Мои любимые возможности в Kotlin

Субъективный взгляд на возможности языка Kotlin

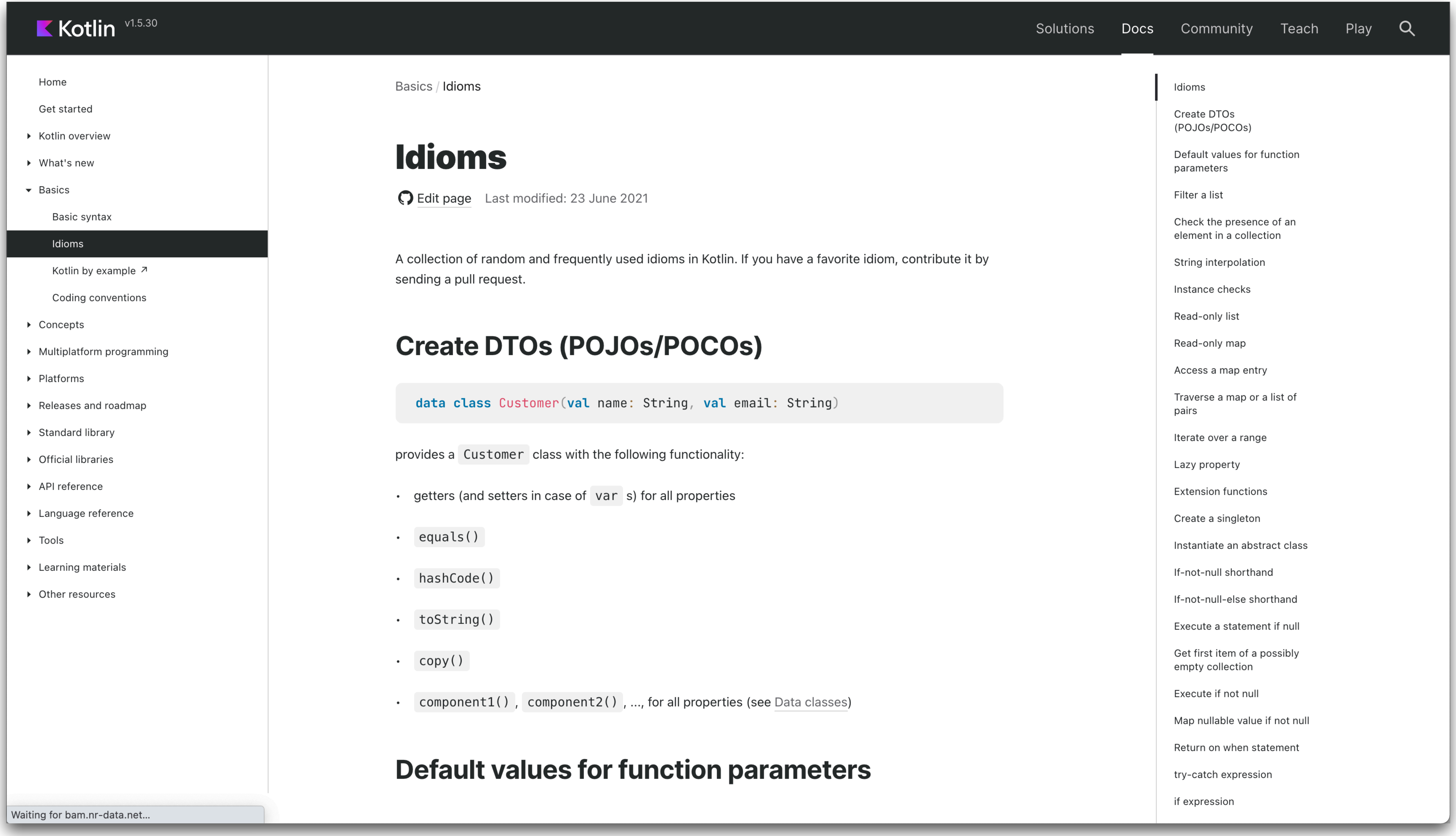
@antonarhipov

Антон Архипов

@antonarhipov

Developer Advocate @ JetBrains

Java™ Champions



Basics / Idioms

Idioms

Edit page Last modified: 23 June 2021

A collection of random and frequently used idioms in Kotlin. If you have a favorite idiom, contribute it by sending a pull request.

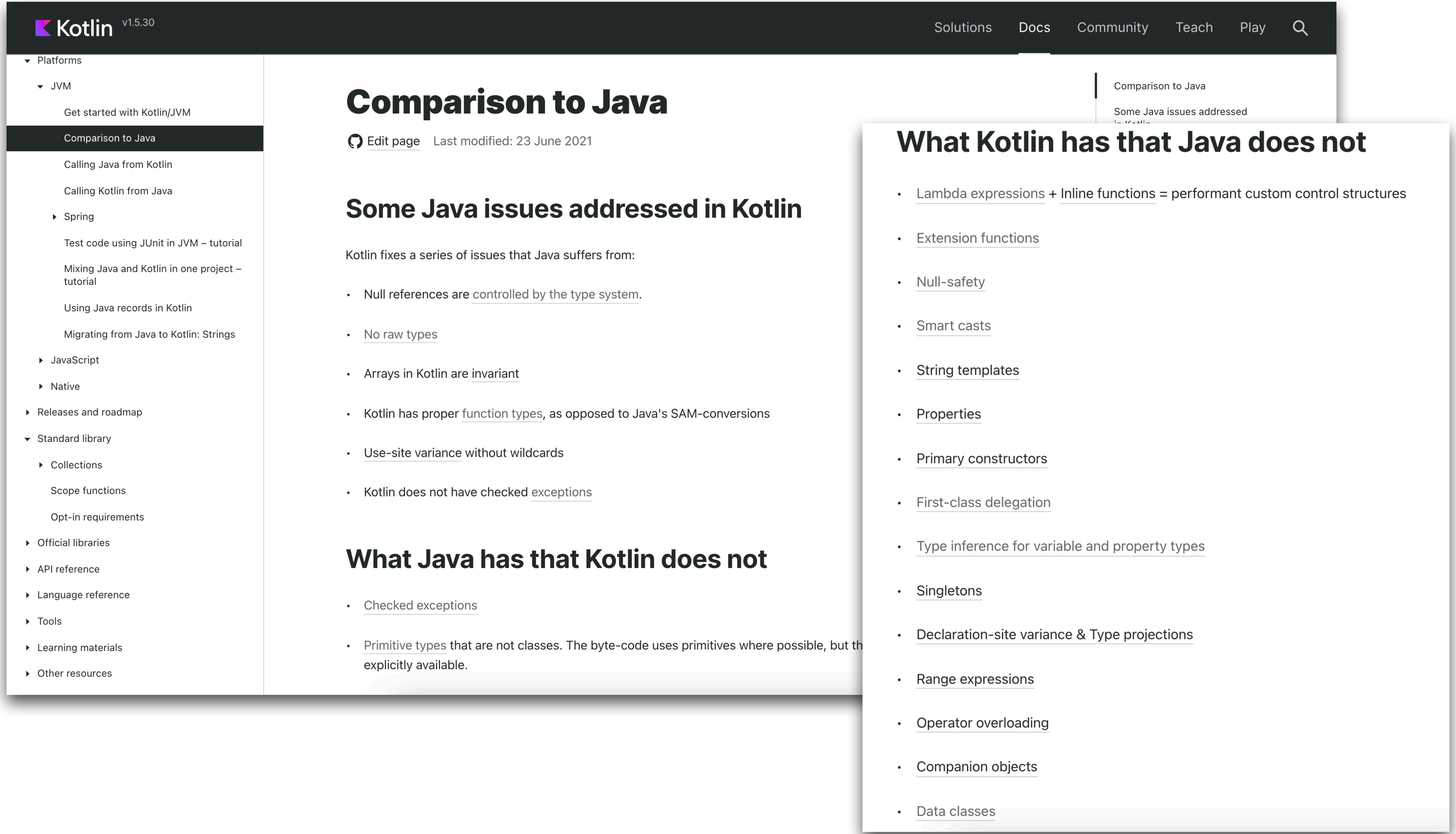
Create DTOs (POJOs/POCOs)

```
data class Customer(val name: String, val email: String)
```

provides a `Customer` class with the following functionality:

- getters (and setters in case of `var` s) for all properties
- `equals()`
- `hashCode()`
- `toString()`
- `copy()`
- `component1()` , `component2()` , ..., for all properties (see Data classes)

Default values for function parameters



- Platforms
  - JVM
    - Get started with Kotlin/JVM
    - Comparison to Java
    - Calling Java from Kotlin
    - Calling Kotlin from Java
    - Spring
    - Test code using JUnit in JVM – tutorial
    - Mixing Java and Kotlin in one project – tutorial
    - Using Java records in Kotlin
    - Migrating from Java to Kotlin: Strings
  - JavaScript
  - Native
- Releases and roadmap
- Standard library
  - Collections
  - Scope functions
  - Opt-in requirements
- Official libraries
- API reference
- Language reference
- Tools
- Learning materials
- Other resources

Comparison to Java

Edit page Last modified: 23 June 2021

Some Java issues addressed in Kotlin

Kotlin fixes a series of issues that Java suffers from:

- Null references are controlled by the type system.
- No raw types
- Arrays in Kotlin are invariant
- Kotlin has proper function types, as opposed to Java's SAM-conversions
- Use-site variance without wildcards
- Kotlin does not have checked exceptions

What Java has that Kotlin does not

- Checked exceptions
- Primitive types that are not classes. The byte-code uses primitives where possible, but th explicitly available.

Comparison to Java

Some Java issues addressed

What Kotlin has that Java does not

- Lambda expressions + Inline functions = performant custom control structures
- Extension functions
- Null-safety
- Smart casts
- String templates
- Properties
- Primary constructors
- First-class delegation
- Type inference for variable and property types
- Singletons
- Declaration-site variance & Type projections
- Range expressions
- Operator overloading
- Companion objects
- Data classes

1. Функции

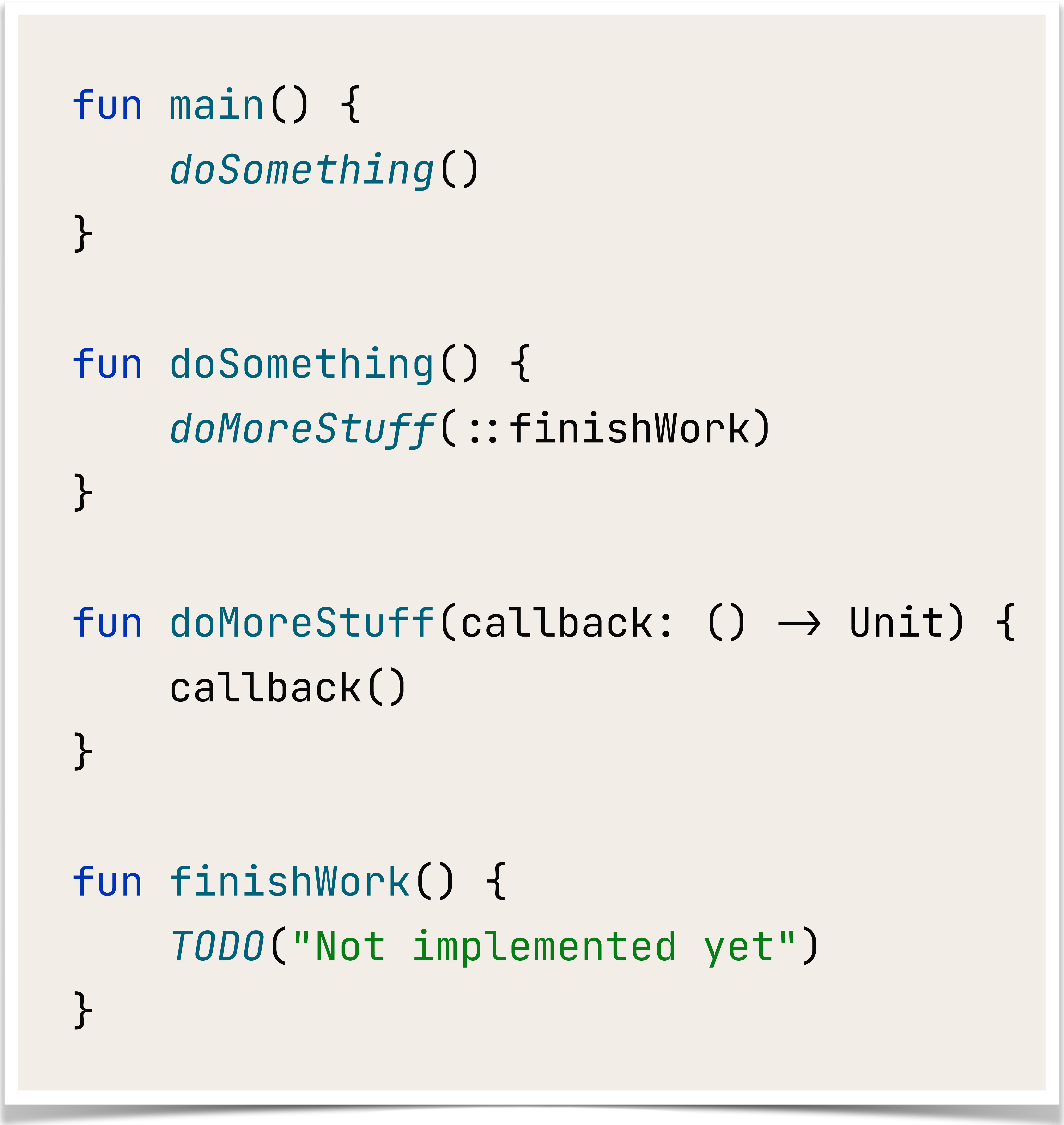
```kotlin
fun main() {
    doSomething()
}

fun doSomething() {

}
```

```kotlin
fun main() {
    doSomething()
}

fun doSomething() {
    doMoreStuff(::finishWork)
}
```

```kotlin
fun main() {
    doSomething()
}

fun doSomething() {
    doMoreStuff(::finishWork)
}

fun doMoreStuff(callback: () -> Unit) {
    callback()
}
```

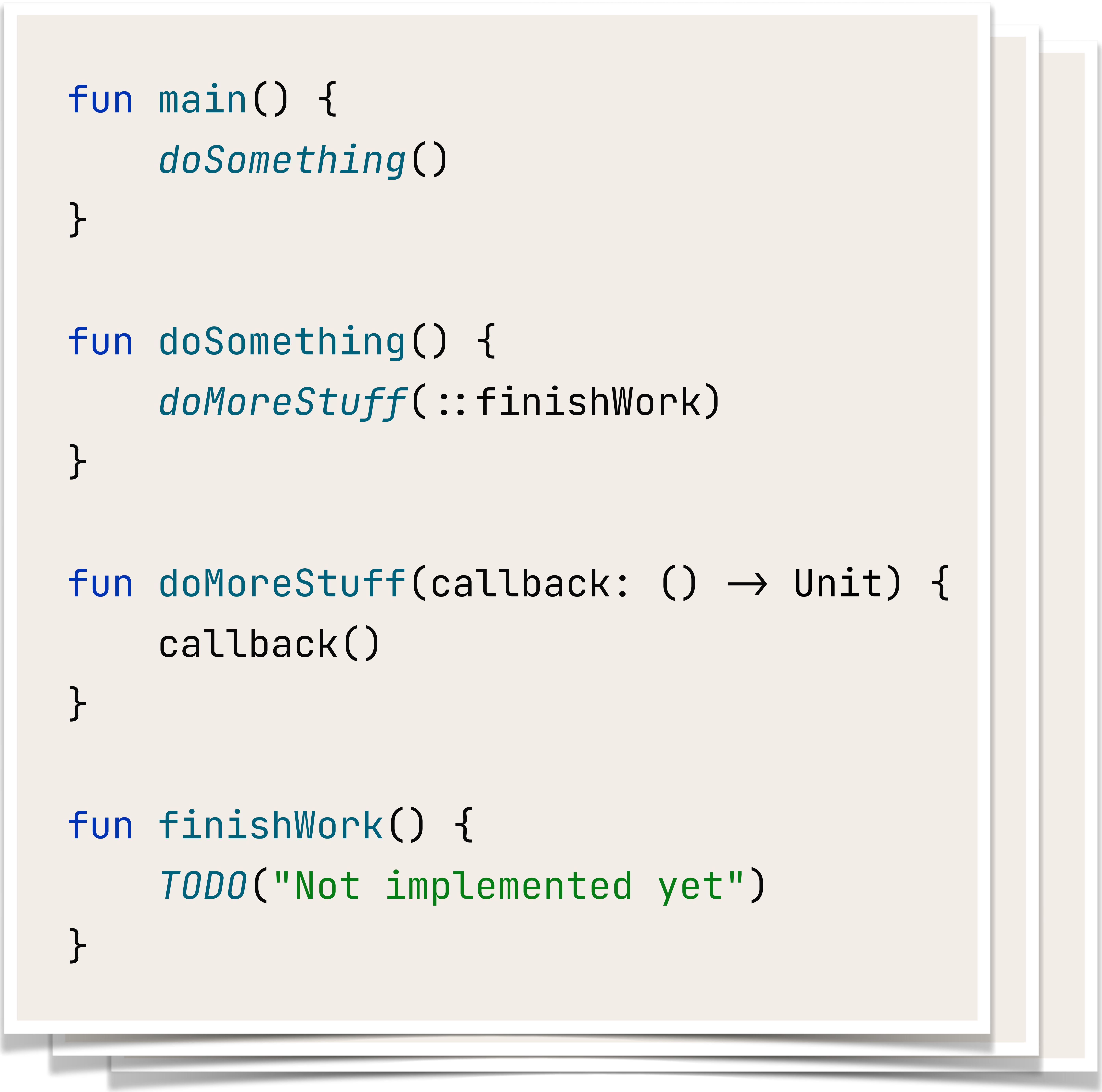
```kotlin
fun main() {
    doSomething()
}

fun doSomething() {
    doMoreStuff(::finishWork)
}

fun doMoreStuff(callback: () -> Unit) {
    callback()
}

fun finishWork() {
    TODO("Not implemented yet")
}
```

Просто функции, никаких классов!



