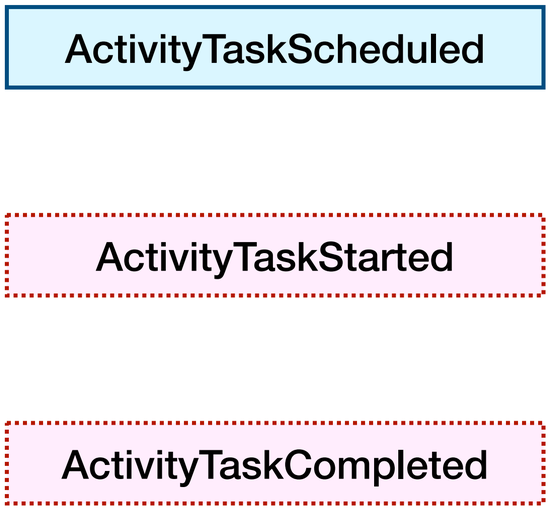




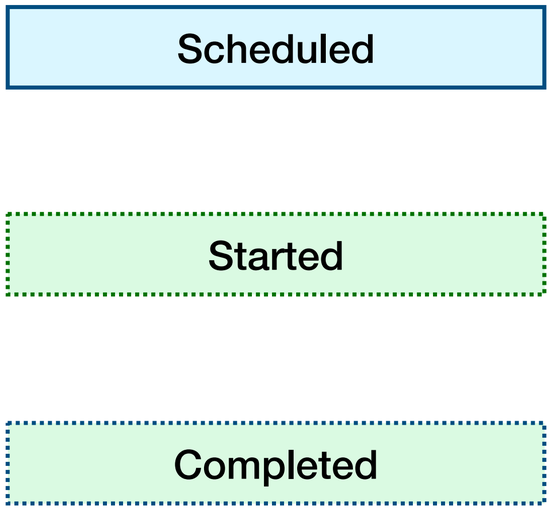


Workflow and Activity Task States

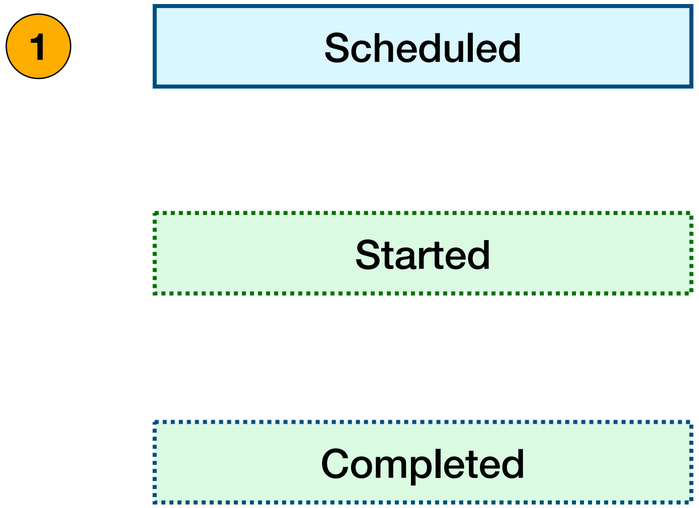
Activity Task Event Sequence



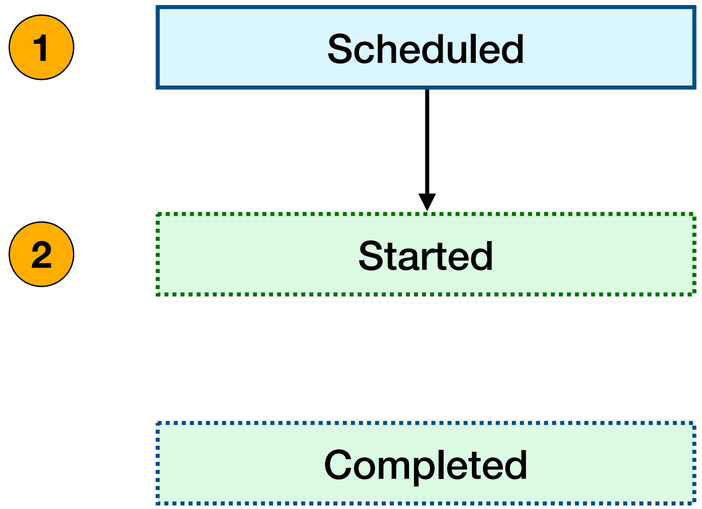
Activity States in that Sequence



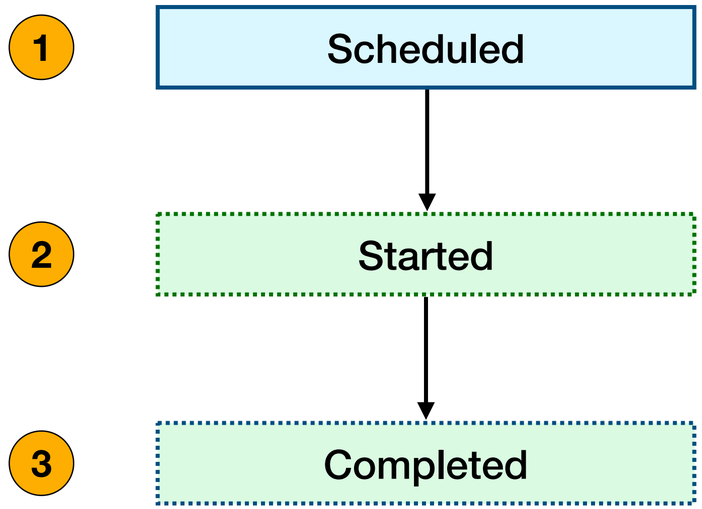
Activity States in that Sequence



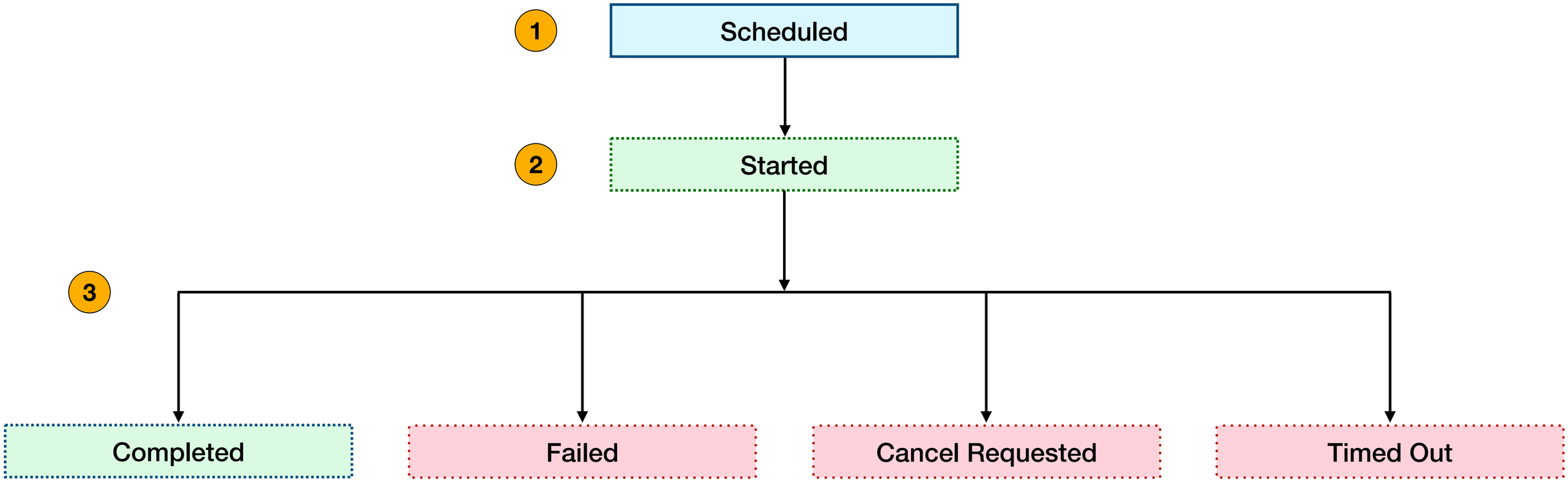
Activity States in that Sequence



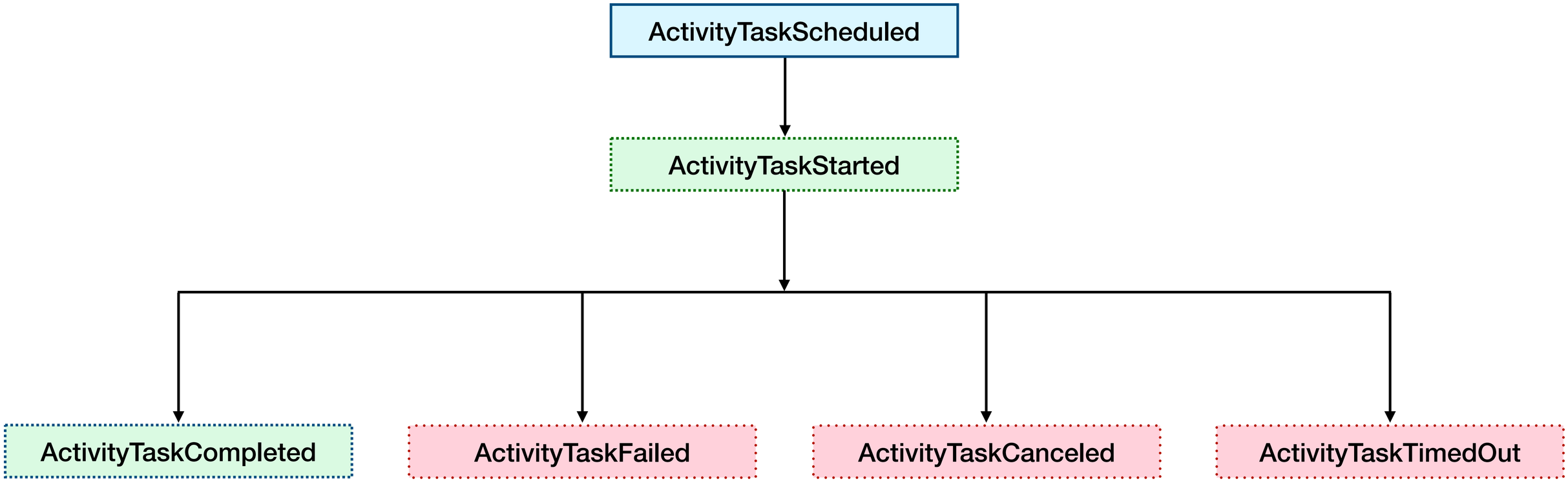
Activity States in that Sequence



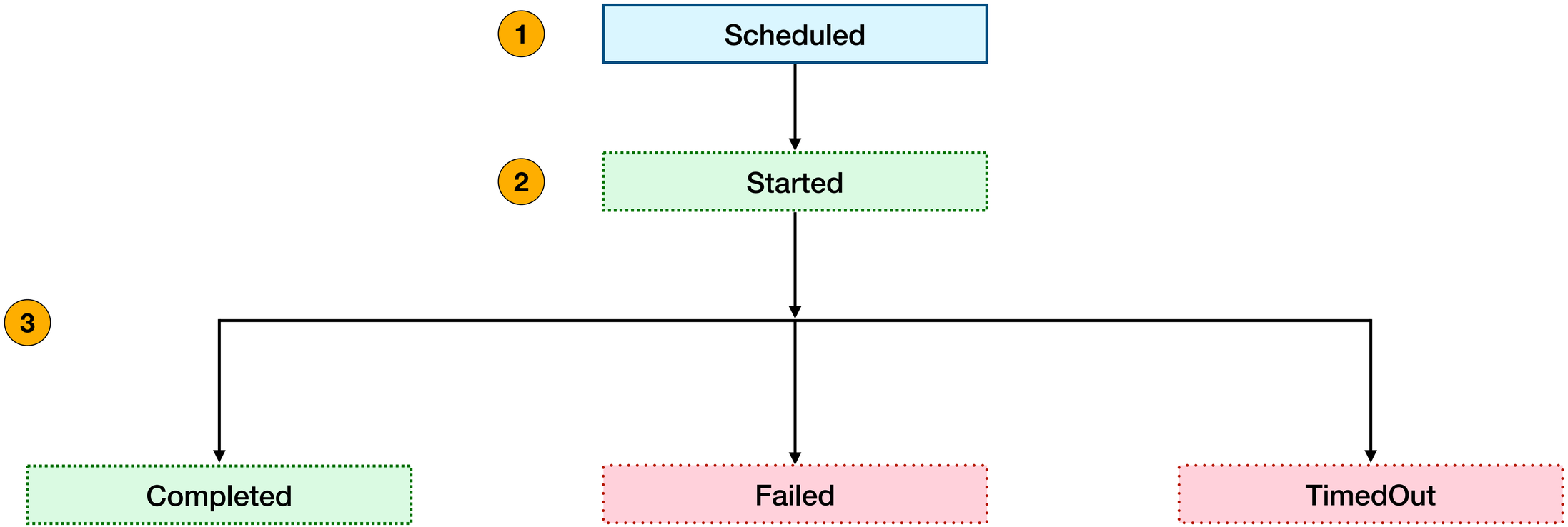
Activity Task States



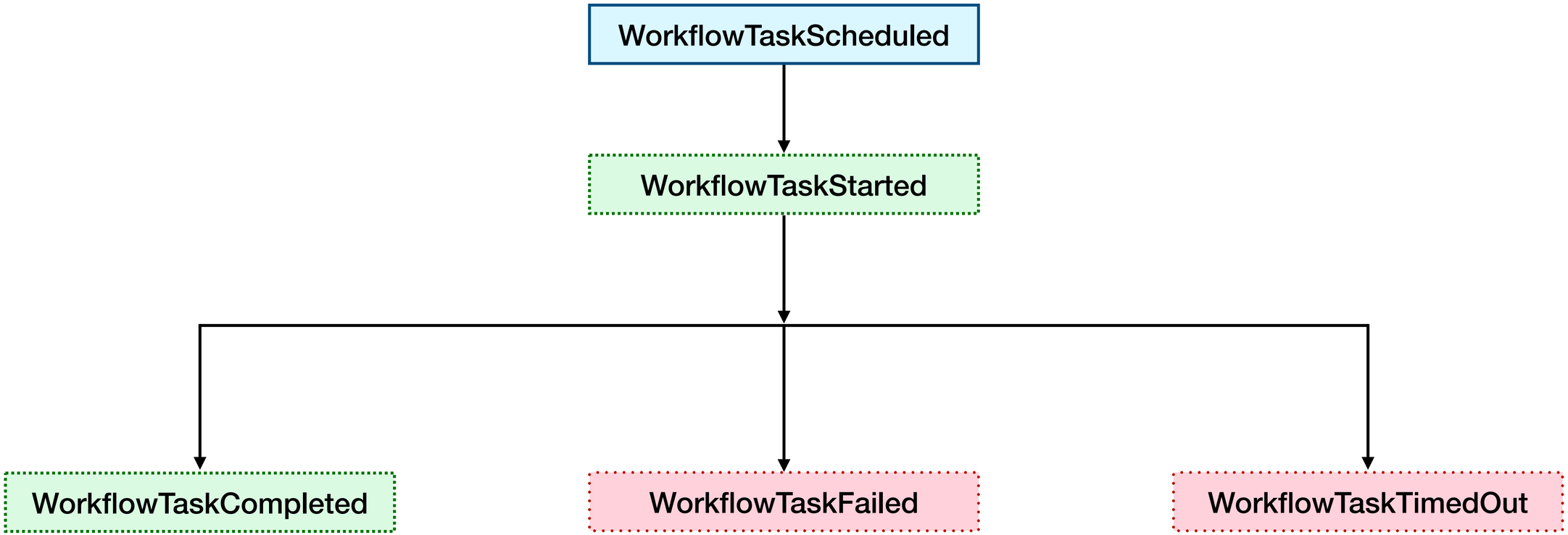
Activity Task Events



Workflow Task States



Workflow Task Events



Sticky Execution

Sticky Execution

- To improve effectiveness of Worker's caching, Temporal use "sticky" execution for Workflow Tasks

Sticky Execution

- **To improve effectiveness of Worker's caching, Temporal use "sticky" execution for Workflow Tasks**
  - A Worker which completed the first Workflow Task is given preference for subsequent Workflow Tasks in the same execution via a Worker-specific Task Queue

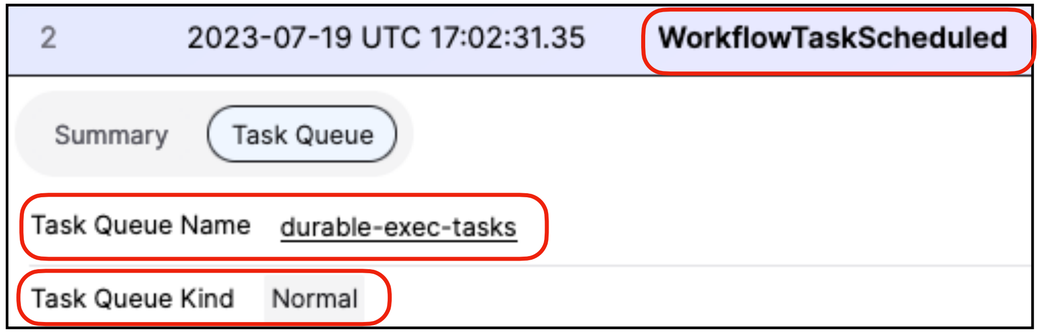
Sticky Execution

- **To improve effectiveness of Worker's caching, Temporal use "sticky" execution for Workflow Tasks**
  - A Worker which completed the first Workflow Task is given preference for subsequent Workflow Tasks in the same execution via a Worker-specific Task Queue
- **Sticky execution is visible in the Web UI**
  - See the Task Queue Name / Kind fields

Sticky Execution

- **To improve effectiveness of Worker's caching, Temporal use "sticky" execution for Workflow Tasks**
  - A Worker which completed the first Workflow Task is given preference for subsequent Workflow Tasks in the same execution via a Worker-specific Task Queue
- **Sticky execution is visible in the Web UI**
  - See the Task Queue Name / Kind fields

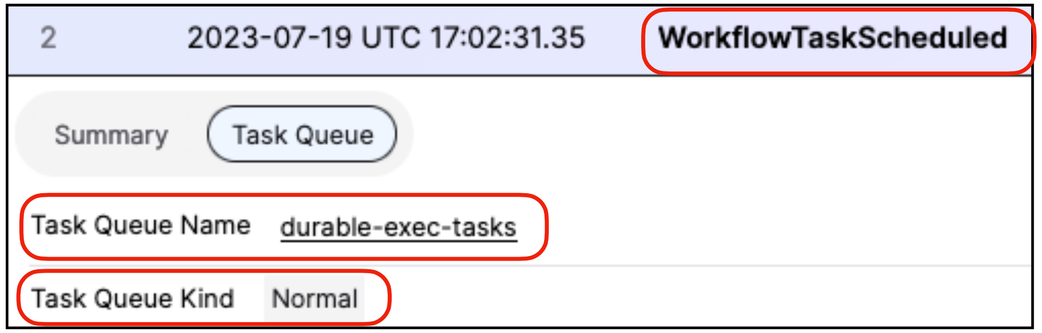
First Workflow Task



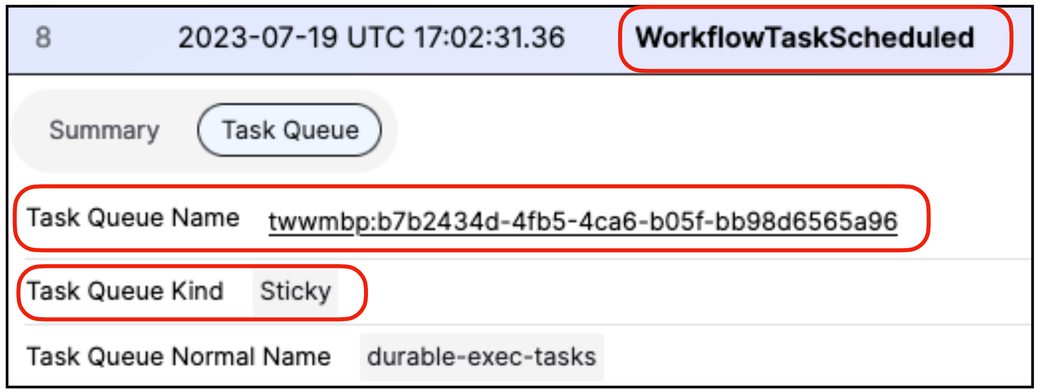
Sticky Execution

- **To improve effectiveness of Worker's caching, Temporal use "sticky" execution for Workflow Tasks**
  - A Worker which completed the first Workflow Task is given preference for subsequent Workflow Tasks in the same execution via a Worker-specific Task Queue
- **Sticky execution is visible in the Web UI**
  - See the Task Queue Name / Kind fields

First Workflow Task



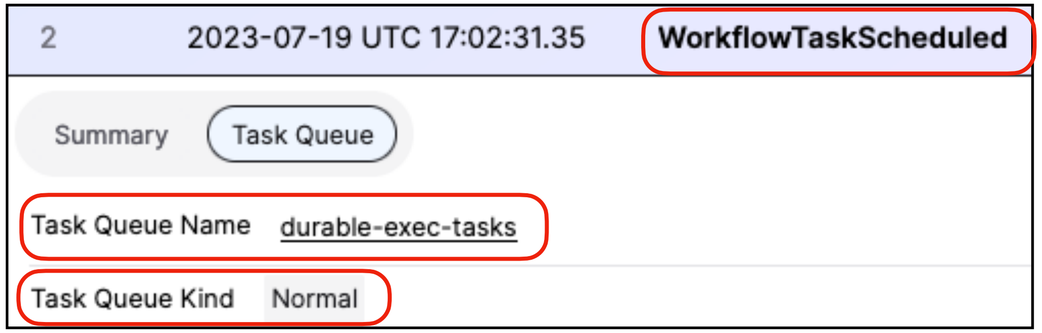
Later Workflow Task



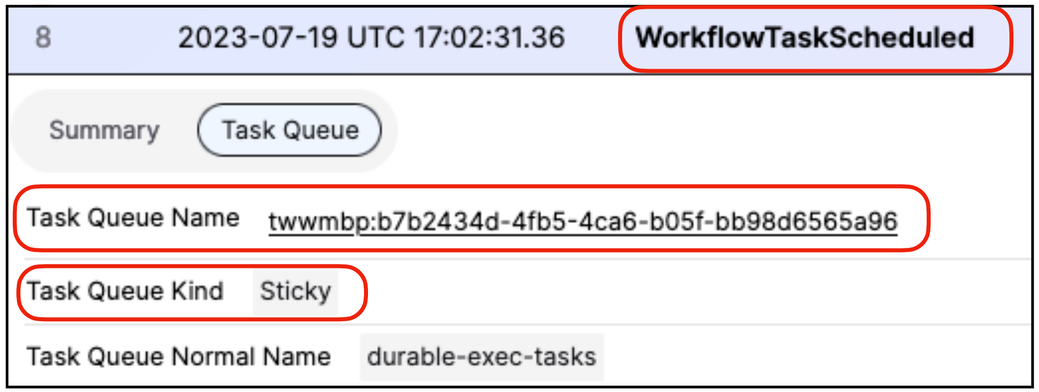
Sticky Execution

- **To improve effectiveness of Worker's caching, Temporal use "sticky" execution for Workflow Tasks**
  - A Worker which completed the first Workflow Task is given preference for subsequent Workflow Tasks in the same execution via a Worker-specific Task Queue
- **Sticky execution is visible in the Web UI**
  - See the Task Queue Name / Kind fields
- **This does not apply to Activity Tasks**

First Workflow Task



Later Workflow Task



Beginner to Builder Bootcamp - Java

00. About this Workshop
01. The Basics of Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
04. Understanding Event History
05. Understanding Workflow Determinism
06. Signals, Queries, and Workflow Updates
07. Timeouts and Retry Policies
08. Recovering from Failure
09. Conclusion

History Replay:

How Temporal Provides Durable Execution

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Start Workflow Execution

```java
String result = workflow.pizzaWorkflow(input);
```

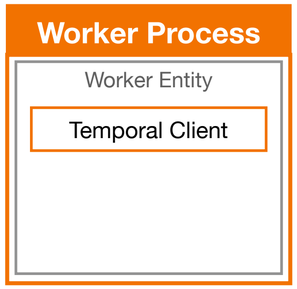
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