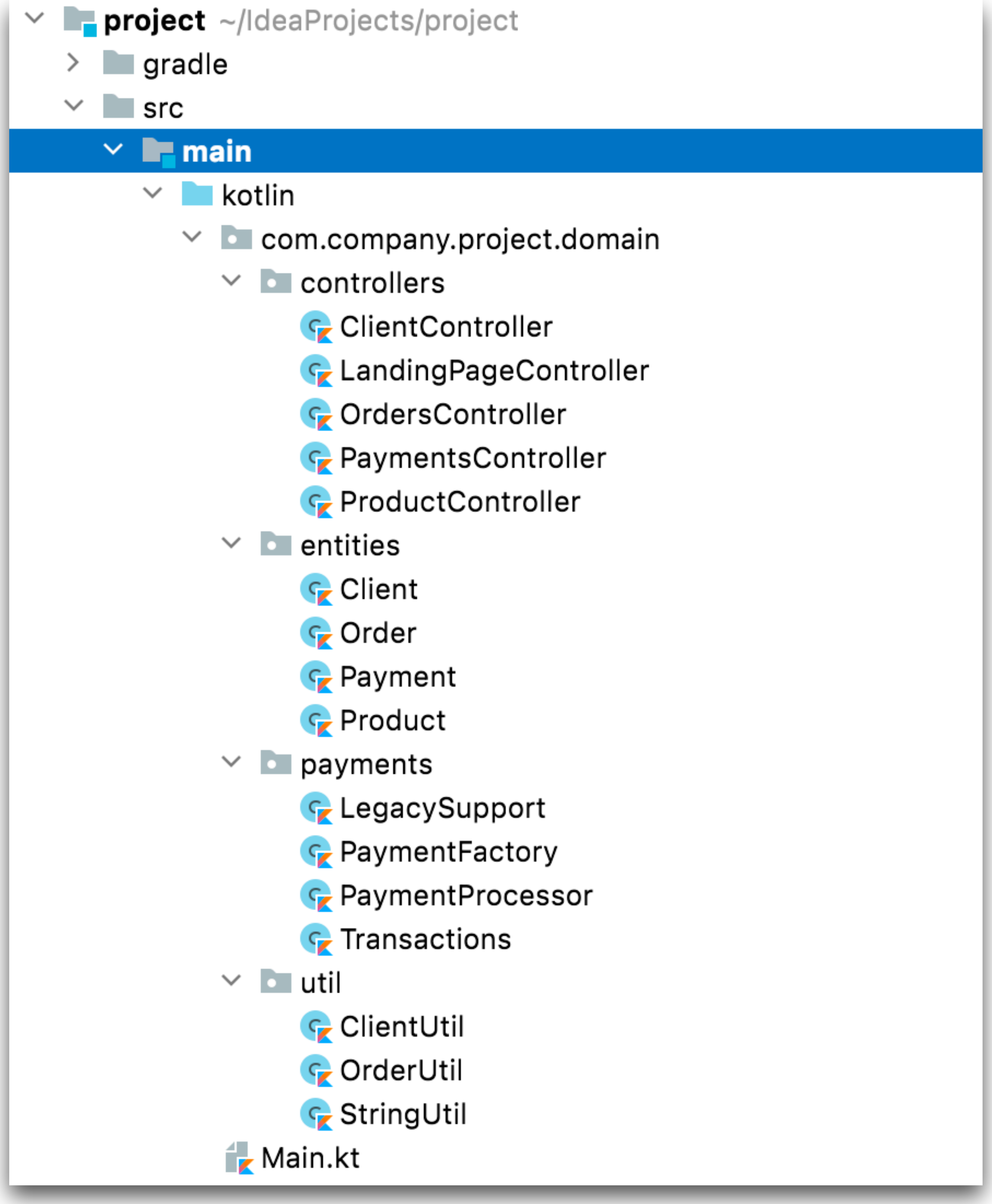
```kotlin
fun main() {
    doSomething()
}

fun doSomething() {
    doMoreStuff(::finishWork)
}

fun doMoreStuff(callback: () -> Unit) {
    callback()
}

fun finishWork() {
    TODO("Not implemented yet")
}
```

Просто функции в файле ...



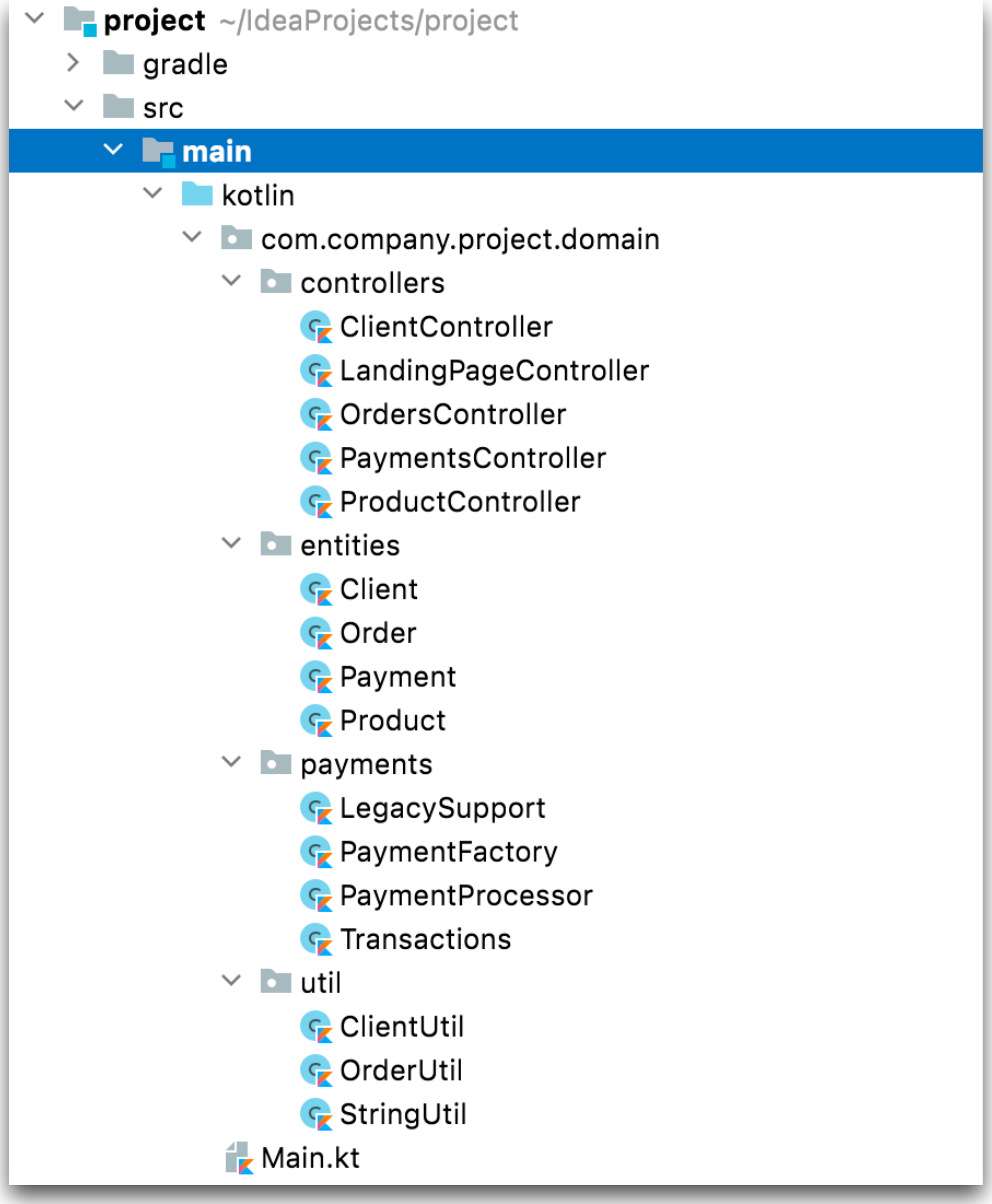
```kotlin
fun main() {
    doSomething()
}

fun doSomething() {
    doMoreStuff(::finishWork)
}

fun doMoreStuff(callback: () -> Unit) {
    callback()
}

fun finishWork() {
    TODO("Not implemented yet")
}
```

Группировка по файлам...

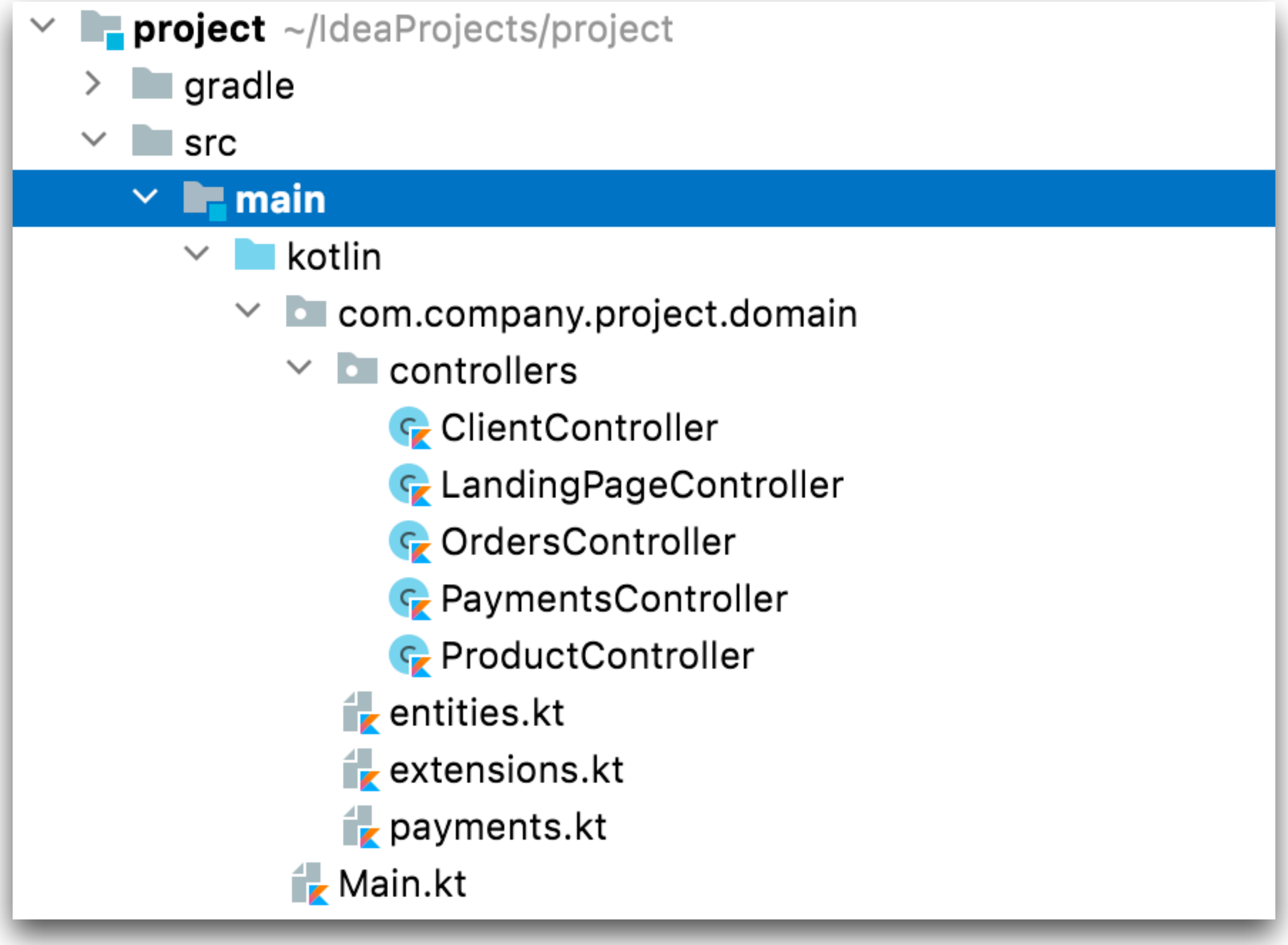


```
project ~/IdeaProjects/project
  gradle
  src
    main
      kotlin
        com.company.project.domain
          controllers
            ClientController
            LandingPageController
            OrdersController
            PaymentsController
            ProductController
          entities
            Client
            Order
            Payment
            Product
          payments
            LegacySupport
            PaymentFactory
            PaymentProcessor
            Transactions
          util
            ClientUtil
            OrderUtil
            StringUtil
        Main.kt
```

Классическая структура проекта

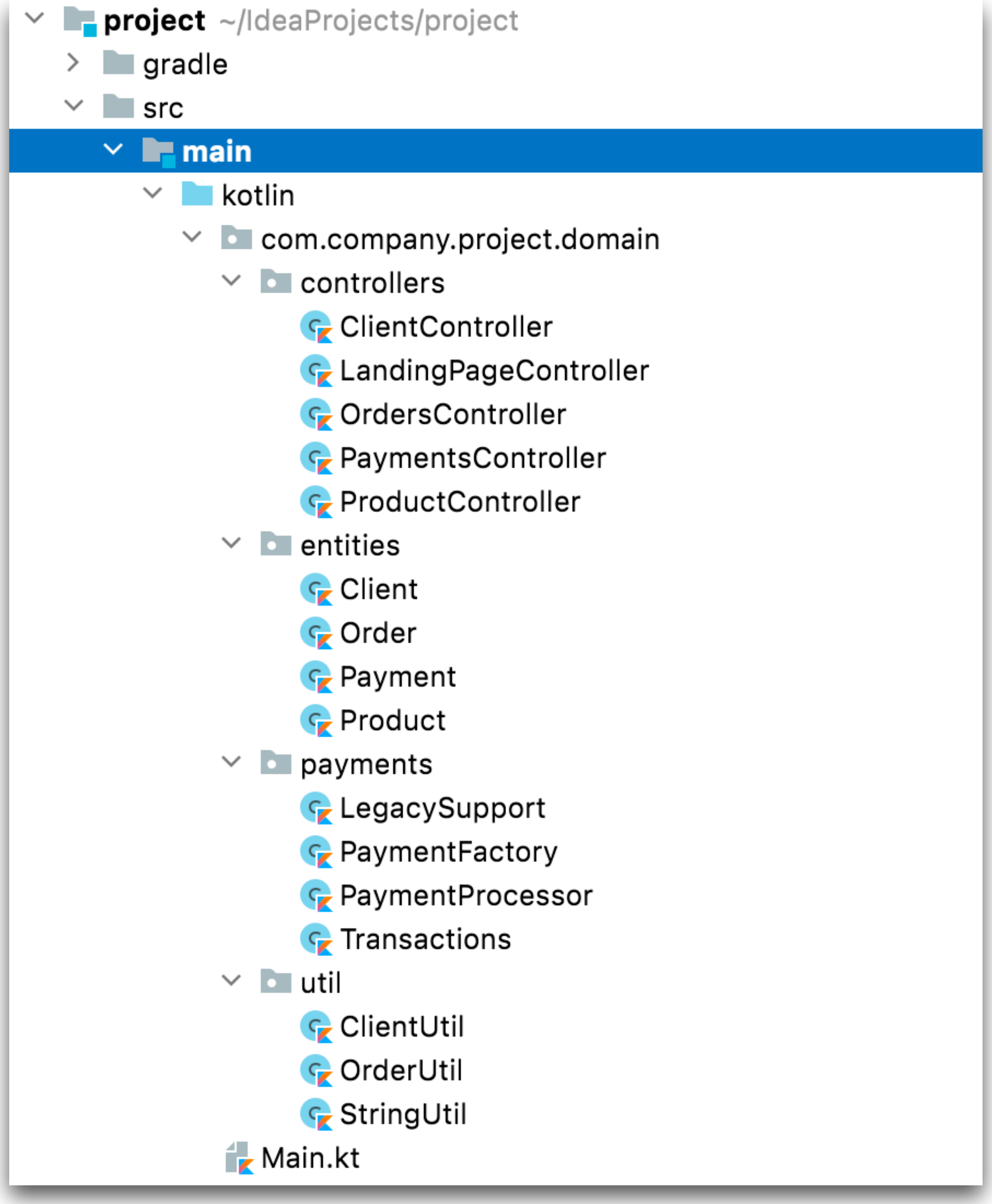


```
project ~/IdeaProjects/project
  gradle
  src
    main
      kotlin
        com.company.project.domain
          controllers
            ClientController
            LandingPageController
            OrdersController
            PaymentsController
            ProductController
          entities
            Client
            Order
            Payment
            Product
          payments
            LegacySupport
            PaymentFactory
            PaymentProcessor
            Transactions
          util
            ClientUtil
            OrderUtil
            StringUtil
        Main.kt
```

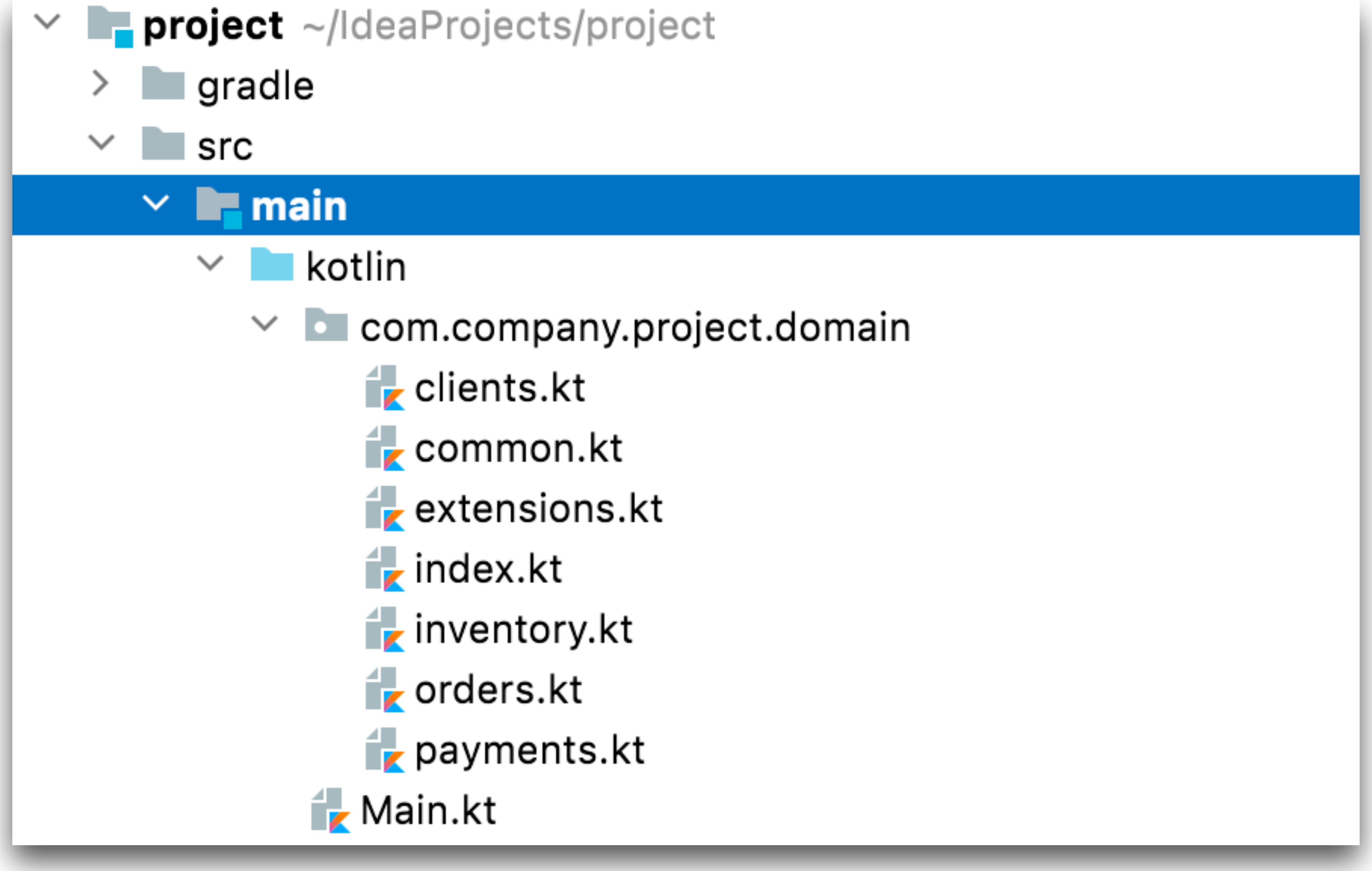


```
project ~/IdeaProjects/project
  gradle
  src
    main
      kotlin
        com.company.project.domain
          controllers
            ClientController
            LandingPageController
            OrdersController
            PaymentsController
            ProductController
          entities.kt
          extensions.kt
          payments.kt
        Main.kt
```

Структура проекта значительно упрощается



```
project ~/IdeaProjects/project
  gradle
  src
    main
      kotlin
        com.company.project.domain
          controllers
            ClientController
            LandingPageController
            OrdersController
            PaymentsController
            ProductController
          entities
            Client
            Order
            Payment
            Product
          payments
            LegacySupport
            PaymentFactory
            PaymentProcessor
            Transactions
          util
            ClientUtil
            OrderUtil
            StringUtil
        Main.kt
```



```
project ~/IdeaProjects/project
  gradle
  src
    main
      kotlin
        com.company.project.domain
          clients.kt
          common.kt
          extensions.kt
          index.kt
          inventory.kt
          orders.kt
          payments.kt
        Main.kt
```

Можно и ещё упростить :)

Вывод 1:

Функции - это фундаментальная деталь для упрощения программ

2. Расширения

```kotlin
class Client(val name: String, val company: String, val twitter: String)

println(Client("Anton", "JetBrains", "@antonarhipov"))
```

```kotlin
class Client(val name: String, val company: String, val twitter: String)

println(Client("Anton", "JetBrains", "@antonarhipov"))

> Client@3ac3fd8b
```

```
class Client(val name: String, val company: String, val twitter: String)

println(toConsole(Client(“Anton", "JetBrains", “@antonarhipov")))

fun toConsole(client: Client) =
    "Client[name=${client.name}, ${client.company}, ${client.twitter}]"
```

```kotlin
class Client(val name: String, val company: String, val twitter: String)

println(toConsole(Client(“Anton", "JetBrains", “@antonarhipov")))

fun toConsole(client: Client) =
    "Client[name=${client.name}, ${client.company}, ${client.twitter}]"
```

```
> Client[name=Anton, JetBrains, @antonarhipov]
```

```
class Client(val name: String, val company: String, val twitter: String)

println(Client("Anton", "JetBrains", “@antonarhipov”).toConsole())

fun Client.toConsole() =
    "Client[name=${name}, ${company}, ${twitter}]"

> Client[name=Anton, JetBrains, @antonarhipov]
```

```
class Client(val name: String, val company: String, val twitter: String)

println(Client("Anton", "JetBrains", “@antonarhipov”).toConsole)

val Client.toConsole: String
    get() = "Client[name=${name}, ${company}, ${twitter}]"

> Client[name=Anton, JetBrains, @antonarhipov]
```

```kotlin
fun findMessageById(id: String) = db.query(
    "select * from messages where id = ?",
    RowMapper { rs, _ ->
        Message(rs.getString("id"), rs.getString("text"))
    },
    id
)
```

```
@Override
public <T> List<T> query(String sql, RowMapper<T> rowMapper, @Nullable Object... args)
      throws DataAccessException {
   return result(query(sql, args, new RowMapperResultSetExtractor<>(rowMapper)));
}
```

```
fun findMessageById(id: String) = db.query(
    "select * from messages where id = ?",
    RowMapper { rs, _ ->
        Message(rs.getString("id"), rs.getString("text"))
    },
    id
)
```

```kotlin
fun findMessageById(id: String) = db.query(
    "select * from messages where id = ?",
    RowMapper { rs, _ ->
        Message(rs.getString("id"), rs.getString("text"))
    },
    id
)
```

```
fun findMessageById(id: String) = db.query(
    """select * from messages where id = ?
       XXXXXXXX XXXXXXXX XXXXXXX XXXXXXXX XXXXXXX XXXXXX
       XXXXXXXXXXXX XXXXXXXXXXXX XXXXXXX XXX XXX XXXX XXX
       XXXXXXXXXXXXXXXXXXXXXXX XXXX XX XX XX XXXXX XX XX
       XXXX XXX XX X XX XXX XXXXXX XXXX
       XXX XXXXXXXXXXXXXXX XXXXXXXXXXX XXXXXXXXX
    """,
    RowMapper { rs, _ →
        Message(
          rs.getString("id"),
          rs.getString("text"))
          rs.getString("xxxxx"))
          rs.getString("xxxxx"))
          rs.getString("xxxxx"))
    },
    id
```

```
fun findMessageById(id: String) = db.query(
    """select * from messages where id = ?
       XXXXXXXX XXXXXXXX XXXXXXX XXXXXXXX XXXXXXX XXXXXX
       XXXXXXXXXXXX XXXXXXXXXXXX XXXXXXX XXX XXX XXXX XXX
       XXXXXXXXXXXXXXXXXXXXXXX XXXX XX XX XX XXXXX XX XX
       XXXX XXX XX X XX XXX XXXXXX XXXX
       XXX XXXXXXXXXXXXXXX XXXXXXXXXXX XXXXXXXXX
    """,
    RowMapper { rs, _ →
        Message(
          rs.getString("id"),
          rs.getString("text"))
          rs.getString("xxxxx"))
          rs.getString("xxxxx"))
          rs.getString("xxxxx"))
    },
    id
```

Параметр находится очень далеко от самого запроса!

```
fun findMessageById(id: String) = db.query(
    "select * from messages where id = ?",
    RowMapper { rs, _ →
        Message(rs.getString("id"), rs.getString("text"))
    },
    id
)
```

```
fun findMessageById(id: String) = db.query(
    "select * from messages where id = ?",
    id,
    RowMapper { rs, _ ->
        Message(rs.getString("id"), rs.getString("text"))
    }
)
```

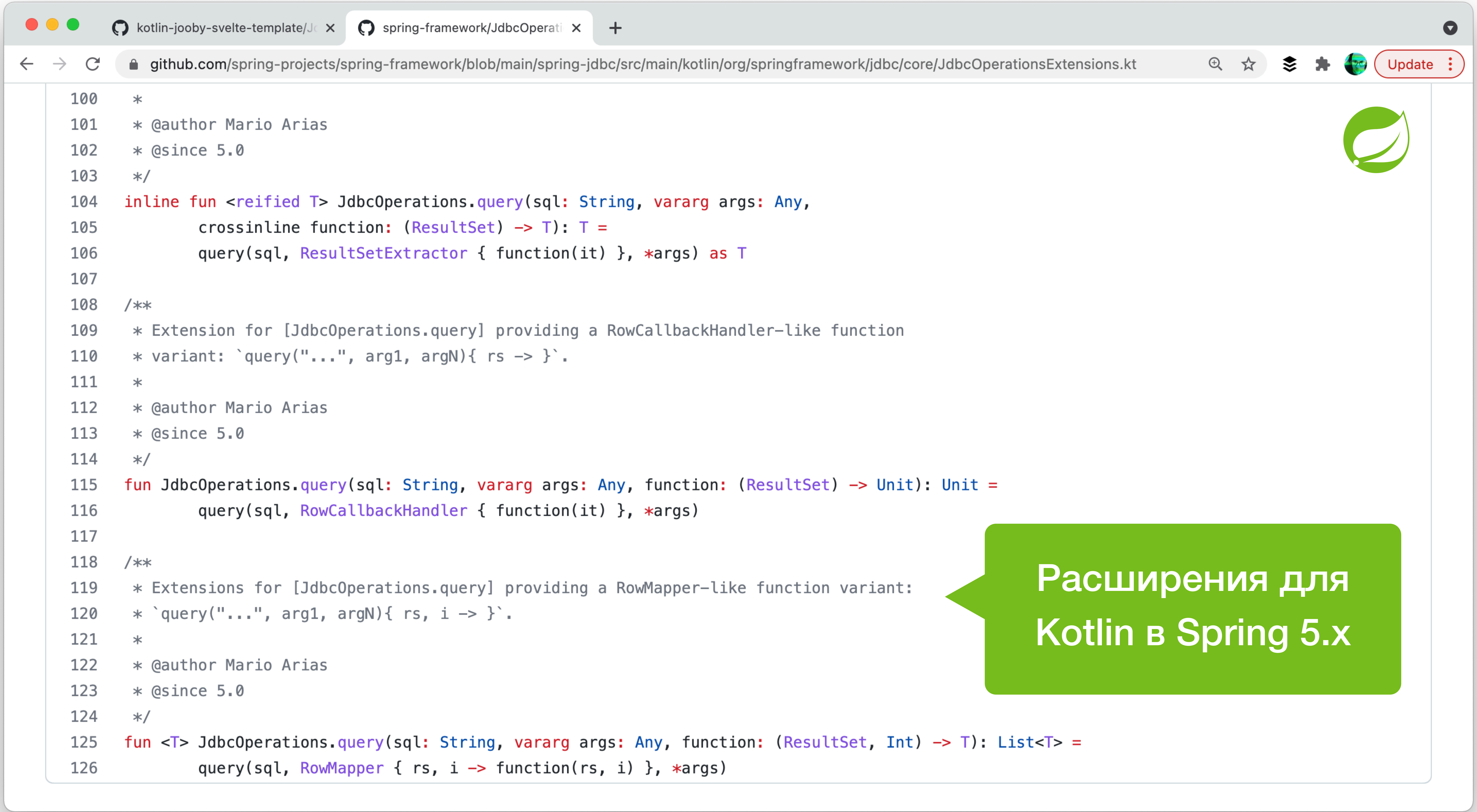
```kotlin
fun findMessageById(id: String) = db.query(
    "select * from messages where id = ?",
    id,
    { rs, _ ->
        Message(rs.getString("id"), rs.getString("text"))
    }
)
```

```
fun findMessageById(id: String) = db.query(
    "select * from messages where id = ?",
    id)
    { rs, _ ->
        Message(rs.getString("id"), rs.getString("text"))
    }
```

```kotlin
fun findMessageById(id: String) = db.query(
    "select * from messages where id = ?", id) { rs, _ ->
    Message(rs.getString("id"), rs.getString("text"))
}
```

```
fun <T> JdbcOperations.query(sql: String, vararg args: Any, function: (ResultSet, Int) -> T): List<T> =
    query(sql, RowMapper { rs, i -> function(rs, i) }, *args)
```

```
fun findMessageById(id: String) = db.query(
    "select * from messages where id = ?", id) { rs, _ ->
    Message(rs.getString("id"), rs.getString("text"))
}
```


kotlin-jooby-svelte-template/J
spring-framework/JdbcOperati
github.com/spring-projects/spring-framework/blob/main/spring-jdbc/src/main/kotlin/org/springframework/jdbc/core/JdbcOperationsExtensions.kt
Update

```kotlin
 *
 * @author Mario Arias
 * @since 5.0
 */
inline fun <reified T> JdbcOperations.query(sql: String, vararg args: Any,
        crossinline function: (ResultSet) -> T): T =
        query(sql, ResultSetExtractor { function(it) }, *args) as T

/**
 * Extension for [JdbcOperations.query] providing a RowCallbackHandler-like function
 * variant: `query("...", arg1, argN){ rs -> }`.
 *
 * @author Mario Arias
 * @since 5.0
 */
fun JdbcOperations.query(sql: String, vararg args: Any, function: (ResultSet) -> Unit): Unit =
        query(sql, RowCallbackHandler { function(it) }, *args)

/**
 * Extensions for [JdbcOperations.query] providing a RowMapper-like function variant:
 * `query("...", arg1, argN){ rs, i -> }`.
 *
 * @author Mario Arias
 * @since 5.0
 */
fun <T> JdbcOperations.query(sql: String, vararg args: Any, function: (ResultSet, Int) -> T): List<T> =
        query(sql, RowMapper { rs, i -> function(rs, i) }, *args)
```

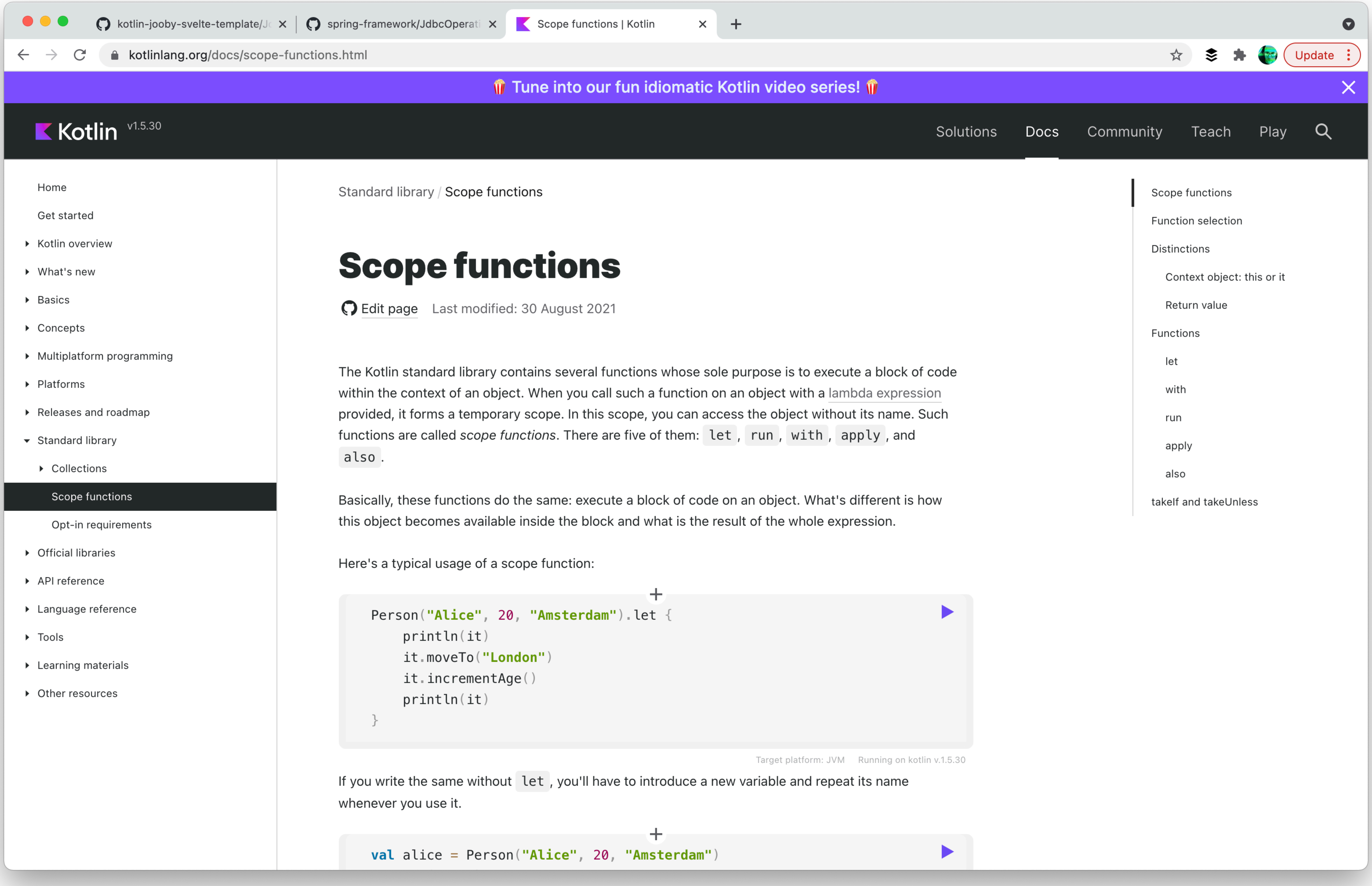
Расширения для Kotlin в Spring 5.x

Вывод 2:

Расширения улучшают читаемость кода
и позволяют адоптировать сторонние библиотеки

3. Контекстные функции

apply, let, run, also, with



Standard library / Scope functions

Scope functions

Edit page Last modified: 30 August 2021

The Kotlin standard library contains several functions whose sole purpose is to execute a block of code within the context of an object. When you call such a function on an object with a lambda expression provided, it forms a temporary scope. In this scope, you can access the object without its name. Such functions are called *scope functions*. There are five of them: `let`, `run`, `with`, `apply`, and `also`.

Basically, these functions do the same: execute a block of code on an object. What's different is how this object becomes available inside the block and what is the result of the whole expression.

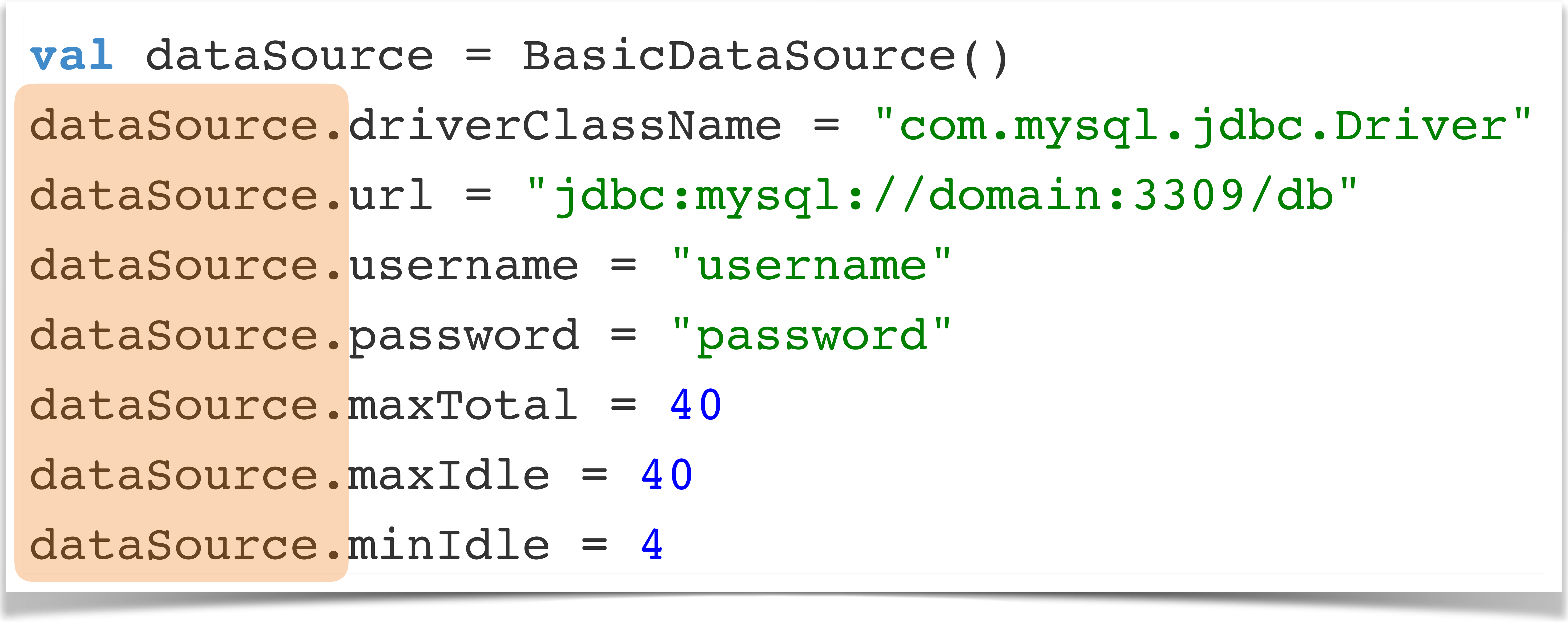
Here's a typical usage of a scope function:

```
Person("Alice", 20, "Amsterdam").let {
    println(it)
    it.moveTo("London")
    it.incrementAge()
    println(it)
}
```