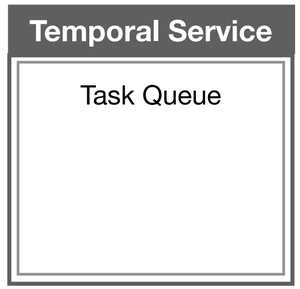
Start Workflow Execution

```java
String result = workflow.pizzaWorkflow(input);
```

```
[
    {
      "orderNumber": "Z1238",
      "customer": {
        "customerID": 12983,
        "name": "María García",
        "email": "maria1985@example.com",
        "phone": "415-555-7418"
      },
      "items": [
        {
          "description": "Large, with pepperoni",
          "price": 1500
        },
        {
          "description": "Small, with mushrooms and onions",
          “price": 1000
        }
      ],
      "isDelivery": true,
      "address": {
        "line1": "701 Mission Street",
        "line2": "Apartment 9C",
        "city": "San Francisco",
        "state": "CA",
        "postalCode": "94103"
      }
    }
]
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

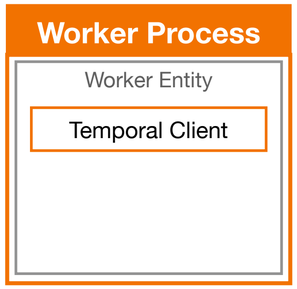


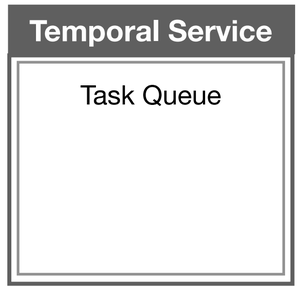


Commands

Events

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



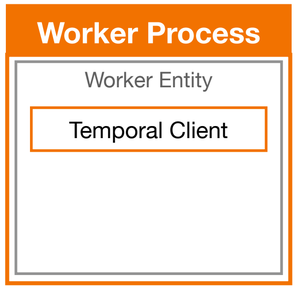


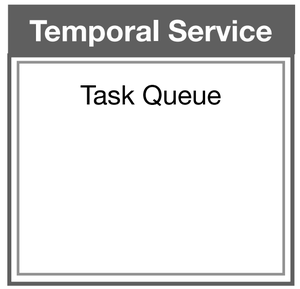
Commands

Events

WorkflowExecutionStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



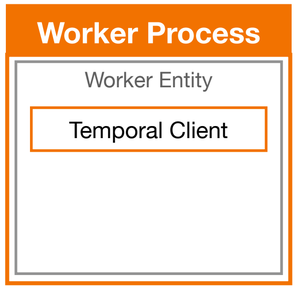


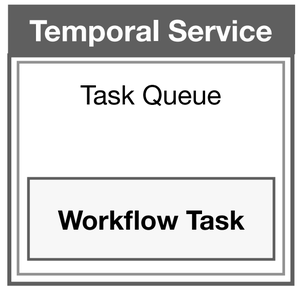
Commands

Events

WorkflowExecutionStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



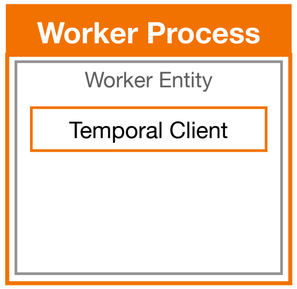


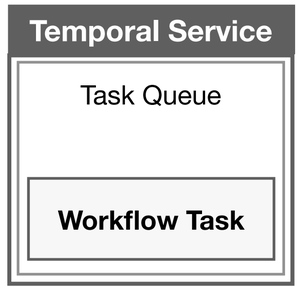
Commands

Events

WorkflowExecutionStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





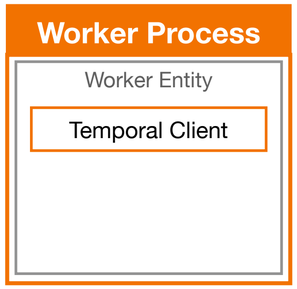
Commands

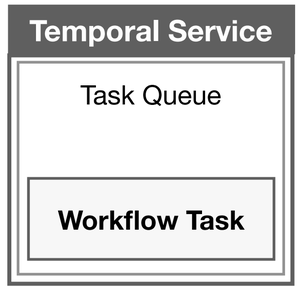
Events

WorkflowExecutionStarted

WorkflowTaskScheduled

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



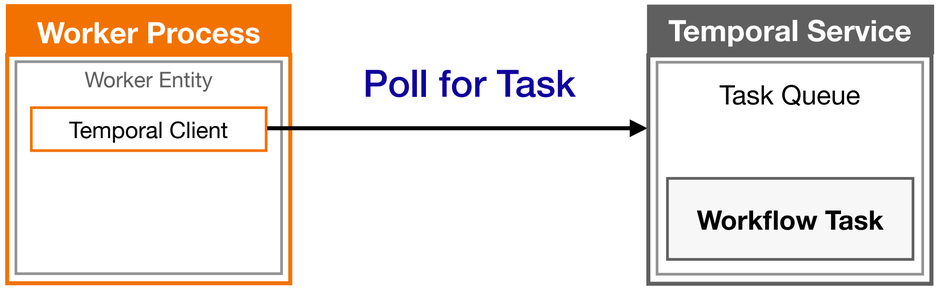


Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

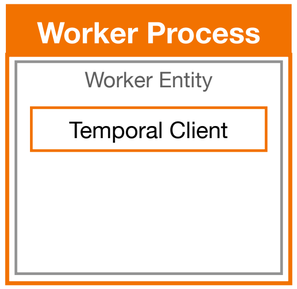


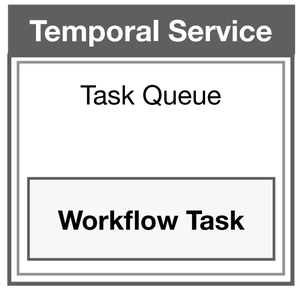
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



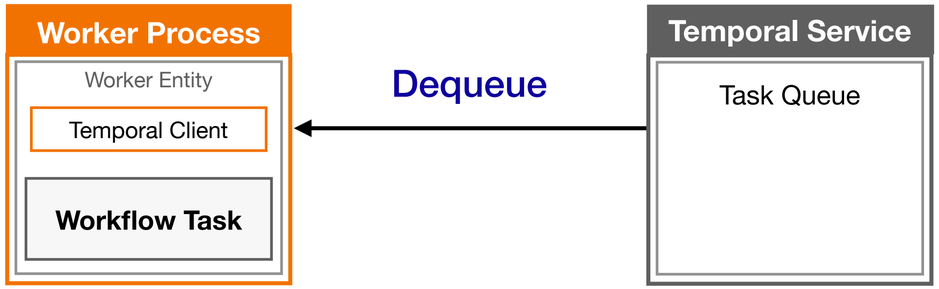


Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

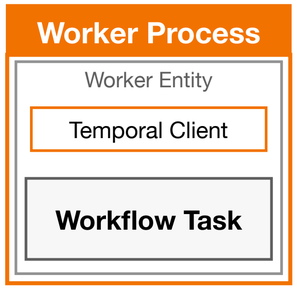


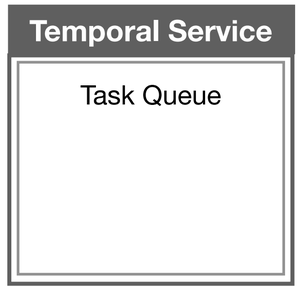
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



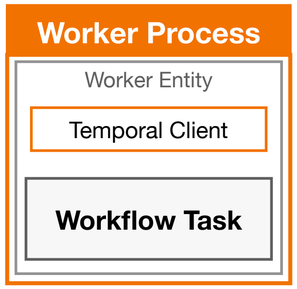


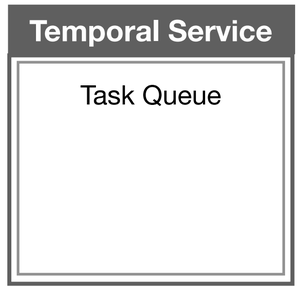
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

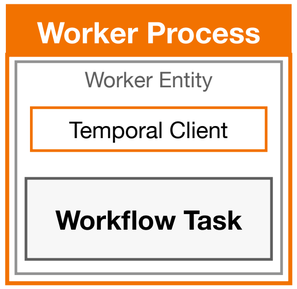
Events

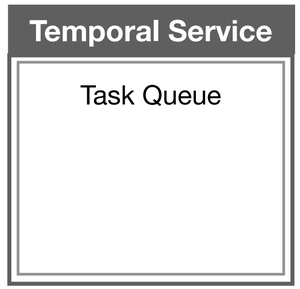
WorkflowExecutionStarted

WorkflowTaskScheduled

WorkflowTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



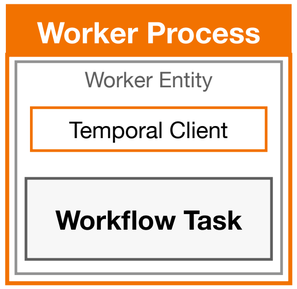


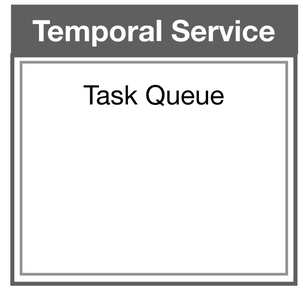
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



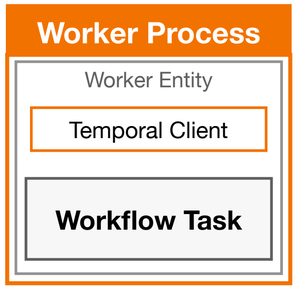


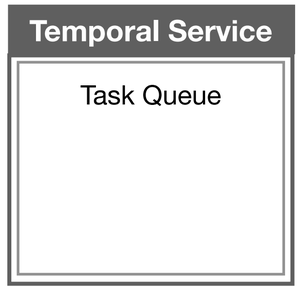
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



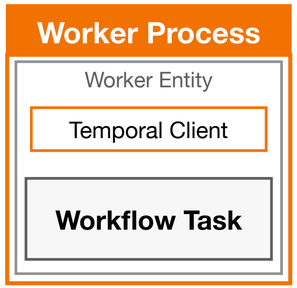


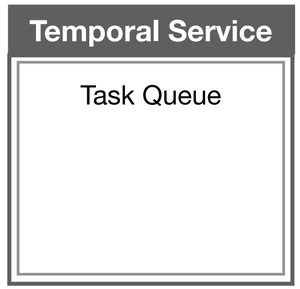
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



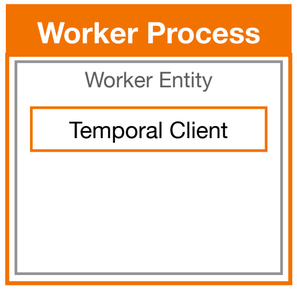


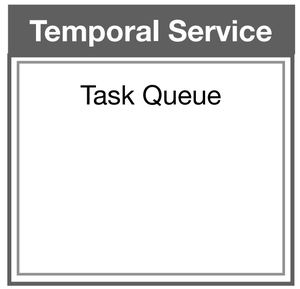
Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



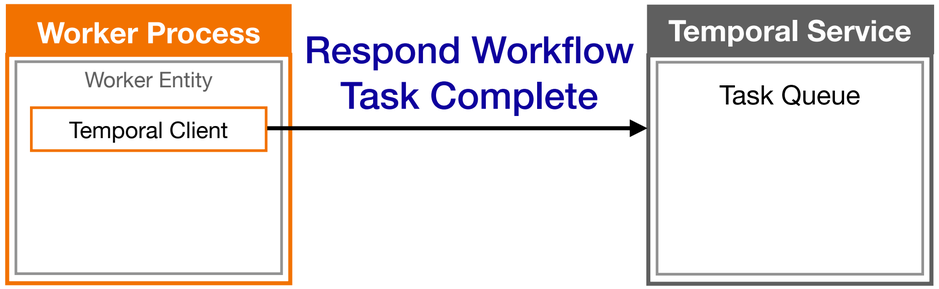


Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

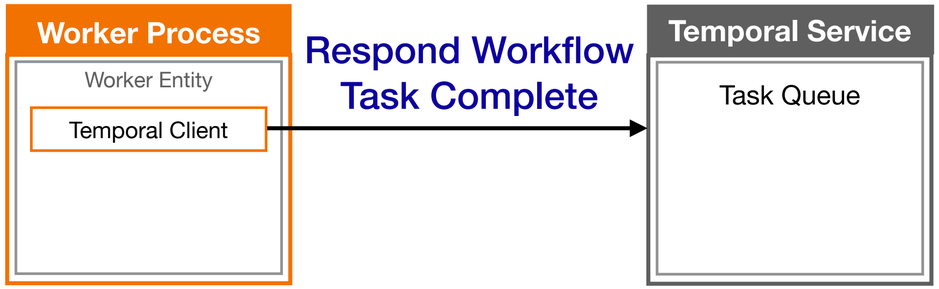


Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

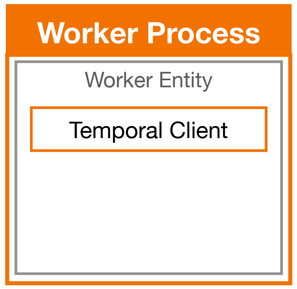


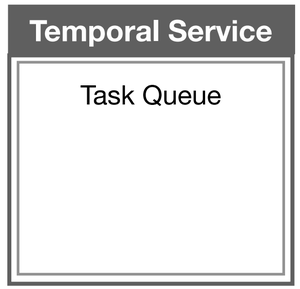
Commands

Events

- WorkflowExecutionStarted
- WorkflowTaskScheduled
- WorkflowTaskStarted
- **WorkflowTaskCompleted**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



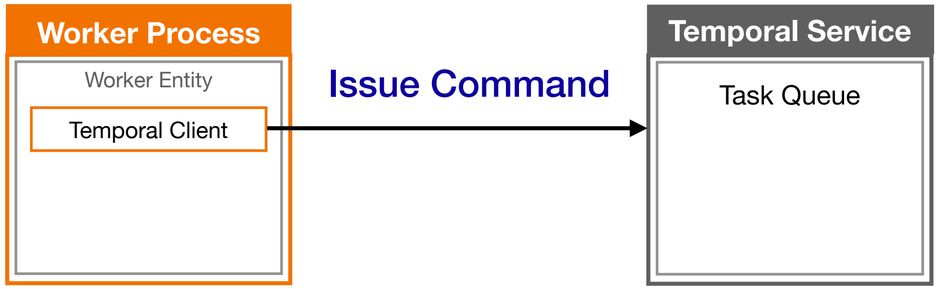


Commands

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

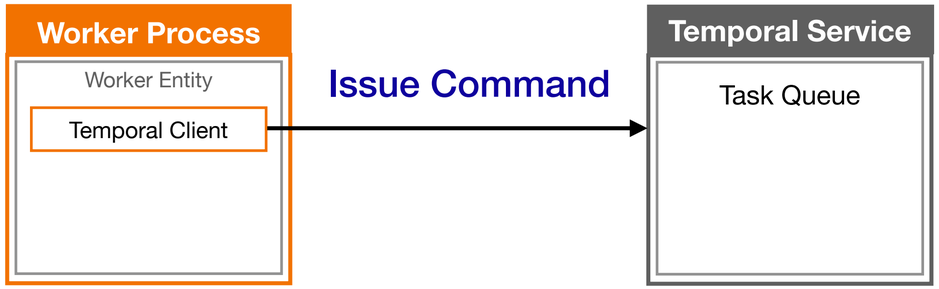


Commands

Events

- WorkflowExecutionStarted
- WorkflowTaskScheduled
- WorkflowTaskStarted
- WorkflowTaskCompleted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

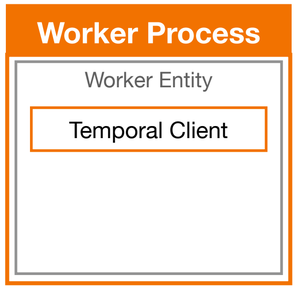
ScheduleActivityTask

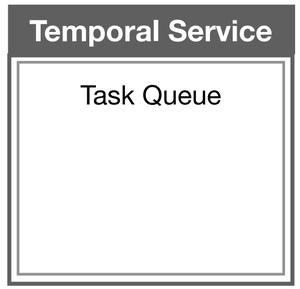
Queue: pizza-tasks
Type: get_distance
Input: “order_number": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

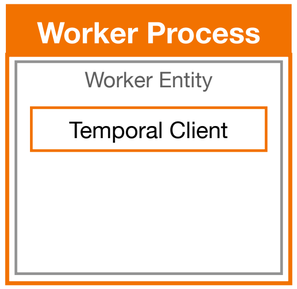
ScheduleActivityTask

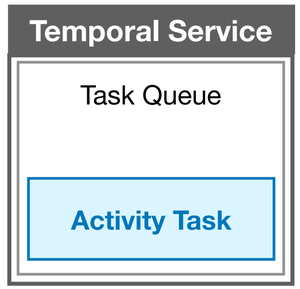
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

- WorkflowExecutionStarted
- WorkflowTaskScheduled
- WorkflowTaskStarted
- WorkflowTaskCompleted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks

Type: getDistance

Input: "orderNumber": "Z1238", ...

Events

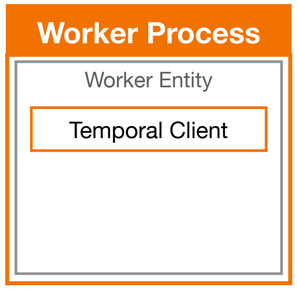
WorkflowExecutionStarted

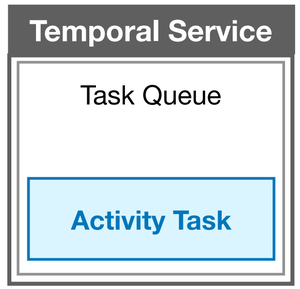
WorkflowTaskScheduled

WorkflowTaskStarted

WorkflowTaskCompleted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks

Type: getDistance

Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted

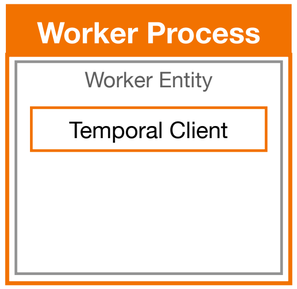
WorkflowTaskScheduled

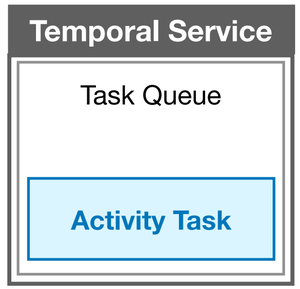
WorkflowTaskStarted

WorkflowTaskCompleted

ActivityTaskScheduled (get_distance)

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks

Type: getDistance

Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted

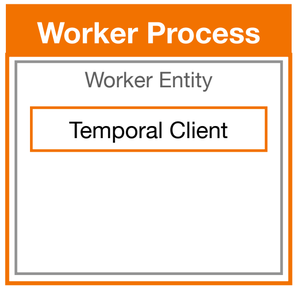
WorkflowTaskScheduled

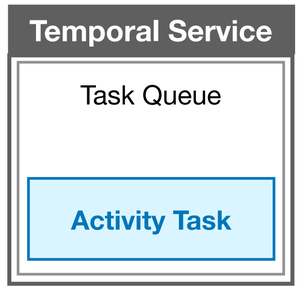
WorkflowTaskStarted

WorkflowTaskCompleted

ActivityTaskScheduled **(getDistance)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks

Type: getDistance

Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted

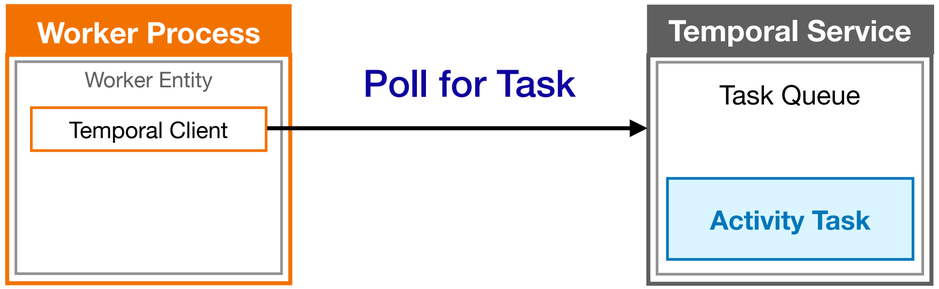
WorkflowTaskScheduled

WorkflowTaskStarted

WorkflowTaskCompleted

ActivityTaskScheduled **(getDistance)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

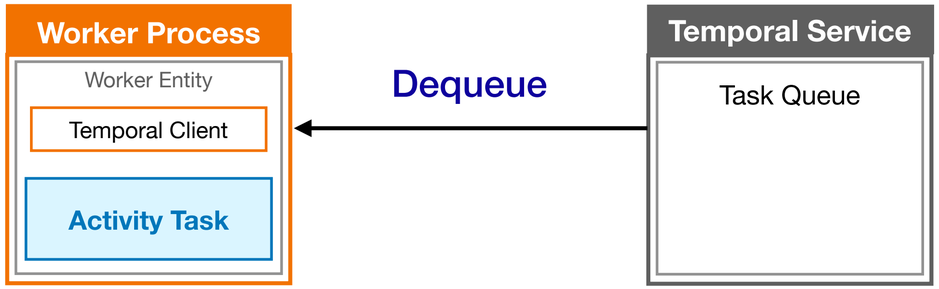
ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

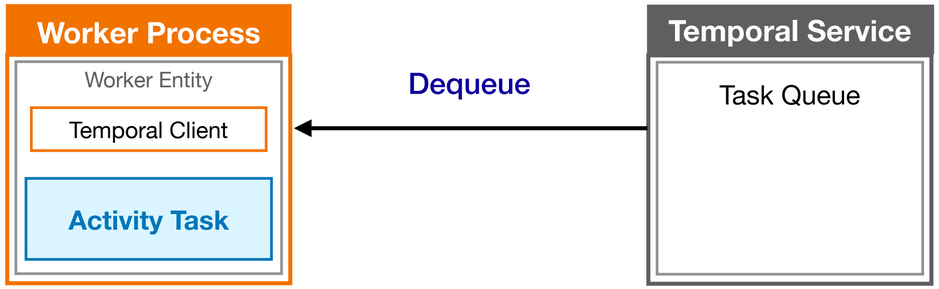
ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

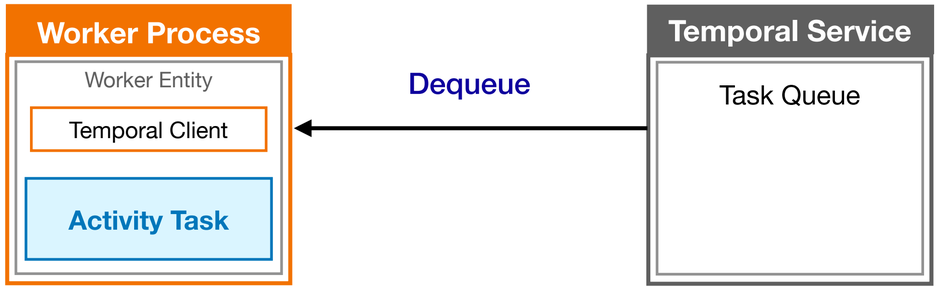
ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

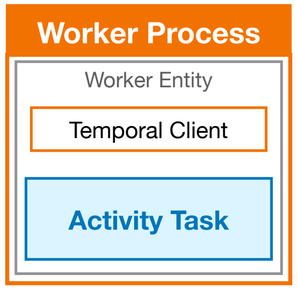
ScheduleActivityTask

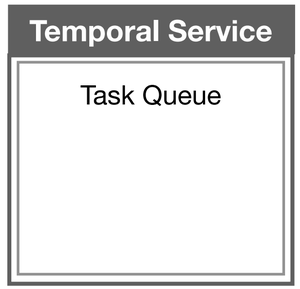
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

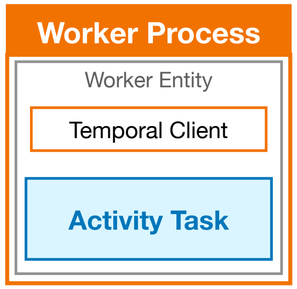
ScheduleActivityTask

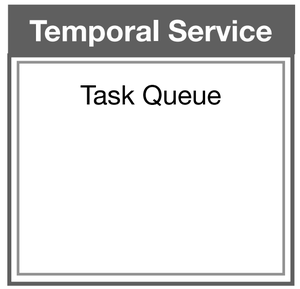
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks

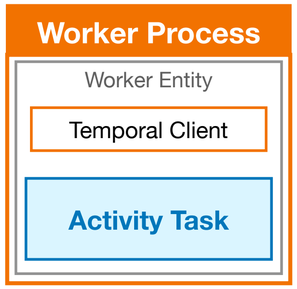
Type: getDistance

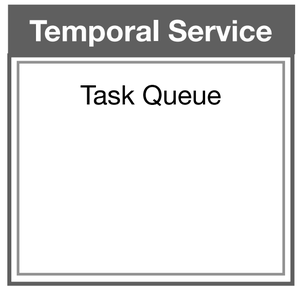
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks

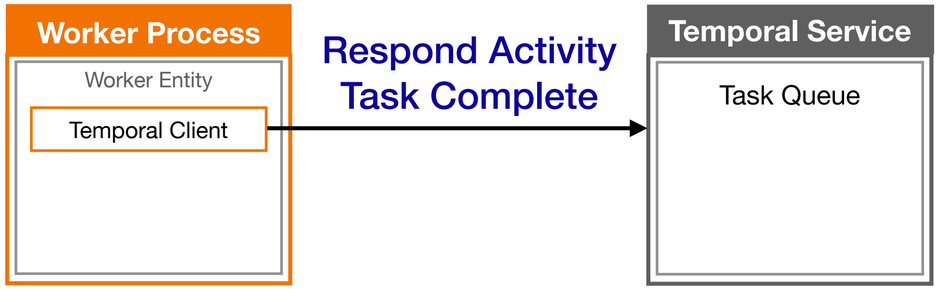
Type: getDistance

Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

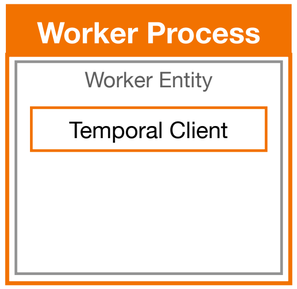
ScheduleActivityTask

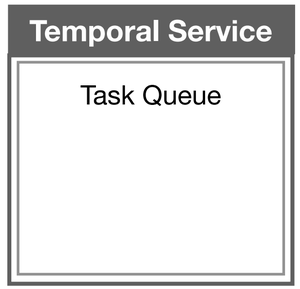
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

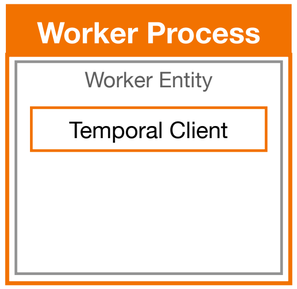
ScheduleActivityTask

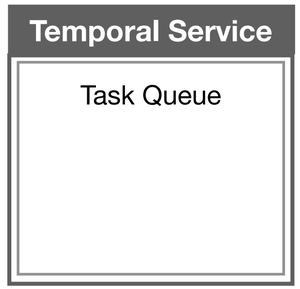
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted **(distance=15)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks

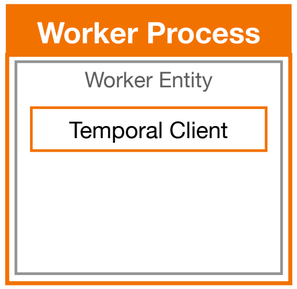
Type: getDistance

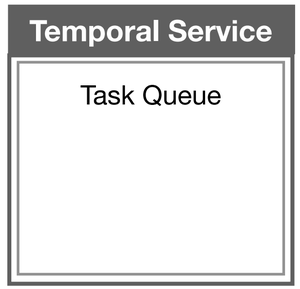
Input: "orderNumber": "Z1238", ...

Events

- WorkflowExecutionStarted
- WorkflowTaskScheduled
- WorkflowTaskStarted
- WorkflowTaskCompleted
- ActivityTaskScheduled **(getDistance)**
- ActivityTaskStarted
- **ActivityTaskCompleted** **(distance=15)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

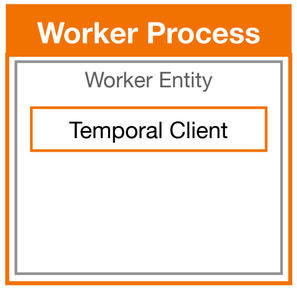
ScheduleActivityTask

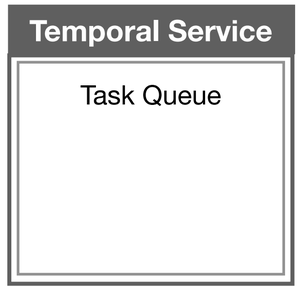
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

- WorkflowExecutionStarted
- WorkflowTaskScheduled
- WorkflowTaskStarted
- WorkflowTaskCompleted
- ActivityTaskScheduled **(getDistance)**
- ActivityTaskStarted
- **ActivityTaskCompleted** **(distance=15)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

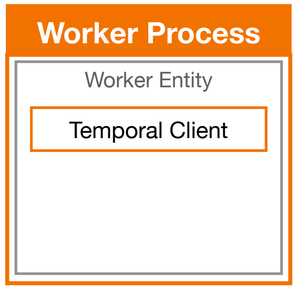
ScheduleActivityTask

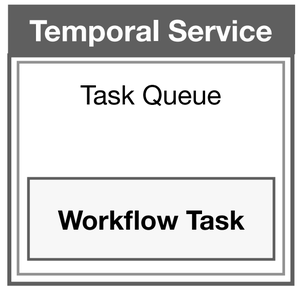
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

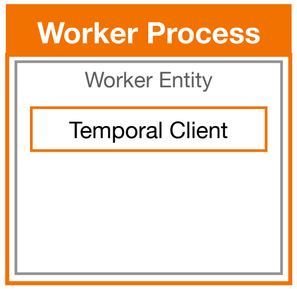
ScheduleActivityTask

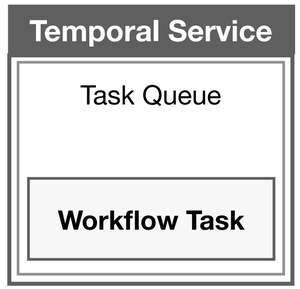
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

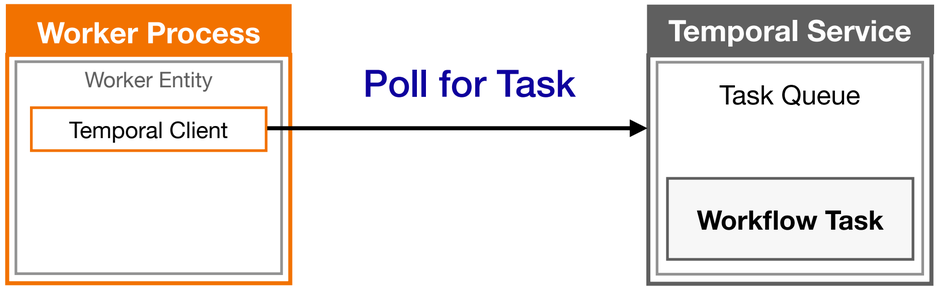
ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

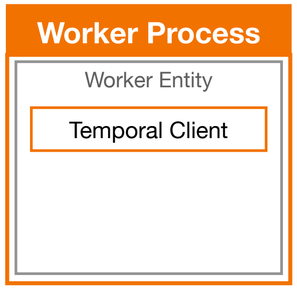
ScheduleActivityTask

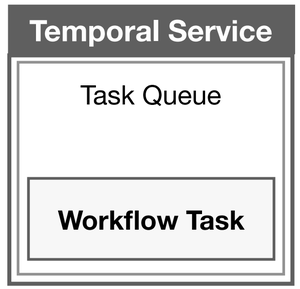
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

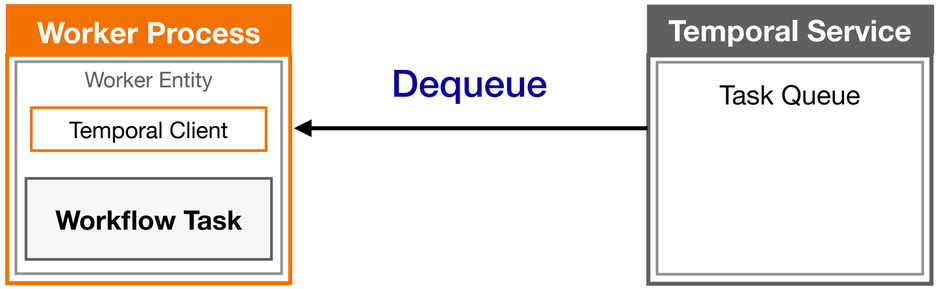
ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

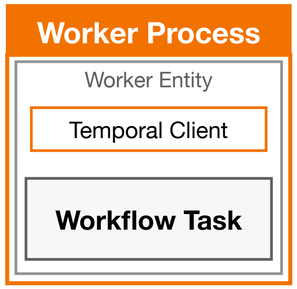
ScheduleActivityTask

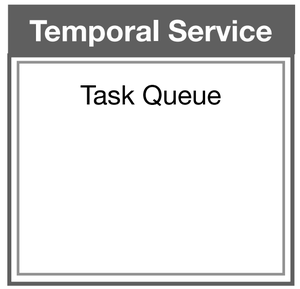
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

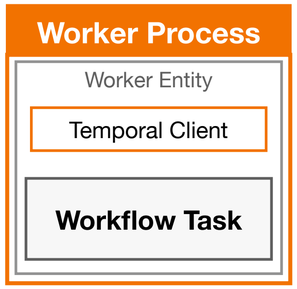
ScheduleActivityTask

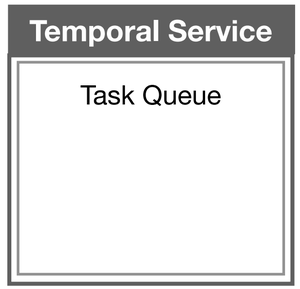
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

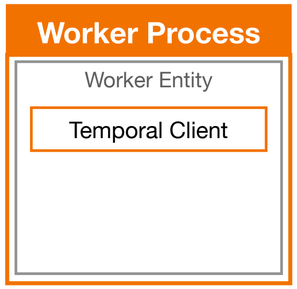
ScheduleActivityTask

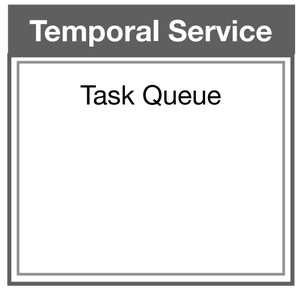
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted
ActivityTaskCompleted **(distance=15)**
WorkflowTaskScheduled
WorkflowTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

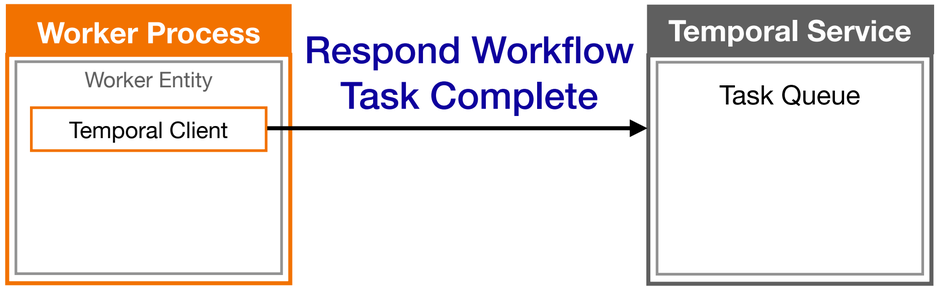
ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

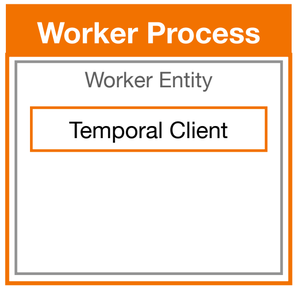
ScheduleActivityTask

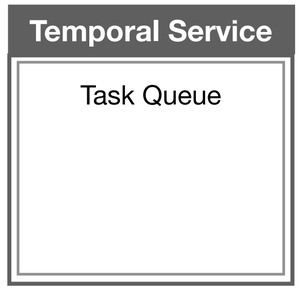
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

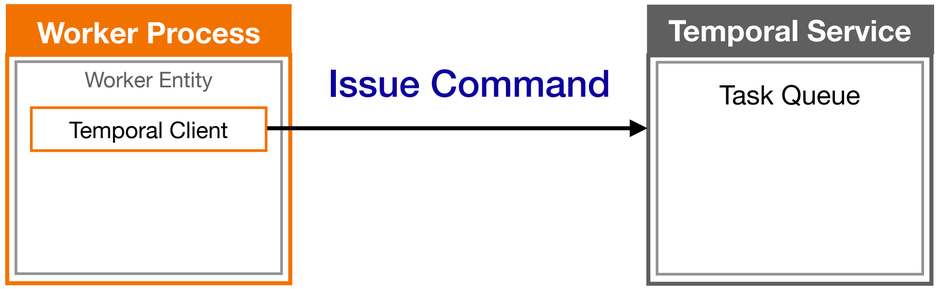
ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

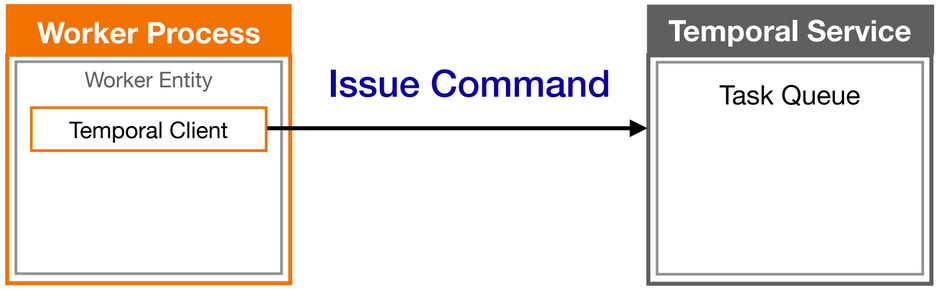
ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

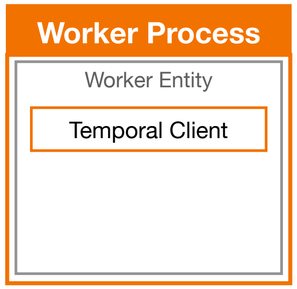
StartTimer

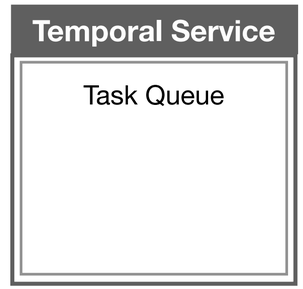
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

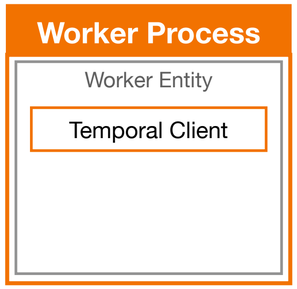
StartTimer

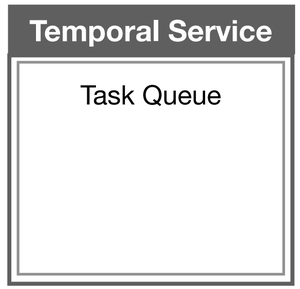
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

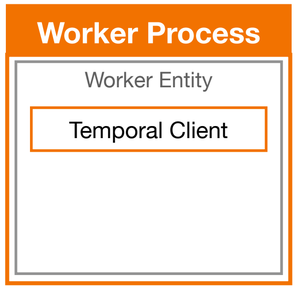
StartTimer

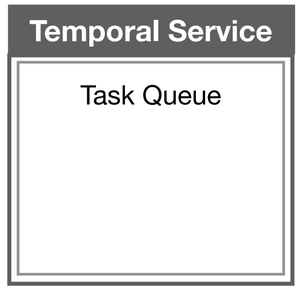
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

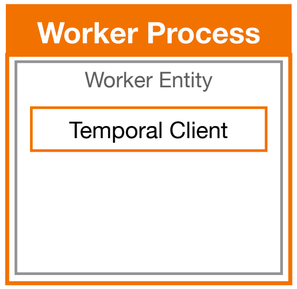
StartTimer

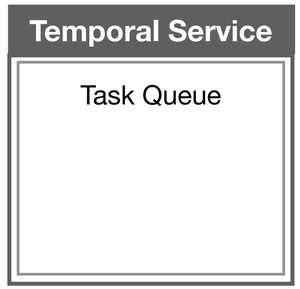
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

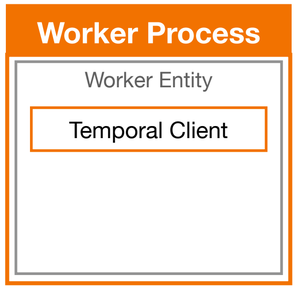
StartTimer

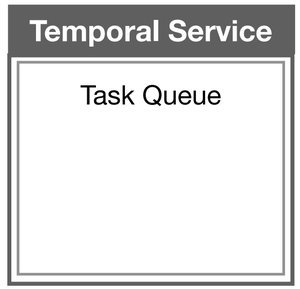
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

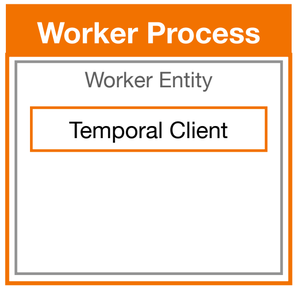
StartTimer

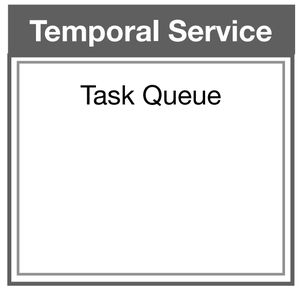
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

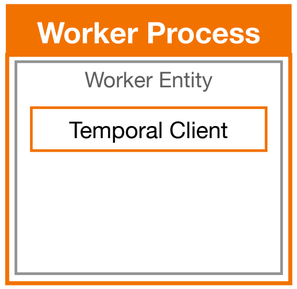
StartTimer

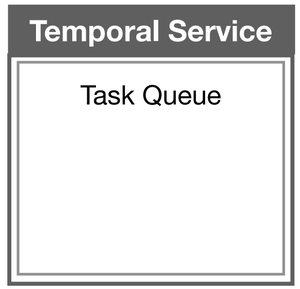
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

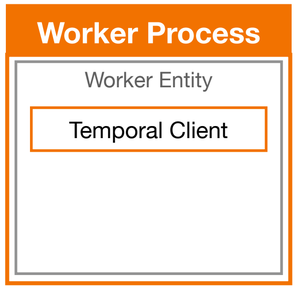
StartTimer

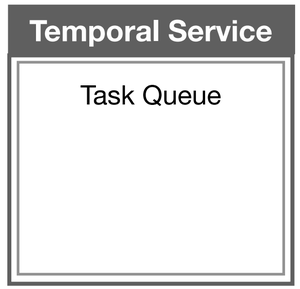
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

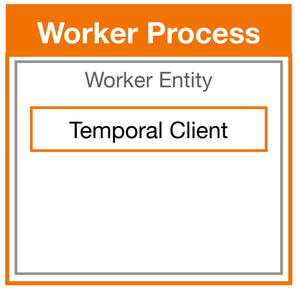
StartTimer

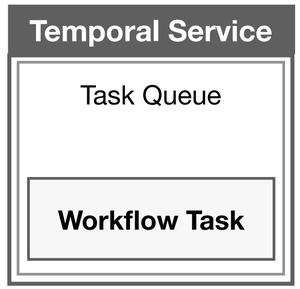
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

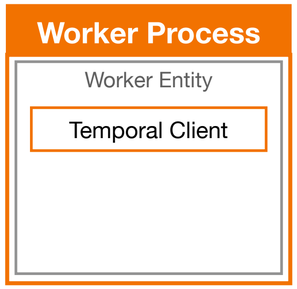
StartTimer

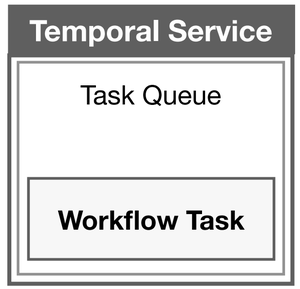
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

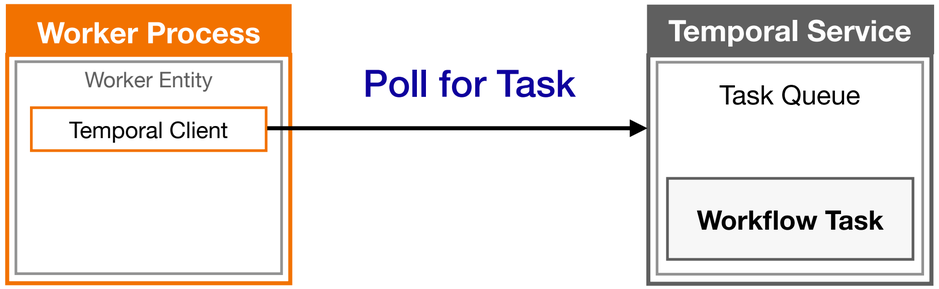
StartTimer

Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

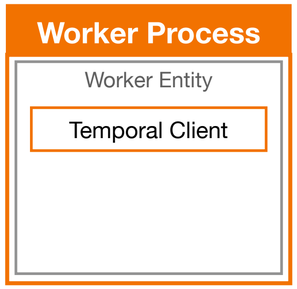
StartTimer

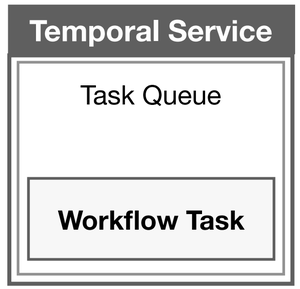
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

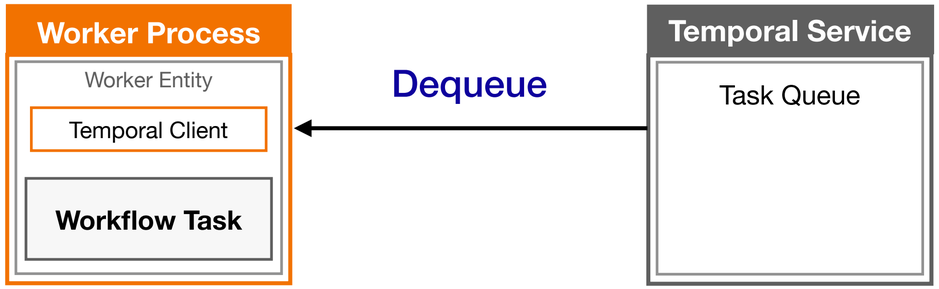
StartTimer

Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

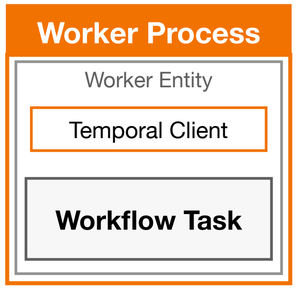
StartTimer

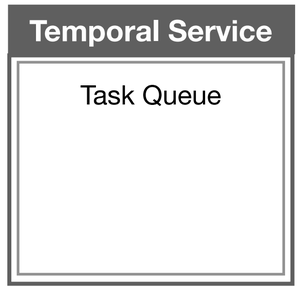
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

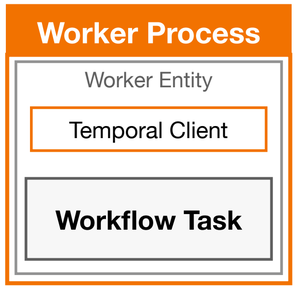
StartTimer

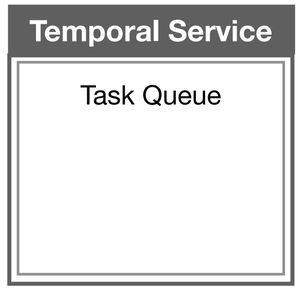
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

StartTimer

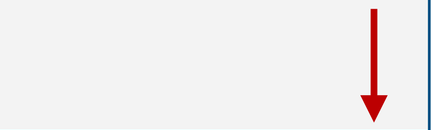
Duration: 30 minutes

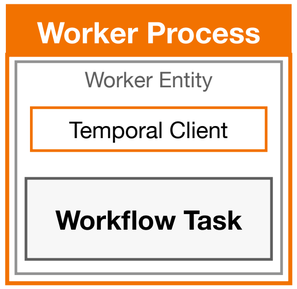
Events

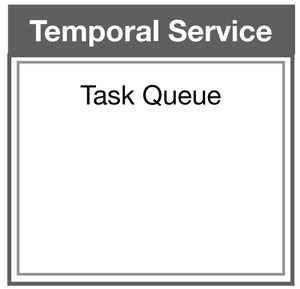
WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Worker crashes here







Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

StartTimer

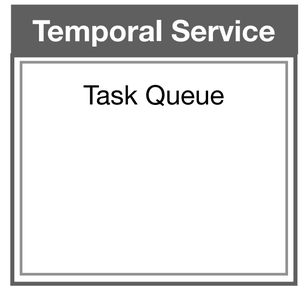
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Worker crashes here



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

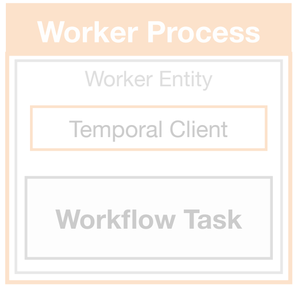
StartTimer

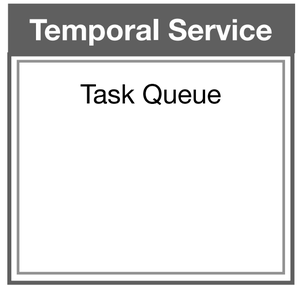
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Worker Process

Workflow Task Timeout Exceeded

Temporal Service

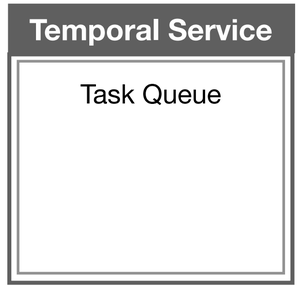
Task Queue

Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

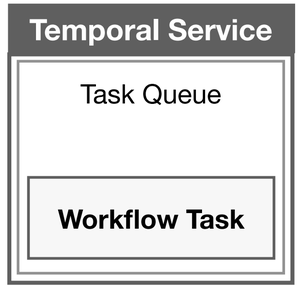


Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

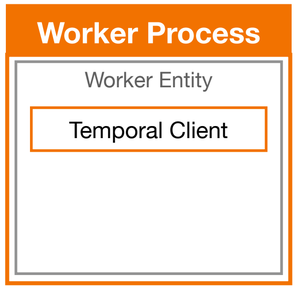


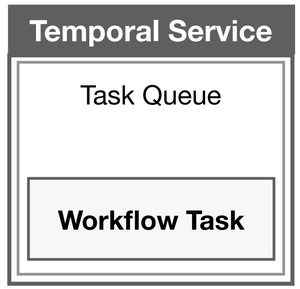
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



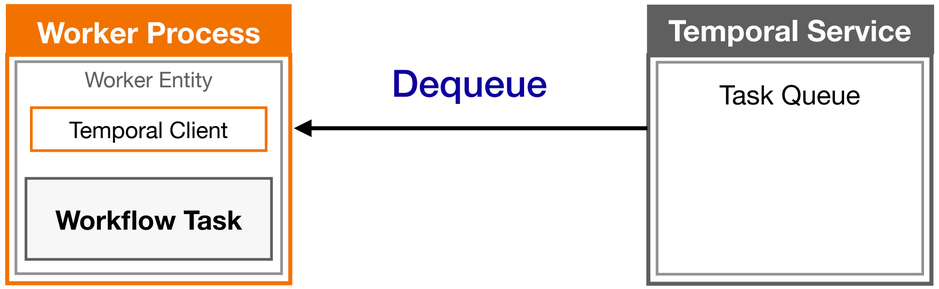


Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

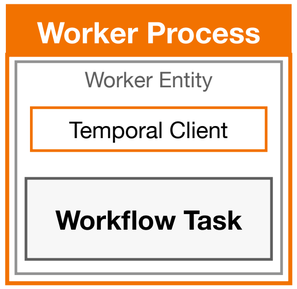


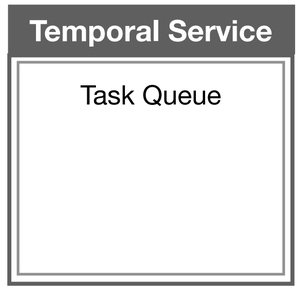
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



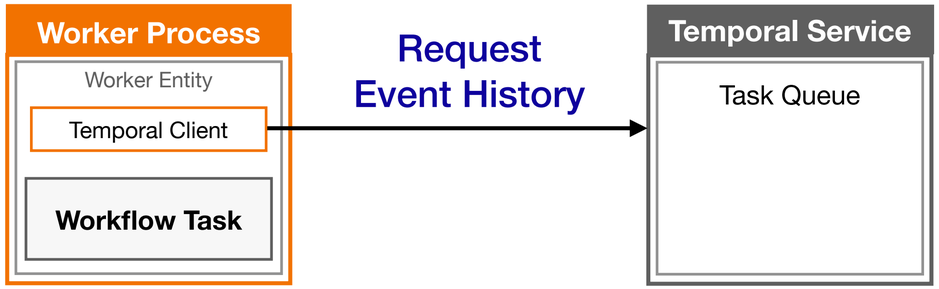


Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

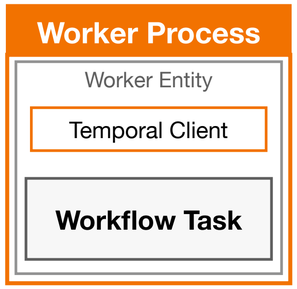


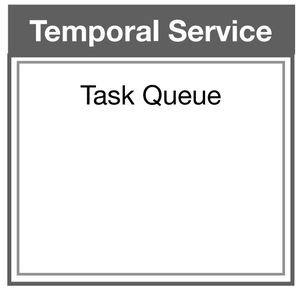
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

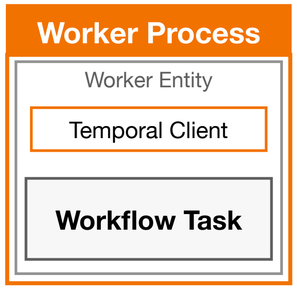
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

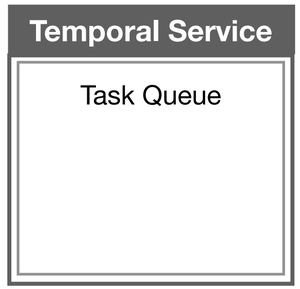
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



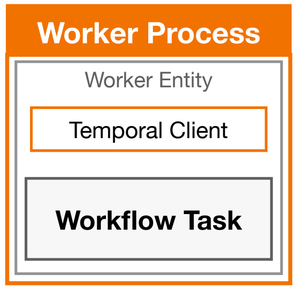


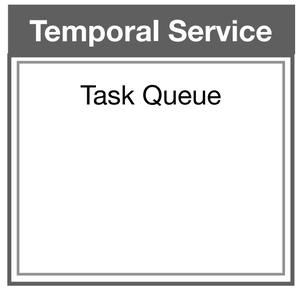
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



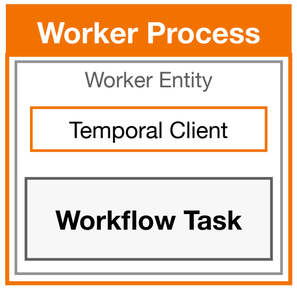


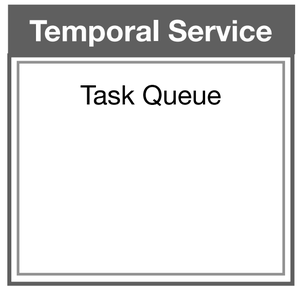
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



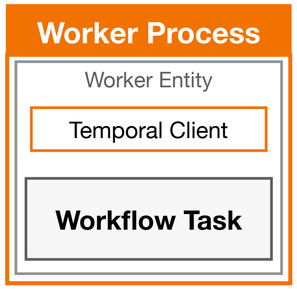


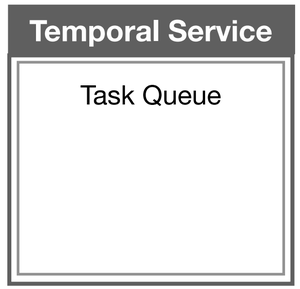
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



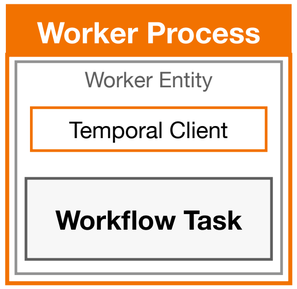


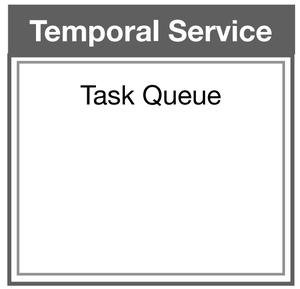
Commands

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

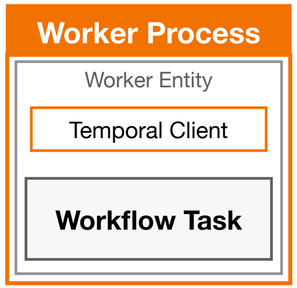
ScheduleActivityTask

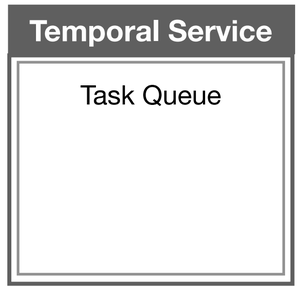
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

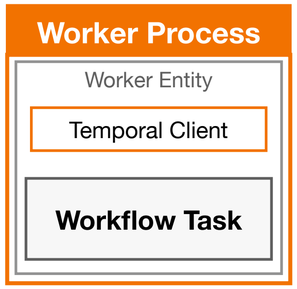
ScheduleActivityTask

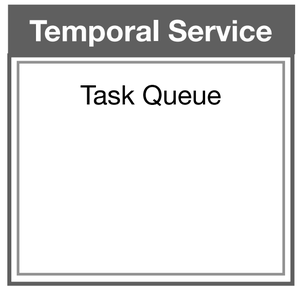
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
ActivityTaskScheduled **(getDistance)**
ActivityTaskStarted
ActivityTaskCompleted **(distance=15)**
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskCompleted
TimerStarted **(30 Minutes)**
TimerFired
WorkflowTaskScheduled
WorkflowTaskStarted
WorkflowTaskTimedOut
WorkflowTaskScheduled
WorkflowTaskStarted

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

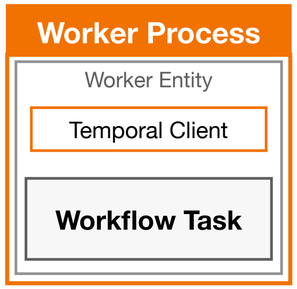
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

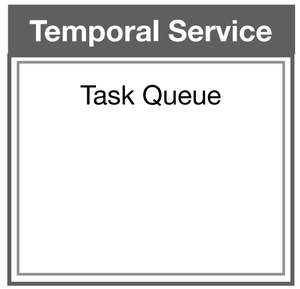
Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

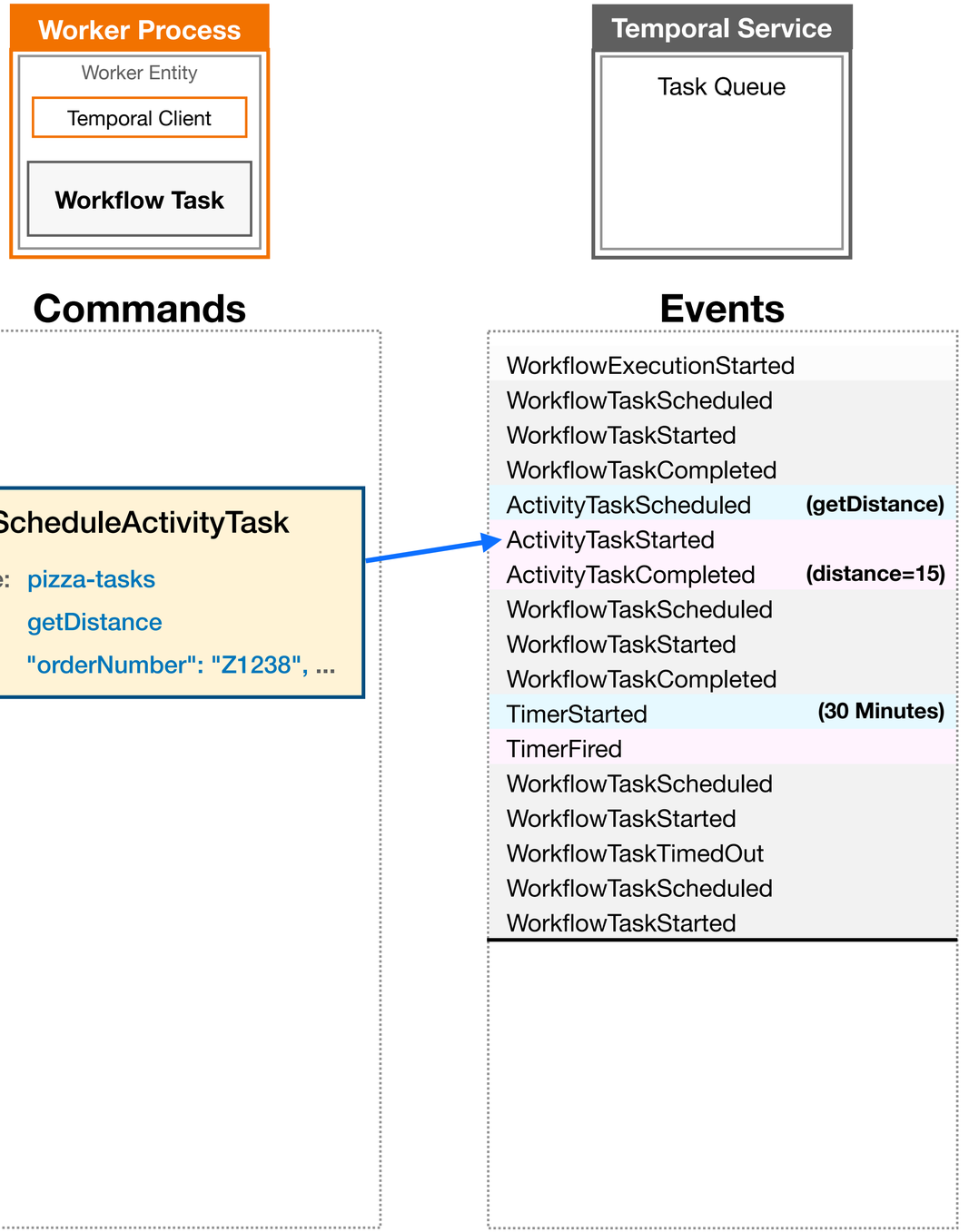
ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

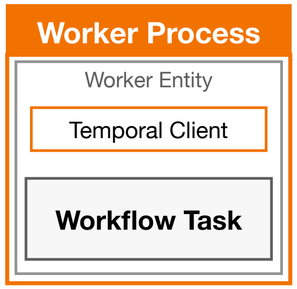
Events

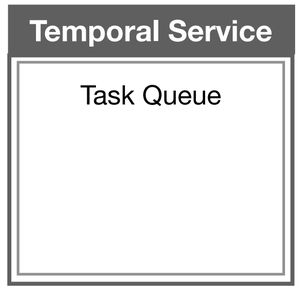
WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

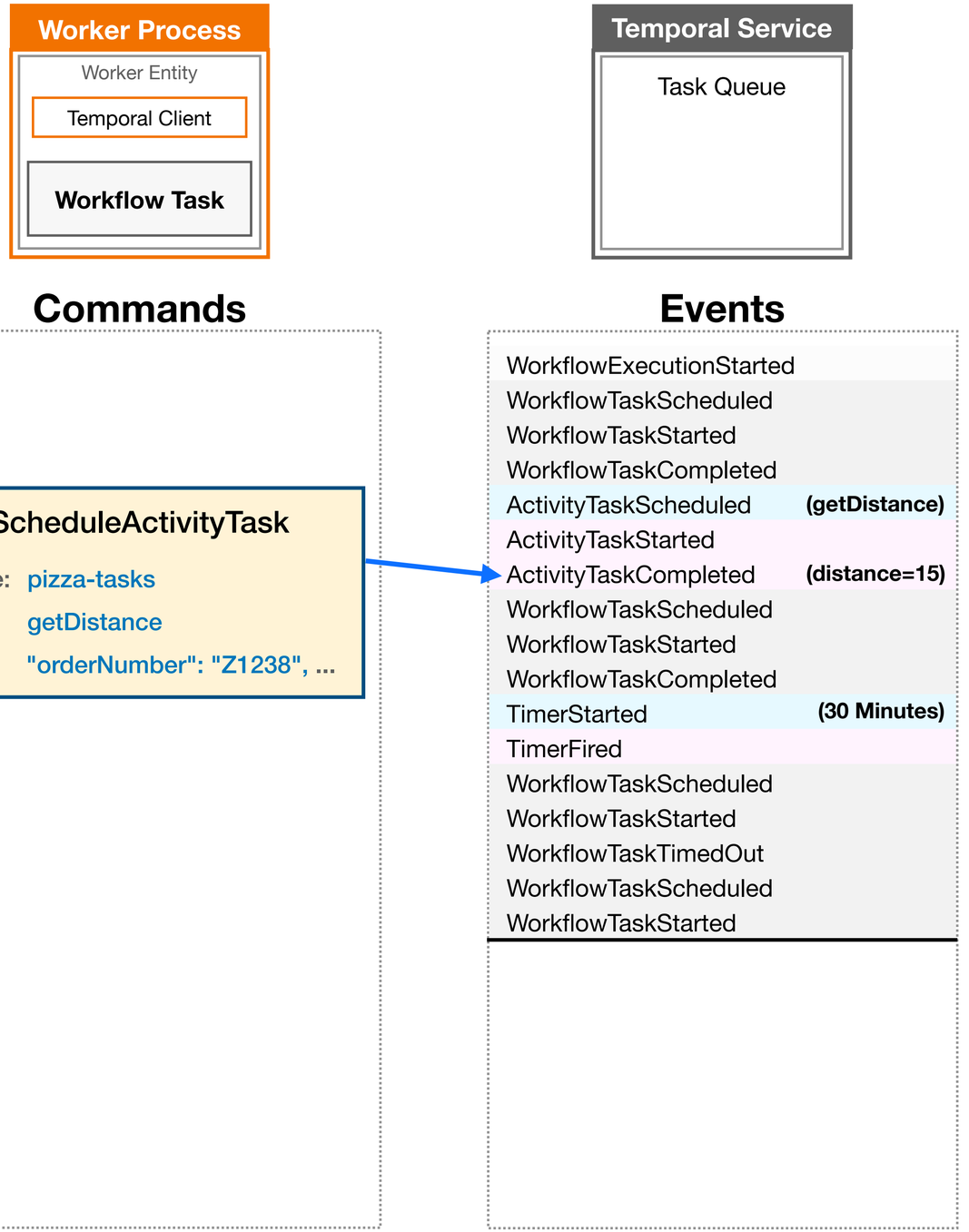
ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

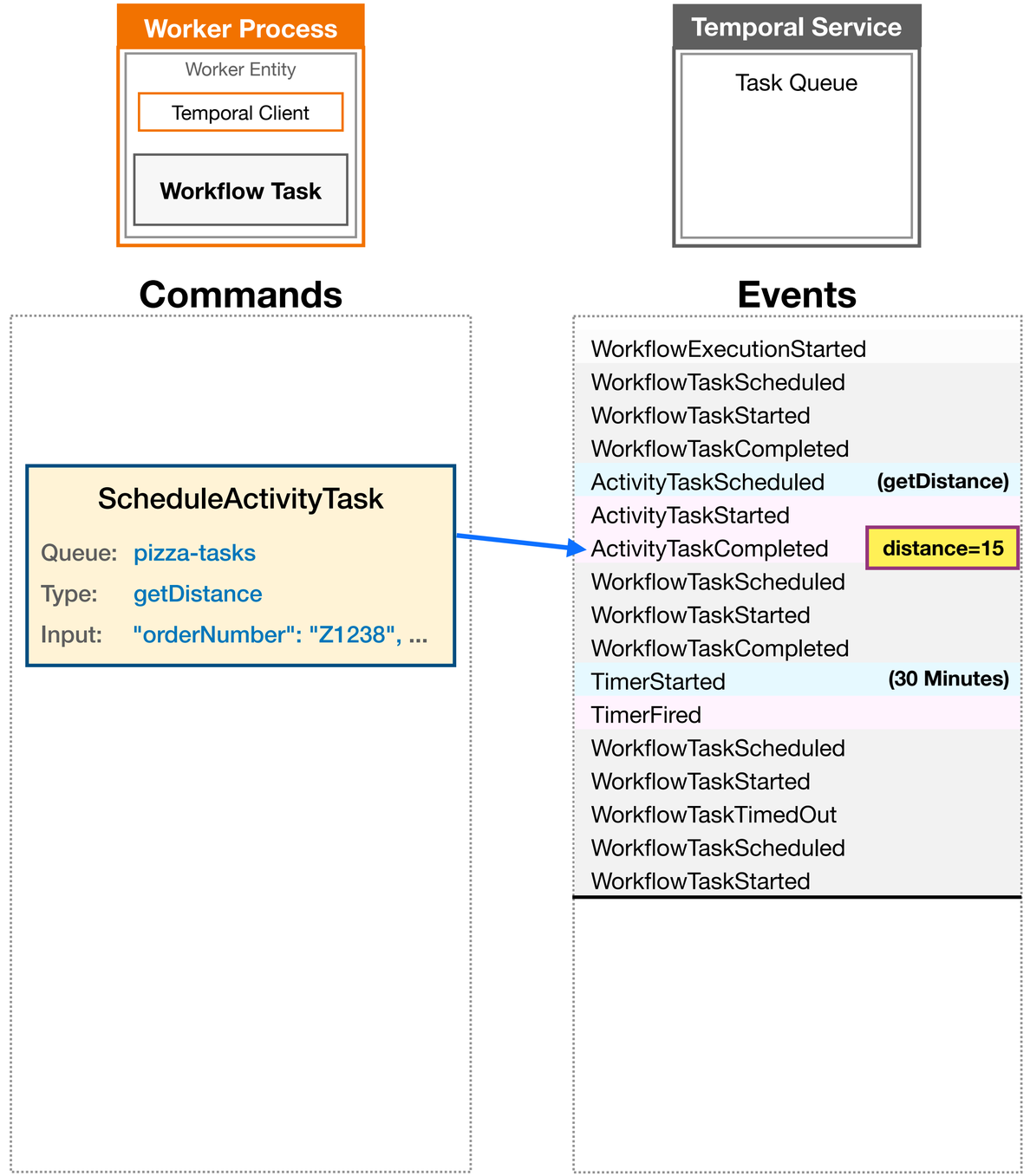
Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

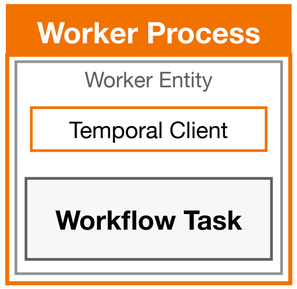
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

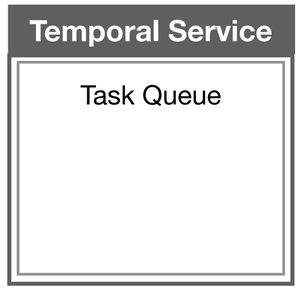