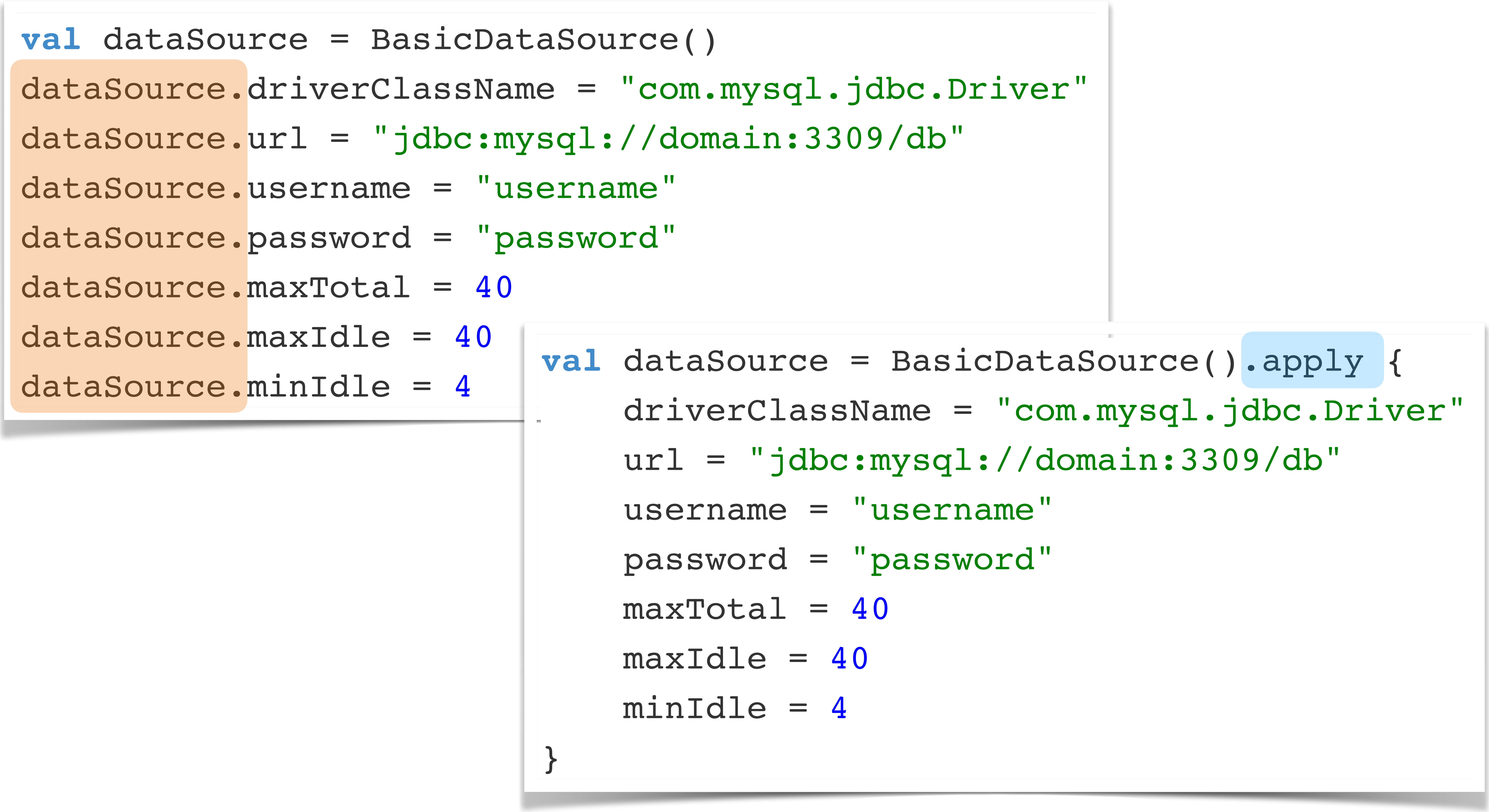
Target platform: JVM Running on kotlin v.1.5.30

If you write the same without `let`, you'll have to introduce a new variable and repeat its name whenever you use it.

```
val alice = Person("Alice", 20, "Amsterdam")
```



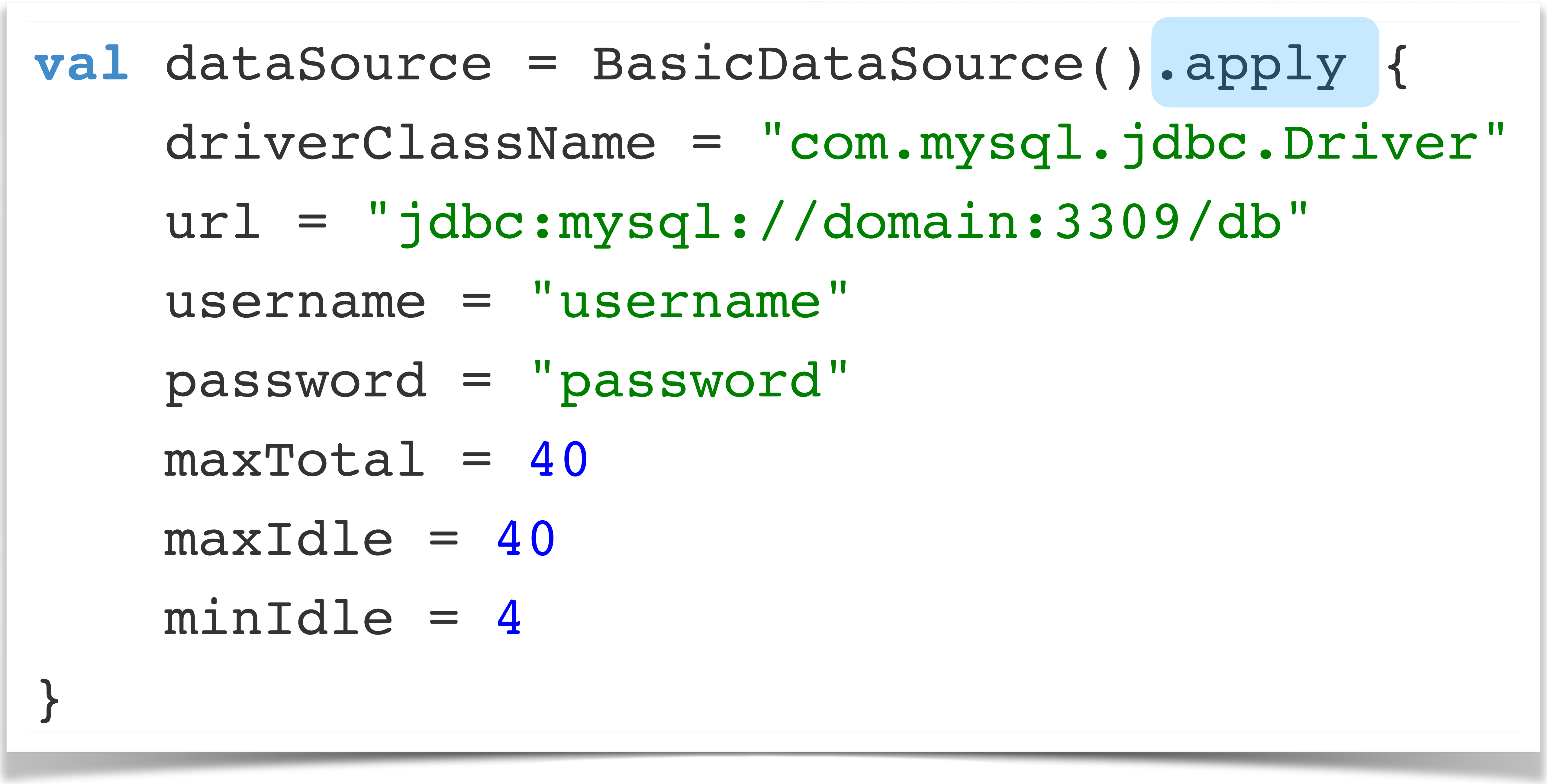
```
val dataSource = BasicDataSource()
dataSource.driverClassName = "com.mysql.jdbc.Driver"
dataSource.url = "jdbc:mysql://domain:3309/db"
dataSource.username = "username"
dataSource.password = "password"
dataSource.maxTotal = 40
dataSource.maxIdle = 40
dataSource.minIdle = 4
```



```kotlin
val dataSource = BasicDataSource()
dataSource.driverClassName = "com.mysql.jdbc.Driver"
dataSource.url = "jdbc:mysql://domain:3309/db"
dataSource.username = "username"
dataSource.password = "password"
dataSource.maxTotal = 40
dataSource.maxIdle = 40
dataSource.minIdle = 4
```

```kotlin
val dataSource = BasicDataSource().apply {
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://domain:3309/db"
    username = "username"
    password = "password"
    maxTotal = 40
    maxIdle = 40
    minIdle = 4
}
```

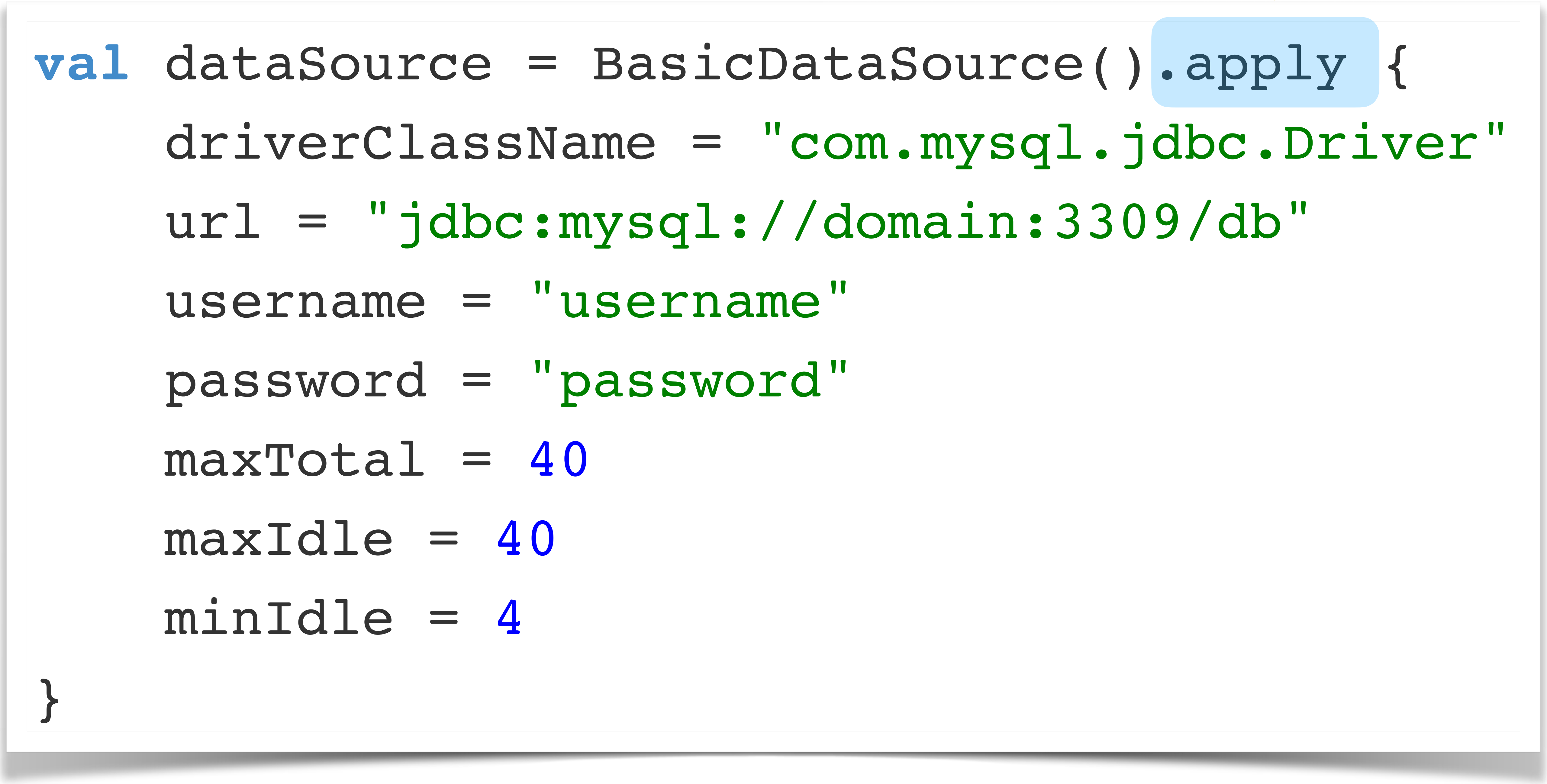
```
public inline fun <T> T.apply(block: T.() -> Unit): T {
    block()
    return this
}
```



```
val dataSource = BasicDataSource().apply {
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://domain:3309/db"
    username = "username"
    password = "password"
    maxTotal = 40
    maxIdle = 40
    minIdle = 4
}
```

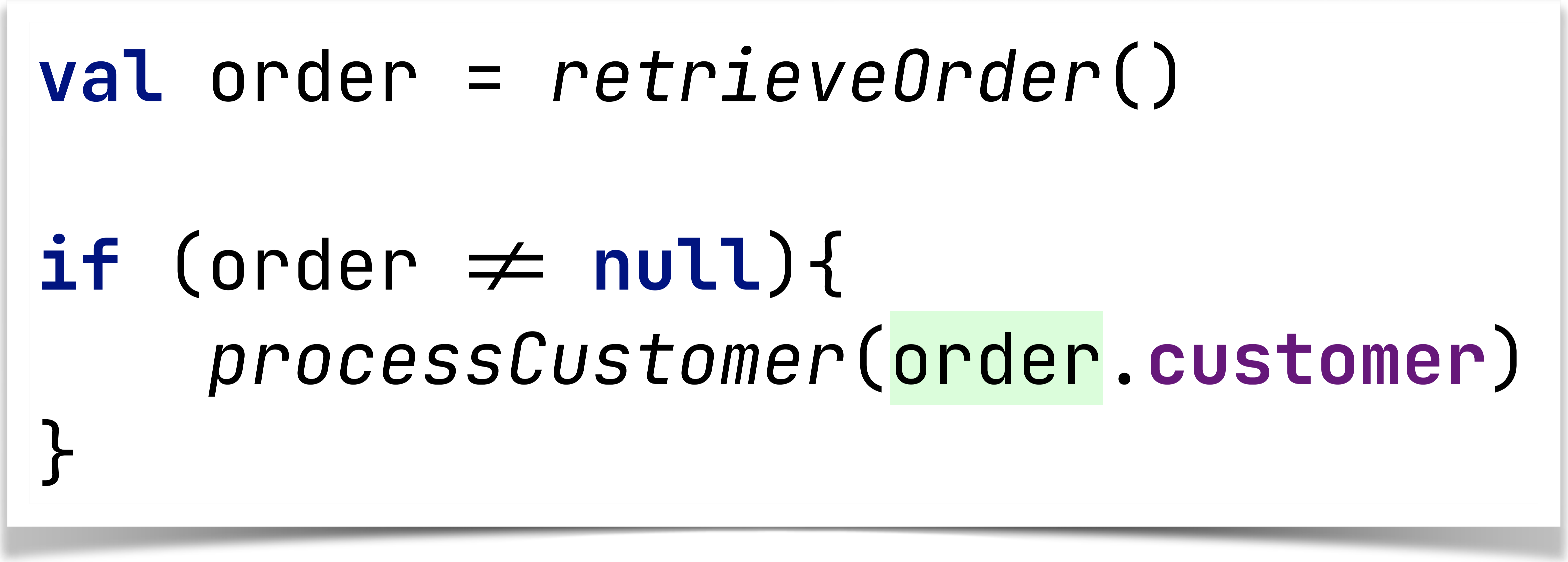
```
public inline fun <T> T.apply(block: T.() -> Unit): T {
    block()
    return this
}
```

“Лямбда с ресивером”



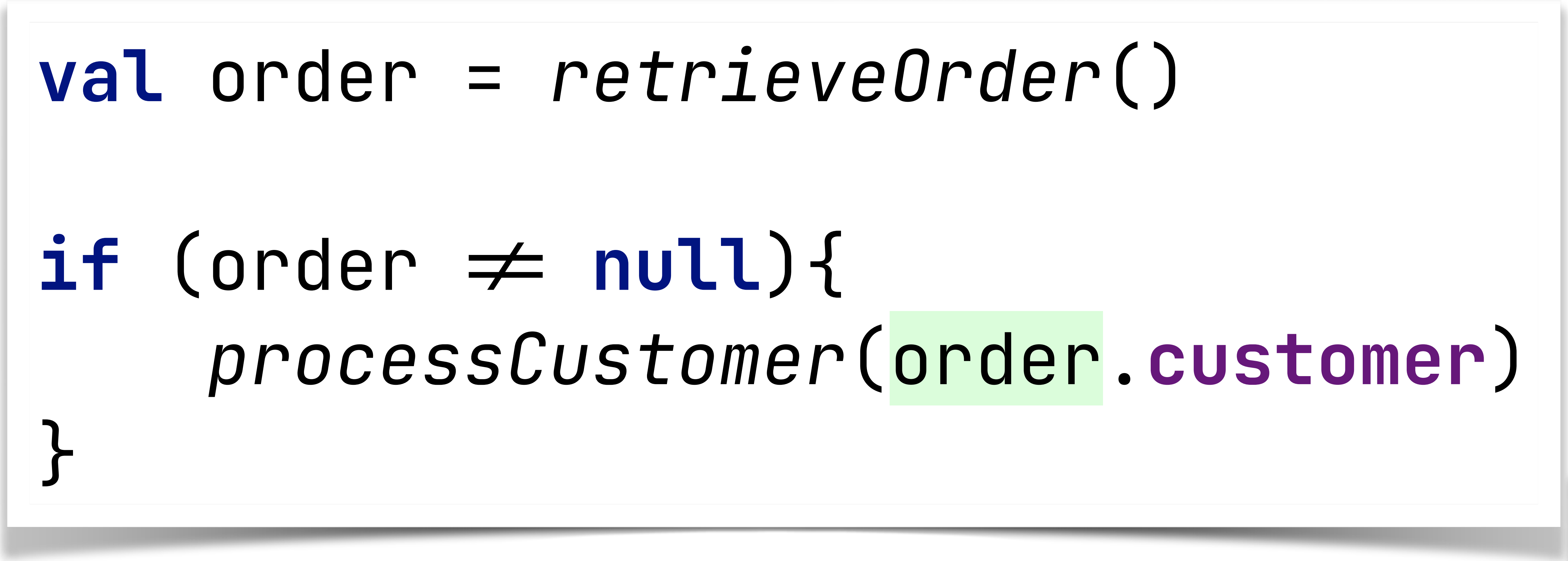
```
val dataSource = BasicDataSource().apply {
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://domain:3309/db"
    username = "username"
    password = "password"
    maxTotal = 40
    maxIdle = 40
    minIdle = 4
}
```

?.let

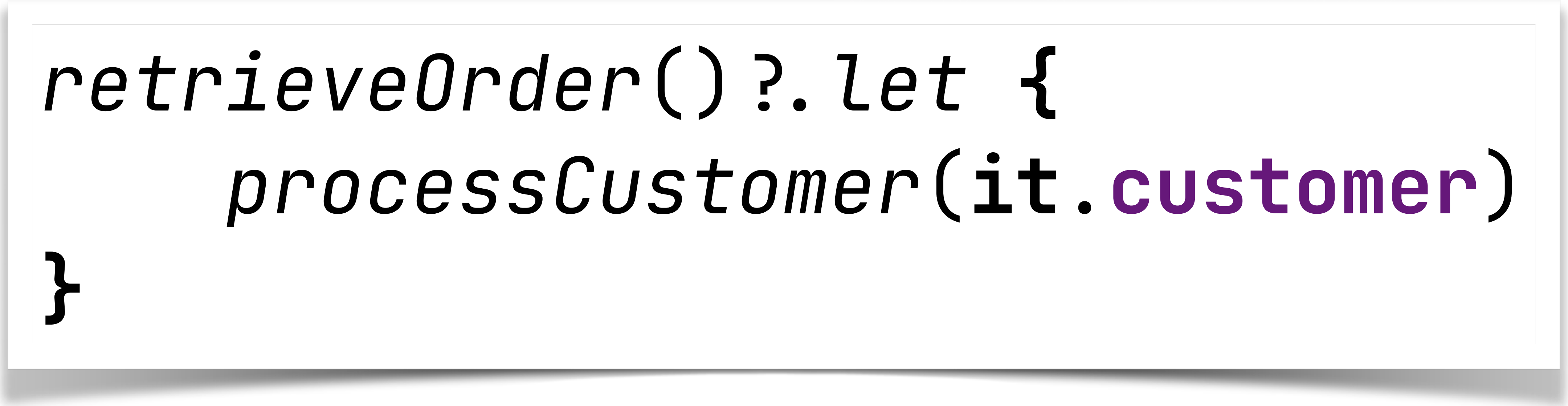


```
val order = retrieveOrder()

if (order ≠ null){
    processCustomer(order.customer)
}
```

?.let

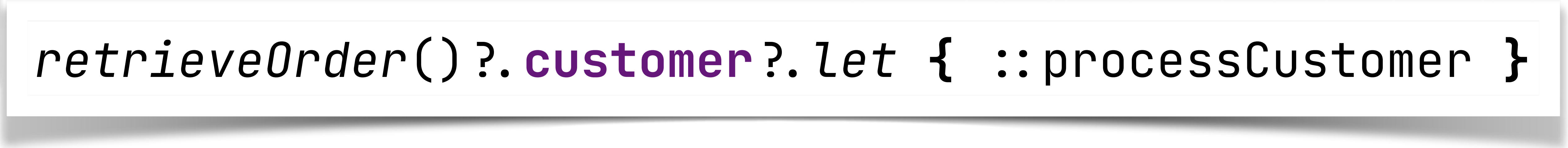


```kotlin
val order = retrieveOrder()

if (order ≠ null){
    processCustomer(order.customer)
}
```



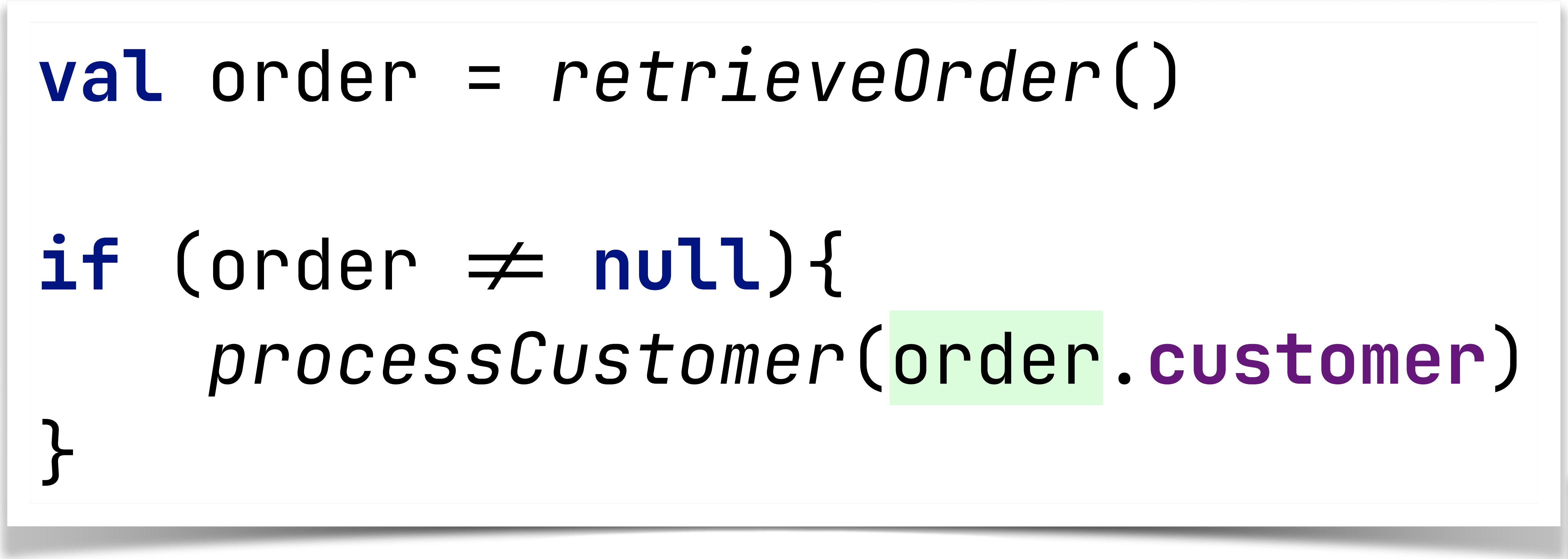
```kotlin
retrieveOrder()?.let {
    processCustomer(it.customer)
}
```

or

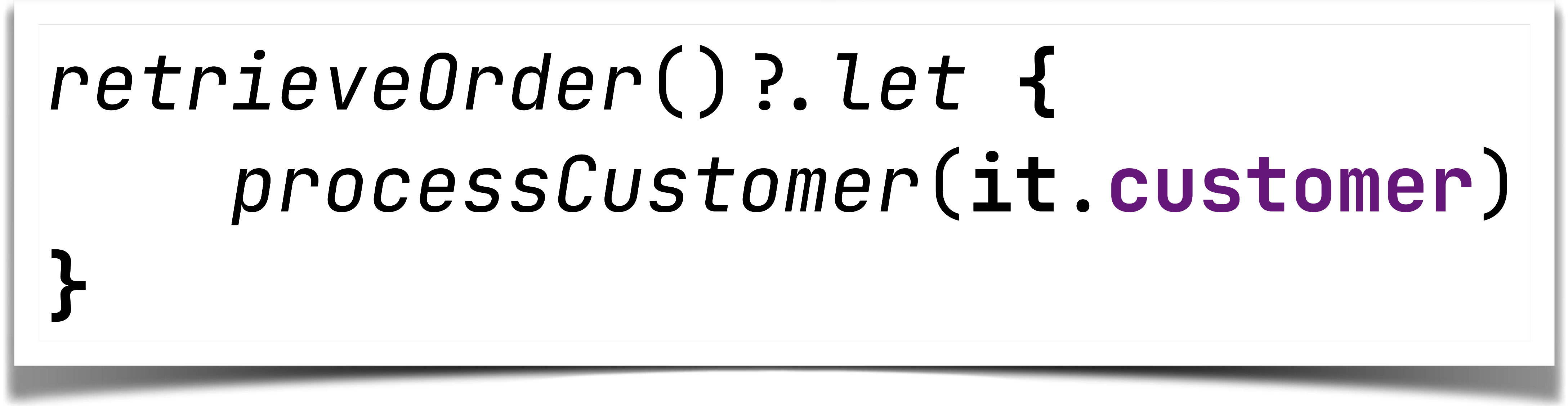


```kotlin
retrieveOrder()?.customer?.let { ::processCustomer }
```

?.let

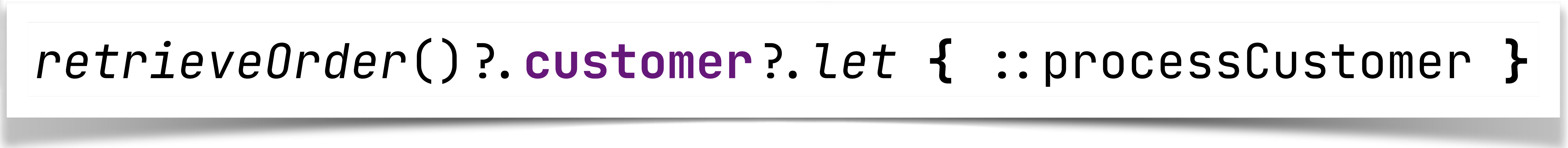


```
val order = retrieveOrder()

if (order ≠ null){
    processCustomer(order.customer)
}
```



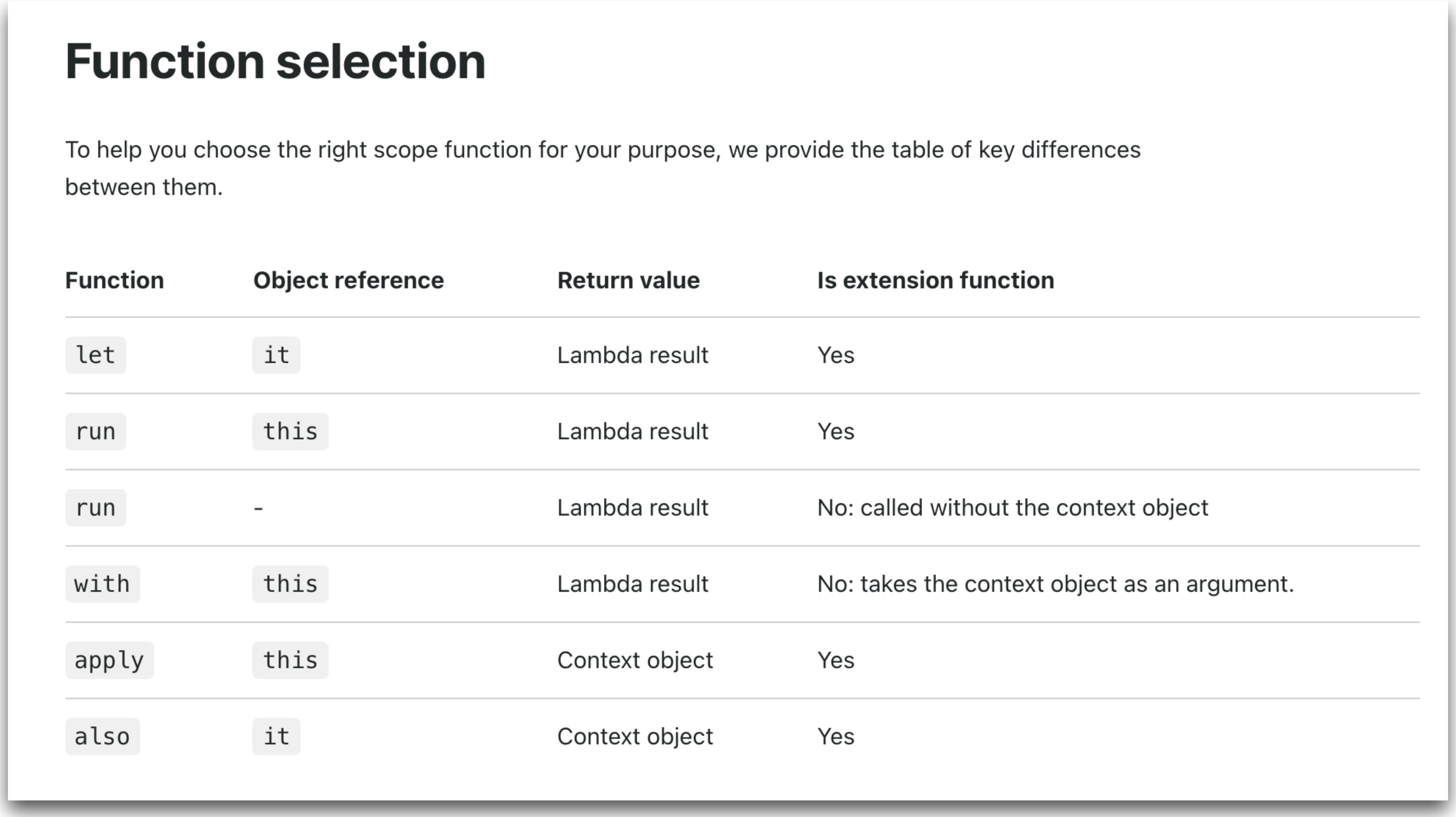
```
retrieveOrder()?.let {
    processCustomer(it.customer)
}
```

or



```
retrieveOrder()?.customer?.let { ::processCustomer }
```

Нет лишней переменной



Function selection

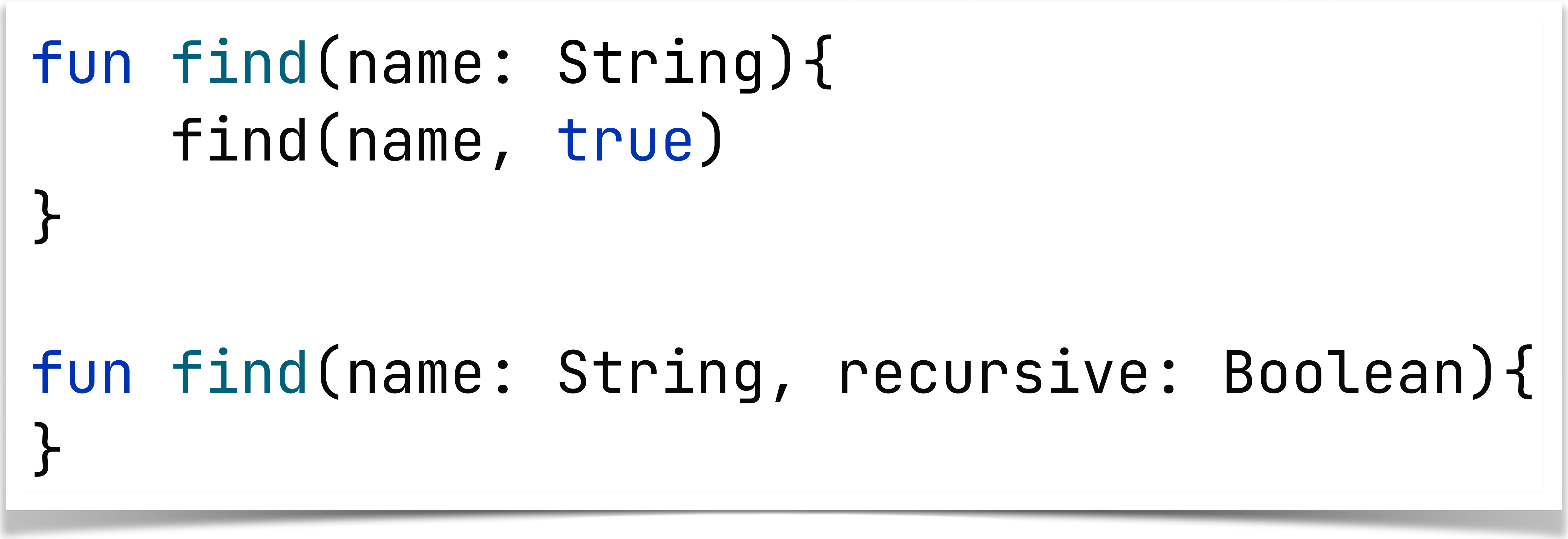
To help you choose the right scope function for your purpose, we provide the table of key differences between them.

Function	Object reference	Return value	Is extension function
`let`	`it`	Lambda result	Yes
`run`	`this`	Lambda result	Yes
`run`	-	Lambda result	No: called without the context object
`with`	`this`	Lambda result	No: takes the context object as an argument.
`apply`	`this`	Context object	Yes
`also`	`it`	Context object	Yes

Вывод 3:

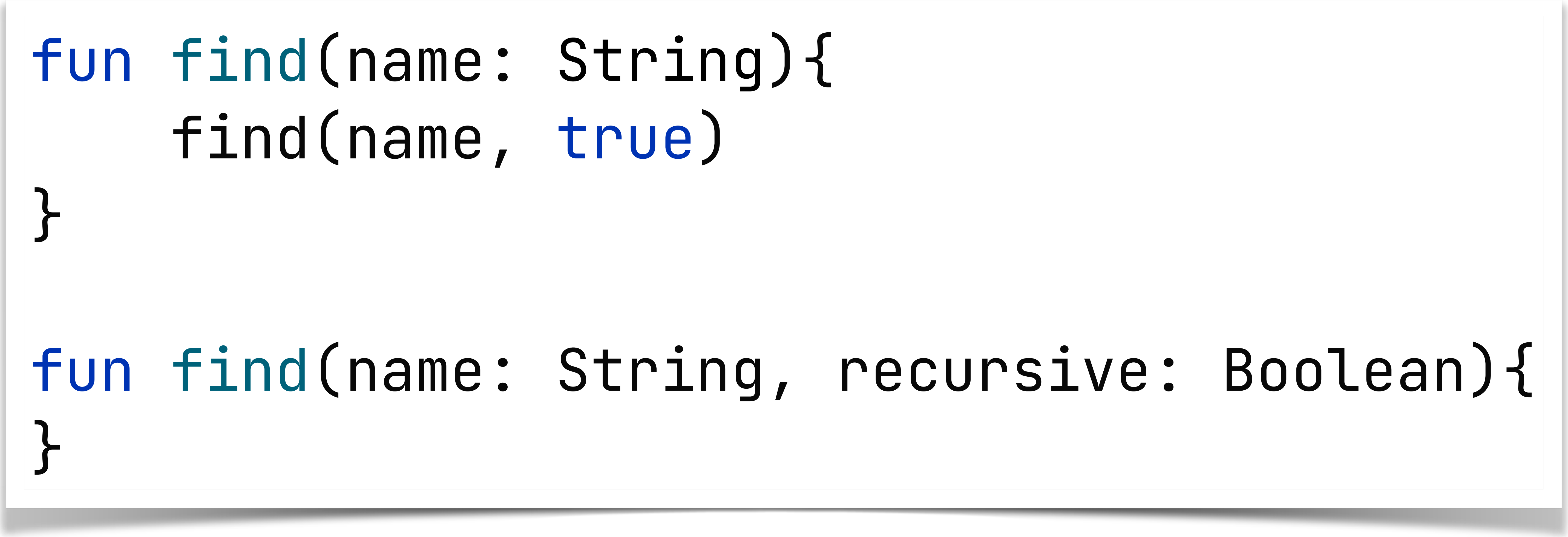
Контекстные функции - очень полезный в хозяйстве инструмент

4. Значения параметров по-умолчанию и именованные параметры



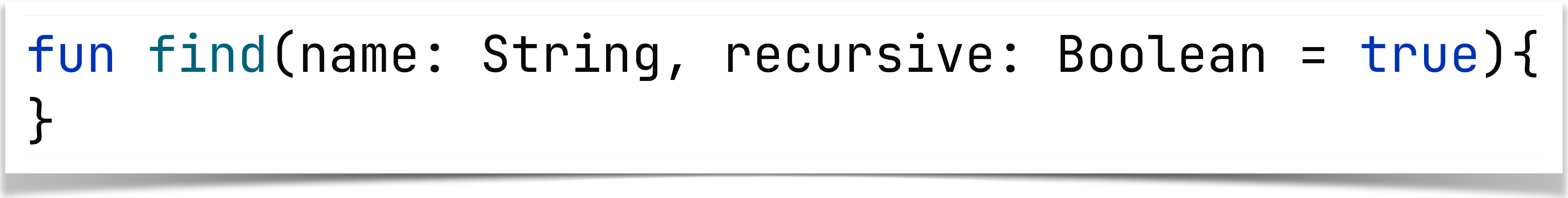
```
fun find(name: String){
    find(name, true)
}

fun find(name: String, recursive: Boolean){
}
```