```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

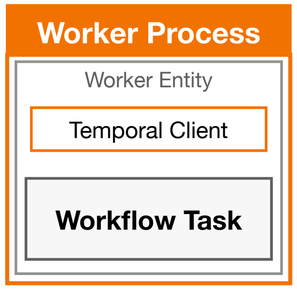
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

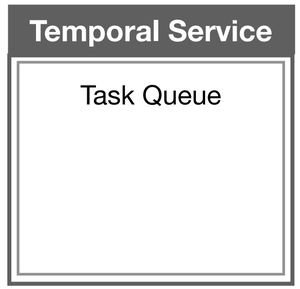
Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

Worker assigns 15 to this variable





Commands

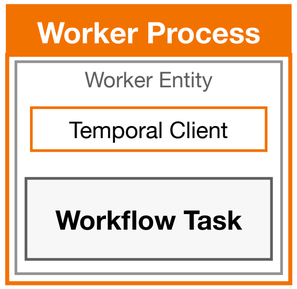
ScheduleActivityTask

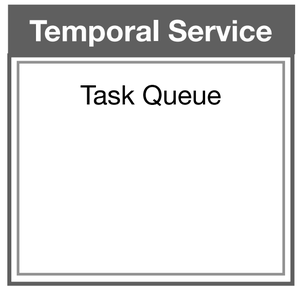
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

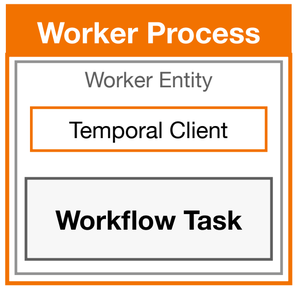
ScheduleActivityTask

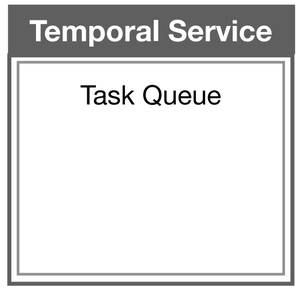
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

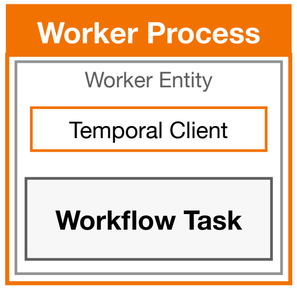
ScheduleActivityTask

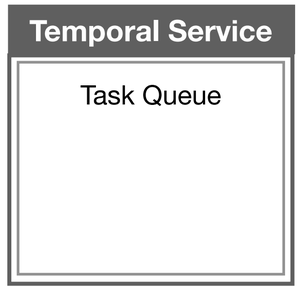
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

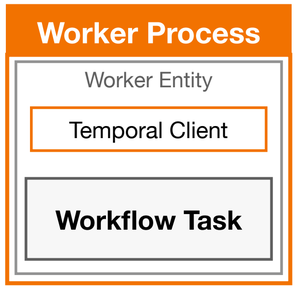
ScheduleActivityTask

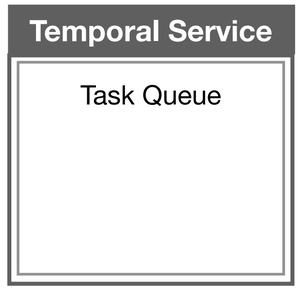
Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

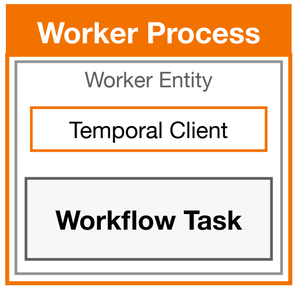
StartTimer

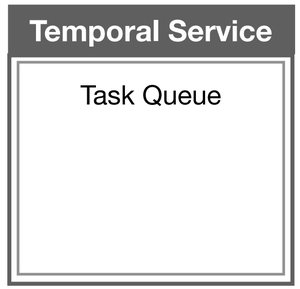
Duration: 30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

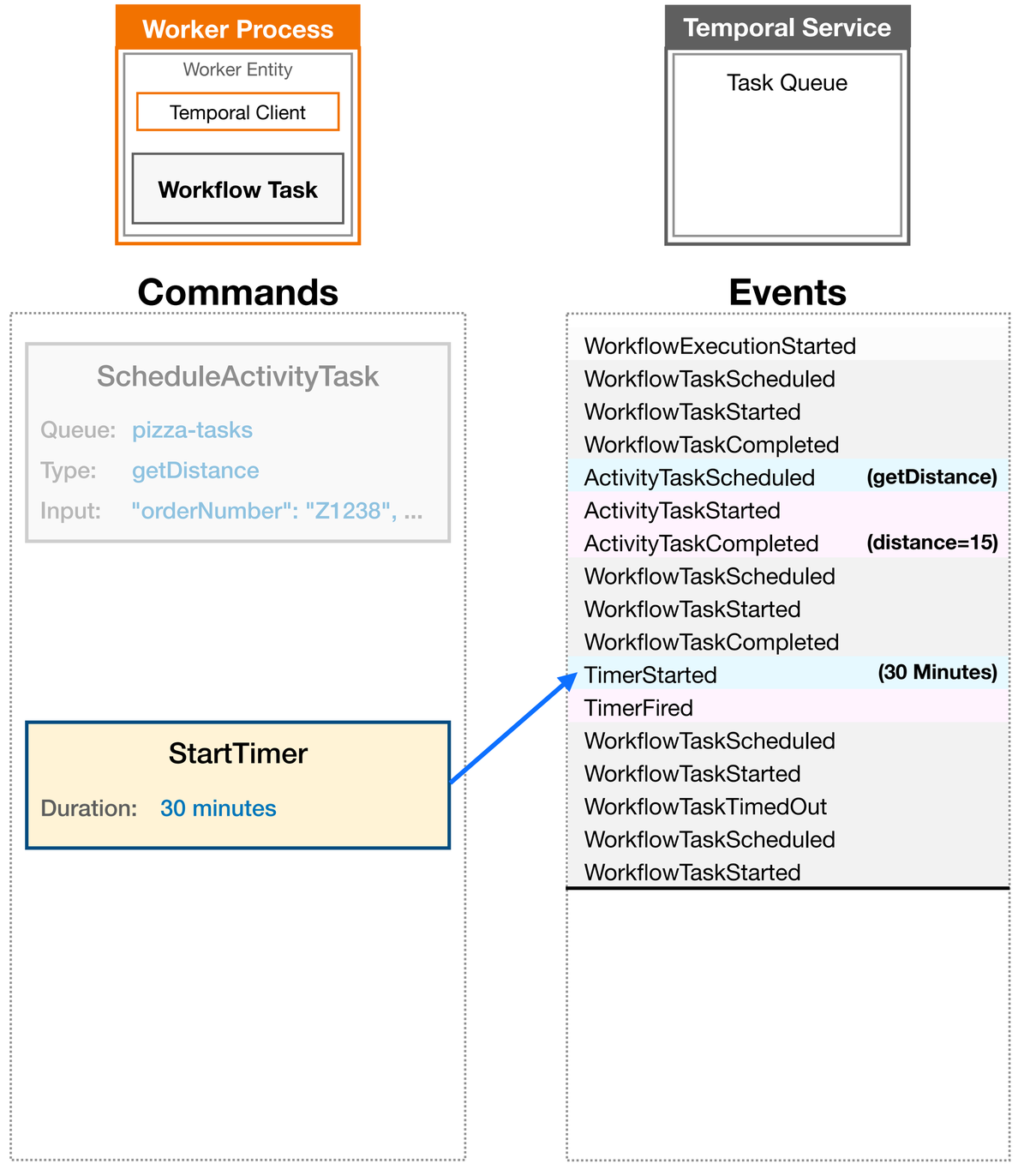
StartTimer

Duration: 30 minutes

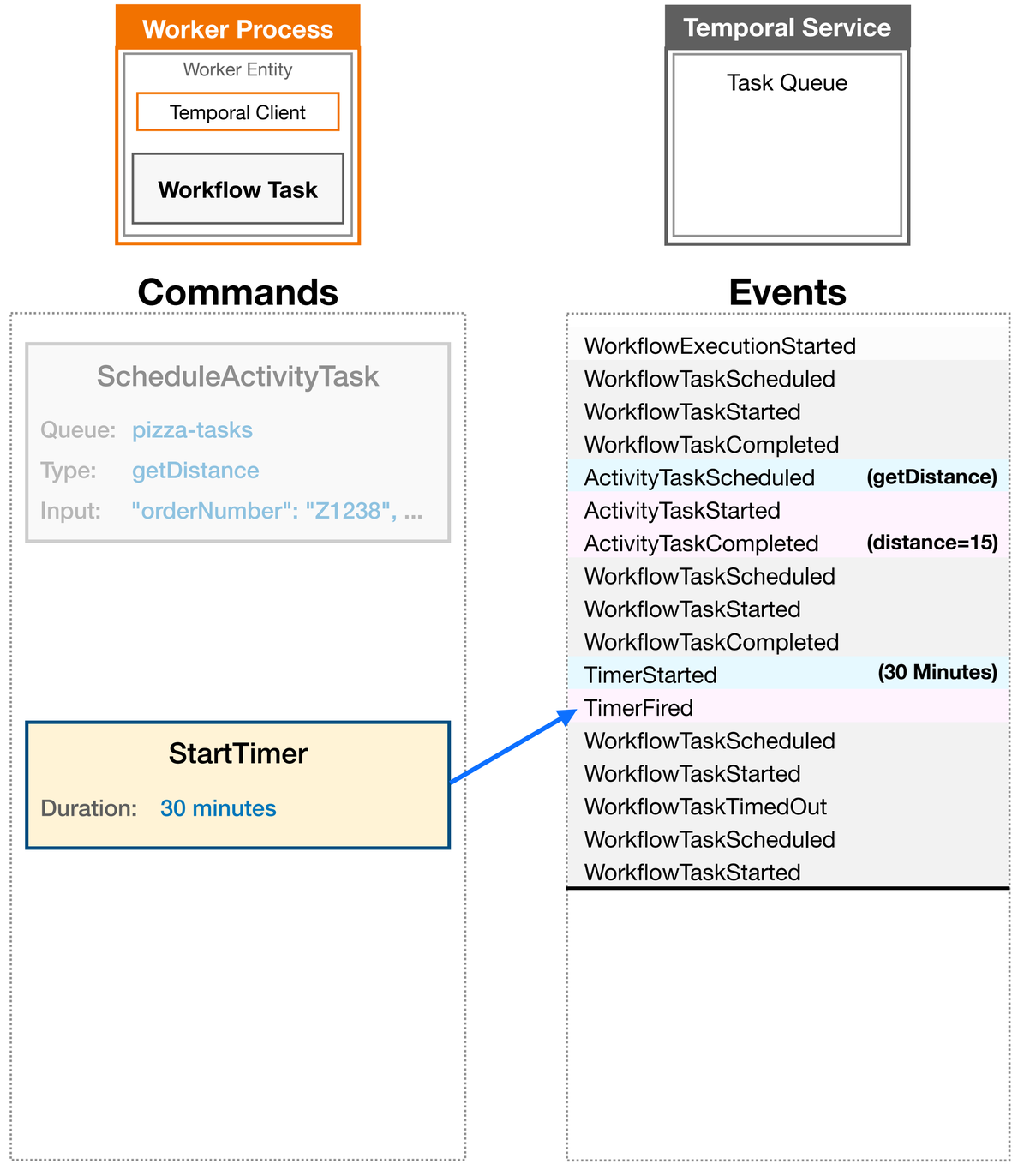
Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

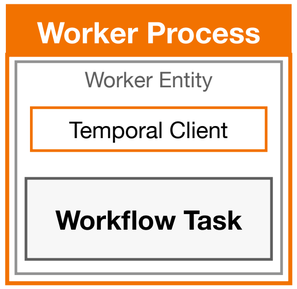
```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```

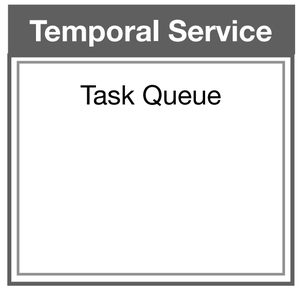


```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

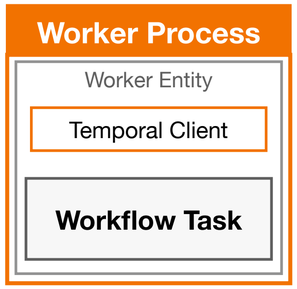
StartTimer

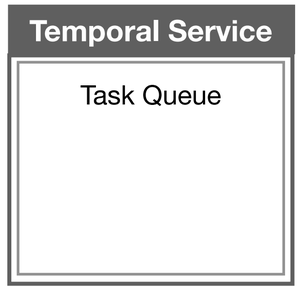
30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

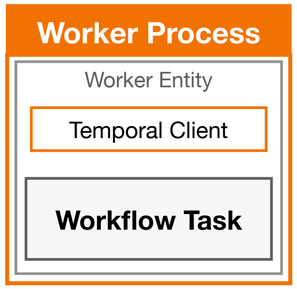
StartTimer

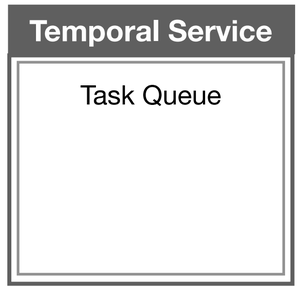
30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

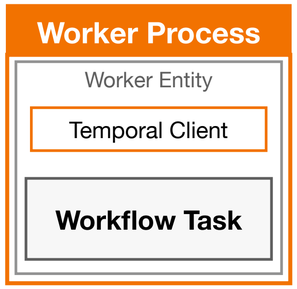
StartTimer

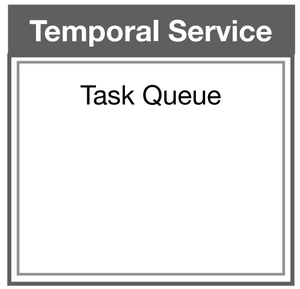
30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

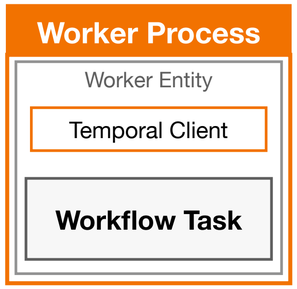
StartTimer

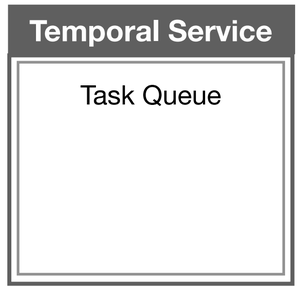
30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

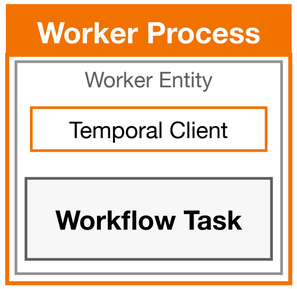
StartTimer

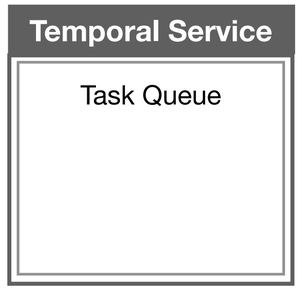
30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

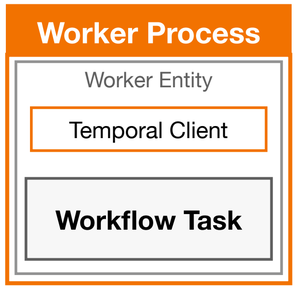
StartTimer

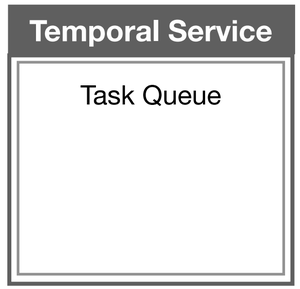
30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

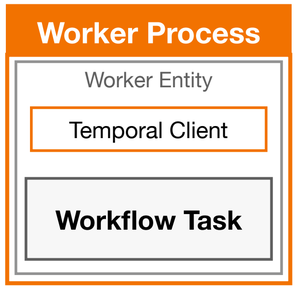
StartTimer

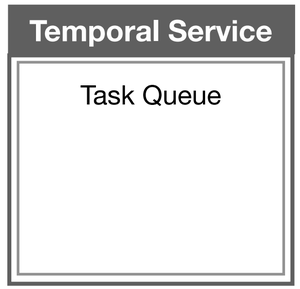
30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

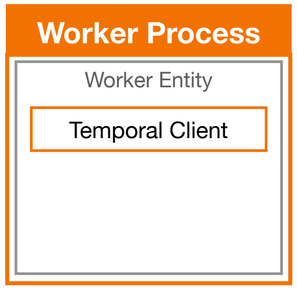
StartTimer

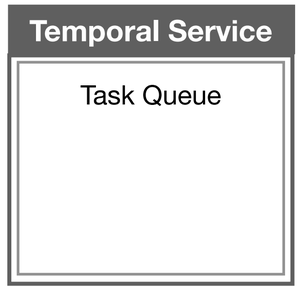
30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

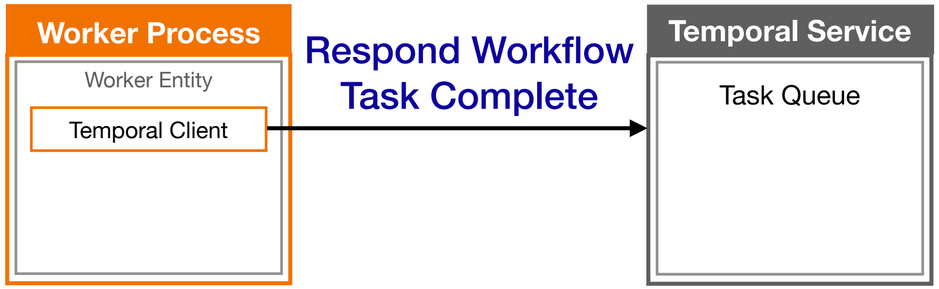
StartTimer

30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

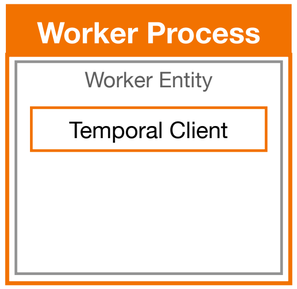
StartTimer

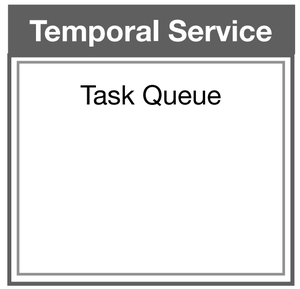
30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

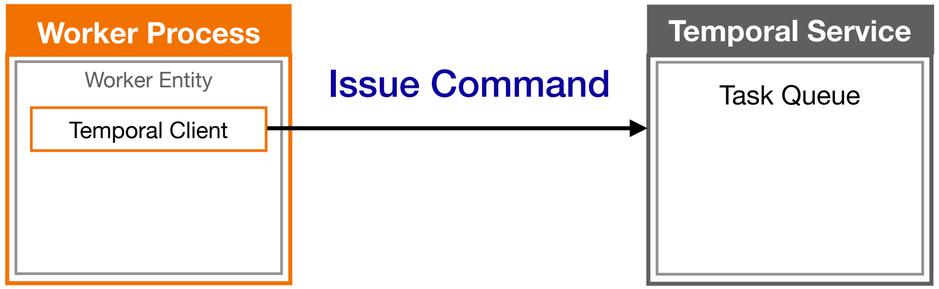
StartTimer

30 minutes

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "orderNumber": "Z1238", ...

StartTimer

30 minutes

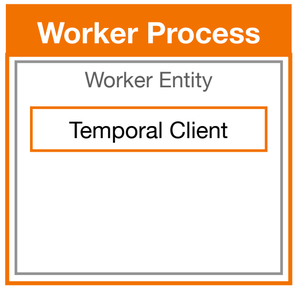
ScheduleActivityTask

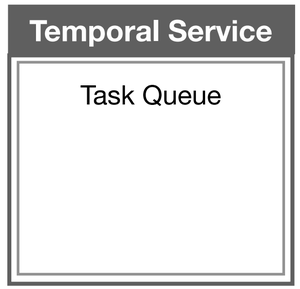
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

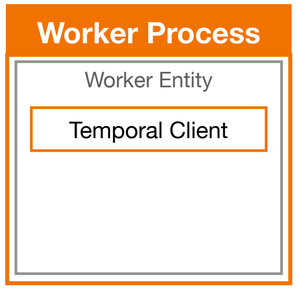
ScheduleActivityTask

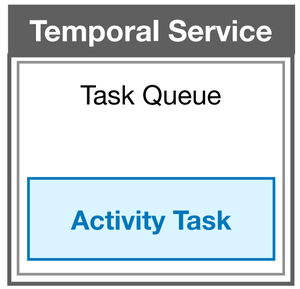
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

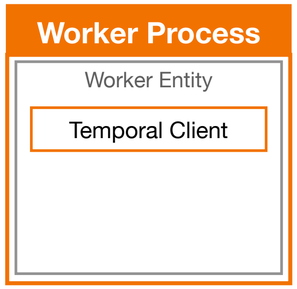
ScheduleActivityTask

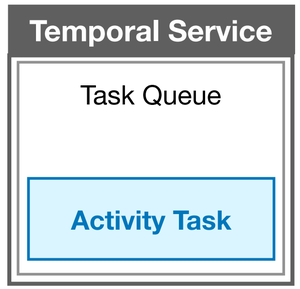
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(send_bill)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

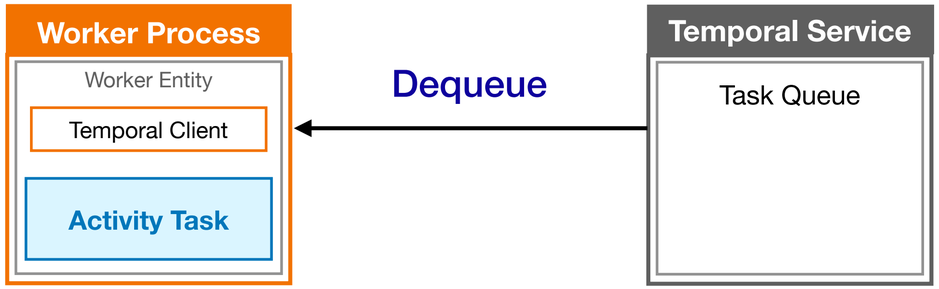
ScheduleActivityTask

Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

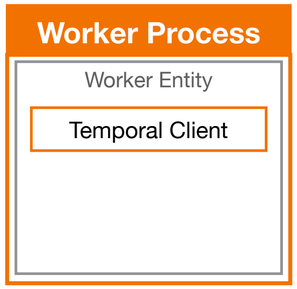
ScheduleActivityTask

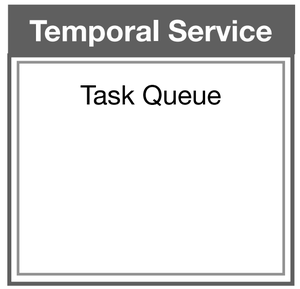
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

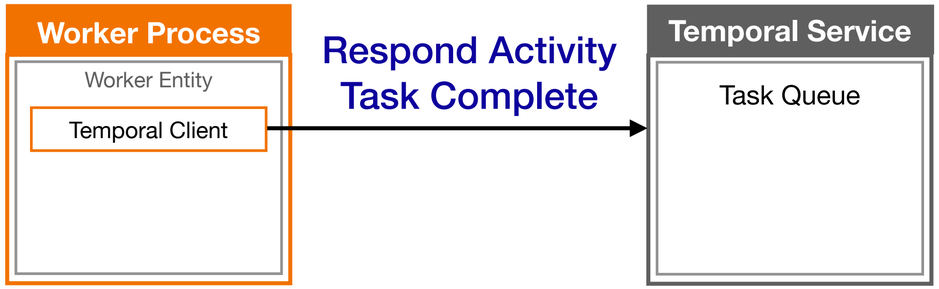
ScheduleActivityTask

Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

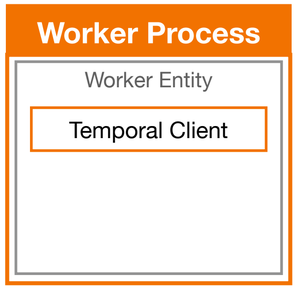
ScheduleActivityTask

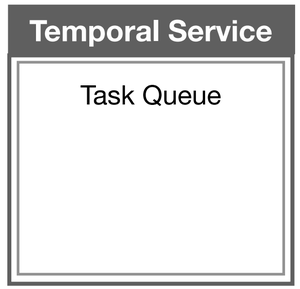
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

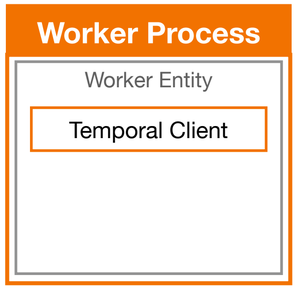
ScheduleActivityTask

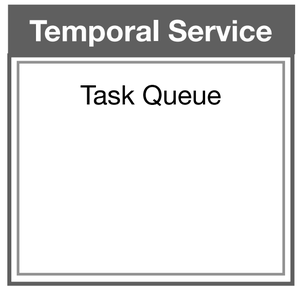
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

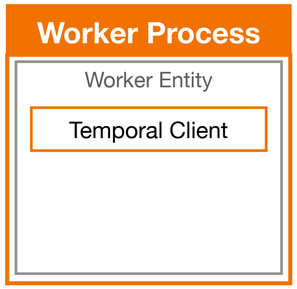
ScheduleActivityTask

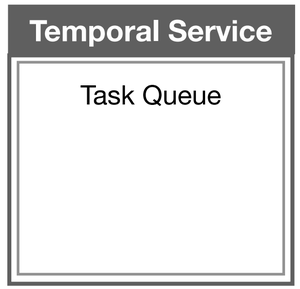
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

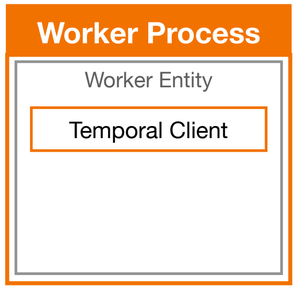
ScheduleActivityTask

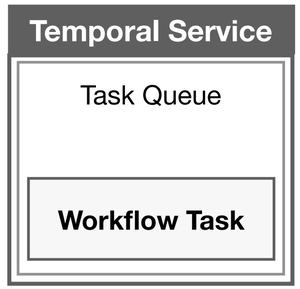
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

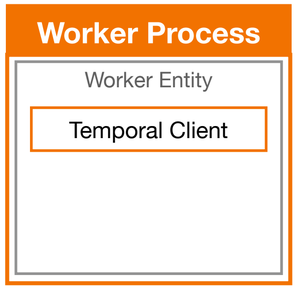
ScheduleActivityTask

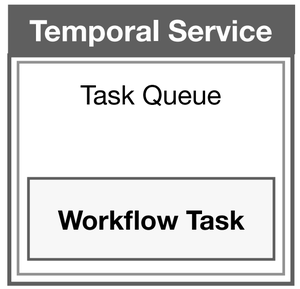
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

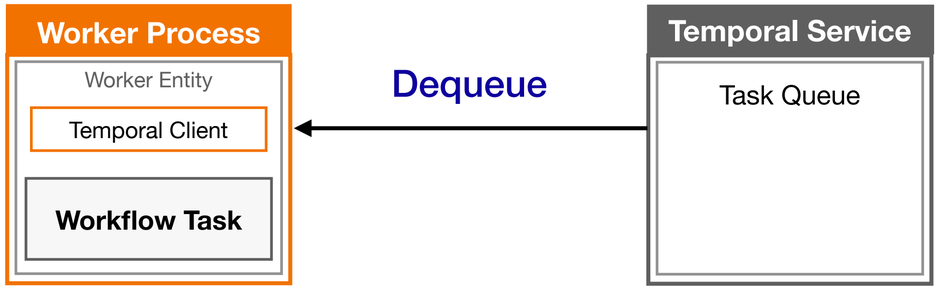
ScheduleActivityTask

Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

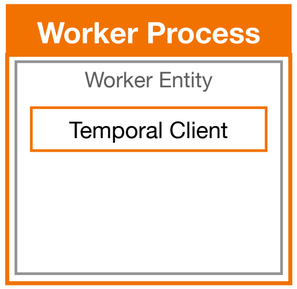
ScheduleActivityTask

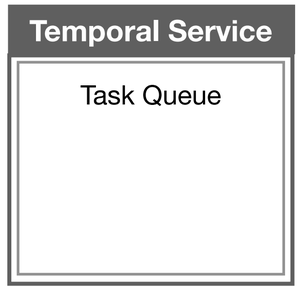
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

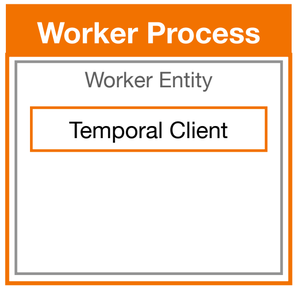
ScheduleActivityTask

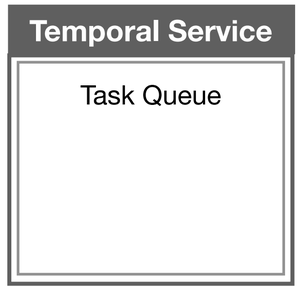
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

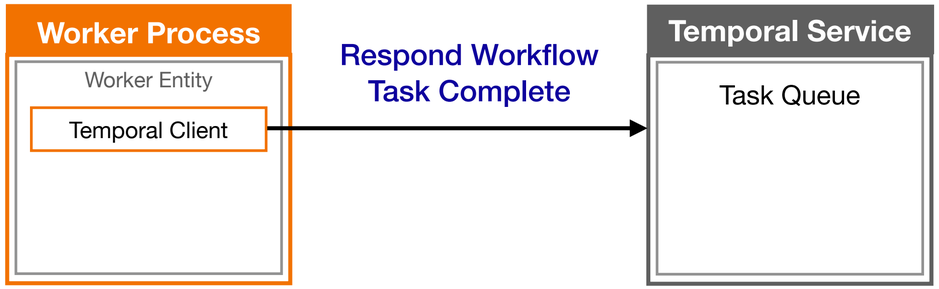
ScheduleActivityTask

Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

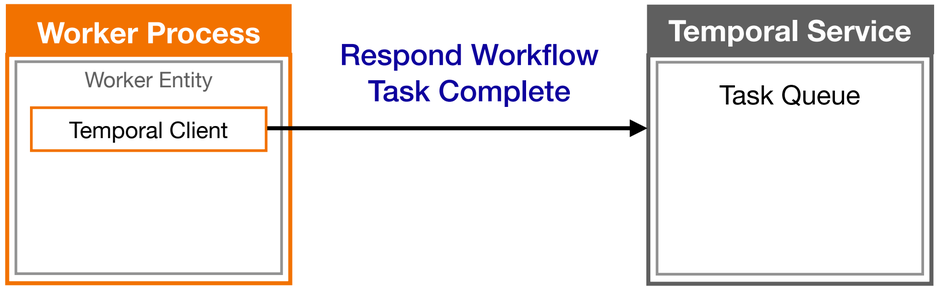
ScheduleActivityTask

Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

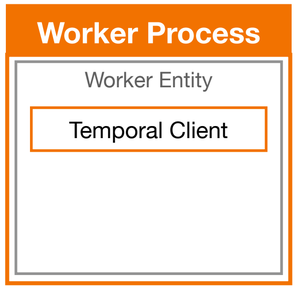
ScheduleActivityTask

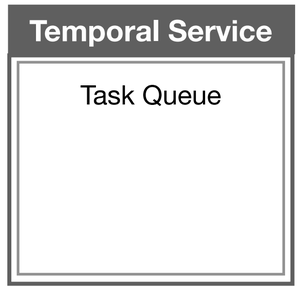
Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

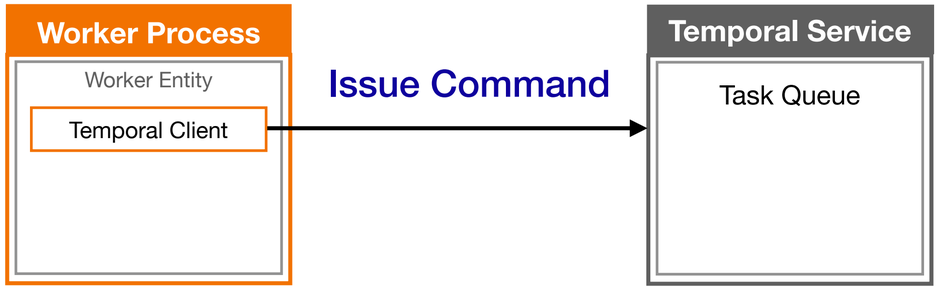
ScheduleActivityTask

Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```



Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

ScheduleActivityTask

Queue: pizza-tasks
Type: sendBill
Input: "customerID": 12983, ...

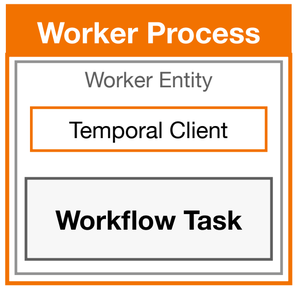
CompleteWorkflowExecution

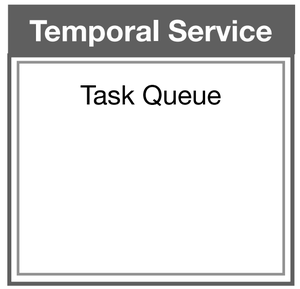
Result: "confirmationNumber": "TPD-26074139"

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

ScheduleActivityTask

Queue: pizza-tasks
Type: sendBill
Input: "CustomerID": 12983, ...

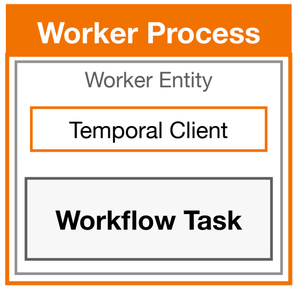
CompleteWorkflowExecution

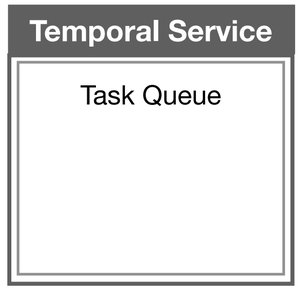
Result: "ConfirmationNumber": "TPD-26074139"

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	

```java
public class PizzaWorkflowImpl implements PizzaWorkflow {

  ActivityOptions options = ActivityOptions.newBuilder()
      .setStartToCloseTimeout(Duration.ofSeconds(5))
      .build();

  private final PizzaActivities activities =
      Workflow.newActivityStub(PizzaActivities.class, options);

  public static final Logger logger = Workflow.getLogger(PizzaWorkflowImpl.class);

  @Override
  public String pizzaWorkflow(Order order) {
    int totalPrice = 0;

    // Iterate over the items and calculate the cost of the order
    for (Pizza pizza : order.getItems()) {
        totalPrice += pizza.getPrice();
    }

    logger.info("Calculated cost of order: " + totalPrice);

    // Execute the getDistance activity
    int distance = activities.getDistance(order.getAddress());

    if (order.isDelivery() && distance > 25) {
      String message = "Customer lives outside the service area";
      throw ApplicationFailure.newFailure(message,
          OutOfServiceAreaException.class.getName());
    }

    // Wait for 30 minutes before billing the customer
    Workflow.sleep(Duration.ofMinutes(30));

    // Create a bill object
    Bill bill = new Bill();
    bill.setCustomerId(order.getCustomer().getCustomerId());
    bill.setAmount(totalPrice);
    bill.setDescription(order.getOrderNumber());

    // Execute the SendBill activity
    String confirmation = activities.sendBill(bill);

    return confirmation;
  }
}
```





Commands

ScheduleActivityTask

Queue: pizza-tasks
Type: getDistance
Input: "OrderNumber": "Z1238", ...

StartTimer

30 minutes

ScheduleActivityTask

Queue: pizza-tasks
Type: sendBill
Input: "CustomerID": 12983, ...

CompleteWorkflowExecution

Result: "ConfirmationNumber": "TPD-26074139"

Events

WorkflowExecutionStarted	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(getDistance)**
ActivityTaskStarted	
ActivityTaskCompleted	**(distance=15)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
TimerStarted	**(30 Minutes)**
TimerFired	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskTimedOut	
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
ActivityTaskScheduled	**(sendBill)**
ActivityTaskStarted	
ActivityTaskCompleted	**(confirmation=...)**
WorkflowTaskScheduled	
WorkflowTaskStarted	
WorkflowTaskCompleted	
WorkflowExecutionCompleted	

Why Temporal Requires Determinism for Workflows

Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      // Sleep for 4 hours
      Workflow.sleep(Duration.ofHours(4));

      String report = activities.runDailyReport(salesData);

    }
}
```

Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      // Sleep for 4 hours
      Workflow.sleep(Duration.ofHours(4));

      String report = activities.runDailyReport(salesData);

    }
}
```

Commands

Events

Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      // Sleep for 4 hours
      Workflow.sleep(Duration.ofHours(4));

      String report = activities.runDailyReport(salesData);

    }
}
```

Commands

ScheduleActivityTask

Type: importSalesData

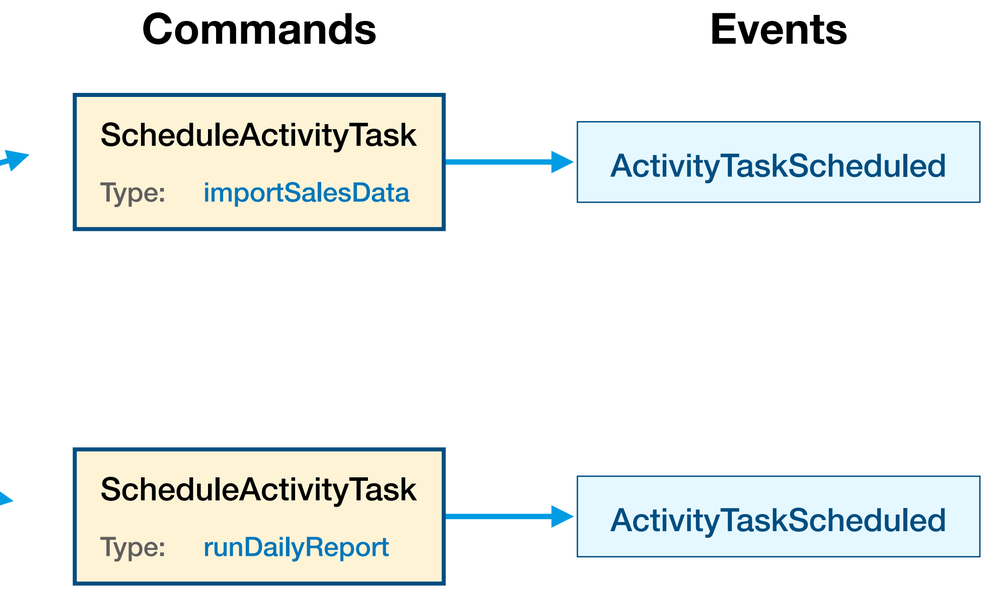
ScheduleActivityTask

Type: runDailyReport

Events

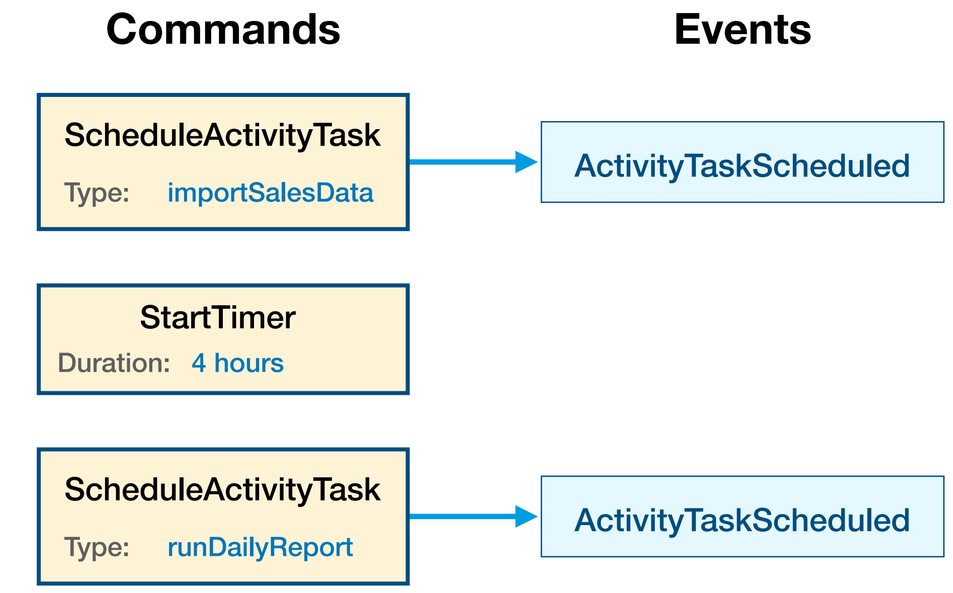
Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      // Sleep for 4 hours
      Workflow.sleep(Duration.ofHours(4));

      String report = activities.runDailyReport(salesData);

    }
}
```



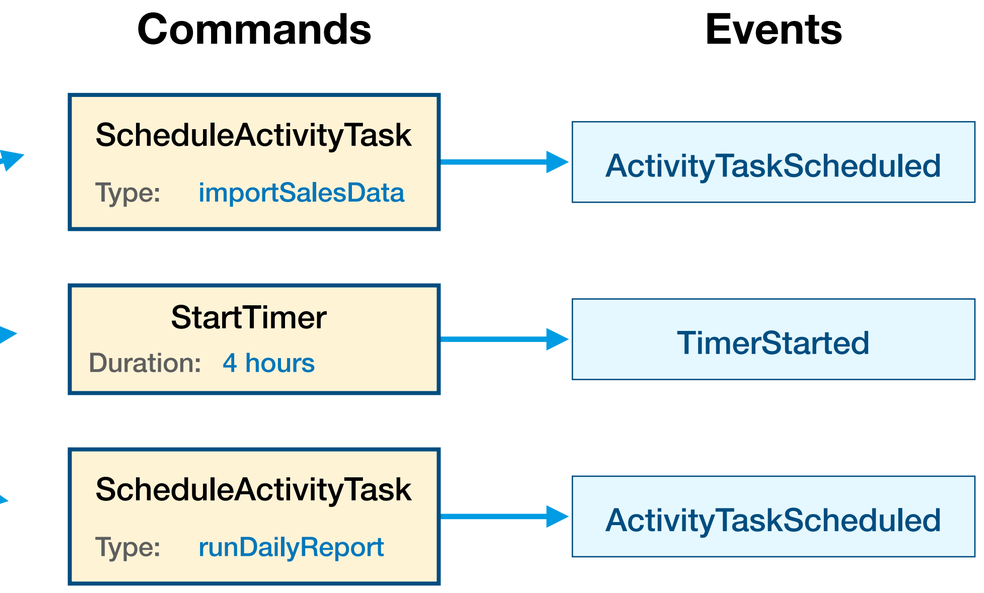
Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      // Sleep for 4 hours
      Workflow.sleep(Duration.ofHours(4));

      String report = activities.runDailyReport(salesData);

    }
}
```



Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      // Sleep for 4 hours
      Workflow.sleep(Duration.ofHours(4));

      String report = activities.runDailyReport(salesData);

    }
}
```



Commands

ScheduleActivityTask

Commands

ScheduleActivityTask

Events

ActivityTaskScheduled

Commands

ScheduleActivityTask

Events

ActivityTaskScheduled

ActivityTaskStarted

Commands

ScheduleActivityTask

Events

ActivityTaskScheduled

ActivityTaskStarted

ActivityTaskCompleted

Commands

ScheduleActivityTask

Events

ActivityTaskScheduled

ActivityTaskStarted

ActivityTaskCompleted

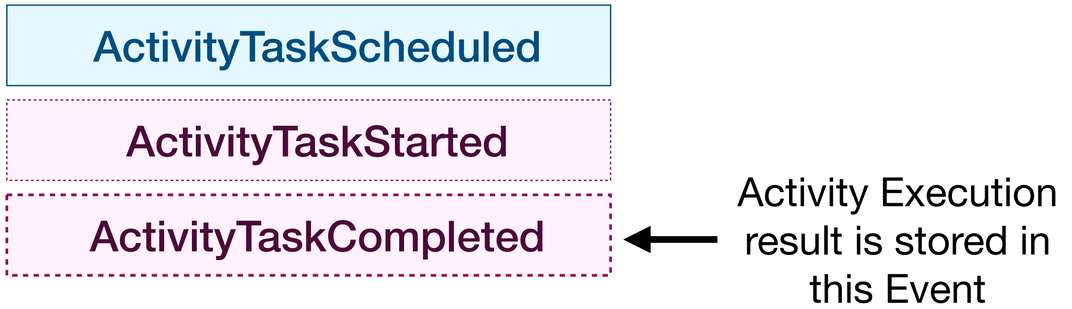
Activity Execution result is stored in this Event

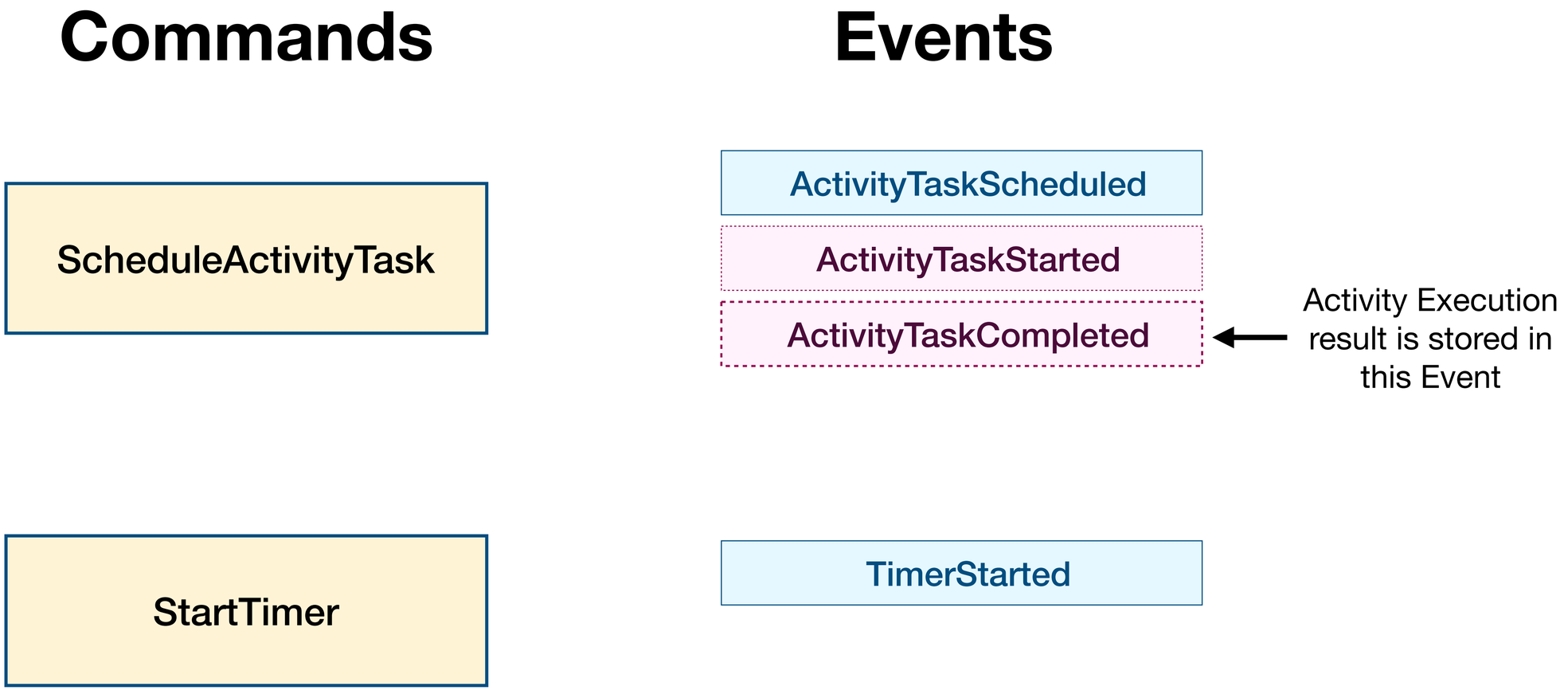
Commands

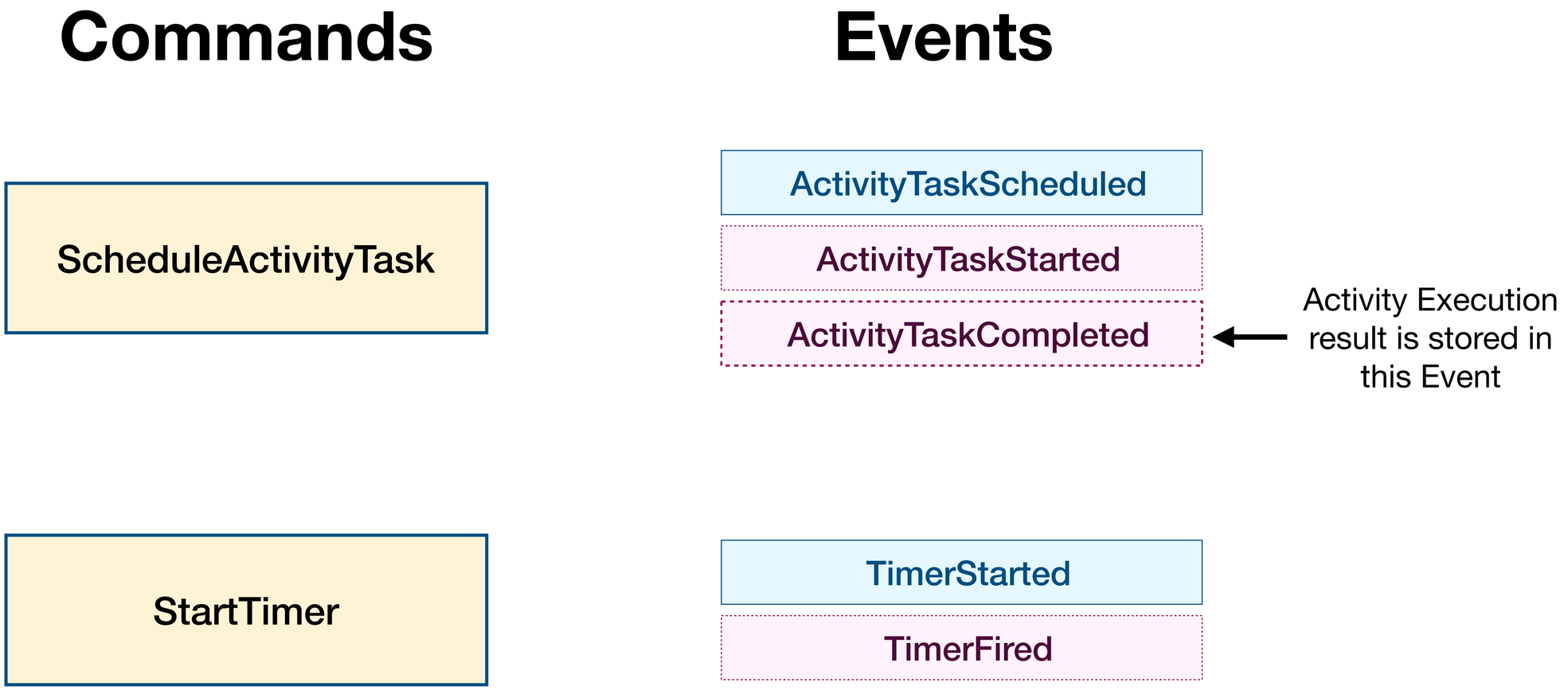
ScheduleActivityTask

StartTimer

Events




Commands
Events
ScheduleActivityTask
ActivityTaskScheduled
ActivityTaskStarted
ActivityTaskCompleted
Activity Execution result is stored in this Event
StartTimer
TimerStarted


Commands
Events
ScheduleActivityTask
ActivityTaskScheduled
ActivityTaskStarted
ActivityTaskCompleted
Activity Execution result is stored in this Event
StartTimer
TimerStarted
TimerFired

Deterministic Workflows:

Deterministic Workflows:

- **A Workflow is deterministic if every execution of its Workflow Definition:**

Deterministic Workflows:

- **A Workflow is deterministic if every execution of its Workflow Definition:**
  - **produces the same Commands**

Deterministic Workflows:

- **A Workflow is deterministic if every execution of its Workflow Definition:**
  - **produces the same Commands**
  - **in the same sequence**

Deterministic Workflows:

- **A Workflow is deterministic if every execution of its Workflow Definition:**
  - **produces the same Commands**
  - **in the same sequence**
  - **given the same input**

Deterministic Workflows:

- **A Workflow is deterministic if every execution of its Workflow Definition:**
  - **produces the same Commands**
  - **in the same sequence**
  - **given the same input**

Temporal's ability to guarantee durable execution of your Workflow depends on deterministic Workflows.

Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      // Sleep for 4 hours
      Workflow.sleep(Duration.ofHours(4));

      String report = activities.runDailyReport(salesData);

    }
}
```

Commands

Events

Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData()

      // Sleep for 4 hours
      Workflow.sleep(Duration.ofHours(4));

      String report = activities.runDailyReport(salesData);

    }
}
```

Commands

ScheduleActivityTask

Type: importSalesData

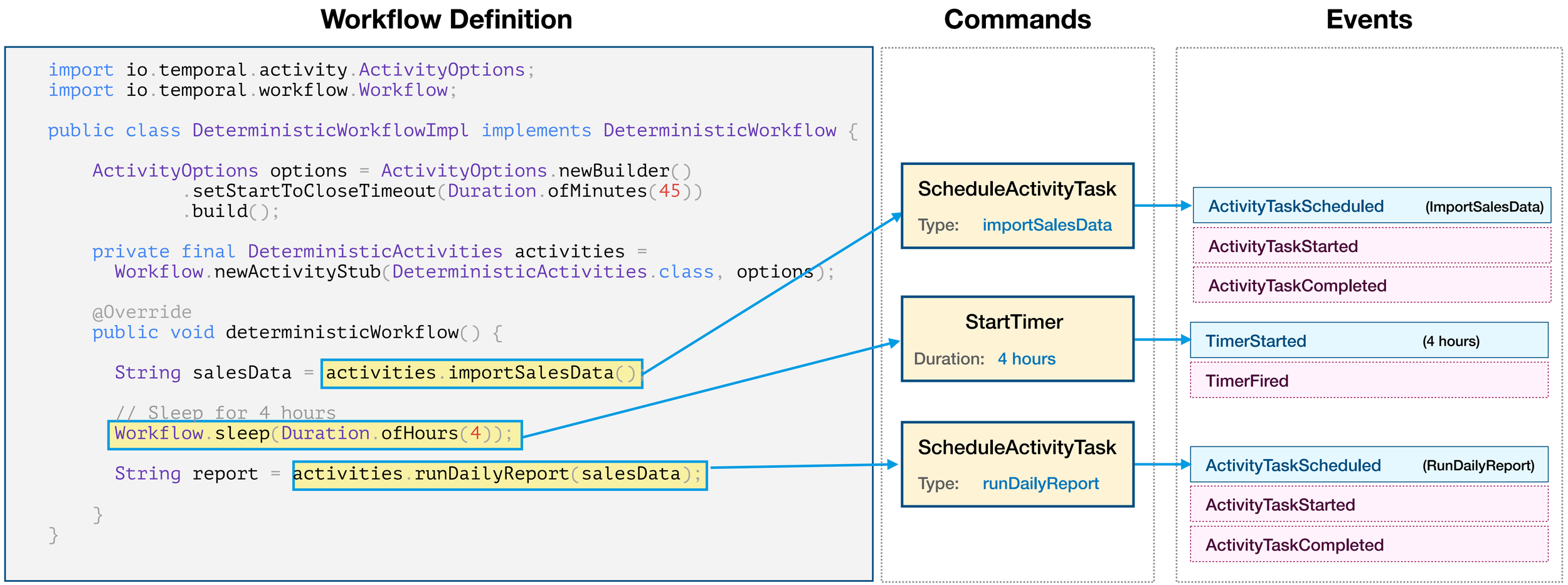
StartTimer

Duration: 4 hours

ScheduleActivityTask

Type: runDailyReport

Events


Workflow Definition
Commands
Events
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;
public class DeterministicWorkflowImpl implements DeterministicWorkflow {
ActivityOptions options = ActivityOptions.newBuilder()
.setStartToCloseTimeout(Duration.ofMinutes(45))
.build();
private final DeterministicActivities activities =
Workflow.newActivityStub(DeterministicActivities.class, options);
@Override
public void deterministicWorkflow() {
String salesData = activities.importSalesData()
// Sleep for 4 hours
Workflow.sleep(Duration.ofHours(4));
String report = activities.runDailyReport(salesData);
}
}
ScheduleActivityTask
Type: importSalesData
StartTimer
Duration: 4 hours
ScheduleActivityTask
Type: runDailyReport
ActivityTaskScheduled (ImportSalesData)
ActivityTaskStarted
ActivityTaskCompleted
TimerStarted (4 hours)
TimerFired
ActivityTaskScheduled (RunDailyReport)
ActivityTaskStarted
ActivityTaskCompleted

Commands Generated

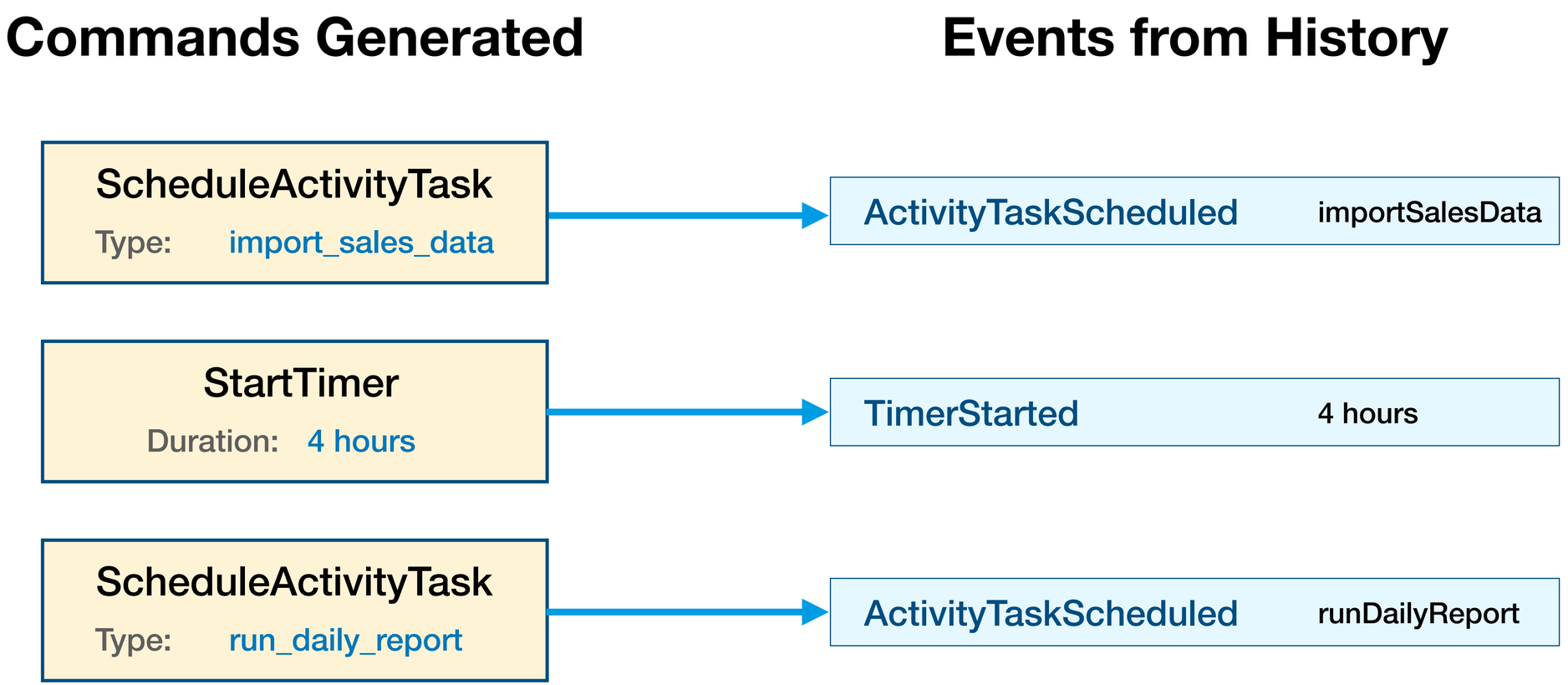
ScheduleActivityTask
Type: import_sales_data

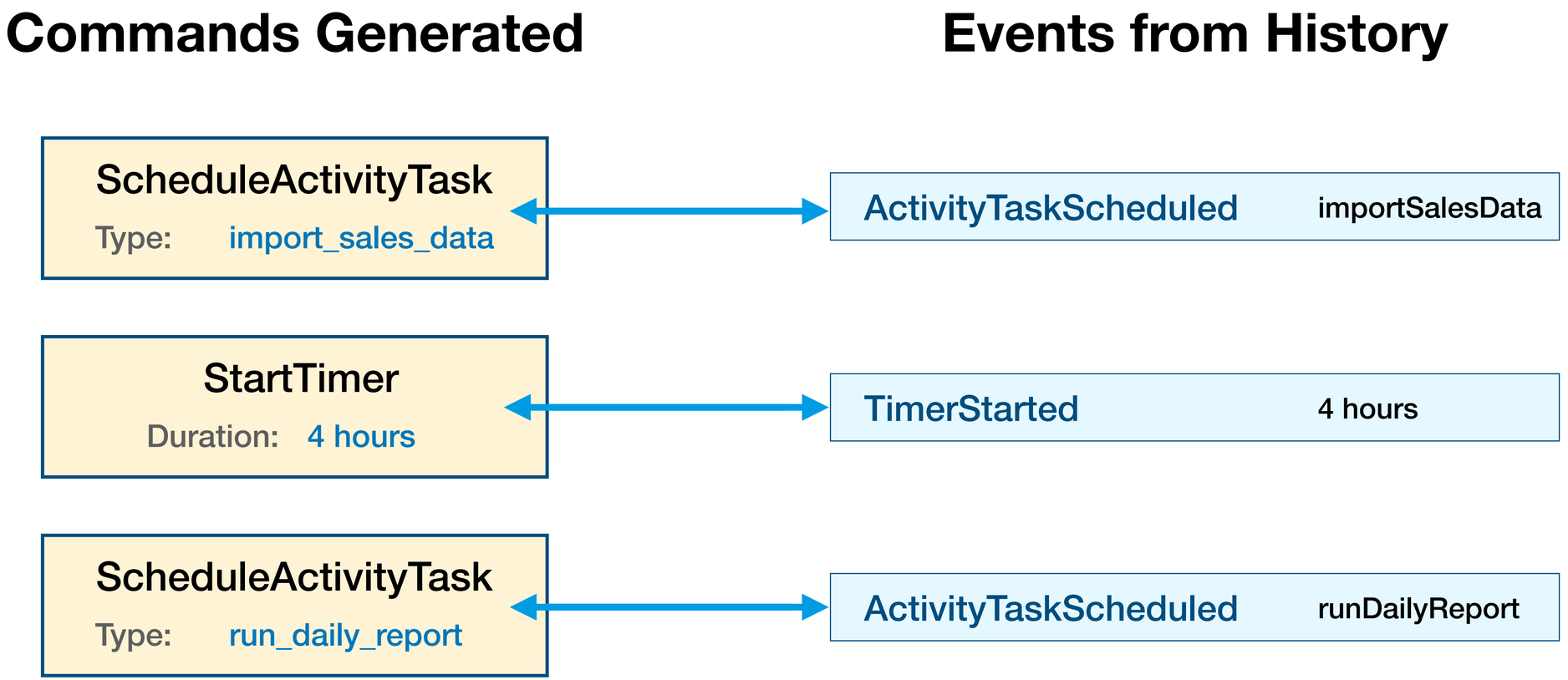
StartTimer
Duration: 4 hours

ScheduleActivityTask
Type: run_daily_report

Events from History

ActivityTaskScheduled	importSalesData
TimerStarted	4 hours
ActivityTaskScheduled	runDailyReport


Commands Generated
Events from History
ScheduleActivityTask
Type: import_sales_data
ActivityTaskScheduled
importSalesData
StartTimer
Duration: 4 hours
TimerStarted
4 hours
ScheduleActivityTask
Type: run_daily_report
ActivityTaskScheduled
runDailyReport


Commands Generated
Events from History
ScheduleActivityTask
Type: import_sales_data
ActivityTaskScheduled
importSalesData
StartTimer
Duration: 4 hours
TimerStarted
4 hours
ScheduleActivityTask
Type: run_daily_report
ActivityTaskScheduled
runDailyReport

Example of a Non-Deterministic Workflow

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

Relevant Events Logged

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

Relevant Events Logged

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: importSalesData

Relevant Events Logged

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: importSalesData

Relevant Events Logged

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: importSalesData

Relevant Events Logged

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Happens to return `84` during this execution

Commands Created

ScheduleActivityTask

Type: importSalesData

Relevant Events Logged

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: importSalesData

Relevant Events Logged

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: importSalesData

Relevant Events Logged

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: importSalesData

StartTimer

Duration: 4 hours

Relevant Events Logged

ActivityTaskScheduled (importSalesData)

ActivityTaskStarted

ActivityTaskCompleted

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: importSalesData

StartTimer

Duration: 4 hours

Relevant Events Logged

ActivityTaskScheduled (importSalesData)

ActivityTaskStarted

ActivityTaskCompleted

TimerStarted (4 hours)

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: importSalesData

StartTimer

Duration: 4 hours

Relevant Events Logged

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted

- TimerStarted (4 hours)
- TimerFired

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: importSalesData

StartTimer

Duration: 4 hours

Relevant Events Logged

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted

- TimerStarted (4 hours)
- TimerFired

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Worker crashes here

Commands Created

ScheduleActivityTask

Type: importSalesData

StartTimer

Duration: 4 hours

Relevant Events Logged

ActivityTaskScheduled (importSalesData)

ActivityTaskStarted

ActivityTaskCompleted

TimerStarted (4 hours)

TimerFired

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

Relevant History Events

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted
- TimerStarted (4 hours)
- TimerFired

Commands Expected (Based on History)

ScheduleActivityTask

Type: ImportSalesData

StartTimer

4 hours

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

Relevant History Events

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted
- TimerStarted (4 hours)
- TimerFired

Commands Expected (Based on History)

ScheduleActivityTask

Type: importSalesData

StartTimer

4 hours

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

Relevant History Events

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted
- TimerStarted (4 hours)
- TimerFired

Commands Expected (Based on History)

ScheduleActivityTask

Type: importSalesData

StartTimer

4 hours

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: ImportSalesData

Relevant History Events

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted
- TimerStarted (4 hours)
- TimerFired

Commands Expected (Based on History)

ScheduleActivityTask

Type: importSalesData

StartTimer

4 hours

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: ImportSalesData

Relevant History Events

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted
- TimerStarted (4 hours)
- TimerFired

Commands Expected (Based on History)

ScheduleActivityTask

Type: importSalesData

StartTimer

4 hours

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: importSalesData

Relevant History Events

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted
- TimerStarted (4 hours)
- TimerFired

Commands Expected (Based on History)

ScheduleActivityTask

Type: importSalesData

StartTimer

4 hours

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Happens to return 14 during this execution

Commands Created

ScheduleActivityTask

Type: importSalesData

Relevant History Events

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted
- TimerStarted (4 hours)
- TimerFired

Commands Expected (Based on History)

ScheduleActivityTask

Type: importSalesData

StartTimer

4 hours

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: importSalesData

Relevant History Events

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted
- TimerStarted (4 hours)
- TimerFired

Commands Expected (Based on History)

ScheduleActivityTask

Type: importSalesData

StartTimer

4 hours

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask

Type: importSalesData

Relevant History Events

- ActivityTaskScheduled (importSalesData)
- ActivityTaskStarted
- ActivityTaskCompleted
- TimerStarted (4 hours)
- TimerFired

Commands Expected (Based on History)

ScheduleActivityTask

Type: importSalesData

StartTimer

4 hours

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask
Type: importSalesData

ScheduleActivityTask
Type: runDailyReport

Relevant History Events

ActivityTaskScheduled (importSalesData)
ActivityTaskStarted
ActivityTaskCompleted

TimerStarted (4 hours)
TimerFired

Commands Expected (Based on History)

ScheduleActivityTask
Type: importSalesData

StartTimer
4 hours

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask
Type: importSalesData

ScheduleActivityTask
Type: runDailyReport

Relevant History Events

ActivityTaskScheduled (importSalesData)
ActivityTaskStarted
ActivityTaskCompleted

TimerStarted (4 hours)
TimerFired

Commands Expected (Based on History)

ScheduleActivityTask
Type: importSalesData

StartTimer
4 hours

A Non-Deterministic Workflow Definition

```
import io.temporal.activity.ActivityOptions;
import io.temporal.workflow.Workflow;

public class DeterministicWorkflowImpl implements DeterministicWorkflow {

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofMinutes(45))
            .build();

    private final DeterministicActivities activities =
      Workflow.newActivityStub(DeterministicActivities.class, options);

    @Override
    public void deterministicWorkflow() {

      String salesData = activities.importSalesData();

      Random random = new Random();
      if(random.nextInt(101) >= 50) {
        // Sleep for 4 hours
        Workflow.sleep(Duration.ofHours(4));
      }

      DailyReport report = new DailyReport();

      report = activities.runDailyReport(salesData);

      // remaining code omitted

    }
}
```

Commands Created

ScheduleActivityTask
Type: importSalesData

ScheduleActivityTask
Type: runDailyReport

Relevant History Events

ActivityTaskScheduled (importSalesData)
ActivityTaskStarted
ActivityTaskCompleted

TimerStarted (4 hours)
TimerFired

Commands Expected (Based on History)

ScheduleActivityTask
Type: importSalesData

StartTimer
4 hours

Using random numbers in a Workflow Definition has resulted in Non-Deterministic Error

Each time a particular Workflow Definition is executed with a given input, it must yield exactly the same commands in exactly the same order.

Common Sources of Non-Determinism

Things to Avoid in a Workflow Definition

Things to Avoid in a Workflow Definition

- **Accessing external systems, such as databases or network services**

Things to Avoid in a Workflow Definition

- **Accessing external systems, such as databases or network services**
  - Instead, use Activities to perform these operations

Things to Avoid in a Workflow Definition

- **Accessing external systems, such as databases or network services**
  - Instead, use Activities to perform these operations
- **Writing business logic or calling methods that rely on system time**

Things to Avoid in a Workflow Definition

- **Accessing external systems, such as databases or network services**
  - Instead, use Activities to perform these operations
- **Writing business logic or calling methods that rely on system time**
  - Instead, use Workflow-safe methods such as Workflow.currentTimeMillis and Workflow.sleep

Things to Avoid in a Workflow Definition

- **Accessing external systems, such as databases or network services**
  - Instead, use Activities to perform these operations
- **Writing business logic or calling methods that rely on system time**
  - Instead, use Workflow-safe methods such as Workflow.currentTimeMillis and Workflow.sleep
- **Working directly with threads**

Things to Avoid in a Workflow Definition

- **Accessing external systems, such as databases or network services**
  - Instead, use Activities to perform these operations
- **Writing business logic or calling methods that rely on system time**
  - Instead, use Workflow-safe methods such as Workflow.currentTimeMillis and Workflow.sleep
- **Working directly with threads**
- **Do not iterate over data structures with unknown ordering**

Things to Avoid in a Workflow Definition

- **Accessing external systems, such as databases or network services**
  - Instead, use Activities to perform these operations
- **Writing business logic or calling methods that rely on system time**
  - Instead, use Workflow-safe methods such as Workflow.currentTimeMillis and Workflow.sleep
- **Working directly with threads**
- **Do not iterate over data structures with unknown ordering**

How Workflow Changes Can Lead to Non-Deterministic Errors

Non-Deterministic *Code* Isn't the Only Danger

Non-Deterministic *Code* Isn't the Only Danger

- As you've just learned, non-deterministic code can cause problems

Non-Deterministic *Code* Isn't the Only Danger

- **As you've just learned, non-deterministic code can cause problems**
  - However, there's also another source of non-deterministic errors

Non-Deterministic *Code* Isn't the Only Danger

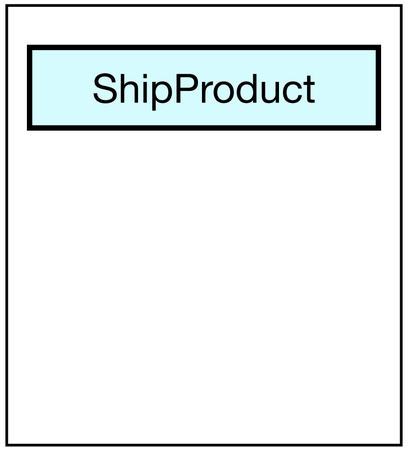
- **As you've just learned, non-deterministic code can cause problems**
  - However, there's also another source of non-deterministic errors
  - This is more subtle. Consider the following scenario

Non-Deterministic *Code* Isn't the Only Danger

- **As you've just learned, non-deterministic code can cause problems**
  - However, there's also another source of non-deterministic errors
  - This is more subtle. Consider the following scenario
    - You deploy and execute the following Workflow, which calls three Activities..

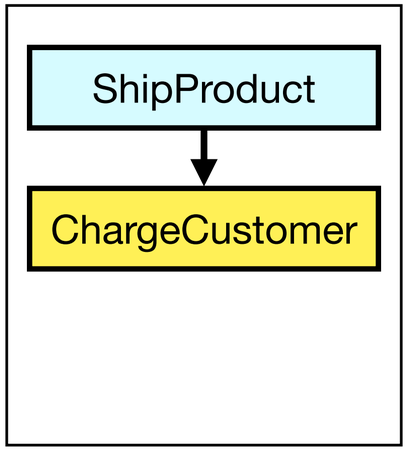
Non-Deterministic *Code* Isn't the Only Danger

- **As you've just learned, non-deterministic code can cause problems**
  - However, there's also another source of non-deterministic errors
  - This is more subtle. Consider the following scenario
    - You deploy and execute the following Workflow, which calls three Activities..



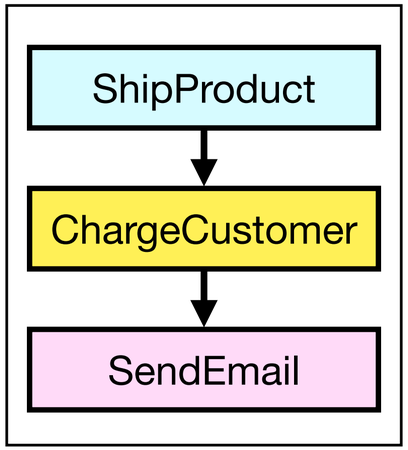
Non-Deterministic *Code* Isn't the Only Danger

- **As you've just learned, non-deterministic code can cause problems**
  - However, there's also another source of non-deterministic errors
  - This is more subtle. Consider the following scenario
    - You deploy and execute the following Workflow, which calls three Activities..



Non-Deterministic *Code* Isn't the Only Danger

- **As you've just learned, non-deterministic code can cause problems**
  - However, there's also another source of non-deterministic errors
  - This is more subtle. Consider the following scenario
    - You deploy and execute the following Workflow, which calls three Activities..



Deployment Leads to Non-Deterministic Error

Deployment Leads to Non-Deterministic Error

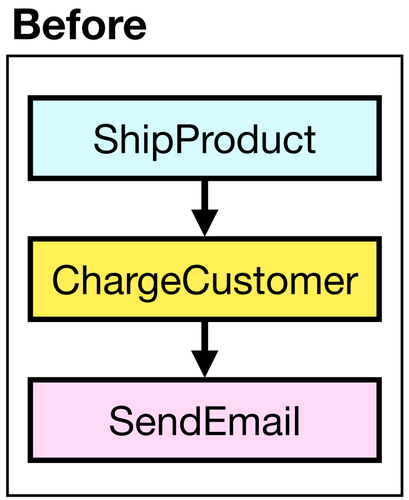
- **While that Workflow is running, you decide to update the code**

Deployment Leads to Non-Deterministic Error

- **While that Workflow is running, you decide to update the code**
  - You now want to charge the customer before shipping the product

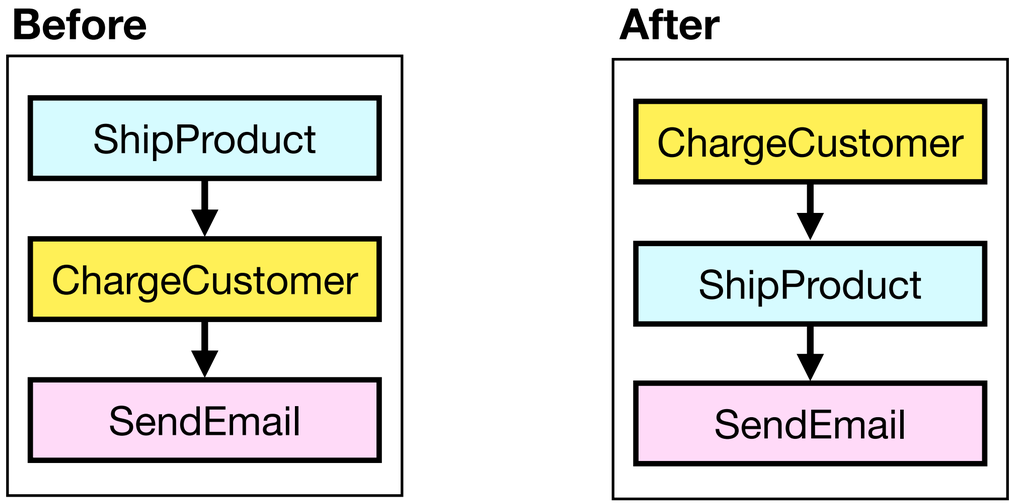
Deployment Leads to Non-Deterministic Error

- **While that Workflow is running, you decide to update the code**
  - You now want to charge the customer before shipping the product



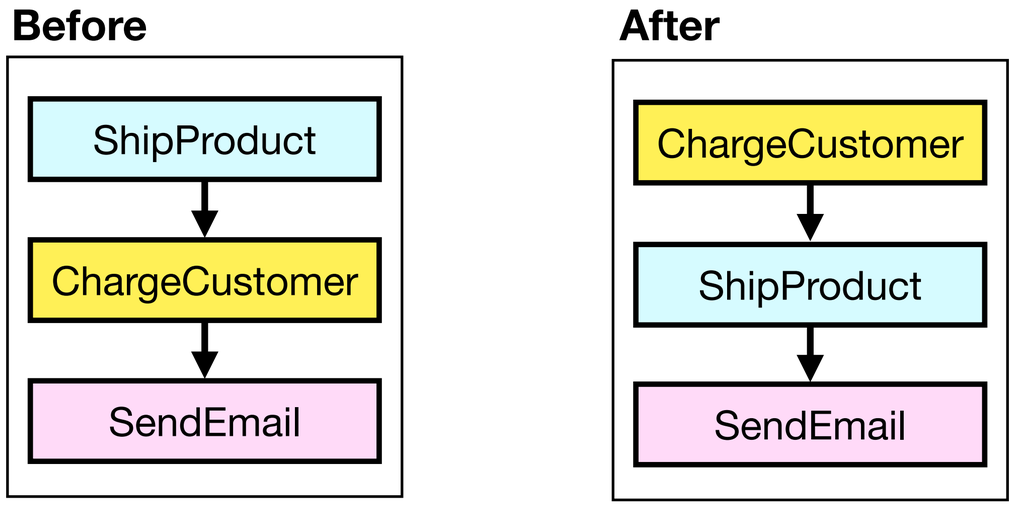
Deployment Leads to Non-Deterministic Error

- **While that Workflow is running, you decide to update the code**
  - You now want to charge the customer before shipping the product



Deployment Leads to Non-Deterministic Error

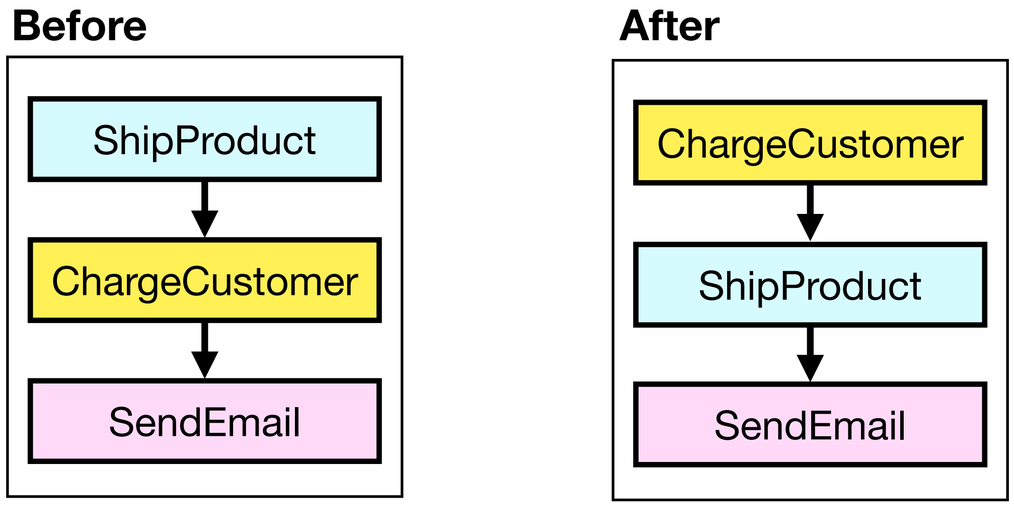
- **While that Workflow is running, you decide to update the code**
  - You now want to charge the customer before shipping the product



  - You deploy the updated code and restart the Worker(s) so that the change takes effect

Deployment Leads to Non-Deterministic Error

- **While that Workflow is running, you decide to update the code**
  - You now want to charge the customer before shipping the product



  - You deploy the updated code and restart the Worker(s) so that the change takes effect

- **What happens to the open execution when you restart the Worker?**

Deployment Leads to Non-Deterministic Error

Deployment Leads to Non-Deterministic Error

- **Problem: Worker cannot restore previous state with the updated code**

Deployment Leads to Non-Deterministic Error

- **Problem: Worker cannot restore previous state with the updated code**
  - Changes to your code updated the ordering of commands

Deployment Leads to Non-Deterministic Error

- **Problem: Worker cannot restore previous state with the updated code**
  - Changes to your code updated the ordering of commands
- **Only an issue if there are open executions at time of deployment**

Deployment Leads to Non-Deterministic Error

- **Problem: Worker cannot restore previous state with the updated code**
  - Changes to your code updated the ordering of commands
- **Only an issue if there are open executions at time of deployment**
- **How to detect?**

Deployment Leads to Non-Deterministic Error

- **Problem: Worker cannot restore previous state with the updated code**
  - Changes to your code updated the ordering of commands
- **Only an issue if there are open executions at time of deployment**
- **How to detect?**
  - Test changes by replaying history of previous executions using new code before deploying

Deployment Leads to Non-Deterministic Error

- **Problem: Worker cannot restore previous state with the updated code**
  - Changes to your code updated the ordering of commands
- **Only an issue if there are open executions at time of deployment**
- **How to detect?**
  - Test changes by replaying history of previous executions using new code before deploying
- **How to prevent?**

Deployment Leads to Non-Deterministic Error

- **Problem: Worker cannot restore previous state with the updated code**
  - Changes to your code updated the ordering of commands
- **Only an issue if there are open executions at time of deployment**
- **How to detect?**
  - Test changes by replaying history of previous executions using new code before deploying
- **How to prevent?**
  - Versioning (see documentation for details)

Deployment Leads to Non-Deterministic Error

- **Problem: Worker cannot restore previous state with the updated code**
  - Changes to your code updated the ordering of commands
- **Only an issue if there are open executions at time of deployment**
- **How to detect?**
  - Test changes by replaying history of previous executions using new code before deploying
- **How to prevent?**
  - Versioning (see documentation for details)
- **How to remediate?**

Deployment Leads to Non-Deterministic Error

- **Problem: Worker cannot restore previous state with the updated code**
  - Changes to your code updated the ordering of commands
- **Only an issue if there are open executions at time of deployment**
- **How to detect?**
  - Test changes by replaying history of previous executions using new code before deploying
- **How to prevent?**
  - Versioning (see documentation for details)
- **How to remediate?**
  - Use Workflow Reset to restart execution to a point before the change was introduced

Deployment Leads to Non-Deterministic Error

- **Problem: Worker cannot restore previous state with the updated code**
  - Changes to your code updated the ordering of commands
- **Only an issue if there are open executions at time of deployment**
- **How to detect?**
  - Test changes by replaying history of previous executions using new code before deploying
- **How to prevent?**
  - Versioning (see documentation for details)
- **How to remediate?**
  - Use Workflow Reset to restart execution to a point before the change was introduced
  - Not always desirable, as any progress made in Activities after the reset point will be lost and reexecuted

Beginner to Builder Bootcamp - Java

00. About this Workshop
01. The Basics of Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
04. Understanding Event History
05. Understanding Workflow Determinism
06. Signals, Queries, and Workflow Updates
07. Timeouts and Retry Policies
08. Recovering from Failure
09. Conclusion

Signaling Your Workflows

What are Signals?

What are Signals?

- Asynchronous message sent to a running Workflow Execution which can be used to change the Workflow's state and control its execution flow

What are Signals?

- Asynchronous message sent to a running Workflow Execution which can be used to change the Workflow's state and control its execution flow
- Useful for Workflows that must react to external events

What are Signals?

- Asynchronous message sent to a running Workflow Execution which can be used to change the Workflow's state and control its execution flow
- Useful for Workflows that must react to external events
- Examples:

What are Signals?