Перегрузка функций



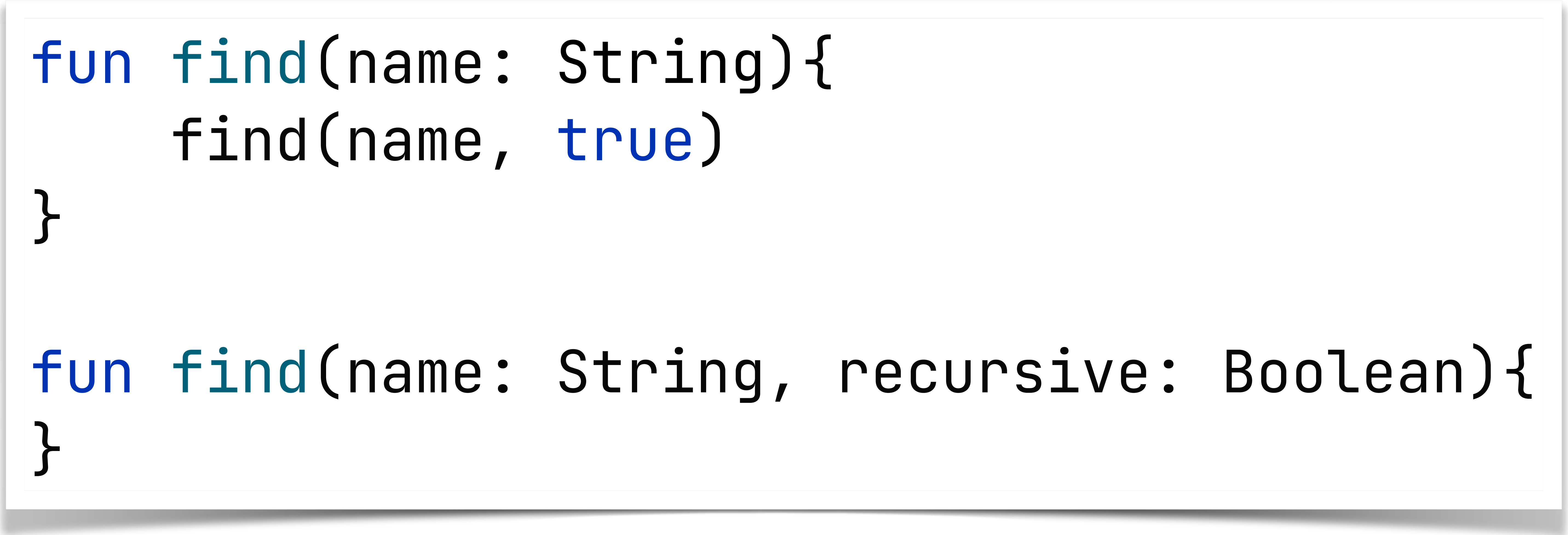
```
fun find(name: String){
    find(name, true)
}

fun find(name: String, recursive: Boolean){
}
```

Перегрузка функций



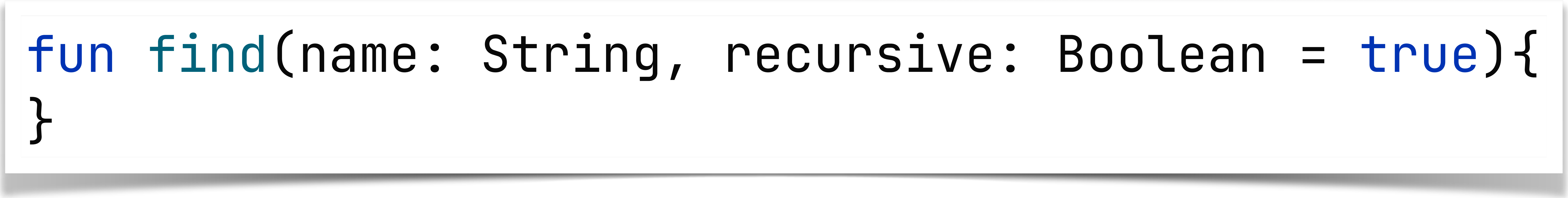
```
fun find(name: String, recursive: Boolean = true){
}
```

Значение параметра по-умолчанию



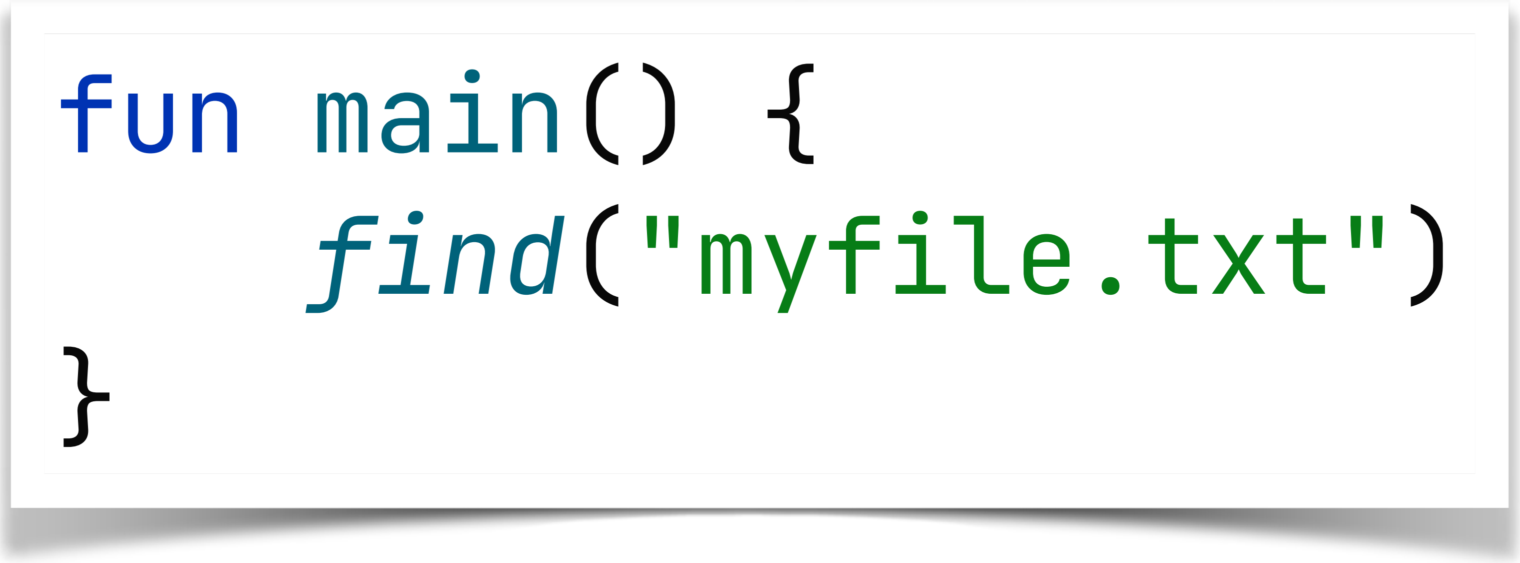
```
fun find(name: String){
    find(name, true)
}

fun find(name: String, recursive: Boolean){
}
```

Перегрузка функций

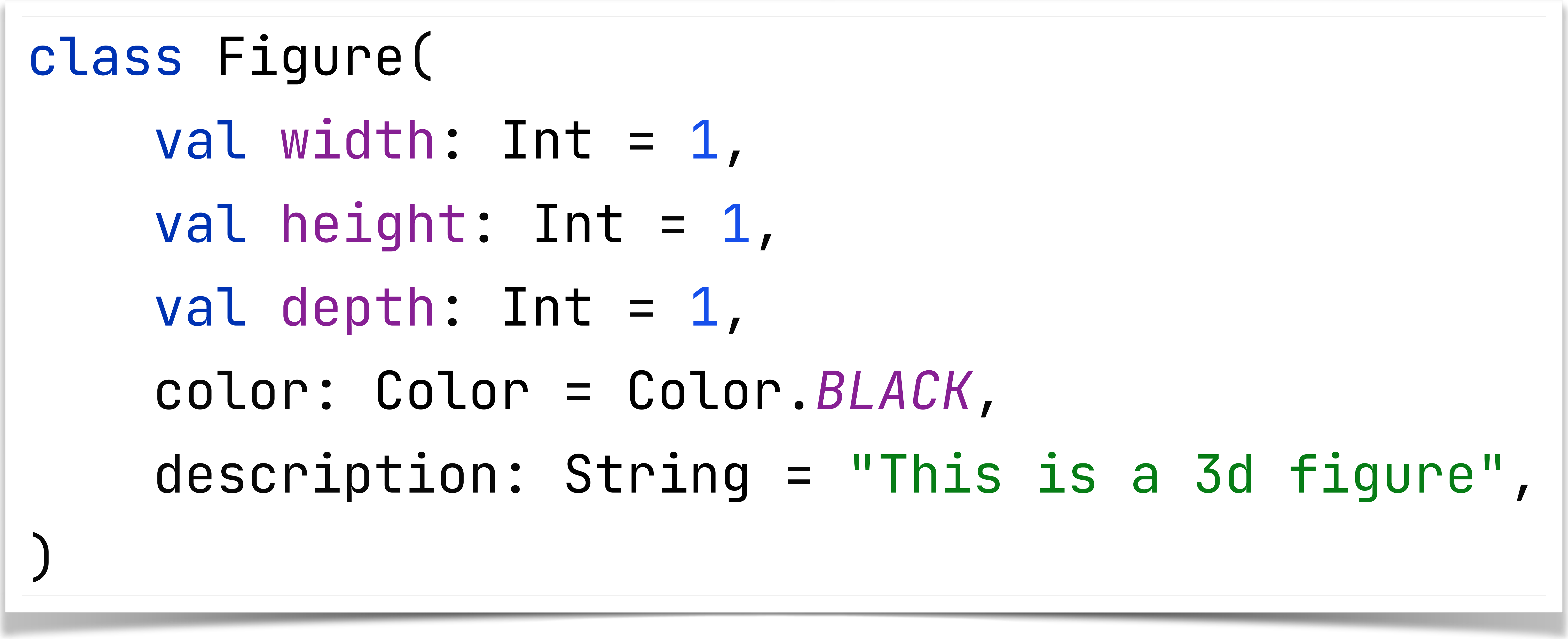


```
fun find(name: String, recursive: Boolean = true){
}
```

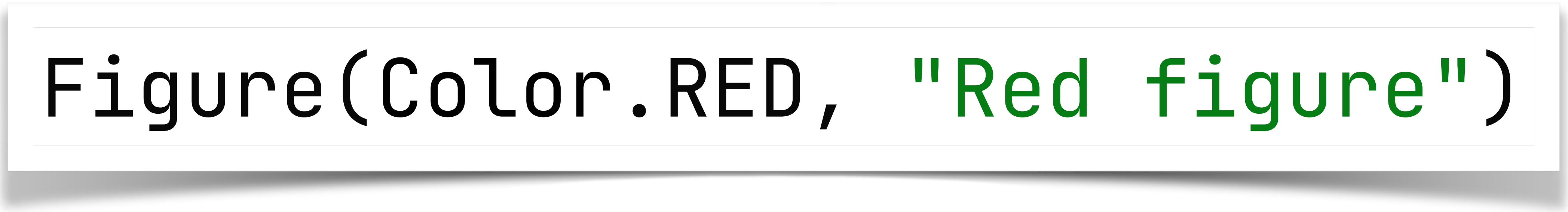
Значение параметра по-умолчанию



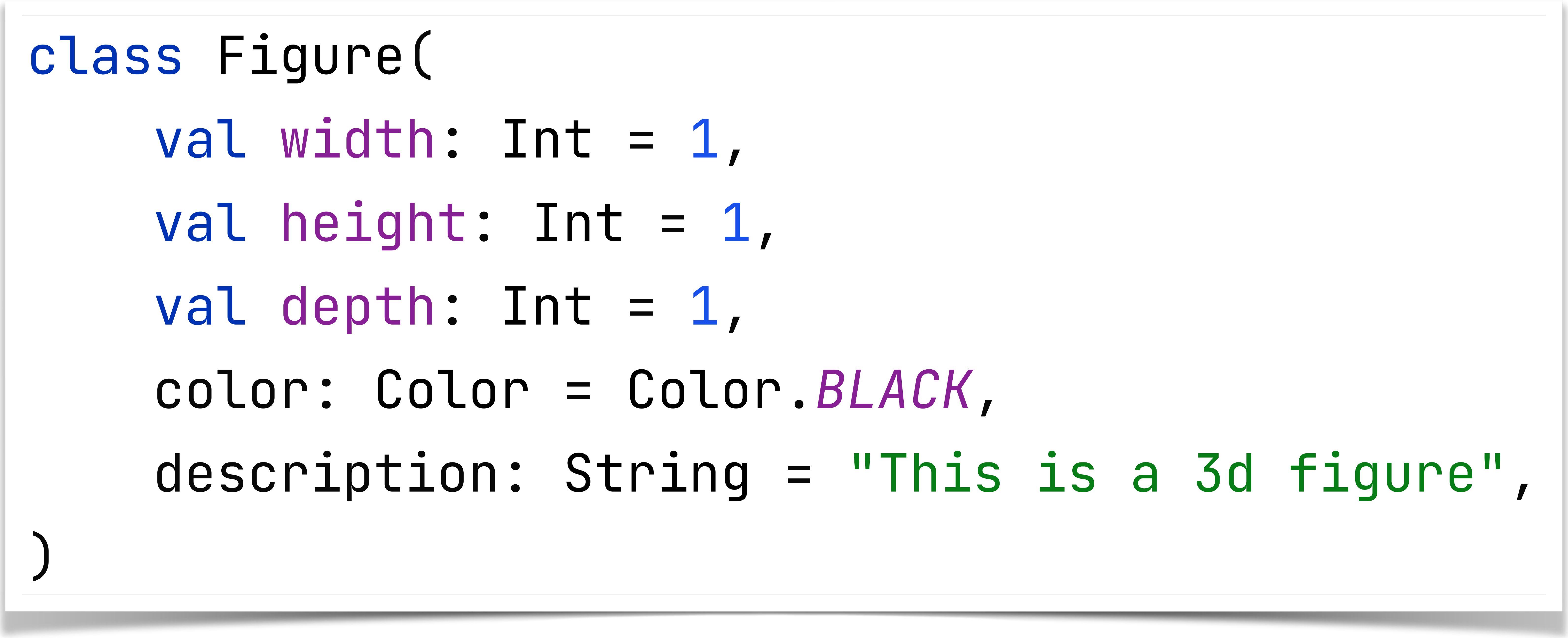
```
fun main() {
    find("myfile.txt")
}
```



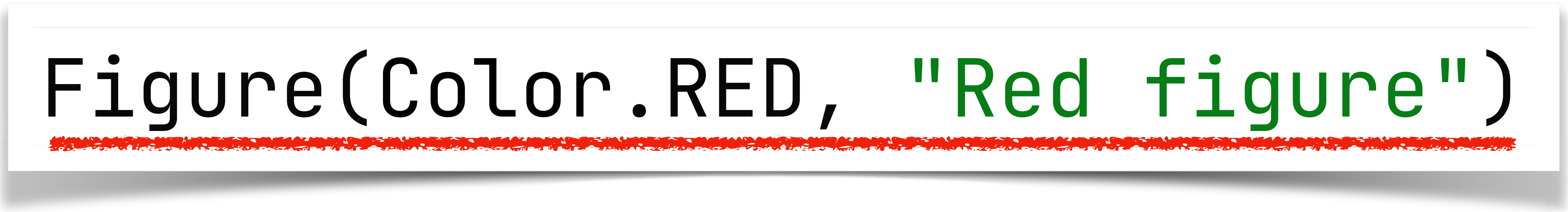
```kotlin
class Figure(
    val width: Int = 1,
    val height: Int = 1,
    val depth: Int = 1,
    color: Color = Color.BLACK,
    description: String = "This is a 3d figure",
)
```



```kotlin
Figure(Color.RED, "Red figure")
```

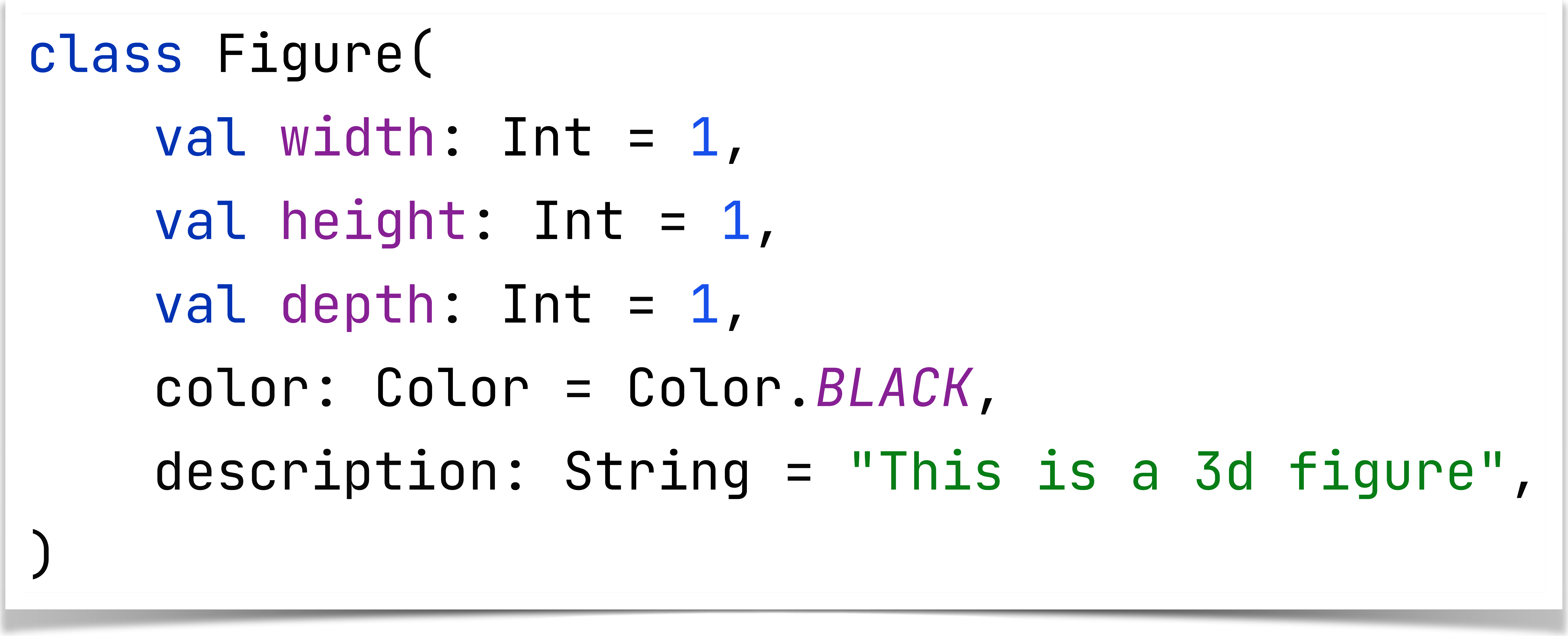


```kotlin
class Figure(
    val width: Int = 1,
    val height: Int = 1,
    val depth: Int = 1,
    color: Color = Color.BLACK,
    description: String = "This is a 3d figure",
)
```

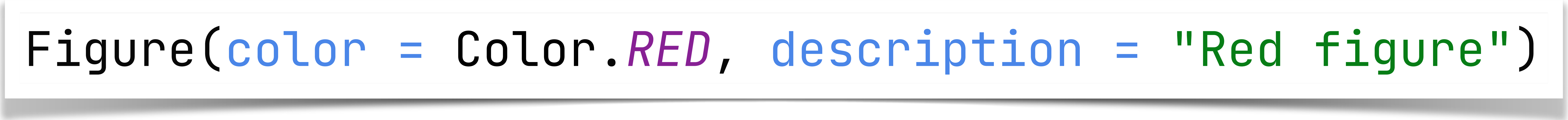


```kotlin
Figure(Color.RED, "Red figure")
```

Ошибка компиляции



```kotlin
class Figure(
    val width: Int = 1,
    val height: Int = 1,
    val depth: Int = 1,
    color: Color = Color.BLACK,
    description: String = "This is a 3d figure",
)
```



```kotlin
Figure(color = Color.RED, description = "Red figure")
```

Вывод(ы) 4:

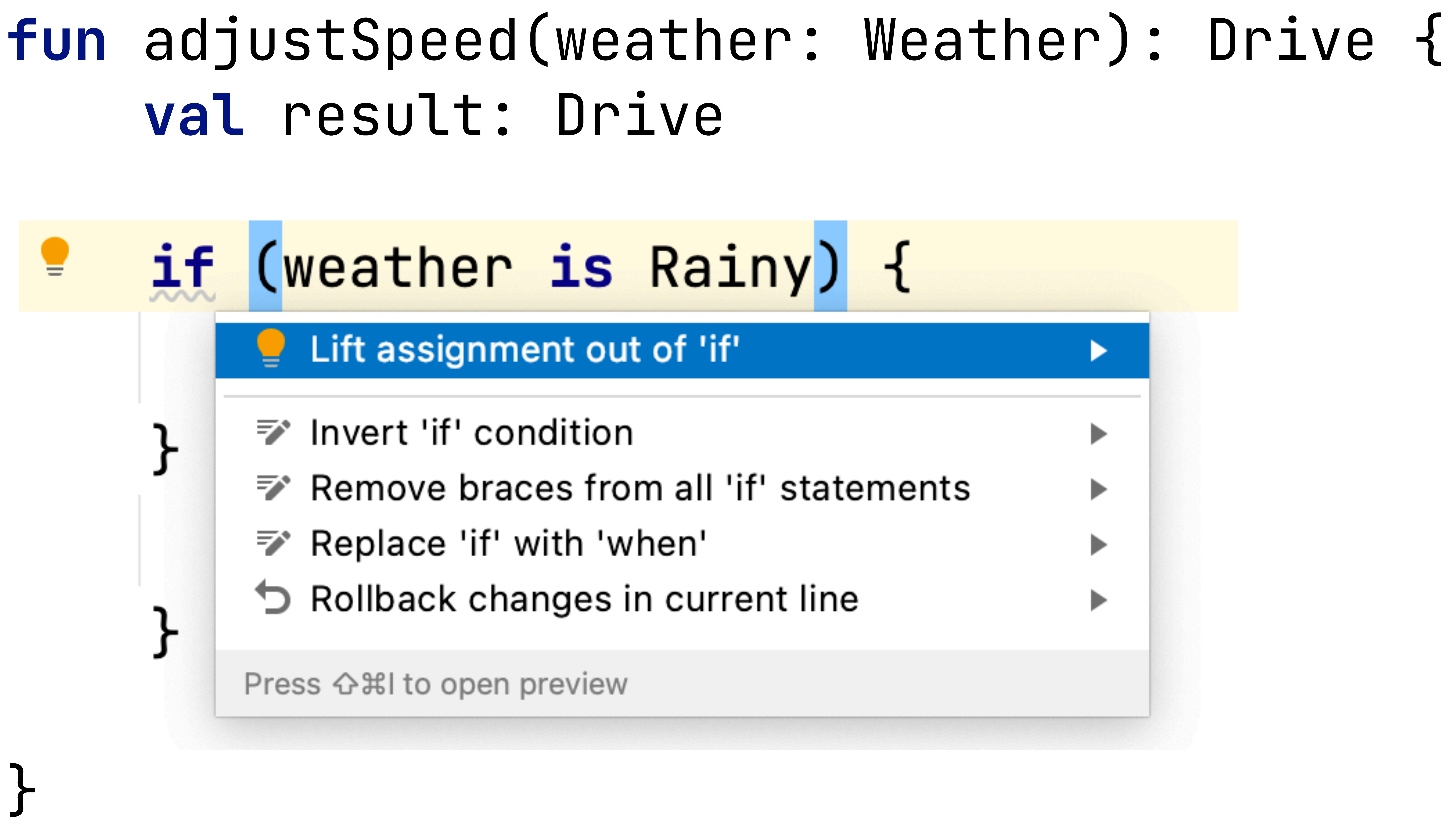
Значения параметров по-умолчанию закрывают надобность в перегрузке функций и конструкторов

Именованные параметры повышают читабельность кода

5. Выражения

try, if, when

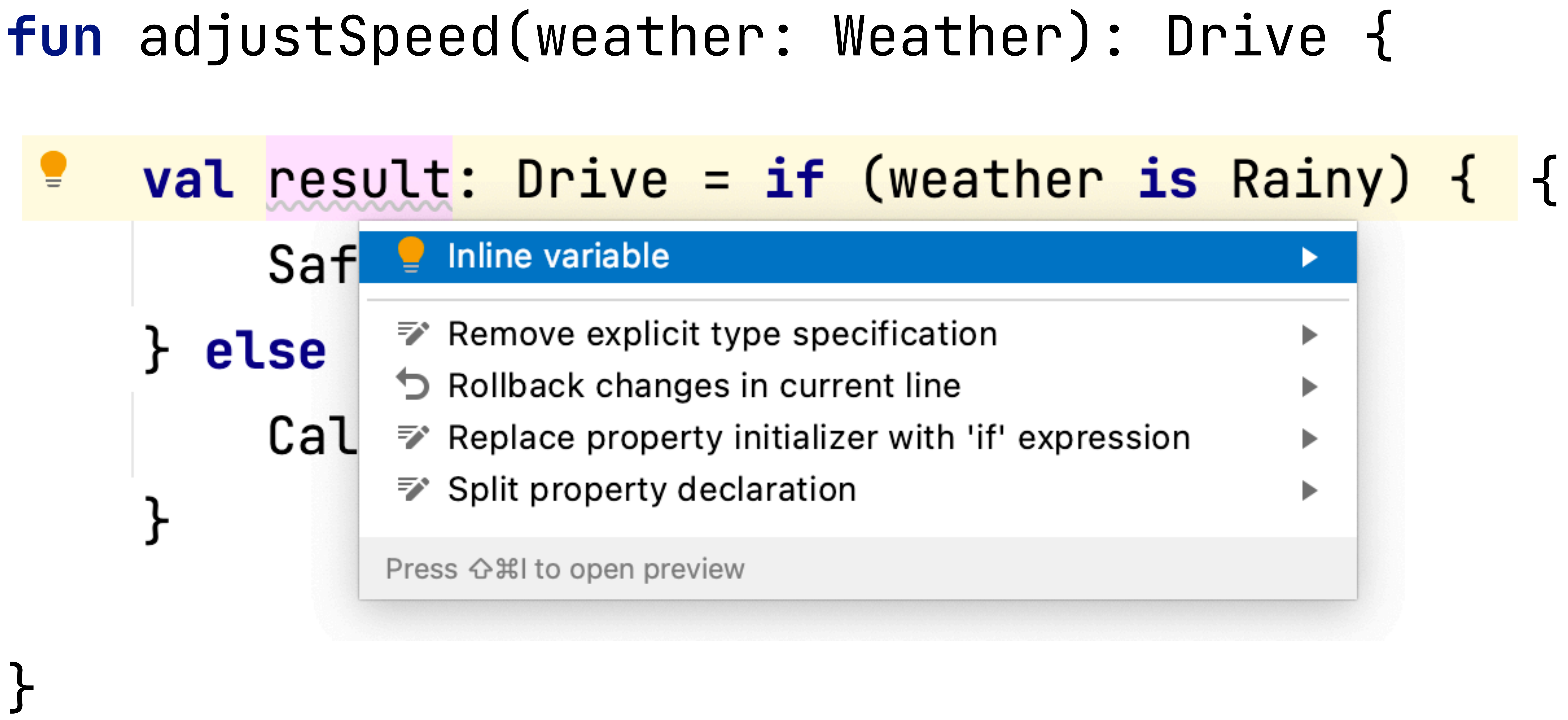
```kotlin
fun adjustSpeed(weather: Weather): Drive {
    val result: Drive

    if (weather is Rainy) {
        result = Safe()
    } else {
        result = Calm()
    }

    return result
}
```



```
fun adjustSpeed(weather: Weather): Drive {
    val result: Drive

    if (weather is Rainy) {

    }

    }

}
```

Lift assignment out of 'if'
Invert 'if' condition
Remove braces from all 'if' statements
Replace 'if' with 'when'
Rollback changes in current line
Press ⇧⌘I to open preview

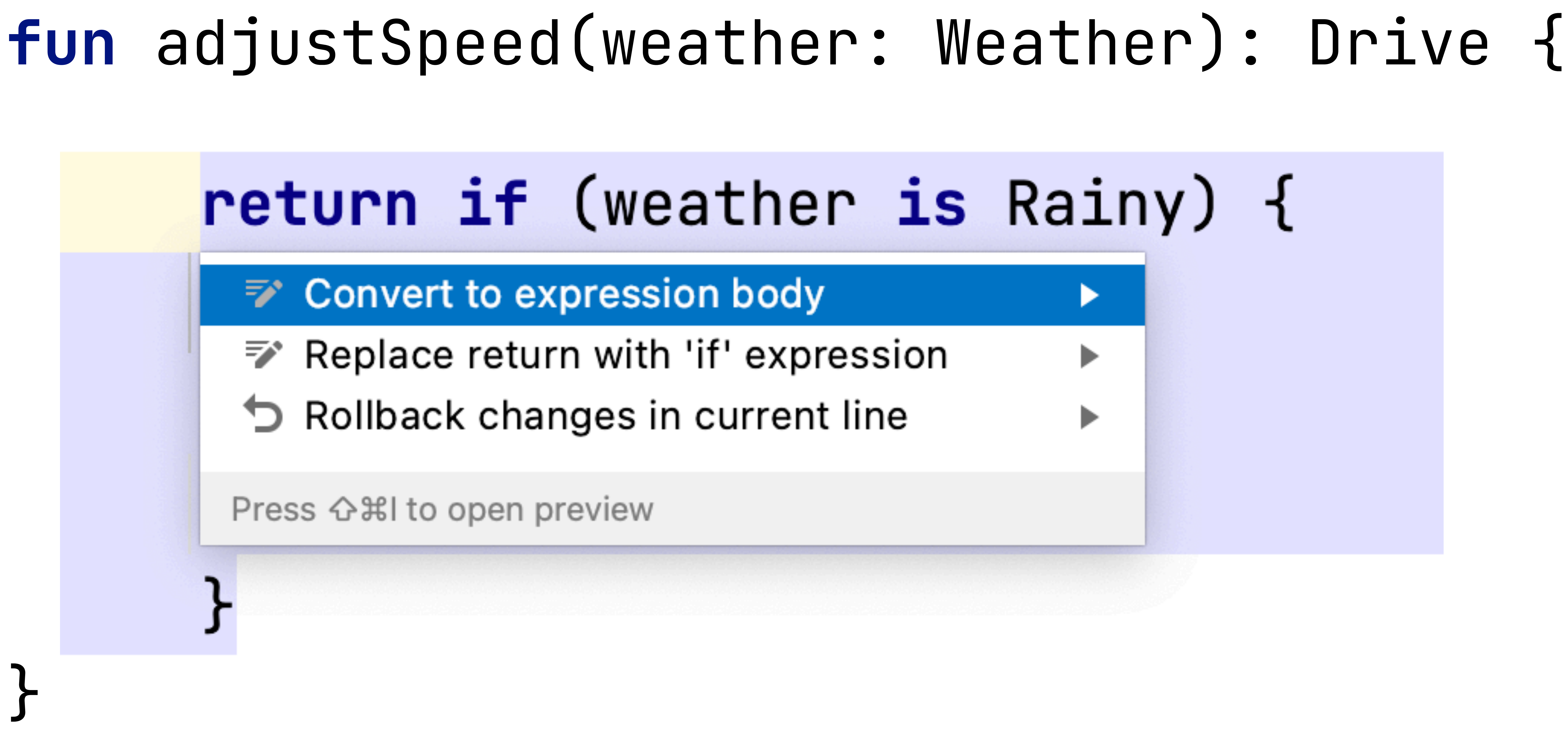
```kotlin
fun adjustSpeed(weather: Weather): Drive {

    val result: Drive = if (weather is Rainy) {
        Safe()
    } else {
        Calm()
    }

    return result
}
```



```
fun adjustSpeed(weather: Weather): Drive {

    val result: Drive = if (weather is Rainy) { {
        Saf
    } else
        Cal
    }

}
```

Inline variable
Remove explicit type specification
Rollback changes in current line
Replace property initializer with 'if' expression
Split property declaration
Press ⇧⌘I to open preview

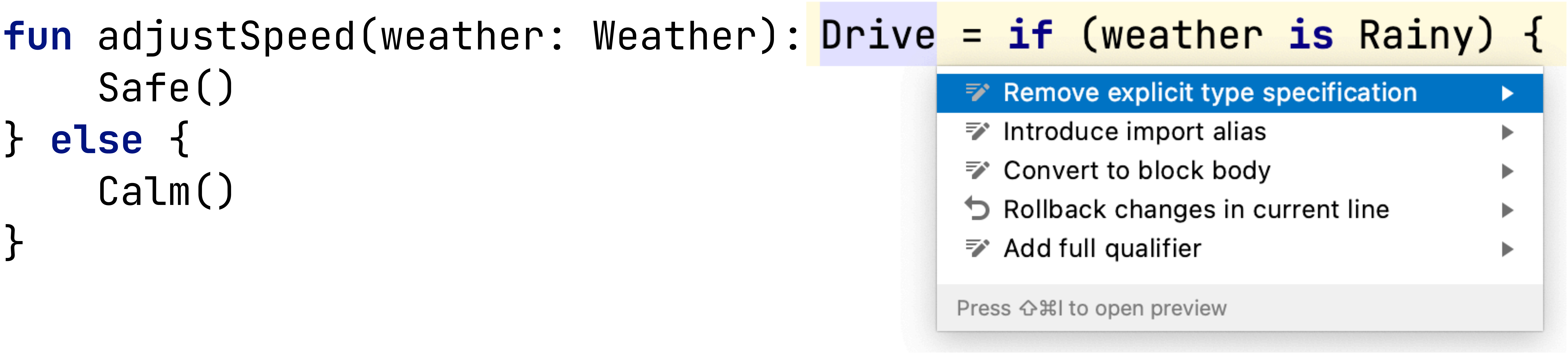
```
fun adjustSpeed(weather: Weather): Drive {

    return if (weather is Rainy) {
        Safe()
    } else {
        Calm()
    }

}
```



```
fun adjustSpeed(weather: Weather): Drive {
    return if (weather is Rainy) {
    }
}
```

Convert to expression body
Replace return with 'if' expression
Rollback changes in current line
Press ⇧⌘I to open preview

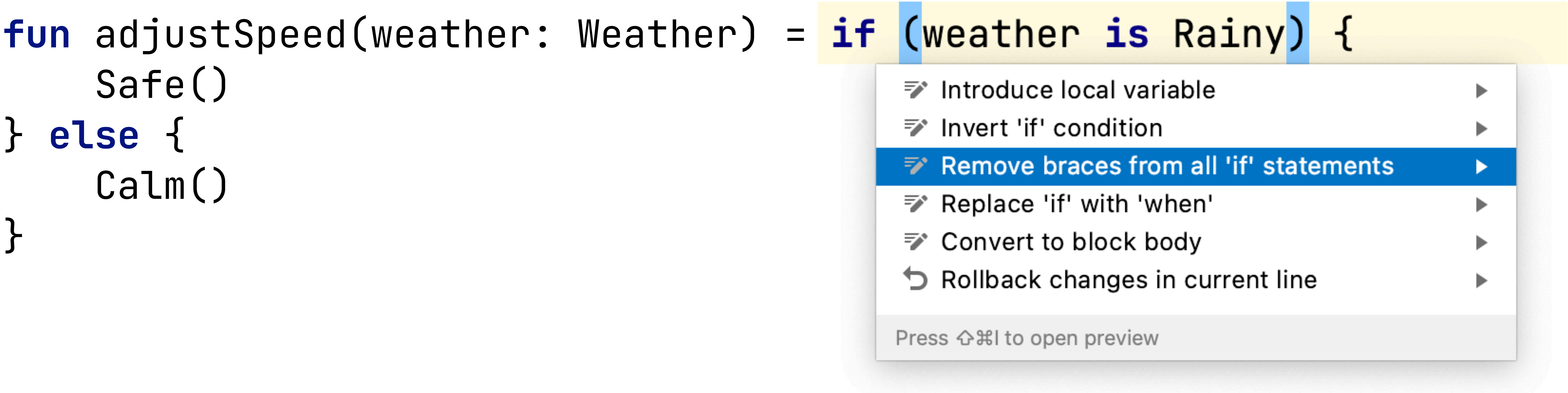
```
fun adjustSpeed(weather: Weather): Drive = if (weather is Rainy) {
    Safe()
} else {
    Calm()
}
```



```
fun adjustSpeed(weather: Weather): Drive = if (weather is Rainy) {
    Safe()
} else {
    Calm()
}
```

Remove explicit type specification
Introduce import alias
Convert to block body
Rollback changes in current line
Add full qualifier
Press ⇧⌘I to open preview

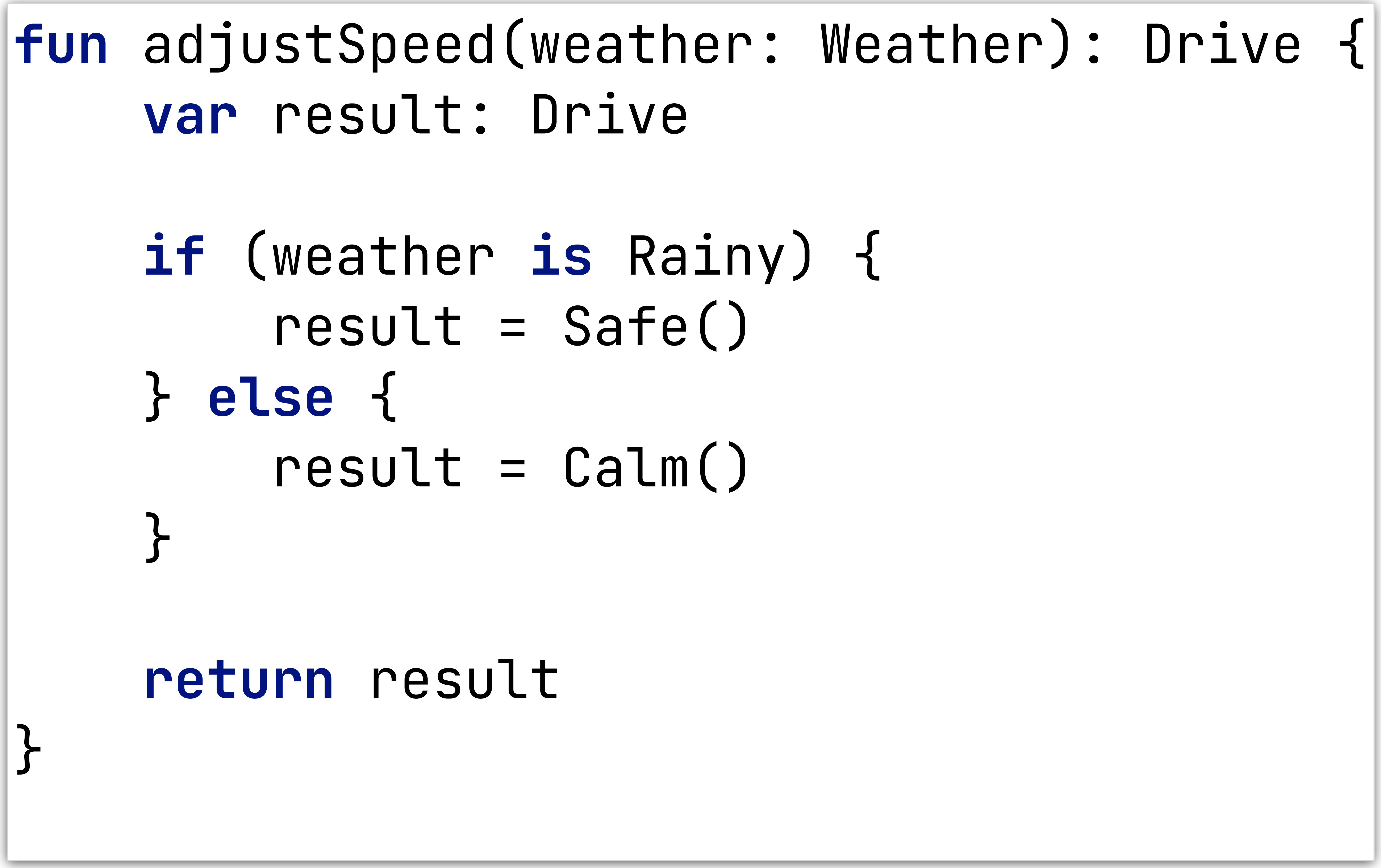
```kotlin
fun adjustSpeed(weather: Weather) = if (weather is Rainy) {
    Safe()
} else {
    Calm()
}
```