- Asynchronous message sent to a running Workflow Execution which can be used to change the Workflow's state and control its execution flow
- Useful for Workflows that must react to external events
- Examples:
  - User interactions in web applications

What are Signals?

- Asynchronous message sent to a running Workflow Execution which can be used to change the Workflow's state and control its execution flow
- Useful for Workflows that must react to external events
- Examples:
  - User interactions in web applications
  - Processing events

What are Signals?

- Asynchronous message sent to a running Workflow Execution which can be used to change the Workflow's state and control its execution flow
- Useful for Workflows that must react to external events
- Examples:
  - User interactions in web applications
  - Processing events
  - Frequently changing data interactions

Developing Signals

Developing Signals

- **There are two steps to adding support for a Signal in your Workflow code**

Developing Signals

- There are two steps to adding support for a Signal in your Workflow code
- Defining the Signal

Developing Signals

- **There are two steps to adding support for a Signal in your Workflow code**
- **Defining the Signal**
  - Specify the name and data structure that Temporal Clients will use to send the Signal

Developing Signals

- **There are two steps to adding support for a Signal in your Workflow code**
- **Defining the Signal**
  - Specify the name and data structure that Temporal Clients will use to send the Signal
- **Handling the Signal**

Developing Signals

- **There are two steps to adding support for a Signal in your Workflow code**
- **Defining the Signal**
  - Specify the name and data structure that Temporal Clients will use to send the Signal
- **Handling the Signal**
  - Write code that will be invoked upon receiving the Signal from a Temporal Client

Defining Signals

Defining Signals

- **Defined in Workflow Interface and handled in Workflow implementation**

Defining Signals

- **Defined in Workflow Interface and handled in Workflow implementation**
- **Each Signal is uniquely identified by a Signal type**

```
@WorkflowInterface
public interface MyWorkflow {

  @WorkflowMethod
  String workflow(String input);

  @SignalMethod
  void mySignal(String data);

}
```

Defining Signals

- **Defined in Workflow Interface and handled in Workflow implementation**
- **Each Signal is uniquely identified by a Signal type**
  - Case-sensitive string value specified by the Client when sending the Signal

```
@WorkflowInterface
public interface MyWorkflow {

  @WorkflowMethod
  String workflow(String input);

  @SignalMethod
  void mySignal(String data);

}
```

Defining Signals

- **Defined in Workflow Interface and handled in Workflow implementation**
- **Each Signal is uniquely identified by a Signal type**
  - Case-sensitive string value specified by the Client when sending the Signal
  - The Signal Type is the name of the Signal Method

```
@WorkflowInterface
public interface MyWorkflow {

  @WorkflowMethod
  String workflow(String input);

  @SignalMethod
  void mySignal(String data);

}
```

Defining Signals

- **Defined in Workflow Interface and handled in Workflow implementation**
- **Each Signal is uniquely identified by a Signal type**
  - Case-sensitive string value specified by the Client when sending the Signal
  - The Signal Type is the name of the Signal Method
  - Signals can include a list of arguments (these must be Serializable)

```
@WorkflowInterface
public interface MyWorkflow {

  @WorkflowMethod
  String workflow(String input);

  @SignalMethod
  void mySignal(String data);

}
```

Defining Signals

```
public class UserInfo {
  private String userId;
  private String groupId;
}


@WorkflowInterface
public interface MyWorkflow {

  @WorkflowMethod
  String workflow(String input);

  @SignalMethod
  void joinSignal(UserInfo user);
}
```

Handling Signals

```
public class MyWorkflowImpl implements MyWorkflow {
  private String signalData;

  @Override
  public void joinSignal(String data) {
    this.signalData = data;
  }
}
```

Handling Signals

- We need to let the Workflow know what to do when it receives a Signal of that type

```
public class MyWorkflowImpl implements MyWorkflow {
  private String signalData;

  @Override
  public void joinSignal(String data) {
    this.signalData = data;
  }
}
```

Handling Signals

- We need to let the Workflow know what to do when it receives a Signal of that type
- Signal handlers are methods defined in the Workflow that listen for incoming Signals of a given type

```
public class MyWorkflowImpl implements MyWorkflow {
  private String signalData;

  @Override
  public void joinSignal(String data) {
    this.signalData = data;
  }
}
```

Handling Signals

- We need to let the Workflow know what to do when it receives a Signal of that type
- Signal handlers are methods defined in the Workflow that listen for incoming Signals of a given type
- Define your handler by implementing the Signal Method as defined in the Interface

```
public class MyWorkflowImpl implements MyWorkflow {
  private String signalData;

  @Override
  public void joinSignal(String data) {
    this.signalData = data;
  }
}
```

Awaiting within a Workflow

Awaiting within a Workflow

- **It is common for the Workflow to await some condition**

Awaiting within a Workflow

- **It is common for the Workflow to await some condition**
  - For example, an approval or confirmation (human-in-the-loop)

Awaiting within a Workflow

- **It is common for the Workflow to await some condition**
  - For example, an approval or confirmation (human-in-the-loop)
- **You can implement this by using Workflow.await()**

Awaiting within a Workflow

- **It is common for the Workflow to await some condition**
  - For example, an approval or confirmation (human-in-the-loop)
- **You can implement this by using Workflow.await()**
  - This pauses execution until the expression supplied to this function resolves to true

Awaiting within a Workflow

- **It is common for the Workflow to await some condition**
  - For example, an approval or confirmation (human-in-the-loop)
- **You can implement this by using Workflow.await()**
  - This pauses execution until the expression supplied to this function resolves to true
  - Once it does resolve to true, execution continues with the next statement

Awaiting within a Workflow

- **It is common for the Workflow to await some condition**
  - For example, an approval or confirmation (human-in-the-loop)
- **You can implement this by using Workflow.await()**
  - This pauses execution until the expression supplied to this function resolves to true
  - Once it does resolve to true, execution continues with the next statement
  - `Workflow.await(Duration.ofDays(30, this.subCancelled)`

Awaiting within a Workflow

- **It is common for the Workflow to await some condition**
  - For example, an approval or confirmation (human-in-the-loop)
- **You can implement this by using Workflow.await()**
  - This pauses execution until the expression supplied to this function resolves to true
  - Once it does resolve to true, execution continues with the next statement
  - `Workflow.await(Duration.ofDays(30, this.subCancelled)`
    - It allows you to limit the wait time by passing a maximum duration as a parameter

Awaiting within a Workflow

- **It is common for the Workflow to await some condition**
  - For example, an approval or confirmation (human-in-the-loop)
- **You can implement this by using Workflow.await()**
  - This pauses execution until the expression supplied to this function resolves to true
  - Once it does resolve to true, execution continues with the next statement
  - `Workflow.await(Duration.ofDays(30, this.subCancelled)`
    - It allows you to limit the wait time by passing a maximum duration as a parameter
    - If this duration is exceeded, the method unblocks and returns false

Waiting for Signals

```
import io.temporal.workflow.Workflow;

boolean continue = Workflow.await(Duration.ofDays(30), this.subCancelled);

if (!continue){
  activities.sendCancelEmail();
}
```

How to Send Signals

- The WorkflowStub is used to obtain a reference, or a "handle", to a specific Workflow Execution

How to Send Signals from Client

```
public class SignalClient {
  public static void main(String[] args) throws Exception {

    WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();

    WorkflowClient client = WorkflowClient.newInstance(service);

    SignalWorkflow handle = client.newWorkflowStub(SignalWorkflow.class, "workflow-id-123");

    handle.joinSignal("user-1", "group-1");

    System.exit(0);
  }
}
```

How to Send Signals from Client

```
public class SignalClient {
  public static void main(String[] args) throws Exception {

    WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();

    WorkflowClient client = WorkflowClient.newInstance(service);

    SignalWorkflow handle = client.newWorkflowStub(SignalWorkflow.class, "workflow-id-123");

    handle.joinSignal("user-1", "group-1");

    System.exit(0);
  }
}
```

Exercise #3: Sending Signals from the Client

- **During this exercise, you will**
  - Define and handle a Signal
  - Retrieve a handle on the Workflow to Signal
  - Send a Signal from the Client
- **Refer to the README.md file in the exercise environment for details**
  - The code is below the **exercises/sending-signals-client**
    - Make your changes to the code in the **practice** subdirectory (look for TODO comments)
    - If you need a hint or want to verify your changes, look at the complete version in the **solution** subdirectory

How to Send Signals from Within a Workflow

How to Send Signals from Within a Workflow

- A signal sent from one Workflow to another is called an *External Signal*

How to Send Signals from Within a Workflow

- A signal sent from one Workflow to another is called an *External Signal*
- Use newExternalWorkflowStub to send an External Signal

How to Send Signals from Within a Workflow

- A signal sent from one Workflow to another is called an *External Signal*
- Use newExternalWorkflowStub to send an External Signal

```
public String signalingWorkflow() {

  MyWorkflow workflow =
      Workflow.newExternalWorkflowStub(MyWorkflow.class, "workflow-id-123");

  workflow.joinSignal("user-1", "group-1");
  return "Success";
}
```

How to Send Signals from Within a Workflow

- A signal sent from one Workflow to another is called an *External Signal*
- Use newExternalWorkflowStub to send an External Signal

```
public String signalingWorkflow() {

  MyWorkflow workflow =
      Workflow.newExternalWorkflowStub(MyWorkflow.class, "workflow-id-123");

  workflow.joinSignal("user-1", "group-1");
  return "Success";
}
```

- This method ensures that the Workflow remains deterministic

Signal-With-Start

Signal-With-Start

- **Signals a Workflow Execution by Workflow ID, starting it if necessary**

```
import io.temporal.client.BatchRequest;

...

public static void signalWithStart() {

  // boilerplate creating Client omitted for brevity

  MyWorkflow workflow = client.newWorkflowStub(MyWorkflow.class, options);

  BatchRequest request = client.newSignalWithStartRequest();
  request.add(workflow::joinSignal, "user-1", "group-1");
```

Signal-With-Start

- **Signals a Workflow Execution by Workflow ID, starting it if necessary**
  - If there is currently an open Workflow Execution with this Workflow ID, it will be signaled

```
import io.temporal.client.BatchRequest;

...

public static void signalWithStart() {

  // boilerplate creating Client omitted for brevity

  MyWorkflow workflow = client.newWorkflowStub(MyWorkflow.class, options);

  BatchRequest request = client.newSignalWithStartRequest();
  request.add(workflow::joinSignal, "user-1", "group-1");
```

Signal-With-Start

- **Signals a Workflow Execution by Workflow ID, starting it if necessary**
  - If there is currently an open Workflow Execution with this Workflow ID, it will be signaled
  - If not, one is started and the Signal immediately delivered to it

```
import io.temporal.client.BatchRequest;

...

public static void signalWithStart() {

  // boilerplate creating Client omitted for brevity

  MyWorkflow workflow = client.newWorkflowStub(MyWorkflow.class, options);

  BatchRequest request = client.newSignalWithStartRequest();
  request.add(workflow::joinSignal, "user-1", "group-1");
```

Signal Limits

Signal Limits

- **There is a limit of 2,000 pending Signals sent per Workflow Execution**

Signal Limits

- **There is a limit of 2,000 pending Signals sent per Workflow Execution**
  - Sending Signals in batches smaller than 2,000 can help you to stay within this limit

Signal Limits

- **There is a limit of 2,000 pending Signals sent per Workflow Execution**
  - Sending Signals in batches smaller than 2,000 can help you to stay within this limit
  - The Continue-As-New feature may also be helpful here

Signal Limits

- **There is a limit of 2,000 pending Signals sent per Workflow Execution**
  - Sending Signals in batches smaller than 2,000 can help you to stay within this limit
  - The Continue-As-New feature may also be helpful here
- **The Temporal Service also limits the total number of Signals *received***

Signal Limits

- **There is a limit of 2,000 pending Signals sent per Workflow Execution**
  - Sending Signals in batches smaller than 2,000 can help you to stay within this limit
  - The Continue-As-New feature may also be helpful here
- **The Temporal Service also limits the total number of Signals *received***
  - Each Workflow Execution is limited to receiving 10,000 Signals during its entire lifespan

Unhandled Command

Unhandled Command

- **This error may occur when a Workflow receives a high volume of Signals in a short period of time**

Unhandled Command

- **This error may occur when a Workflow receives a high volume of Signals in a short period of time**
  - It may be unable to process them all before attempting to complete or transition to another state.

Unhandled Command

- **This error may occur when a Workflow receives a high volume of Signals in a short period of time**
  - It may be unable to process them all before attempting to complete or transition to another state.
  - Acceptable to ignore in Java SDK

Unhandled Command

- **This error may occur when a Workflow receives a high volume of Signals in a short period of time**
  - It may be unable to process them all before attempting to complete or transition to another state.
  - Acceptable to ignore in Java SDK
  - Potentially performance degradation

Optional Exercise #4: Sending an External Signal

- **This is an optional exercise**
  - We will do this together if time allows; otherwise, you may do it on your own later
- **During this exercise, you will**
  - Define and handle a Signal
  - Retrieve a handle on the Workflow to Signal
  - Send an External Signal
  - Use a Temporal Client to submit execution requests for both Workflows
- **Refer to the README.md file in the exercise environment for details**
  - The code is below the **exercises/sending-signals-external**
    - Make your changes to the code in the **practice** subdirectory (look for TODO comments)
    - If you need a hint or want to verify your changes, look at the complete version in the **solution** subdirectory

Querying Your Workflows

What are Queries?

What are Queries?

- **Used to retrieve the state during or even after a Workflow Execution**

What are Queries?

- **Used to retrieve the state during or even after a Workflow Execution**
- **Examples of when you would use Queries :**

What are Queries?

- **Used to retrieve the state during or even after a Workflow Execution**
- **Examples of when you would use Queries** :
  - Monitoring progress of long-running Workflows

What are Queries?

- **Used to retrieve the state during or even after a Workflow Execution**
- **Examples of when you would use Queries** :
  - Monitoring progress of long-running Workflows
  - Retrieving results

Query Attributes

Query Attributes

- Do not mutate state of Workflow Execution

Query Attributes

- Do not mutate state of Workflow Execution
- Do not execute any Workflow command

Query Attributes

- Do not mutate state of Workflow Execution
- Do not execute any Workflow command
- Queries are read-only and must complete synchronously

Query Attributes

- Do not mutate state of Workflow Execution
- Do not execute any Workflow command
- Queries are read-only and must complete synchronously
- Possible to send to a closed Workflow Execution

Query Attributes

- Do not mutate state of Workflow Execution
- Do not execute any Workflow command
- Queries are read-only and must complete synchronously
- Possible to send to a closed Workflow Execution
- Queries will return the most recent state

Query Attributes

- Do not mutate state of Workflow Execution
- Do not execute any Workflow command
- Queries are read-only and must complete synchronously
- Possible to send to a closed Workflow Execution
- Queries will return the most recent state
- A single Workflow can support multiple Queries

Developing Queries pt. 1

1. Defining the Query Type (name)

```
import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.WorkflowMethod;
import io.temporal.workflow.QueryMethod;

@WorkflowInterface
public interface MyWorkflow {

  @WorkflowMethod
  String workflow(String input);

  @QueryMethod
  String queryStatus();
}
```

Developing Queries pt. 2

2. Handling the Query

```
public class MyWorkflowImpl implements MyWorkflow {

  private String status;

  // Workflow code omitted for brevity

  public String queryStatus() {
    return this.status;
  }

}
```

Defining a Query

Defining a Query

- Query Type: A unique name used to identify the query

Defining a Query

- Query Type: A unique name used to identify the query
- Arguments must be serializable

Defining a Query

- Query Type: A unique name used to identify the query
- Arguments must be serializable
- Use the @QueryMethod annotation

```
import io.temporal.workflow.WorkflowInterface;
import io.temporal.workflow.QueryMethod;

@WorkflowInterface
public interface MyWorkflow {

  // Workflow Definition omitted for brevity

  @QueryMethod
  String queryStatus();
}
```

Handling a Query

Handling a Query

- Query handlers: specify what Workflow should do upon receiving a Query

Handling a Query

- Query handlers: specify what Workflow should do upon receiving a Query
- Define the Query Method to handle Queries in a Workflow

```
public class MyWorkflowImpl implements MyWorkflow {

  private String status;

  // Workflow code omitted for brevity

  @Override
  public String queryStatus() {
    return this.status;
  }
}
```

Sending a Query with the SDK

- Use Workflow Stub to invoke the Workflow's Query method
  - You can do this to query both open and closed Workflow Executions

```
public class QueryClient {
  public static void main(String[] args) throws Exception {

    WorkflowServiceStubs service = WorkflowServiceStubs.newLocalServiceStubs();

    WorkflowClient client = WorkflowClient.newInstance(service);

    MyWorkflow workflow = client.newWorkflowStub(MyWorkflow.class, "state-id-0");

    System.out.println(workflow.queryStatus());

    System.exit(0);
  }
```

Sending a Query with the CLI

```
$ temporal workflow query \
    --workflow-id="state-id-0" \
    --type="getValueQuery" \
    --input=\"meaning-of-life\"
```

Exercise #5: Querying Workflows

- **During this exercise, you will**
  - Define and handle a Query
  - Create a handle on a Workflow to be queried
  - Call the Query from the Client
  - Send a Query from the Command Line
- **Refer to the README.md file in the exercise environment for details**
  - The code is below the **exercises/querying-workflows**
    - Make your changes to the code in the **practice** subdirectory (look for TODO comments)
    - If you need a hint or want to verify your changes, look at the complete version in the **solution** subdirectory

Workflow Updates

What are Updates?

What are Updates?

- **An update is a trackable, synchronous request to a Workflow Execution**

What are Updates?

- **An update is a trackable, synchronous request to a Workflow Execution**
  - Comparable to sending a Signal and a Query at the same time

What are Updates?

- **An update is a trackable, synchronous request to a Workflow Execution**
  - Comparable to sending a Signal and a Query at the same time
- **Unlike Signals, sender *must* wait until Worker processes the Update**

What are Updates?

- **An update is a trackable, synchronous request to a Workflow Execution**
  - Comparable to sending a Signal and a Query at the same time
- **Unlike Signals, sender *must* wait until Worker processes the Update**
  - The sender *may* wait further to receive a returned value

What are Updates?

- **An update is a trackable, synchronous request to a Workflow Execution**
  - Comparable to sending a Signal and a Query at the same time
- **Unlike Signals, sender *must* wait until Worker processes the Update**
  - The sender *may* wait further to receive a returned value
  - If you need an asynchronous response, use a Signal instead

What are Updates?

- **An update is a trackable, synchronous request to a Workflow Execution**
  - Comparable to sending a Signal and a Query at the same time
- **Unlike Signals, sender *must* wait until Worker processes the Update**
  - The sender *may* wait further to receive a returned value
  - If you need an asynchronous response, use a Signal instead
- **A Workflow cannot send an Update to another Workflow**

Update Handlers

- **Defining an Update Handlers is similar to a Signal and Query Handler**

```
public class GreetingWorkflowImpl implements GreetingWorkflow {

    private Language language = Language.ENGLISH;

    @Override
    public String greet() {
        // ... workflow logic
    }

    @Override
    public Language setLanguage(Language newLanguage) {
        Language previousLanguage = this.language;
        this.language = newLanguage;
        return previousLanguage;
    }
}
```

Sending Updates via the Java API

- **Use the Temporal Client's UpdateWorkflow method to send an update**
  - This returns a handle, which you can use to retrieve the Update's result

```
WorkflowStub workflowStub = client.newUntypedWorkflowStub(
    we.getWorkflowId(),
    Optional.of(we.getFirstExecutionRunId()),
    Optional.empty()
);

UpdateOptions<Language> updateOptions = UpdateOptions.<Language>newBuilder(Language.class)
    .setUpdateName(GreetingWorkflow.SET_LANGUAGE_UPDATE)
    .setWaitForStage(WorkflowUpdateStage.ACCEPTED)
    .build();

WorkflowUpdateHandle<Language> updateHandle = workflowStub.startUpdate(
    updateOptions,
    Language.CHINESE
);

Language previousLang = updateHandle.getResultAsync().get(15, TimeUnit.SECONDS);
```

Update Validators

Update Validators

- **Validators can be used to reject an Update before it is written to history**

Update Validators

- **Validators can be used to reject an Update before it is written to history**
  - To reject an Update, return an error in the Validator

Update Validators

- **Validators can be used to reject an Update before it is written to history**
  - To reject an Update, return an error in the Validator
- **If there is no Validator, each Update will be accepted**

Adding Validators to Update Handlers

```
@WorkflowInterface
public interface GreetingWorkflow {

    @WorkflowMethod
    String greet();

    @UpdateMethod(name = "set-language")
    Language setLanguage(Language newLanguage);

    @UpdateValidatorMethod(updateName = "set-language")
    void validateSetLanguage(Language newLanguage);
}
```

Adding Validators to Update Handlers

```
public class GreetingWorkflowImpl implements GreetingWorkflow {

    private Language language = Language.ENGLISH;

    @Override
    public String greet() {
        // ... workflow logic
    }

    @Override
    public Language setLanguage(Language newLanguage) {
        Language previousLanguage = this.language;
        this.language = newLanguage;
        return previousLanguage;
    }

    @Override
    public void validateSetLanguage(Language newLanguage) {
        if (!greetings.containsKey(newLanguage)) {
            throw new IllegalArgumentException(newLanguage + " is an unsupported language");
        }
    }
}
```

Update-with-Start and Dynamic Updates

Update-with-Start and Dynamic Updates

- **Update-With-Start checks if there is currently a running Workflow Execution with the given Workflow ID**

Update-with-Start and Dynamic Updates

- **Update-With-Start checks if there is currently a running Workflow Execution with the given Workflow ID**
  - Just as Signal-With-Start does

Update-with-Start and Dynamic Updates

- **Update-With-Start checks if there is currently a running Workflow Execution with the given Workflow ID**
  - Just as Signal-With-Start does
- **If one exists, it will be sent the Update**

Update-with-Start and Dynamic Updates

- **Update-With-Start checks if there is currently a running Workflow Execution with the given Workflow ID**
  - Just as Signal-With-Start does
- **If one exists, it will be sent the Update**
  - If it does not, one will be started and then immediately sent the Update

Update-with-Start and Dynamic Updates

- **Update-With-Start checks if there is currently a running Workflow Execution with the given Workflow ID**
  - Just as Signal-With-Start does
- **If one exists, it will be sent the Update**
  - If it does not, one will be started and then immediately sent the Update
- **Dynamic Update Handlers work the same as Dynamic Signal Handlers**

Beginner to Builder Bootcamp - Java

00. About this Workshop
01. The Basics of Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
04. Understanding Event History
05. Understanding Workflow Determinism
06. Signals, Queries, and Workflow Updates
▶ **07. Timeouts and Retry Policies**
08. Recovering from Failure
09. Conclusion

Timeouts

What are Timeouts?

What are Timeouts?

- **A predefined duration provided for an operation to complete**

What are Timeouts?

- **A predefined duration provided for an operation to complete**
- **Temporal uses timeouts for two primary reasons:**

What are Timeouts?

- **A predefined duration provided for an operation to complete**
- **Temporal uses timeouts for two primary reasons:**
  - Detect failure

What are Timeouts?

- **A predefined duration provided for an operation to complete**
- **Temporal uses timeouts for two primary reasons:**
  - Detect failure
  - Establish a maximum time duration for your business logic

Activity Timeouts

Activity Timeouts

- **Controls the maximum duration of an aspect of an Activity Execution**

Activity Timeouts

- Controls the maximum duration of an aspect of an Activity Execution
- A measure of the time it takes to transition between one state to another

Activity Timeouts

- Controls the maximum duration of an aspect of an Activity Execution
- A measure of the time it takes to transition between one state to another
- Specified as an argument in the ActivityOptions class

Activity Timeouts

- Controls the maximum duration of an aspect of an Activity Execution
- A measure of the time it takes to transition between one state to another
- Specified as an argument in the ActivityOptions class
- As with an Activity that fails, an Activity that times out will be retried

Activity Timeouts

- **Controls the maximum duration of an aspect of an Activity Execution**
- **A measure of the time it takes to transition between one state to another**
- **Specified as an argument in the ActivityOptions class**
- **As with an Activity that fails, an Activity that times out will be retried**
  - Based on details specified in the Retry Policy

Review of Activity Task States

Order	Event Type	Event Description

Review of Activity Task States

Order	Event Type	Event Description
1	ActivityTaskScheduled	

Review of Activity Task States

Order	Event Type	Event Description
1	ActivityTaskScheduled	Temporal Service adds the Activity Task to the Task Queue

Review of Activity Task States

Order	Event Type	Event Description
1	ActivityTaskScheduled	Temporal Service adds the Activity Task to the Task Queue

Review of Activity Task States

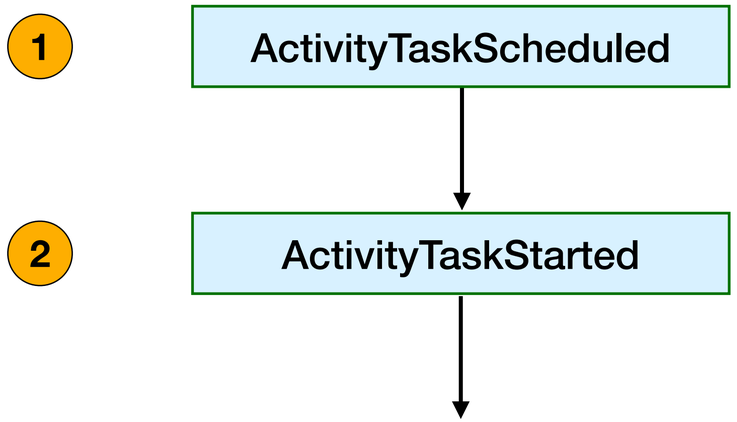
Order	Event Type	Event Description
1	ActivityTaskScheduled	Temporal Service adds the Activity Task to the Task Queue
2	ActivityTaskStarted	

Review of Activity Task States

Order	Event Type	Event Description
1	ActivityTaskScheduled	Temporal Service adds the Activity Task to the Task Queue
2	ActivityTaskStarted	Worker accepts the Activity Task; it's removed from the Task Queue)

Review of Activity Task States

Order	Event Type	Event Description
1	ActivityTaskScheduled	Temporal Service adds the Activity Task to the Task Queue
2	ActivityTaskStarted	Worker accepts the Activity Task; it's removed from the Task Queue)



Review of Activity Task States

Order	Event Type	Event Description
1	ActivityTaskScheduled	Temporal Service adds the Activity Task to the Task Queue
2	ActivityTaskStarted	Worker accepts the Activity Task; it's removed from the Task Queue)
3	ActivityTaskCompleted	

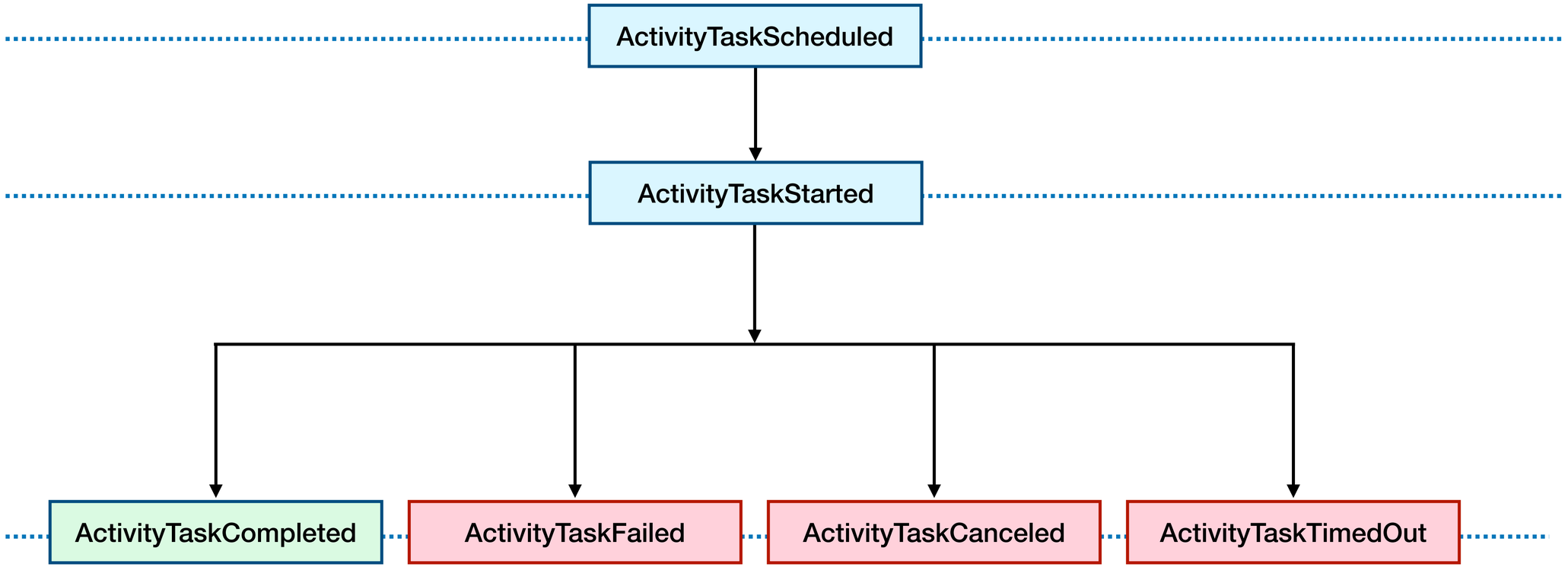
(One of many closed states)

Review of Activity Task States

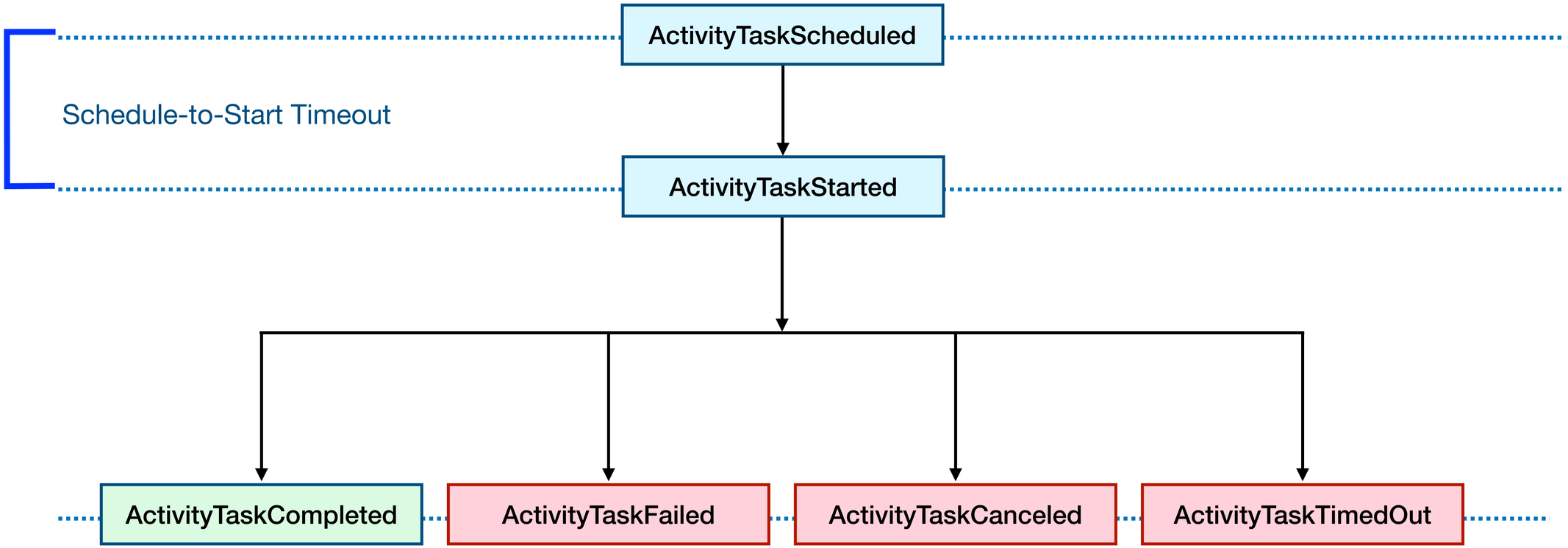
Order	Event Type	Event Description
1	ActivityTaskScheduled	Temporal Service adds the Activity Task to the Task Queue
2	ActivityTaskStarted	Worker accepts the Activity Task; it's removed from the Task Queue)
3	ActivityTaskCompleted	Worker reports result of Activity Execution to the Temporal Service

(One of many closed states)

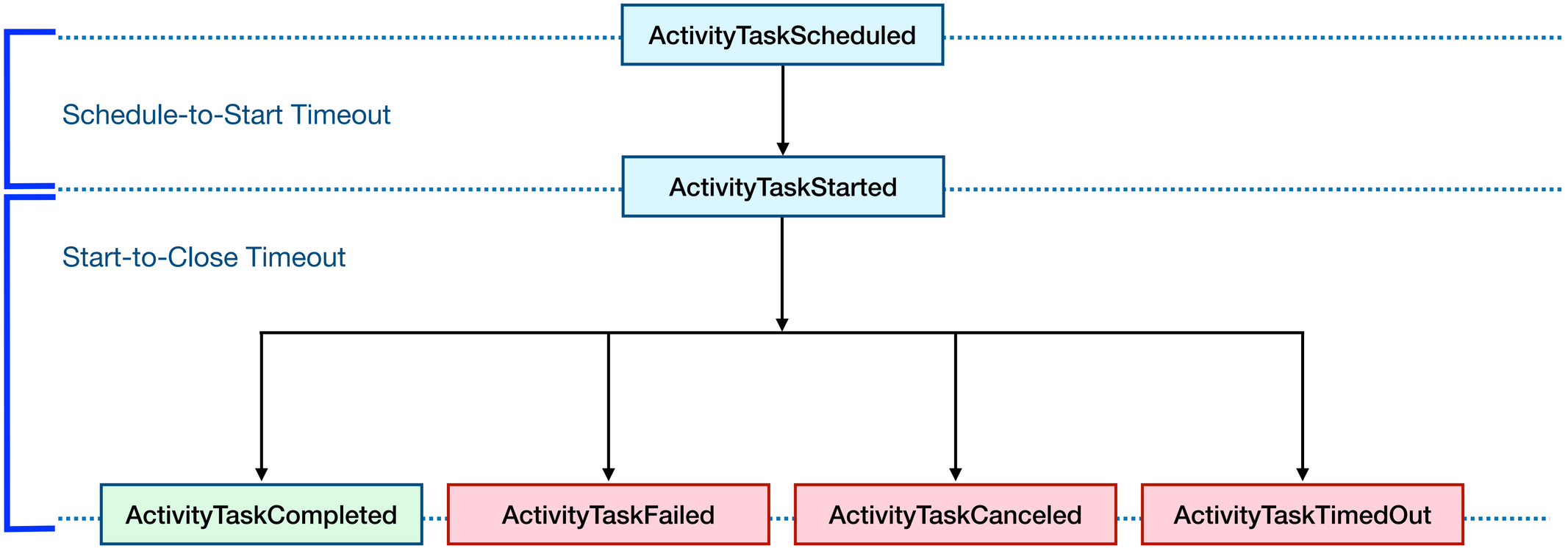
Understanding Activity Timeout Names



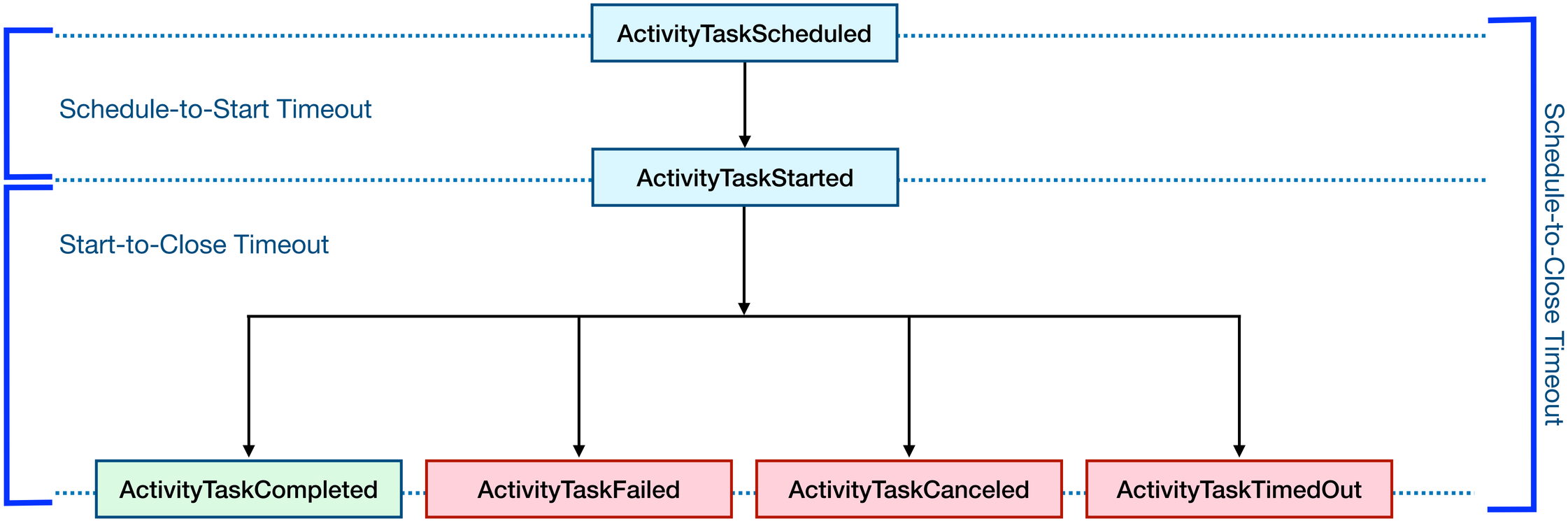
Understanding Activity Timeout Names



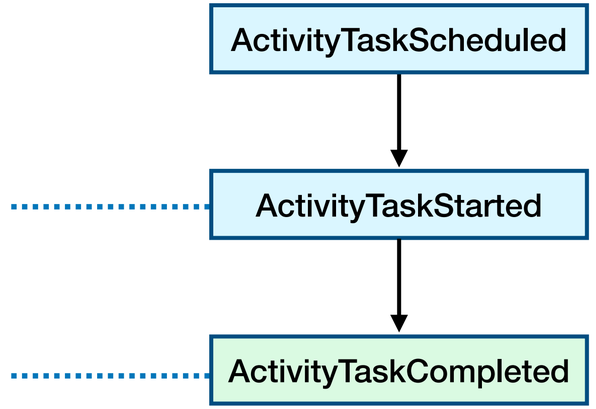
Understanding Activity Timeout Names



Understanding Activity Timeout Names



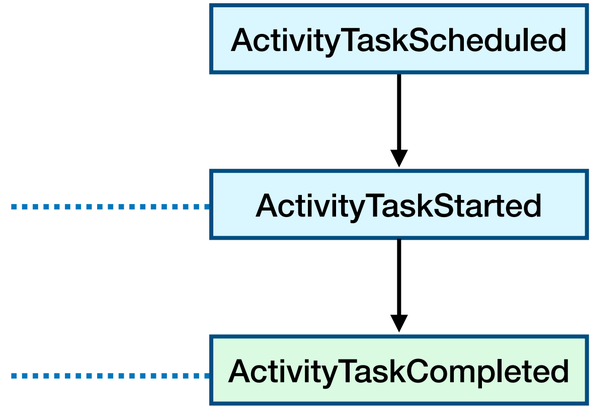
Start-to-Close Timeout



Start-to-Close Timeout

- **Limits maximum time allowed for a single Activity *Task* Execution**

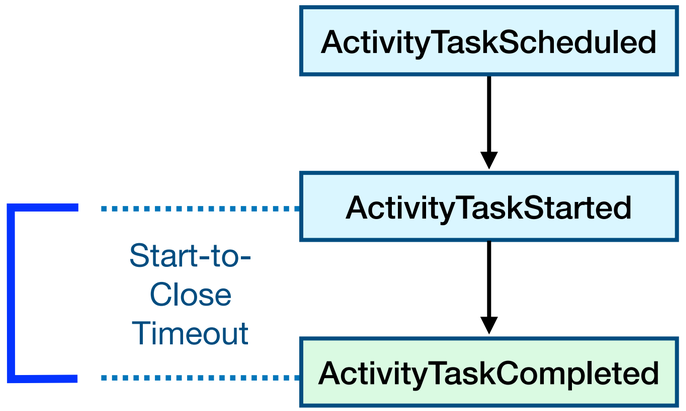
```
ActivityOptions options = ActivityOptions.newBuilder()
    .setStartToCloseTimeout(Duration.ofSeconds(5))
    .build();
```



Start-to-Close Timeout

- **Limits maximum time allowed for a single Activity *Task* Execution**

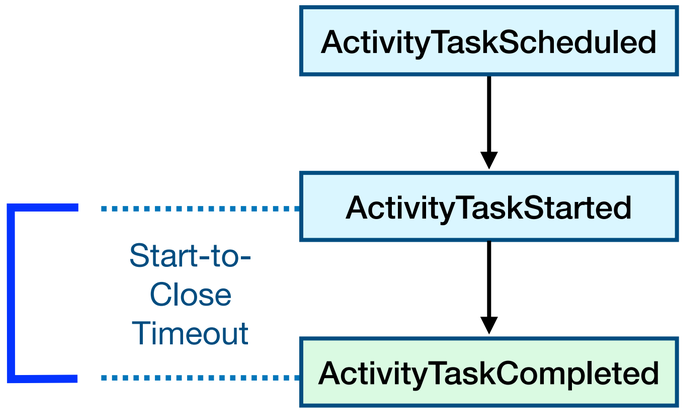
```
ActivityOptions options = ActivityOptions.newBuilder()
    .setStartToCloseTimeout(Duration.ofSeconds(5))
    .build();
```



Start-to-Close Timeout

- **Limits maximum time allowed for a single Activity *Task* Execution**
  - Time is reset for each retry attempt, since that will take place in a new Activity Task

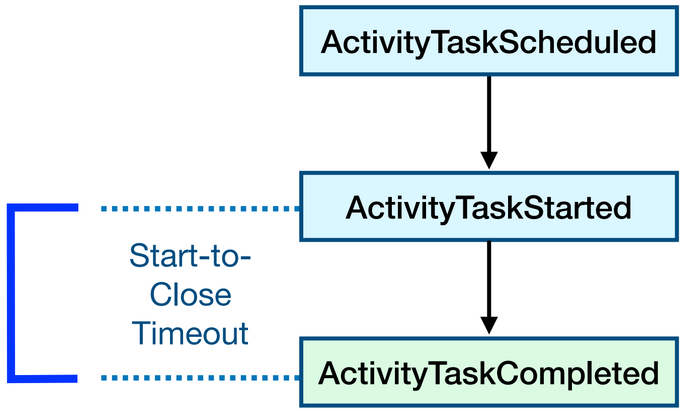
```
ActivityOptions options = ActivityOptions.newBuilder()
    .setStartToCloseTimeout(Duration.ofSeconds(5))
    .build();
```



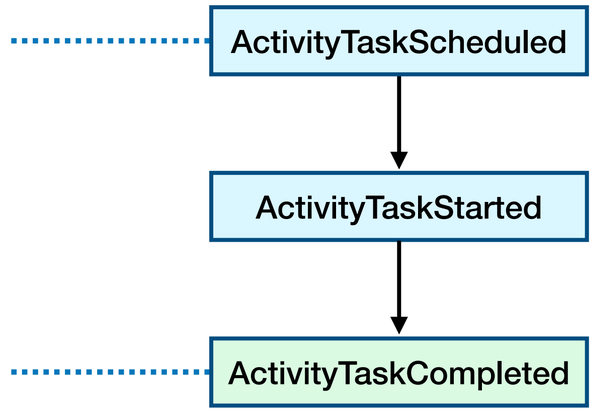
Start-to-Close Timeout

- **Limits maximum time allowed for a single Activity *Task* Execution**
  - Time is reset for each retry attempt, since that will take place in a new Activity Task
  - Recommended: Set duration slightly longer than *maximum* time you expect the Activity will take

```
ActivityOptions options = ActivityOptions.newBuilder()
    .setStartToCloseTimeout(Duration.ofSeconds(5))
    .build();
```



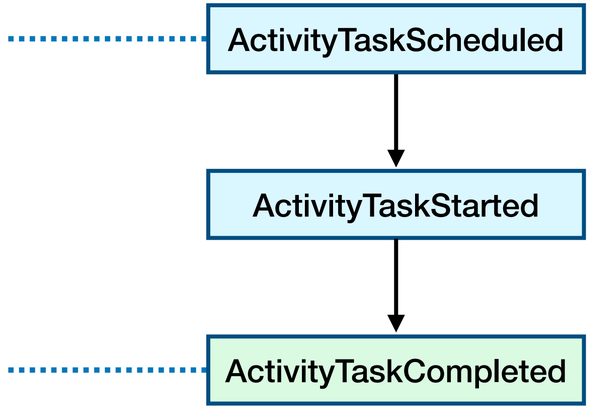
Schedule-to-Close Timeout



Schedule-to-Close Timeout

- **Limits maximum time allowed for entire Activity Execution**

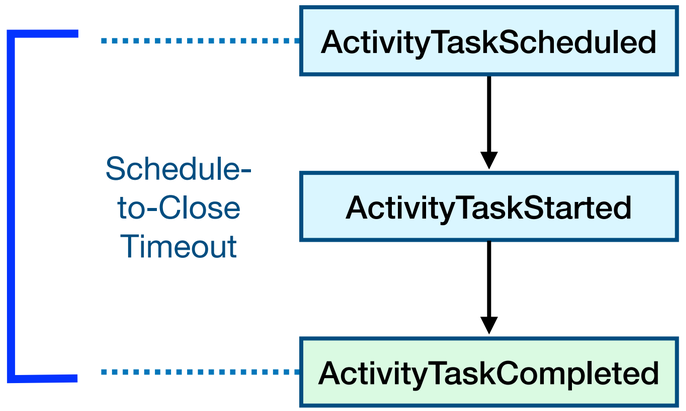
```
ActivityOptions options = ActivityOptions.newBuilder()
    .setScheduleToCloseTimeout(Duration.ofSeconds(5))
    .build();
```



Schedule-to-Close Timeout

- **Limits maximum time allowed for entire Activity Execution**

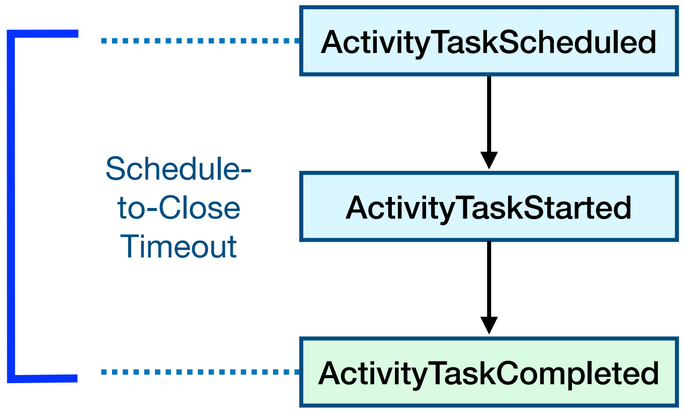
```
ActivityOptions options = ActivityOptions.newBuilder()
    .setScheduleToCloseTimeout(Duration.ofSeconds(5))
    .build();
```



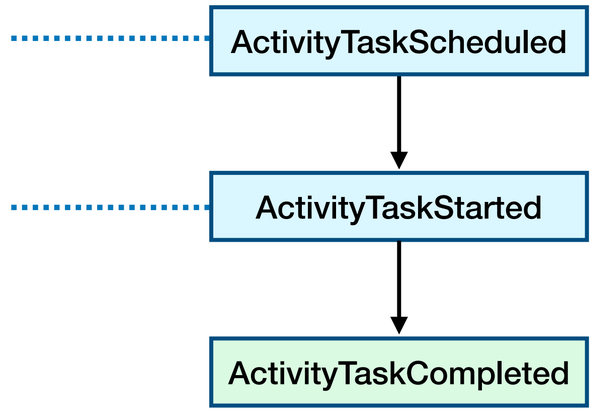
Schedule-to-Close Timeout

- **Limits maximum time allowed for entire Activity Execution**
  - Because it includes all retries, it is typically less predictable than a Start-to-Close Timeout

```
ActivityOptions options = ActivityOptions.newBuilder()
    .setScheduleToCloseTimeout(Duration.ofSeconds(5))
    .build();
```



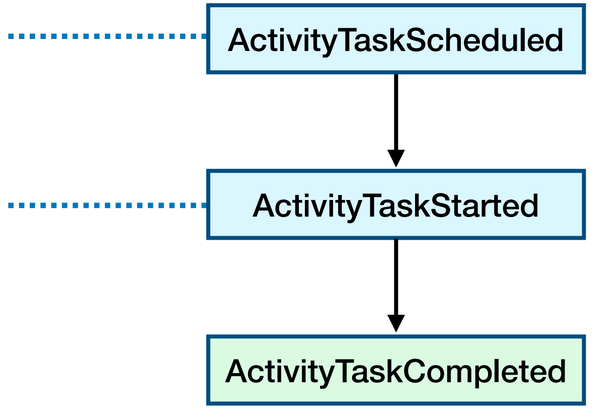
Schedule-to-Start Timeout



Schedule-to-Start Timeout

- **Limits maximum time allowed for Activity Task to remain in Task Queue**

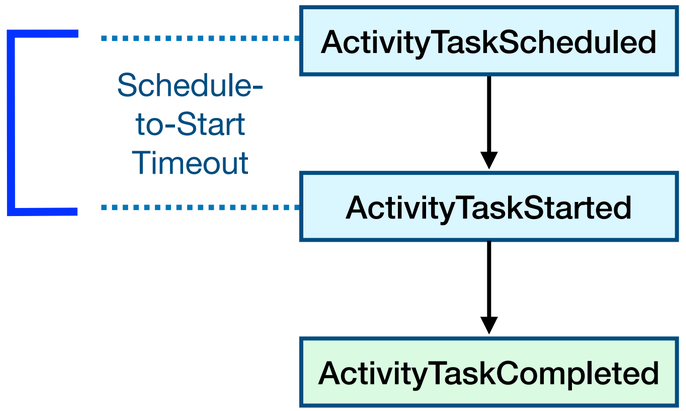
```
ActivityOptions options = ActivityOptions.newBuilder()
    .setScheduleToStartTimeout(Duration.ofSeconds(5))
    .setStartToCloseTimeout(Duration.ofSeconds(10))
    .build();
```



Schedule-to-Start Timeout

- **Limits maximum time allowed for Activity Task to remain in Task Queue**

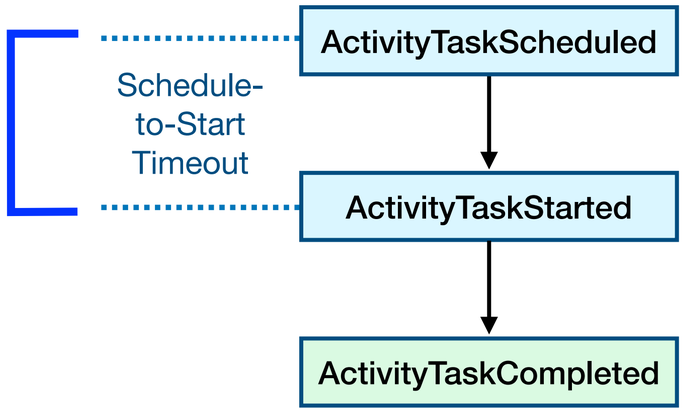
```
ActivityOptions options = ActivityOptions.newBuilder()
    .setScheduleToStartTimeout(Duration.ofSeconds(5))
    .setStartToCloseTimeout(Duration.ofSeconds(10))
    .build();
```



Schedule-to-Start Timeout

- **Limits maximum time allowed for Activity Task to remain in Task Queue**
  - Ensures the Activity is started within a specified time frame, though it's seldom recommended

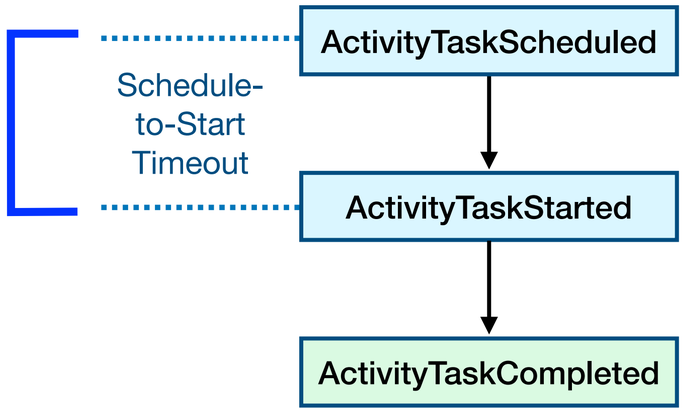
```
ActivityOptions options = ActivityOptions.newBuilder()
    .setScheduleToStartTimeout(Duration.ofSeconds(5))
    .setStartToCloseTimeout(Duration.ofSeconds(10))
    .build();
```



Schedule-to-Start Timeout

- **Limits maximum time allowed for Activity Task to remain in Task Queue**
  - Ensures the Activity is started within a specified time frame, though it's seldom recommended
  - If set, it is done *in addition to* a Start-to-Close or Schedule-to-Close Timeout

```
ActivityOptions options = ActivityOptions.newBuilder()
    .setScheduleToStartTimeout(Duration.ofSeconds(5))
    .setStartToCloseTimeout(Duration.ofSeconds(10))
    .build();
```



Activity Timeout Best Practices

Activity Timeout Best Practices

- **You are required to set a Schedule-to-Close or Start-to-Close Timeout**

Activity Timeout Best Practices

- **You are required to set a Schedule-to-Close or Start-to-Close Timeout**
  - It can be difficult to predict how long execution might take when retries are involved

Activity Timeout Best Practices

- **You are required to set a Schedule-to-Close or Start-to-Close Timeout**
  - It can be difficult to predict how long execution might take when retries are involved
  - Therefore, setting Start-to-Close is usually the better choice

Activity Timeout Best Practices

- **You are required to set a Schedule-to-Close or Start-to-Close Timeout**
  - It can be difficult to predict how long execution might take when retries are involved
  - Therefore, setting Start-to-Close is usually the better choice
- **Retry Policies allow you to specify a maximum number of retry attempts**

Activity Timeout Best Practices

- **You are required to set a Schedule-to-Close or Start-to-Close Timeout**
  - It can be difficult to predict how long execution might take when retries are involved
  - Therefore, setting Start-to-Close is usually the better choice
- **Retry Policies allow you to specify a maximum number of retry attempts**
  - However, using Timeouts to limit the duration is typically more useful

Activity Timeout Best Practices

- **You are required to set a Schedule-to-Close or Start-to-Close Timeout**
  - It can be difficult to predict how long execution might take when retries are involved
  - Therefore, setting Start-to-Close is usually the better choice
- **Retry Policies allow you to specify a maximum number of retry attempts**
  - However, using Timeouts to limit the duration is typically more useful
  - Business logic tends to be concerned with how long something takes (for example, SLAs)

Workflow Timeouts

Workflow Timeouts

- **Control the maximum duration of a different aspect of a Workflow Execution**

Workflow Timeouts

- **Control the maximum duration of a different aspect of a Workflow Execution**
  - Worklfow Timeouts are set when starting the Workflow Execution

Workflow Timeouts

- **Control the maximum duration of a different aspect of a Workflow Execution**
  - Worklfow Timeouts are set when starting the Workflow Execution
- **Thre Types of Workflow Timeouts**

Workflow Timeouts

- **Control the maximum duration of a different aspect of a Workflow Execution**
  - Worklfow Timeouts are set when starting the Workflow Execution
- **Thre Types of Workflow Timeouts**
  - Workflow Execution Timeouts

Workflow Timeouts

- **Control the maximum duration of a different aspect of a Workflow Execution**
  - Worklfow Timeouts are set when starting the Workflow Execution
- **Thre Types of Workflow Timeouts**
  - Workflow Execution Timeouts
  - Workflow Run Timeouts

Workflow Timeouts

- **Control the maximum duration of a different aspect of a Workflow Execution**
  - Worklfow Timeouts are set when starting the Workflow Execution
- **Thre Types of Workflow Timeouts**
  - Workflow Execution Timeouts
  - Workflow Run Timeouts
  - Workflow Task Timeouts

Workflow Timeouts

- **Control the maximum duration of a different aspect of a Workflow Execution**
  - Worklfow Timeouts are set when starting the Workflow Execution
- **Thre Types of Workflow Timeouts**
  - Workflow Execution Timeouts
  - Workflow Run Timeouts
  - Workflow Task Timeouts
- **We generally do not recommend setting Workflow Timeouts**

Workflow Execution Timeout

Workflow Execution Timeout

- **The maximum amount of time that a single Workflow Execution can be executed**

```
WorkflowOptions options = WorkflowOptions.newBuilder()
    .setWorkflowId(workflowID)
    .setTaskQueue(Constants.TASK_QUEUE_NAME)
    .setWorkflowExecutionTimeout(Duration.ofSeconds(10))
    .build();
```

Workflow Execution Timeout

- **The maximum amount of time that a single Workflow Execution can be executed**
  - Including Retries and any usage of Continue-As-New

```
WorkflowOptions options = WorkflowOptions.newBuilder()
    .setWorkflowId(workflowID)
    .setTaskQueue(Constants.TASK_QUEUE_NAME)
    .setWorkflowExecutionTimeout(Duration.ofSeconds(10))
    .build();
```

Workflow Execution Timeout

- **The maximum amount of time that a single Workflow Execution can be executed**
  - Including Retries and any usage of Continue-As-New
- **Default is infinite**

```
WorkflowOptions options = WorkflowOptions.newBuilder()
    .setWorkflowId(workflowID)
    .setTaskQueue(Constants.TASK_QUEUE_NAME)
    .setWorkflowExecutionTimeout(Duration.ofSeconds(10))
    .build();
```

Workflow Run Timeout

Workflow Run Timeout

- **A Workflow Run is the instance of a specific Workflow Execution**

```
WorkflowOptions options = WorkflowOptions.newBuilder()
    .setWorkflowId(workflowID)
    .setTaskQueue(Constants.TASK_QUEUE_NAME)
    .setWorkflowRunTimeout(Duration.ofSeconds(10))
    .build();
```

Workflow Run Timeout

- **A Workflow Run is the instance of a specific Workflow Execution**
  - Restricts the maximum duration of a single Workflow Run

```
WorkflowOptions options = WorkflowOptions.newBuilder()
    .setWorkflowId(workflowID)
    .setTaskQueue(Constants.TASK_QUEUE_NAME)
    .setWorkflowRunTimeout(Duration.ofSeconds(10))
    .build();
```

Workflow Run Timeout

- **A Workflow Run is the instance of a specific Workflow Execution**
  - Restricts the maximum duration of a single Workflow Run
  - This does not include retries or Continue-As-New

```
WorkflowOptions options = WorkflowOptions.newBuilder()
    .setWorkflowId(workflowID)
    .setTaskQueue(Constants.TASK_QUEUE_NAME)
    .setWorkflowRunTimeout(Duration.ofSeconds(10))
    .build();
```

Workflow Run Timeout

- **A Workflow Run is the instance of a specific Workflow Execution**
  - Restricts the maximum duration of a single Workflow Run
  - This does not include retries or Continue-As-New
- **Default is infinite**

```
WorkflowOptions options = WorkflowOptions.newBuilder()
    .setWorkflowId(workflowID)
    .setTaskQueue(Constants.TASK_QUEUE_NAME)
    .setWorkflowRunTimeout(Duration.ofSeconds(10))
    .build();
```

Workflow Task Timeout

Workflow Task Timeout

- **Restricts the maximum amount of time that a Worker can execute a Workflow Task**

```
WorkflowOptions options = WorkflowOptions.newBuilder()
    .setWorkflowId(workflowID)
    .setTaskQueue(Constants.TASK_QUEUE_NAME)
    .setWorkflowTaskTimeout(Duration.ofSeconds(20))
    .build();
```

Workflow Task Timeout

- **Restricts the maximum amount of time that a Worker can execute a Workflow Task**
  - Begins from when the Worker has accepted that Workflow Task through its completion

```
WorkflowOptions options = WorkflowOptions.newBuilder()
    .setWorkflowId(workflowID)
    .setTaskQueue(Constants.TASK_QUEUE_NAME)
    .setWorkflowTaskTimeout(Duration.ofSeconds(20))
    .build();
```

Workflow Task Timeout

- **Restricts the maximum amount of time that a Worker can execute a Workflow Task**
  - Begins from when the Worker has accepted that Workflow Task through its completion
  - Primarily available to detect when a Worker has become unavailable

```
WorkflowOptions options = WorkflowOptions.newBuilder()
    .setWorkflowId(workflowID)
    .setTaskQueue(Constants.TASK_QUEUE_NAME)
    .setWorkflowTaskTimeout(Duration.ofSeconds(20))
    .build();
```

Workflow Task Timeout

- **Restricts the maximum amount of time that a Worker can execute a Workflow Task**
  - Begins from when the Worker has accepted that Workflow Task through its completion
  - Primarily available to detect when a Worker has become unavailable
- **Default value of is ten seconds**

```
WorkflowOptions options = WorkflowOptions.newBuilder()
    .setWorkflowId(workflowID)
    .setTaskQueue(Constants.TASK_QUEUE_NAME)
    .setWorkflowTaskTimeout(Duration.ofSeconds(20))
    .build();
```

Best Practices

Best Practices

- **We generally do not recommend setting Workflow Timeouts**

Best Practices

- **We generally do not recommend setting Workflow Timeouts**
  - No, really, we REALLY don't recommend it.

Best Practices

- **We generally do not recommend setting Workflow Timeouts**
  - No, really, we REALLY don't recommend it.
- **If you need to perform an action inside your Workflow after a specific period time, we recommend using a Timer**

Activity Heartbeats

Activity Heartbeats

- **A periodic message sent by the Activity to the Temporal Service that serves multiple purposes:**

Activity Heartbeats

- **A periodic message sent by the Activity to the Temporal Service that serves multiple purposes:**
  - Progress indication

Activity Heartbeats

- **A periodic message sent by the Activity to the Temporal Service that serves multiple purposes:**
  - Progress indication
  - Cancellation Detection

Activity Heartbeats

- **A periodic message sent by the Activity to the Temporal Service that serves multiple purposes:**
  - Progress indication
  - Cancellation Detection
  - Worker Health Check

How to Send a Heartbeat Message

```
for(int x = 0; x < 10; x++){
  Activity.getExecutionContext().heartbeat(x);
  try {
    Thread.sleep(Duration.ofSeconds(1));
  } catch (InterruptedException e) {
    continue;
  }
}
```

Heartbeats and Cancellations

Heartbeats and Cancellations

- **For an Activity to be cancellable, it must perform Heartbeating**

Heartbeats and Cancellations

- **For an Activity to be cancellable, it must perform Heartbeating**
  - When an Activity sends a Heartbeat, the Temporal Service checks if there has been a cancellation request.

Heartbeats and Cancellations

- **For an Activity to be cancellable, it must perform Heartbeating**
  - When an Activity sends a Heartbeat, the Temporal Service checks if there has been a cancellation request.
  - If a cancellation request exists, it is included in the response to the Heartbeat.

Heartbeats and Cancellations

- **For an Activity to be cancellable, it must perform Heartbeating**
  - When an Activity sends a Heartbeat, the Temporal Service checks if there has been a cancellation request.
  - If a cancellation request exists, it is included in the response to the Heartbeat.
  - This allows the Activity to respond promptly to cancellation requests.

Heartbeats and Cancellations

- **For an Activity to be cancellable, it must perform Heartbeating**
  - When an Activity sends a Heartbeat, the Temporal Service checks if there has been a cancellation request.
  - If a cancellation request exists, it is included in the response to the Heartbeat.
  - This allows the Activity to respond promptly to cancellation requests.
- **If you need to cancel a long-running Activity Execution, make sure it is configured to send Heartbeats periodically**

Heartbeat Timeout

Heartbeat Timeout

- **The maximum time allowed between Activity Heartbeats**

Heartbeat Timeout

- **The maximum time allowed between Activity Heartbeats**
- **The Heartbeat Timeout must be set in order for Temporal to track the Heartbeats sent by the Activity**

```
ActivityOptions options = ActivityOptions.newBuilder()
    .setStartToCloseTimeout(Duration.ofMinutes(5))
    .setHeartbeatTimeout(Duration.ofSeconds(3))
    .build();
```

Heartbeat Timeout

- **The maximum time allowed between Activity Heartbeats**
- **The Heartbeat Timeout must be set in order for Temporal to track the Heartbeats sent by the Activity**
  - If this Timeout is not set, any Heartbeats sent by the Activity will be ignored.

```
ActivityOptions options = ActivityOptions.newBuilder()
    .setStartToCloseTimeout(Duration.ofMinutes(5))
    .setHeartbeatTimeout(Duration.ofSeconds(3))
    .build();
```

Heartbeat Timeout Best Practices

Heartbeat Timeout Best Practices

- **To ensure efficient, handling of long-running Activities:**

Heartbeat Timeout Best Practices

- **To ensure efficient, handling of long-running Activities:**
  - Set Start-to-Close Timeout slightly longer than the maximum duration of your Activity

Heartbeat Timeout Best Practices

- **To ensure efficient, handling of long-running Activities:**
  - Set Start-to-Close Timeout slightly longer than the maximum duration of your Activity
  - Your Heartbeat Timeout should be fairly short

Heartbeat Timeout Best Practices

- **To ensure efficient, handling of long-running Activities:**
  - Set Start-to-Close Timeout slightly longer than the maximum duration of your Activity
  - Your Heartbeat Timeout should be fairly short
- **Must send Heartbeats at intervals shorter than the Heartbeat Timeout**

Heartbeat Throttling

Heartbeat Throttling

- **Heartbeats may be throttled by the Worker**

Heartbeat Throttling

- **Heartbeats may be throttled by the Worker**
  - Throttled Heartbeats are not sent to the Temporal Service

Heartbeat Throttling

- **Heartbeats may be throttled by the Worker**
  - Throttled Heartbeats are not sent to the Temporal Service
- **Throttling reduces network traffic and load on the Temporal Service**

Heartbeat Throttling

- **Heartbeats may be throttled by the Worker**
  - Throttled Heartbeats are not sent to the Temporal Service
- **Throttling reduces network traffic and load on the Temporal Service**
  - Suppresses Heartbeats that aren't necessary to prevent a Heartbeat Timeout

Heartbeat Throttling

- **Heartbeats may be throttled by the Worker**
  - Throttled Heartbeats are not sent to the Temporal Service
- **Throttling reduces network traffic and load on the Temporal Service**
  - Suppresses Heartbeats that aren't necessary to prevent a Heartbeat Timeout
- **Does not apply to the final Heartbeat in the case of Activity Failure**

Heartbeat Throttling

Activity ID	Details	
4	**Activity Type**	**pollDeliveryDriver**
	Attempt	1
	Maximum Attempts	5
	Last Heartbeat	
	State	PENDING_ACTIVITY_STATE_STARTED
	Last Started Time	2024-08-08 UTC 01:28:12.76
	Last Worker Identity	45943@Angelas-MBP

Retry Policies

Retry Policies

Retry Policies

- **By default, Temporal automatically retries an Activity that fails**

Retry Policies

- **By default, Temporal automatically retries an Activity that fails**
  - Activities are associated with a Retry Policy by default

Retry Policies

- **By default, Temporal automatically retries an Activity that fails**
  - Activities are associated with a Retry Policy by default
  - A Retry Policy defines the details of how those retries are carried out

Retry Policies

- **By default, Temporal automatically retries an Activity that fails**
  - Activities are associated with a Retry Policy by default
  - A Retry Policy defines the details of how those retries are carried out
- **Unlike Activities, Workflow Executions are not retried by default**

Retry Policies

- **By default, Temporal automatically retries an Activity that fails**
  - Activities are associated with a Retry Policy by default
  - A Retry Policy defines the details of how those retries are carried out
- **Unlike Activities, Workflow Executions are not retried by default**
  - Workflows are not associated with a Retry Policy by default

Retry Policies

- **By default, Temporal automatically retries an Activity that fails**
  - Activities are associated with a Retry Policy by default
  - A Retry Policy defines the details of how those retries are carried out
- **Unlike Activities, Workflow Executions are not retried by default**
  - Workflows are not associated with a Retry Policy by default
  - While failed *Workflow Executions* are not retried automatically, failed *Workflow Tasks* are

Retry Policies

- **By default, Temporal automatically retries an Activity that fails**
  - Activities are associated with a Retry Policy by default
  - A Retry Policy defines the details of how those retries are carried out
- **Unlike Activities, Workflow Executions are not retried by default**
  - Workflows are not associated with a Retry Policy by default
  - While failed *Workflow Executions* are not retried automatically, failed *Workflow Tasks* are
  - Workflow Tasks retry automatically and indefinitely

Retry Policy for Activities

Method	Specifies	Default Value

Retry Policy for Activities

- Customize RetryPolicy by calling methods on RetryOptions.Builder

Method	Specifies	Default Value

Retry Policy for Activities

- Customize RetryPolicy by calling methods on RetryOptions.Builder

Method	Specifies	Default Value
setInitialInterval	Duration before the first retry	1 second

Retry Policy for Activities

- Customize RetryPolicy by calling methods on RetryOptions.Builder

Method	Specifies	Default Value
setInitialInterval	Duration before the first retry	1 second
setBackoffCoefficient	Multiplier used for subsequent retries	2.0

Retry Policy for Activities

- Customize RetryPolicy by calling methods on RetryOptions.Builder

Method	Specifies	Default Value
setInitialInterval	Duration before the first retry	1 second
setBackoffCoefficient	Multiplier used for subsequent retries	2.0
setMaximumInterval	Maximum duration between retries, in seconds	100 * InitialInterval

Retry Policy for Activities

- Customize RetryPolicy by calling methods on RetryOptions.Builder

Method	Specifies	Default Value
setInitialInterval	Duration before the first retry	1 second
setBackoffCoefficient	Multiplier used for subsequent retries	2.0
setMaximumInterval	Maximum duration between retries, in seconds	100 * InitialInterval
setMaximumAttempts	Maximum number of retry attempts before giving up	0 (unlimited)

Retry Policy for Activities

- Customize RetryPolicy by calling methods on RetryOptions.Builder

Method	Specifies	Default Value
setInitialInterval	Duration before the first retry	1 second
setBackoffCoefficient	Multiplier used for subsequent retries	2.0
setMaximumInterval	Maximum duration between retries, in seconds	100 * InitialInterval
setMaximumAttempts	Maximum number of retry attempts before giving up	0 (unlimited)
setDoNotRetry	List of application failure types that won’t be retried	[] (empty array)

Retry Policy for Activities

- Customize RetryPolicy by calling methods on RetryOptions.Builder

Method	Specifies	Default Value
setInitialInterval	Duration before the first retry	1 second
setBackoffCoefficient	Multiplier used for subsequent retries	2.0
setMaximumInterval	Maximum duration between retries, in seconds	100 * InitialInterval
setMaximumAttempts	Maximum number of retry attempts before giving up	0 (unlimited)
setDoNotRetry	List of application failure types that won't be retried	[] (empty array)

```
RetryOptions retryOptions = RetryOptions.newBuilder()
    .setInitialInterval(Duration.ofSeconds(20))
    .setBackoffCoefficient(3.0)
    .setMaximumInterval(Duration.ofSeconds(360))
    .setDoNotRetry(CreditCardProcException.class.getName(), "ExpiredCardError")
    .build();
```

Defining Errors as Non-Retryable Types

Defining Errors as Non-Retryable Types

- **Not always appropriate to designate an Activity Execution as retryable or not when throwing an ApplicationFailure from the Activity**

Defining Errors as Non-Retryable Types

- **Not always appropriate to designate an Activity Execution as retryable or not when throwing an ApplicationFailure from the Activity**
  - The implementer likely knows which errors are retryable or not

Defining Errors as Non-Retryable Types

- **Not always appropriate to designate an Activity Execution as retryable or not when throwing an ApplicationFailure from the Activity**
  - The implementer likely knows which errors are retryable or not
  - Others using a shared library may not wish to fail on that specific error

Defining Errors as Non-Retryable Types

- **Not always appropriate to designate an Activity Execution as retryable or not when throwing an ApplicationFailure from the Activity**
  - The implementer likely knows which errors are retryable or not
  - Others using a shared library may not wish to fail on that specific error
- **Throw the exception, designate it as non-retryable in the Retry Policy**

```
RetryOptions retryOptions = RetryOptions.newBuilder()
    .setDoNotRetry(CreditCardProcException.class.getName(), “ExpiredCardError")
    .build();
```

Defining Errors as Non-Retryable Types

- **Not always appropriate to designate an Activity Execution as retryable or not when throwing an ApplicationFailure from the Activity**
  - The implementer likely knows which errors are retryable or not
  - Others using a shared library may not wish to fail on that specific error
- **Throw the exception, designate it as non-retryable in the Retry Policy**
  - Known as a non-retryable error type

```
RetryOptions retryOptions = RetryOptions.newBuilder()
    .setDoNotRetry(CreditCardProcException.class.getName(), “ExpiredCardError")
    .build();
```

Non-Retryable Errors vs Non-Retryable Error Types

Non-Retryable Errors vs Non-Retryable Error Types

- **Non-Retryable Errors**

Non-Retryable Errors vs Non-Retryable Error Types

- **Non-Retryable Errors**
  - Thrown directy from the method

Non-Retryable Errors vs Non-Retryable Error Types

- **Non-Retryable Errors**
  - Thrown directy from the method
  - Failure is deemed unrecoverable by the implementer

Non-Retryable Errors vs Non-Retryable Error Types

- **Non-Retryable Errors**
  - Thrown directy from the method
  - Failure is deemed unrecoverable by the implementer
- **Non-Retryable Error Types**

Non-Retryable Errors vs Non-Retryable Error Types

- **Non-Retryable Errors**
  - Thrown directy from the method
  - Failure is deemed unrecoverable by the implementer
- **Non-Retryable Error Types**
  - Specified in a Retry Policy

Non-Retryable Errors vs Non-Retryable Error Types

- **Non-Retryable Errors**
  - Thrown directy from the method
  - Failure is deemed unrecoverable by the implementer
- **Non-Retryable Error Types**
  - Specified in a Retry Policy
  - Failure is left to be handled by the invoker

Retry Policy Configurations for Failures

Retry Policy Configurations for Failures

- **Transient failure**

Retry Policy Configurations for Failures

- **Transient failure**
  - Resolved by retrying the operation immediately after the failure

Retry Policy Configurations for Failures

- **Transient failure**
  - Resolved by retrying the operation immediately after the failure
  - Default Policy typically a good option

Retry Policy Configurations for Failures

- **Transient failure**
  - Resolved by retrying the operation immediately after the failure
  - Default Policy typically a good option
- **Intermittent failure:**

Retry Policy Configurations for Failures

- **Transient failure**
  - Resolved by retrying the operation immediately after the failure
  - Default Policy typically a good option
- **Intermittent failure:**
  - Addressed by retrying the operation, spread out over a longer period of time

Retry Policy Configurations for Failures

- **Transient failure**
  - Resolved by retrying the operation immediately after the failure
  - Default Policy typically a good option
- **Intermittent failure:**
  - Addressed by retrying the operation, spread out over a longer period of time
  - Configure a custom BackoffCoefficient & Maximum Interval

Retry Policy Configurations for Failures

- **Transient failure**
  - Resolved by retrying the operation immediately after the failure
  - Default Policy typically a good option
- **Intermittent failure:**
  - Addressed by retrying the operation, spread out over a longer period of time
  - Configure a custom BackoffCoefficient & Maximum Interval
- **Permanent failure:**

Retry Policy Configurations for Failures

- **Transient failure**
  - Resolved by retrying the operation immediately after the failure
  - Default Policy typically a good option
- **Intermittent failure:**
  - Addressed by retrying the operation, spread out over a longer period of time
  - Configure a custom BackoffCoefficient & Maximum Interval
- **Permanent failure:**
  - Cannot be resolved solely through retries, needs manual intervention

Retry Policy Configurations for Failures

- **Transient failure**
  - Resolved by retrying the operation immediately after the failure
  - Default Policy typically a good option
- **Intermittent failure:**
  - Addressed by retrying the operation, spread out over a longer period of time
  - Configure a custom BackoffCoefficient & Maximum Interval
- **Permanent failure:**
  - Cannot be resolved solely through retries, needs manual intervention
  - Configure Non-Retryable Error Types

Retry Policy Configurations for Failures

- **Transient failure**
  - Resolved by retrying the operation immediately after the failure
  - Default Policy typically a good option
- **Intermittent failure:**
  - Addressed by retrying the operation, spread out over a longer period of time
  - Configure a custom BackoffCoefficient & Maximum Interval
- **Permanent failure:**
  - Cannot be resolved solely through retries, needs manual intervention
  - Configure Non-Retryable Error Types
- **Define a Retry Policy for failures you anticpate**

Custom Retry Policy for Activity Execution

```
import io.temporal.common.RetryOptions;

public class GreetingWorkflowImpl implements GreetingWorkflow {

    RetryOptions retryOptions = RetryOptions.newBuilder()
        .setInitialInterval(Duration.ofSeconds(15))
        .setBackoffCoefficient(2.0)
        .setMaximumInterval(Duration.ofSeconds(60))
        .setMaximumAttempts(100)
        .build();

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofSeconds(5))
            .setRetryOptions(retryOptions)
            .build();

    private final GreetingActivities activities =
            Workflow.newActivityStub(GreetingActivities.class, options);

    @Override
    public String greetSomeone(String name) {
        // .. remainder of Workflow code would follow
```

Custom Retry Policy for Activity Execution

```
import io.temporal.common.RetryOptions;

public class GreetingWorkflowImpl implements GreetingWorkflow {

    RetryOptions retryOptions = RetryOptions.newBuilder()
        .setInitialInterval(Duration.ofSeconds(15))
        .setBackoffCoefficient(2.0)
        .setMaximumInterval(Duration.ofSeconds(60))
        .setMaximumAttempts(100)
        .build();

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofSeconds(5))
            .setRetryOptions(retryOptions)
            .build();

    private final GreetingActivities activities =
            Workflow.newActivityStub(GreetingActivities.class, options);

    @Override
    public String greetSomeone(String name) {
        // .. remainder of Workflow code would follow
```

Import this Class from the SDK

Custom Retry Policy for Activity Execution

```
import io.temporal.common.RetryOptions;

public class GreetingWorkflowImpl implements GreetingWorkflow {

    RetryOptions retryOptions = RetryOptions.newBuilder()
        .setInitialInterval(Duration.ofSeconds(15))
        .setBackoffCoefficient(2.0)
        .setMaximumInterval(Duration.ofSeconds(60))
        .setMaximumAttempts(100)
        .build();

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofSeconds(5))
            .setRetryOptions(retryOptions)
            .build();

    private final GreetingActivities activities =
            Workflow.newActivityStub(GreetingActivities.class, options);

    @Override
    public String greetSomeone(String name) {
        // .. remainder of Workflow code would follow
```

1 Import this Class from the SDK

2 Specify your policy values

Custom Retry Policy for Activity Execution

```
import io.temporal.common.RetryOptions;

public class GreetingWorkflowImpl implements GreetingWorkflow {

    RetryOptions retryOptions = RetryOptions.newBuilder()
        .setInitialInterval(Duration.ofSeconds(15))
        .setBackoffCoefficient(2.0)
        .setMaximumInterval(Duration.ofSeconds(60))
        .setMaximumAttempts(100)
        .build();

    ActivityOptions options = ActivityOptions.newBuilder()
            .setStartToCloseTimeout(Duration.ofSeconds(5))
            .setRetryOptions(retryOptions)
            .build();

    private final GreetingActivities activities =
            Workflow.newActivityStub(GreetingActivities.class, options);

    @Override
    public String greetSomeone(String name) {
        // .. remainder of Workflow code would follow
```

1 Import this Class from the SDK

2 Specify your policy values

3 Associate the Policy with the Activity Execution

Common Use Cases for Defining a Custom Retry Policy

Common Use Cases for Defining a Custom Retry Policy

- **Making calls to a service experiencing heavy load**

Common Use Cases for Defining a Custom Retry Policy

- **Making calls to a service experiencing heavy load**
  - Setting a higher MaximumInterval to allow for longer delays between retries

Common Use Cases for Defining a Custom Retry Policy

- **Making calls to a service experiencing heavy load**
  - Setting a higher MaximumInterval to allow for longer delays between retries
- **If an external service implements rate limiting**

Common Use Cases for Defining a Custom Retry Policy

- **Making calls to a service experiencing heavy load**
  - Setting a higher MaximumInterval to allow for longer delays between retries
- **If an external service implements rate limiting**
  - Use a longer BackoffCoefficient to avoid triggering this limit

Common Use Cases for Defining a Custom Retry Policy

- **Making calls to a service experiencing heavy load**
  - Setting a higher MaximumInterval to allow for longer delays between retries
- **If an external service implements rate limiting**
  - Use a longer BackoffCoefficient to avoid triggering this limit
- **A service charges for each call received**

Common Use Cases for Defining a Custom Retry Policy

- **Making calls to a service experiencing heavy load**
  - Setting a higher MaximumInterval to allow for longer delays between retries
- **If an external service implements rate limiting**
  - Use a longer BackoffCoefficient to avoid triggering this limit
- **A service charges for each call received**
  - One of the few instances where setting Maximum Attempts is appropriate

Best Practices for Retry Policies

Best Practices for Retry Policies

- **Don’t unnecessarily set maximum attempts to 1**

Best Practices for Retry Policies

- **Don’t unnecessarily set maximum attempts to 1**
  - Not a substitue for idempotency

Best Practices for Retry Policies

- **Don’t unnecessarily set maximum attempts to 1**
  - Not a substitue for idempotency
  - Setting MaximumAttempts to **DOES NOT** mean Only Once Execution

Best Practices for Retry Policies

- **Don’t unnecessarily set maximum attempts to 1**
  - Not a substitue for idempotency
  - Setting MaximumAttempts to **DOES NOT** mean Only Once Execution
- **Recognize that each Activity Execution can have its own retry policy**

Best Practices for Retry Policies

- **Don’t unnecessarily set maximum attempts to 1**
  - Not a substitue for idempotency
  - Setting MaximumAttempts to **DOES NOT** mean Only Once Execution
- **Recognize that each Activity Execution can have its own retry policy**
  - You can call the same Activity with different Retry Policies

Best Practices for Retry Policies

- **Don’t unnecessarily set maximum attempts to 1**
  - Not a substitue for idempotency
  - Setting MaximumAttempts to **DOES NOT** mean Only Once Execution
- **Recognize that each Activity Execution can have its own retry policy**
  - You can call the same Activity with different Retry Policies
- **Avoid retry policies for Workflow Executions**

Customizing a Retry Policy for a Specific Activity

```
// Create Activity Stub with Retry Policy One
RetryOptions retryOptionsOne = RetryOptions.newBuilder()
    .setInitialInterval(Duration.ofSeconds(5))
    .setBackoffCoefficient(3.0)
    .build();

ActivityOptions optionsOne = ActivityOptions.newBuilder()
    .setStartToCloseTimeout(Duration.ofMinutes(5))
    .setRetryOptions(retryOptionsOne)
    .setHeartbeatTimeout(Duration.ofSeconds(3))
    .build();

private final MyActivities activitiesOne =
    Workflow.newActivityStub(MyActivities.class, optionsOne);

// Create Activity Stub with Retry Policy Two
RetryOptions retryOptionsTwo = RetryOptions.newBuilder()
    .setInitialInterval(Duration.ofSeconds(10))
    .setBackoffCoefficient(1.25)
    .build();

ActivityOptions optionsTwo = ActivityOptions.newBuilder()
    .setStartToCloseTimeout(Duration.ofMinutes(5))
    .setRetryOptions(retryOptionsTwo)
    .setHeartbeatTimeout(Duration.ofSeconds(3))
    .build();

private final MyActivities activitiesTwo =
    Workflow.newActivityStub(MyActivities.class, optionsTwo);
```

```
public String myWorkflow(String name){

    // Calling the greet Activity useing Retry Policy One
    String resultOne = activitiesOne.greet(name);

    // Calling the greet Activity useing Retry Policy Two
    String resultTwo = activitiesTwo.greet(name);

    return resultOne + " " + resultTwo;
}
```

Retry Policy for Workflow Executions

Retry Policy for Workflow Executions

- **Workflow Executions do not retry by default**

Retry Policy for Workflow Executions

- **Workflow Executions do not retry by default**
  - Workflow failures typically indicate a permanent failure

Retry Policy for Workflow Executions

- **Workflow Executions do not retry by default**
  - Workflow failures typically indicate a permanent failure
- **We do not recommend associating a Retry Policy with your Workflow Execution**

Exercise #6: Non-Retryable Error Types

- **During this exercise, you will**
  - Configure non-retry able error types for Activities
  - Implement customized retry policies for Activities
  - Add Heartbeats and Heartbeat timeouts to help users monitor the health of Activities
- **Refer to the README.md file in the exercise environment for details**
  - The code is below the **exercises/non-retryable-error-types**
    - Make your changes to the code in the **practice** subdirectory (look for TODO comments)
    - If you need a hint or want to verify your changes, look at the complete version in the **solution** subdirectory

Beginner to Builder Bootcamp - Java

00. About this Workshop
01. The Basics of Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
04. Understanding Event History
05. Understanding Workflow Determinism
06. Signals, Queries, and Workflow Updates
07. Timeouts and Retry Policies
08. Recovering from Failure
09. Conclusion

Handling a Workflow Execution that Cannot Complete

Handling a Workflow Execution that Cannot Complete

- **Canceling your Workflow Execution**

Handling a Workflow Execution that Cannot Complete

- **Canceling your Workflow Execution**
  - Provides a graceful way to stop the Workflow Execution

Handling a Workflow Execution that Cannot Complete

- **Canceling your Workflow Execution**
  - Provides a graceful way to stop the Workflow Execution
- **Terminating your Workflow Execution**

Handling a Workflow Execution that Cannot Complete

- **Canceling your Workflow Execution**
  - Provides a graceful way to stop the Workflow Execution
- **Terminating your Workflow Execution**
  - Forcefully halts th Workflow Execution

Handling a Workflow Execution that Cannot Complete

- **Canceling your Workflow Execution**
  - Provides a graceful way to stop the Workflow Execution
- **Terminating your Workflow Execution**
  - Forcefully halts th Workflow Execution
- **Resetting your Workflow Execution**

Handling a Workflow Execution that Cannot Complete

- **Canceling your Workflow Execution**
  - Provides a graceful way to stop the Workflow Execution
- **Terminating your Workflow Execution**
  - Forcefully halts th Workflow Execution
- **Resetting your Workflow Execution**
  - Resets the Workflow Execution state to a previous point in the Event History

Rollback Actions and the Saga Pattern

Rollback Actions and the Saga Pattern

- **A saga is a pattern used in distributed systems to manage a sequence of local transactions**

Rollback Actions and the Saga Pattern

- A saga is a pattern used in distributed systems to manage a sequence of local transactions
- If any transaction in the sequence fails, the saga executes actions to rollback the previous operations.

Rollback Actions and the Saga Pattern

- **A saga is a pattern used in distributed systems to manage a sequence of local transactions**

- **If any transaction in the sequence fails, the saga executes actions to rollback the previous operations.**
  - This is known as a compensating action.

Rollback Actions and the Saga Pattern

- **A saga is a pattern used in distributed systems to manage a sequence of local transactions**
- **If any transaction in the sequence fails, the saga executes actions to rollback the previous operations.**
  - This is known as a compensating action.
- **Examples:**

Rollback Actions and the Saga Pattern

- **A saga is a pattern used in distributed systems to manage a sequence of local transactions**

- **If any transaction in the sequence fails, the saga executes actions to rollback the previous operations.**
  - This is known as a compensating action.

- **Examples:**
  - E-Commerce Transaction

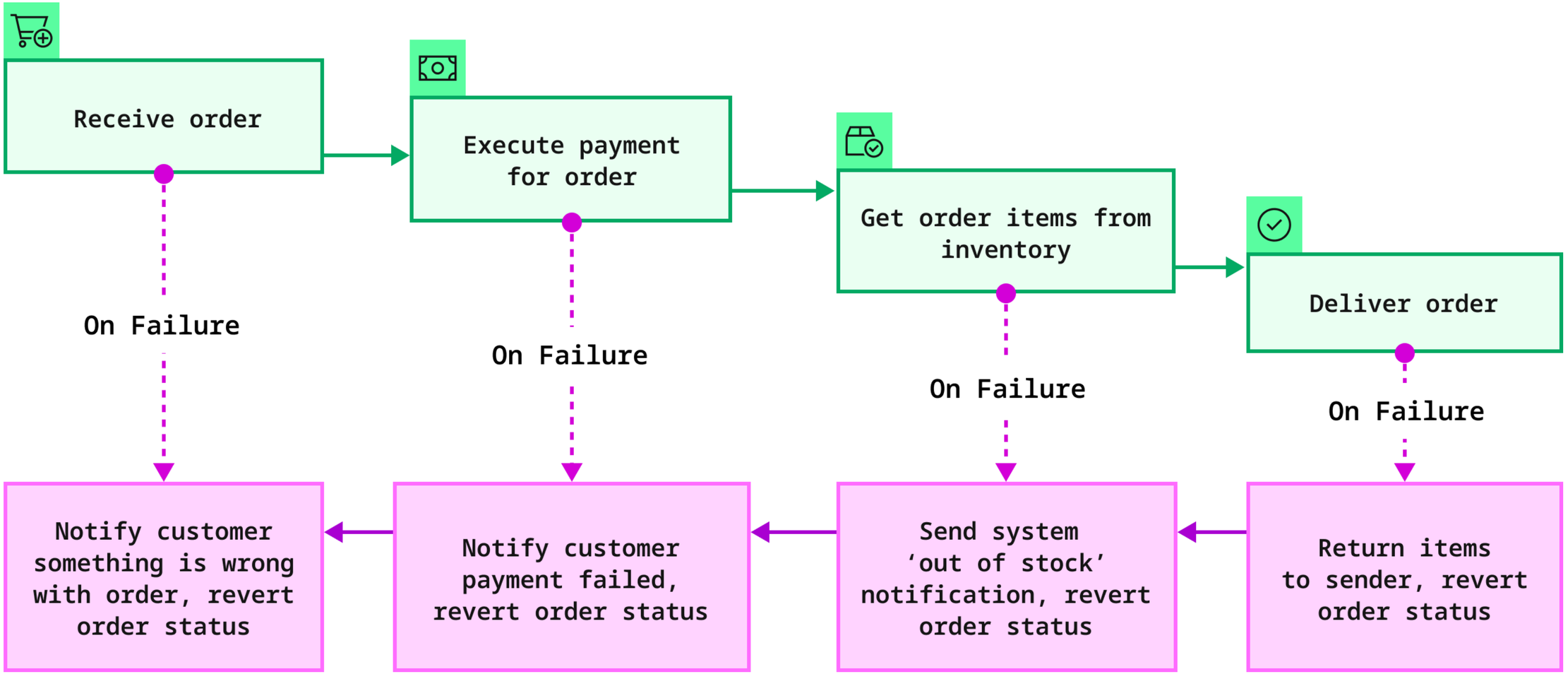
Rollback Actions and the Saga Pattern

- **A saga is a pattern used in distributed systems to manage a sequence of local transactions**

- **If any transaction in the sequence fails, the saga executes actions to rollback the previous operations.**
  - This is known as a compensating action.

- **Examples:**
  - E-Commerce Transaction
  - Distributed Data Updates

Rollback Actions and the Saga Pattern



Rollback Actions and the Saga Pattern

```
import io.temporal.workflow.Saga;

// within Workflow method

Saga saga = new Saga(new Saga.Options.Builder().build());
try {
  saga.addCompensation(activities::refundHotel, billingInfo);
  activities.bookHotel(billingInfo);
} catch (ActivityFailure e) {
  // Compensating action for booking failure
  saga.compensate();
}
```

Rollback Actions and the Saga Pattern

```
import io.temporal.workflow.Saga;

// within Workflow method

Saga saga = new Saga(new Saga.Options.Builder().build());
try {
  saga.addCompensation(activities::revertInventory, order.getItems());
  String inventoryResult = activities.updateInventory(order.getItems());


  saga.addCompensation(activities::refundBill, bill);
  String creditCardConfirmation = activities.sendBill(bill);
} catch (ActivityFailure e) {
  saga.compensate();
  throw e;
}
```

Optional Exercise #7
Implementing a Rollback Action with the Saga Pattern

- **This is an optional exercise**
  - We will do this together if time allows; otherwise, you may do it on your own later
- **During this exercise, you will**
  - Orchestrate Activities using a Saga pattern to implement compensating transactions
  - Handle failures with rollback logic
- **Refer to the README.md file in the exercise environment for details**
  - The code is below the **exercises/rollback-with-saga**
    - Make your changes to the code in the **practice** subdirectory (look for TODO comments)
    - If you need a hint or want to verify your changes, look at the complete version in the **solution** subdirectory

Beginner to Builder Bootcamp - Java

00. About this Workshop
01. The Basics of Temporal
02. Improving Your Temporal Application Code
03. Using Timers in a Workflow Definition
04. Understanding Event History
05. Understanding Workflow Determinism
06. Signals, Queries, and Workflow Updates
07. Timeouts and Retry Policies
08. Recovering from Failure
09. **Conclusion**

Conclusion (1)

Conclusion (1)

- **Temporal is an open-source Durable Execution platform**

Conclusion (1)

- **Temporal is an open-source Durable Execution platform**
  - Temporal offers SDKs for several programming languages, including Java

Conclusion (1)

- **Temporal is an open-source Durable Execution platform**
  - Temporal offers SDKs for several programming languages, including Java
- **Workflows are the core abstraction in a Temporal application**

Conclusion (1)

- **Temporal is an open-source Durable Execution platform**
  - Temporal offers SDKs for several programming languages, including Java
- **Workflows are the core abstraction in a Temporal application**
  - They are Interfaces + Implementations that have the benefit of Durable Execution

Conclusion (1)

- **Temporal is an open-source Durable Execution platform**
  - Temporal offers SDKs for several programming languages, including Java
- **Workflows are the core abstraction in a Temporal application**
  - They are Interfaces + Implementations that have the benefit of Durable Execution
  - Workflows are required to be deterministic

Conclusion (1)

- **Temporal is an open-source Durable Execution platform**
  - Temporal offers SDKs for several programming languages, including Java
- **Workflows are the core abstraction in a Temporal application**
  - They are Interfaces + Implementations that have the benefit of Durable Execution
  - Workflows are required to be deterministic
- **Activities are Interfaces + Implementations that encapsulate non-deterministic code**

Conclusion (1)

- **Temporal is an open-source Durable Execution platform**
  - Temporal offers SDKs for several programming languages, including Java
- **Workflows are the core abstraction in a Temporal application**
  - They are Interfaces + Implementations that have the benefit of Durable Execution
  - Workflows are required to be deterministic
- **Activities are Interfaces + Implementations that encapsulate non-deterministic code**
  - They are automatically retried upon failure

Conclusion (1)

- **Temporal is an open-source Durable Execution platform**
  - Temporal offers SDKs for several programming languages, including Java
- **Workflows are the core abstraction in a Temporal application**
  - They are Interfaces + Implementations that have the benefit of Durable Execution
  - Workflows are required to be deterministic
- **Activities are Interfaces + Implementations that encapsulate non-deterministic code**
  - They are automatically retried upon failure
  - You can change this behavior with a custom Retry Policy

Conclusion (2)

Conclusion (2)

- **Workers are responsible for executing your application code**

Conclusion (2)

- **Workers are responsible for executing your application code**
  - You must restart Workers after deploying a code change

Conclusion (2)

- **Workers are responsible for executing your application code**
  - You must restart Workers after deploying a code change
- **The Temporal Service orchestrates code execution**

Conclusion (2)

- **Workers are responsible for executing your application code**
  - You must restart Workers after deploying a code change
- **The Temporal Service orchestrates code execution**
  - The Temporal Service maintains dynamically-created Task Queues

Conclusion (2)

- **Workers are responsible for executing your application code**
  - You must restart Workers after deploying a code change
- **The Temporal Service orchestrates code execution**
  - The Temporal Service maintains dynamically-created Task Queues
  - Workers continuously poll a Task Queue and accept tasks if they have spare capacity

Conclusion (2)

- **Workers are responsible for executing your application code**
  - You must restart Workers after deploying a code change
- **The Temporal Service orchestrates code execution**
  - The Temporal Service maintains dynamically-created Task Queues
  - Workers continuously poll a Task Queue and accept tasks if they have spare capacity
- **The Web UI is a powerful tool for gaining insight into your application**

Conclusion (2)

- **Workers are responsible for executing your application code**
  - You must restart Workers after deploying a code change
- **The Temporal Service orchestrates code execution**
  - The Temporal Service maintains dynamically-created Task Queues
  - Workers continuously poll a Task Queue and accept tasks if they have spare capacity
- **The Web UI is a powerful tool for gaining insight into your application**
  - It displays current and recent Workflow Executions

Conclusion (2)

- **Workers are responsible for executing your application code**
  - You must restart Workers after deploying a code change
- **The Temporal Service orchestrates code execution**
  - The Temporal Service maintains dynamically-created Task Queues
  - Workers continuously poll a Task Queue and accept tasks if they have spare capacity
- **The Web UI is a powerful tool for gaining insight into your application**
  - It displays current and recent Workflow Executions
  - The Web UI shows inputs, outputs, failures, and the Event History

Conclusion (3)

Conclusion (3)

- **Timers are used to pause a Workflow Execution**

Conclusion (3)

- **Timers are used to pause a Workflow Execution**
  - These are maintained by the Temporal Service, so they will survive a Worker crash

Conclusion (3)

- **Timers are used to pause a Workflow Execution**
  - These are maintained by the Temporal Service, so they will survive a Worker crash
  - Because of Durable Execution, you can reliably use Timers with arbitrarily long durations

Conclusion (3)

- **Timers are used to pause a Workflow Execution**
  - These are maintained by the Temporal Service, so they will survive a Worker crash
  - Because of Durable Execution, you can reliably use Timers with arbitrarily long durations
- **The Event History is an append-only log detailing a Workflow Execution**

Conclusion (3)

- **Timers are used to pause a Workflow Execution**
  - These are maintained by the Temporal Service, so they will survive a Worker crash
  - Because of Durable Execution, you can reliably use Timers with arbitrarily long durations
- **The Event History is an append-only log detailing a Workflow Execution**
  - It is maintained by the Temporal Service

Conclusion (3)

- **Timers are used to pause a Workflow Execution**
  - These are maintained by the Temporal Service, so they will survive a Worker crash
  - Because of Durable Execution, you can reliably use Timers with arbitrarily long durations
- **The Event History is an append-only log detailing a Workflow Execution**
  - It is maintained by the Temporal Service
  - This data is helpful for debugging and is the key to how Temporal achieves Durable Execution

Conclusion (3)

- **Timers are used to pause a Workflow Execution**
  - These are maintained by the Temporal Service, so they will survive a Worker crash
  - Because of Durable Execution, you can reliably use Timers with arbitrarily long durations
- **The Event History is an append-only log detailing a Workflow Execution**
  - It is maintained by the Temporal Service
  - This data is helpful for debugging and is the key to how Temporal achieves Durable Execution
- **Temporal uses History Replay to achieve Durable Execution**

Conclusion (3)

- **Timers are used to pause a Workflow Execution**
  - These are maintained by the Temporal Service, so they will survive a Worker crash
  - Because of Durable Execution, you can reliably use Timers with arbitrarily long durations
- **The Event History is an append-only log detailing a Workflow Execution**
  - It is maintained by the Temporal Service
  - This data is helpful for debugging and is the key to how Temporal achieves Durable Execution
- **Temporal uses History Replay to achieve Durable Execution**
  - Essentially, the Worker selectively re-runs code based on what is in the Event History

Conclusion (4)

Conclusion (4)

- **Signals are a way of passing data to a running Workflow Execution**

Conclusion (4)

- **Signals are a way of passing data to a running Workflow Execution**
  - This can be used to change its internal state

Conclusion (4)

- **Signals are a way of passing data to a running Workflow Execution**
  - This can be used to change its internal state
- **Queries are a way of retrieving data from a Workflow Execution**

Conclusion (4)

- **Signals are a way of passing data to a running Workflow Execution**
  - This can be used to change its internal state
- **Queries are a way of retrieving data from a Workflow Execution**
  - This is used to inspect its state, but is not allowed to change it

Conclusion (4)

- **Signals are a way of passing data to a running Workflow Execution**
  - This can be used to change its internal state
- **Queries are a way of retrieving data from a Workflow Execution**
  - This is used to inspect its state, but is not allowed to change it
- **Workflow Updates are similar to combining a Signal with a Query**

Conclusion (4)

- **Signals are a way of passing data to a running Workflow Execution**
  - This can be used to change its internal state
- **Queries are a way of retrieving data from a Workflow Execution**
  - This is used to inspect its state, but is not allowed to change it
- **Workflow Updates are similar to combining a Signal with a Query**
  - It enables you to supply and retrieve data to a Workflow Execution with a single call

Conclusion (4)

- **Signals are a way of passing data to a running Workflow Execution**
  - This can be used to change its internal state
- **Queries are a way of retrieving data from a Workflow Execution**
  - This is used to inspect its state, but is not allowed to change it
- **Workflow Updates are similar to combining a Signal with a Query**
  - It enables you to supply and retrieve data to a Workflow Execution with a single call
  - Workflow Update Handlers can optionally use a Validator to reject changes

Conclusion (5)

Conclusion (5)

- **Temporal provides built-in support for Retries and Timeouts**

Conclusion (5)

- **Temporal provides built-in support for Retries and Timeouts**
  - It is common to customize Timeouts and Retry Policies for Activity Executions

Conclusion (5)

- **Temporal provides built-in support for Retries and Timeouts**
  - It is common to customize Timeouts and Retry Policies for Activity Executions
  - It is unusual and seldom recommended to customize them for Workflow Executions

Conclusion (5)

- **Temporal provides built-in support for Retries and Timeouts**
  - It is common to customize Timeouts and Retry Policies for Activity Executions
  - It is unusual and seldom recommended to customize them for Workflow Executions
  - You can use Activity Heartbeating to quickly detect failure of a long-running Activity

Conclusion (5)

- **Temporal provides built-in support for Retries and Timeouts**
  - It is common to customize Timeouts and Retry Policies for Activity Executions
  - It is unusual and seldom recommended to customize them for Workflow Executions
  - You can use Activity Heartbeating to quickly detect failure of a long-running Activity
- **Saga is a common pattern for failure recovery in distributed systems**

Conclusion (5)

- **Temporal provides built-in support for Retries and Timeouts**
  - It is common to customize Timeouts and Retry Policies for Activity Executions
  - It is unusual and seldom recommended to customize them for Workflow Executions
  - You can use Activity Heartbeating to quickly detect failure of a long-running Activity
- **Saga is a common pattern for failure recovery in distributed systems**
- **There are two options for using the Temporal Service**

Conclusion (5)

- **Temporal provides built-in support for Retries and Timeouts**
  - It is common to customize Timeouts and Retry Policies for Activity Executions
  - It is unusual and seldom recommended to customize them for Workflow Executions
  - You can use Activity Heartbeating to quickly detect failure of a long-running Activity
- **Saga is a common pattern for failure recovery in distributed systems**
- **There are two options for using the Temporal Service**
  - Self-hosted: install, configure, and operate it yourself

Conclusion (5)

- **Temporal provides built-in support for Retries and Timeouts**
  - It is common to customize Timeouts and Retry Policies for Activity Executions
  - It is unusual and seldom recommended to customize them for Workflow Executions
  - You can use Activity Heartbeating to quickly detect failure of a long-running Activity
- **Saga is a common pattern for failure recovery in distributed systems**
- **There are two options for using the Temporal Service**
  - Self-hosted: install, configure, and operate it yourself
  - Temporal Cloud: a fully-managed service that’s an alternative to self-hosting

Temporal Service Options for Production

Temporal Service Options for Production

- **Self-Hosted**

Temporal Service Options for Production

- **Self-Hosted**
  - The temporal CLI provides an easy way to run the Temporal Service for development

Temporal Service Options for Production

- **Self-Hosted**
  - The temporal CLI provides an easy way to run the Temporal Service for development
  - Using Docker Compose is another alternative for small-scale development

Temporal Service Options for Production

- **Self-Hosted**
  - The temporal CLI provides an easy way to run the Temporal Service for development
  - Using Docker Compose is another alternative for small-scale development
  - Production deployments often run on Kubernetes

Temporal Service Options for Production

- **Self-Hosted**
  - The temporal CLI provides an easy way to run the Temporal Service for development
  - Using Docker Compose is another alternative for small-scale development
  - Production deployments often run on Kubernetes
- **Temporal Cloud**

Temporal Service Options for Production

- **Self-Hosted**
  - The temporal CLI provides an easy way to run the Temporal Service for development
  - Using Docker Compose is another alternative for small-scale development
  - Production deployments often run on Kubernetes
- **Temporal Cloud**
  - Access to a Temporal Service run by experts via our fully-managed cloud service

Temporal Service Options for Production

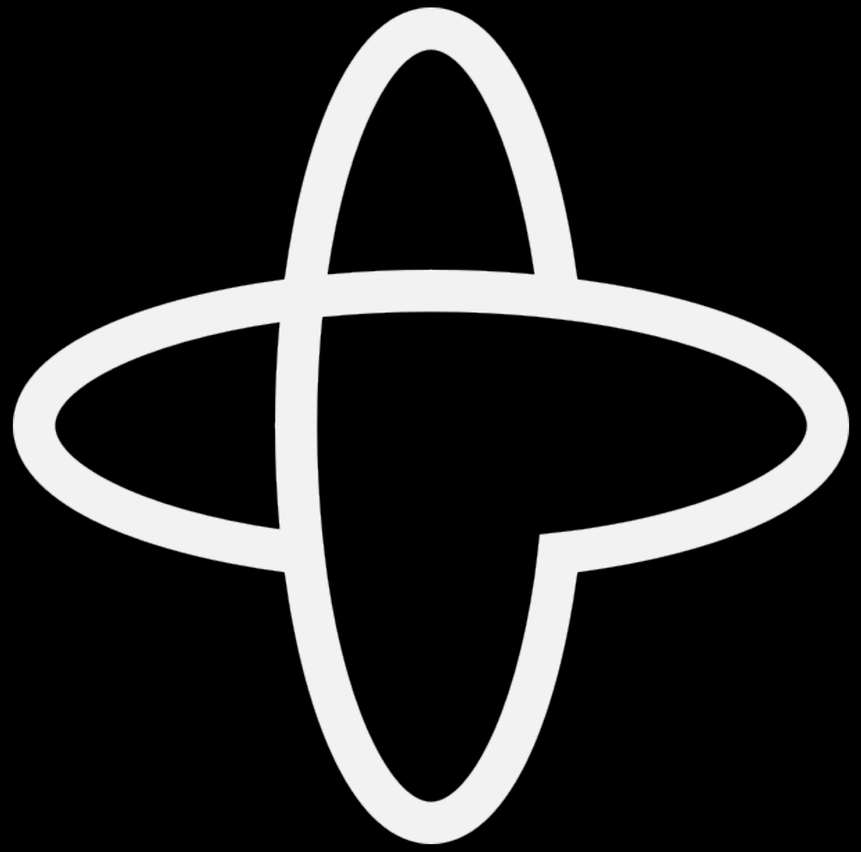
- **Self-Hosted**
  - The temporal CLI provides an easy way to run the Temporal Service for development
  - Using Docker Compose is another alternative for small-scale development
  - Production deployments often run on Kubernetes
- **Temporal Cloud**
  - Access to a Temporal Service run by experts via our fully-managed cloud service
    - Dependable: 99.9% uptime SLA and 24x7 production support

Temporal Service Options for Production

- **Self-Hosted**
  - The temporal CLI provides an easy way to run the Temporal Service for development
  - Using Docker Compose is another alternative for small-scale development
  - Production deployments often run on Kubernetes
- **Temporal Cloud**
  - Access to a Temporal Service run by experts via our fully-managed cloud service
    - Dependable: 99.9% uptime SLA and 24x7 production support
    - Frees your organization from having to plan, deploy, operate, and support a self-hosted Temporal Service

Temporal Service Options for Production

- **Self-Hosted**
  - The temporal CLI provides an easy way to run the Temporal Service for development
  - Using Docker Compose is another alternative for small-scale development
  - Production deployments often run on Kubernetes
- **Temporal Cloud**
  - Access to a Temporal Service run by experts via our fully-managed cloud service
    - Dependable: 99.9% uptime SLA and 24x7 production support
    - Frees your organization from having to plan, deploy, operate, and support a self-hosted Temporal Service
  - Your application runs on your own infrastructure



Thank You