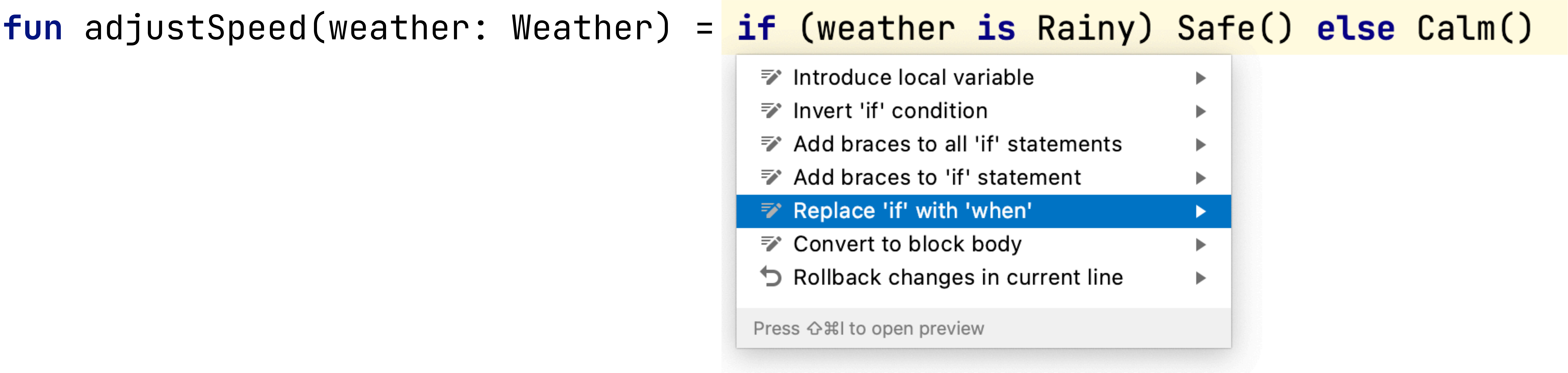
```
fun adjustSpeed(weather: Weather) = if (weather is Rainy) {
    Safe()
} else {
    Calm()
}
```

Introduce local variable
Invert 'if' condition
Remove braces from all 'if' statements
Replace 'if' with 'when'
Convert to block body
Rollback changes in current line
Press ⇧⌘I to open preview

```kotlin
fun adjustSpeed(weather: Weather) = if (weather is Rainy) Safe() else Calm()
fun adjustSpeed(weather: Weather) = if (weather is Rainy) Safe() else Calm()
```



```
fun adjustSpeed(weather: Weather): Drive {
    var result: Drive

    if (weather is Rainy) {
        result = Safe()
    } else {
        result = Calm()
    }

    return result
}
```



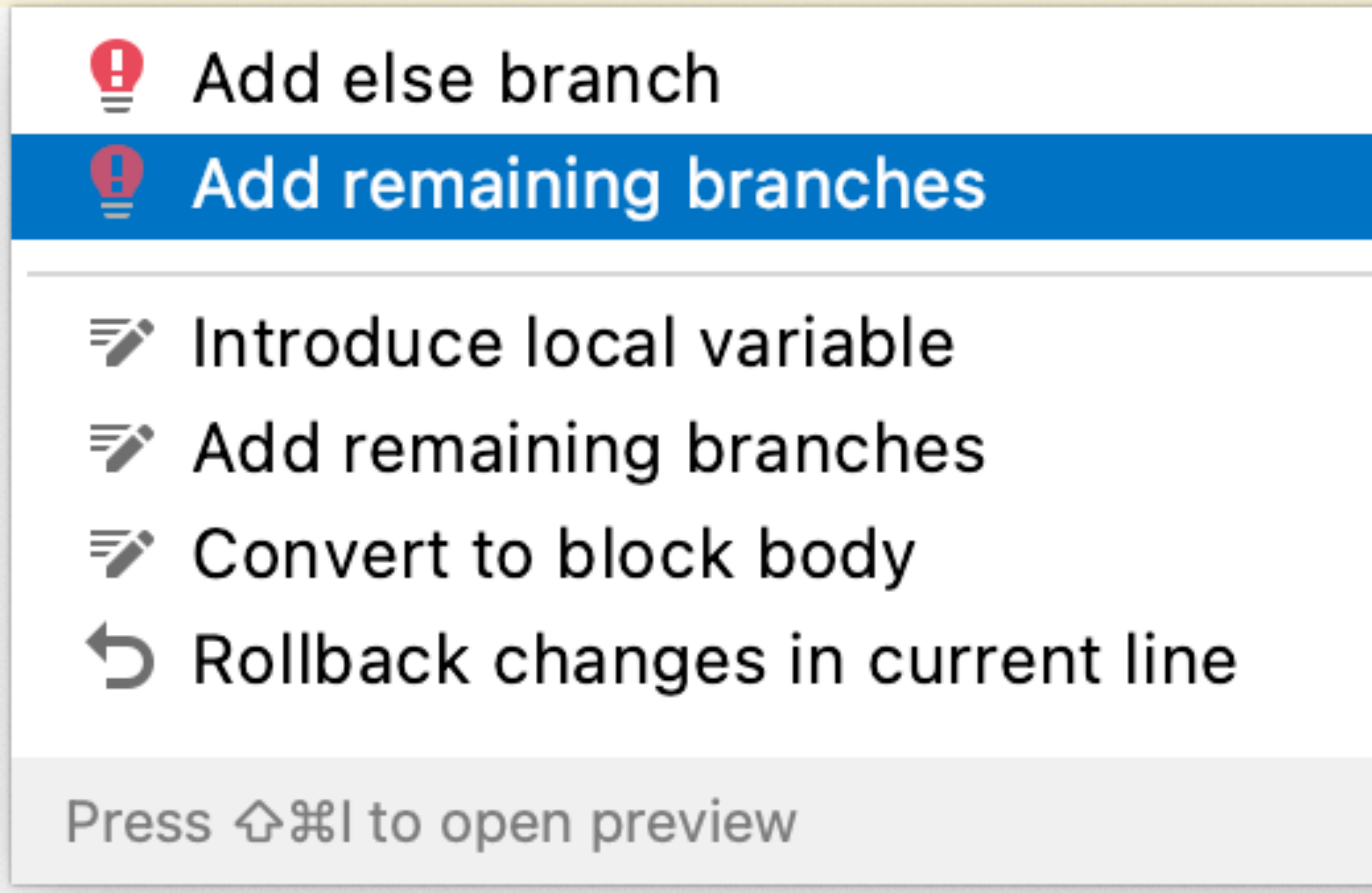
```kotlin
fun adjustSpeed(weather: Weather) = if (weather is Rainy) Safe() else Calm()
```

Introduce local variable
Invert 'if' condition
Add braces to all 'if' statements
Add braces to 'if' statement
Replace 'if' with 'when'
Convert to block body
Rollback changes in current line
Press ⇧⌘I to open preview

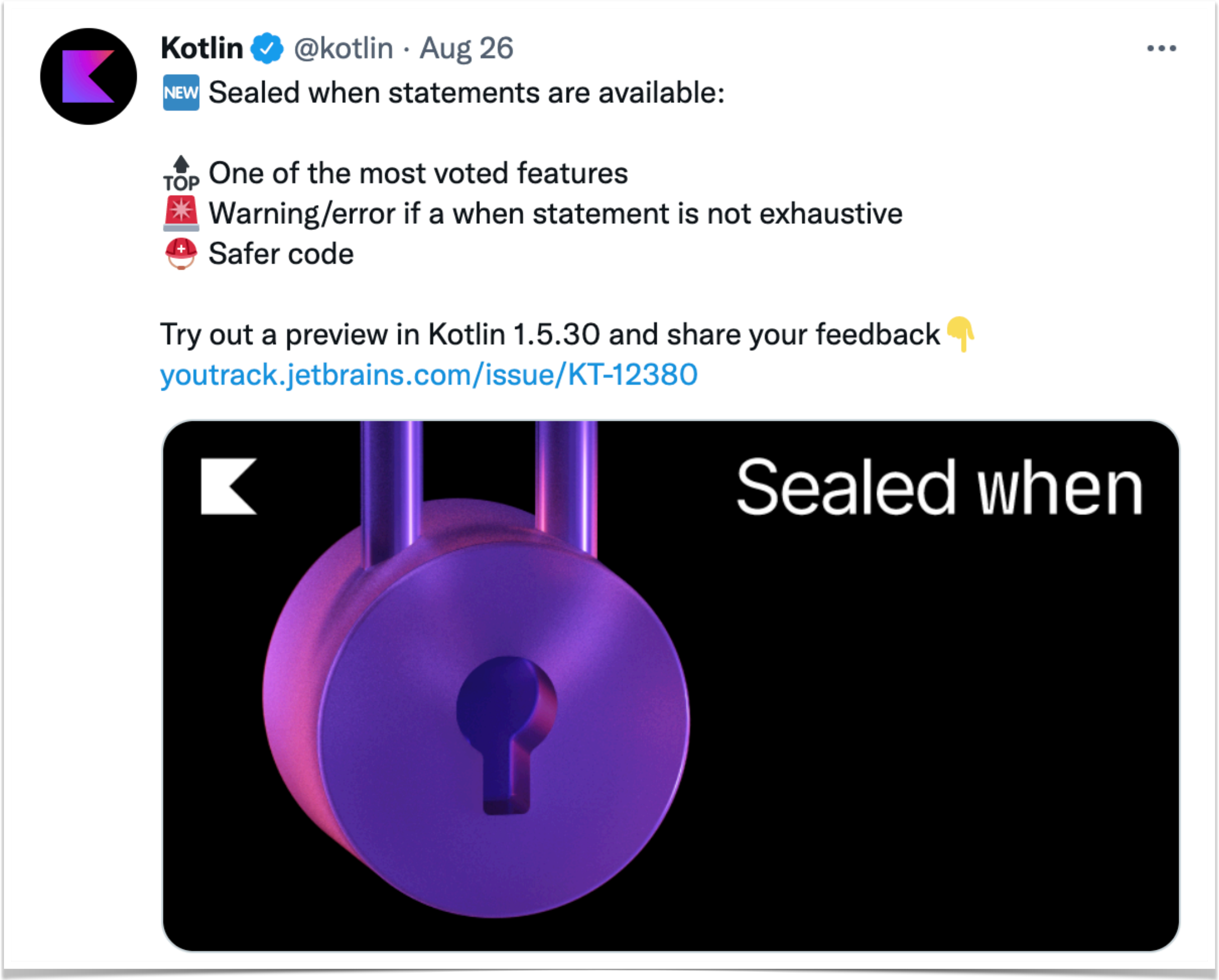
```kotlin
abstract class Weather
class Sunny : Weather()
class Rainy : Weather()


fun adjustSpeed(weather: Weather) = when (weather) {
    is Rainy -> Safe()
    else -> Calm()
}
```

```
sealed class Weather
class Sunny : Weather()
class Rainy : Weather()


fun adjustSpeed(weather: Weather) = when (weather) {
    is Rainy -> Safe()
//    else -> Calm()
}
```

```
sealed class Weather
class Sunny : Weather()
class Rainy : Weather()


fun adjustSpeed(weather: Weather) = when (weather) {
    is Rainy → Safe()
//    else → Calm()
}
```



```kotlin
sealed class Weather
class Sunny : Weather()
class Rainy : Weather()


fun adjustSpeed(weather: Weather) = when (weather) {
    is Rainy -> Safe()
    is Sunny -> Calm()
}
```



Kotlin @kotlin · Aug 26

🆕 Sealed when statements are available:

🔝 One of the most voted features
🚨 Warning/error if a when statement is not exhaustive
⛑️ Safer code

Try out a preview in Kotlin 1.5.30 and share your feedback 👇
youtrack.jetbrains.com/issue/KT-12380

Вывод 5:

Используйте `when` + `sealed` классы

6. Функциональные типы

```
fun <T> someFunction(function: () -> T): T {
    …
}
```

```
fun <T> someFunction(function: () -> T): T {
    …
}
```

Функциональный тип

```kotlin
fun <T> someFunction(function: () -> T): T {
    …
}


fun someOtherFunction(){
    val s: String = someFunction { "Hello" }
}
```

```kotlin
typealias Action<T> = () -> T

fun <T> someFunction(function: Action<T>): T {
    …
}


fun someOtherFunction(){
    val s: String = someFunction { "Hello" }
}
```

```
typealias Action<T> = () -> T
```

```kotlin
typealias Action<T> = () -> T

class MyAction<T> : Action<T> {
    override fun invoke(): T {
        TODO("Not yet implemented")
    }
}
```

```kotlin
typealias Action<T> = () -> T

class MyAction<T> : Action<T> {
    override fun invoke(): T {
        TODO("Not yet implemented")
    }
}

fun <T> someFunction(function: Action<T>): T {
    …
}
```

```kotlin
typealias Action<T> = () -> T

class MyAction<T>(val param: String) : Action<T> {
    override fun invoke(): T {
        TODO("Not yet implemented")
    }
}

fun <T> someFunction(function: Action<T>): T {
    …
}

fun someOtherFunction(){
    val s: String = someFunction(MyAction("Greetings"))
}
```

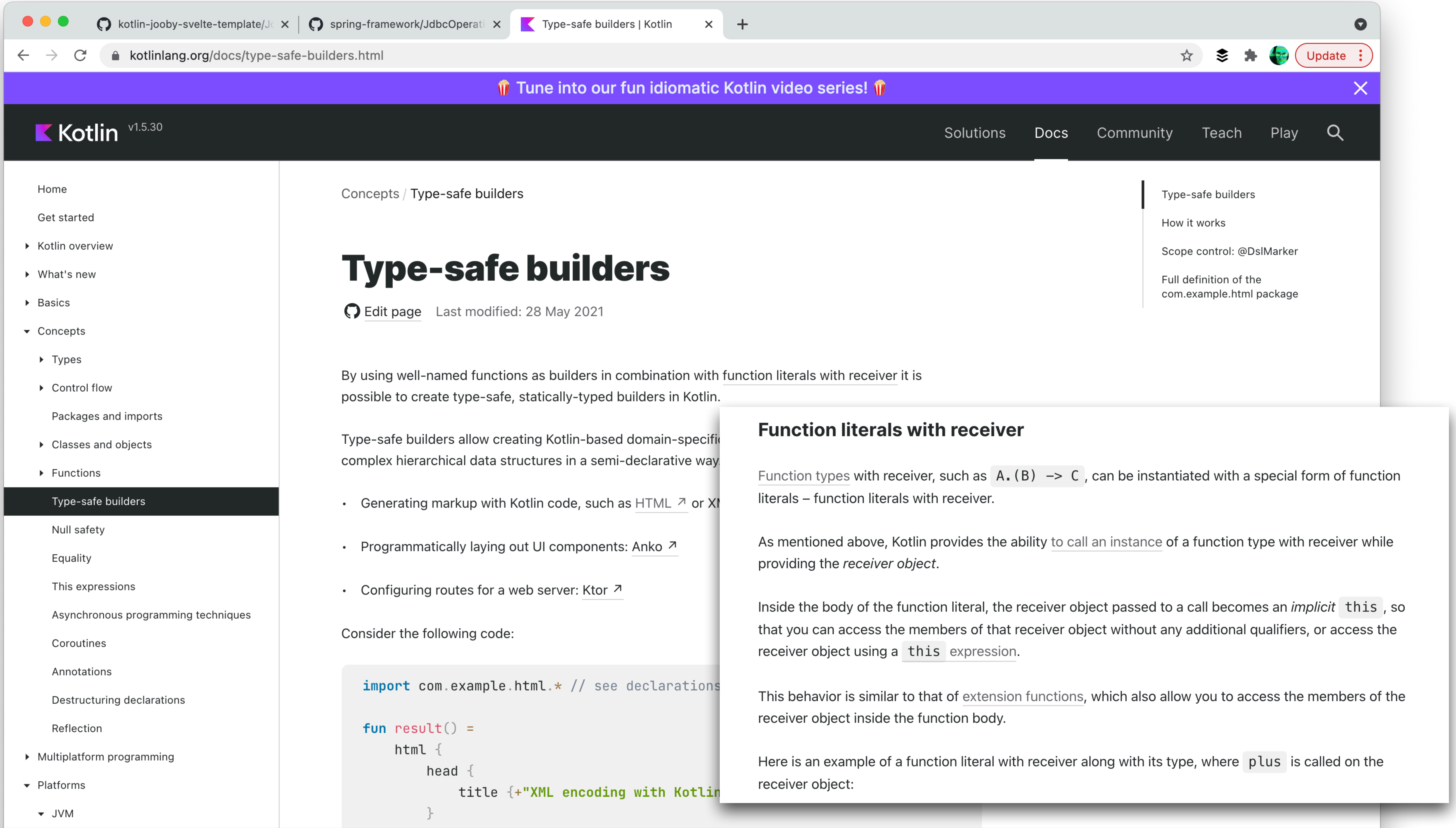
Вывод 6:

Мало иметь функции высшего порядка, нужны дополнительные возможности работы с ними

Вывод от частного к общему:

Мало иметь некую возможность в ЯП, надо иметь дополнительные инструменты работы с этой возможностью

7. X.(Y) → Z


kotlin-jooby-svelte-template/J
spring-framework/JdbcOperati
Type-safe builders | Kotlin
kotlinlang.org/docs/type-safe-builders.html
Update
🍿 Tune into our fun idiomatic Kotlin video series! 🍿
Kotlin v1.5.30
Solutions
Docs
Community
Teach
Play
Home
Get started
Kotlin overview
What's new
Basics
Concepts
Types
Control flow
Packages and imports
Classes and objects
Functions
Type-safe builders
Null safety
Equality
This expressions
Asynchronous programming techniques
Coroutines
Annotations
Destructuring declarations
Reflection
Multiplatform programming
Platforms
JVM
Type-safe builders
How it works
Scope control: @DslMarker
Full definition of the com.example.html package

Concepts / Type-safe builders

Type-safe builders

Edit page Last modified: 28 May 2021

By using well-named functions as builders in combination with function literals with receiver it is possible to create type-safe, statically-typed builders in Kotlin.

Type-safe builders allow creating Kotlin-based domain-specifi complex hierarchical data structures in a semi-declarative way

- Generating markup with Kotlin code, such as HTML ↗ or X
- Programmatically laying out UI components: Anko ↗
- Configuring routes for a web server: Ktor ↗

Consider the following code:

```
import com.example.html.* // see declarations

fun result() =
    html {
        head {
            title {+"XML encoding with Kotlin
        }
```

Function literals with receiver

Function types with receiver, such as `A.(B) -> C`, can be instantiated with a special form of function literals – function literals with receiver.

As mentioned above, Kotlin provides the ability to call an instance of a function type with receiver while providing the *receiver object*.

Inside the body of the function literal, the receiver object passed to a call becomes an *implicit* `this`, so that you can access the members of that receiver object without any additional qualifiers, or access the receiver object using a `this` expression.

This behavior is similar to that of extension functions, which also allow you to access the members of the receiver object inside the function body.

Here is an example of a function literal with receiver along with its type, where `plus` is called on the receiver object:

```
val ds = BasicDataSource().apply {
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://domain:3309/db"
    username = "username"
    password = "password"
}
```

```kotlin
val ds = BasicDataSource().apply {
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://domain:3309/db"
    username = "username"
    password = "password"
}
```

```kotlin
public inline fun <T> T.apply(block: T.() -> Unit): T {
    block()
    return this
}
```

```
val ds = BasicDataSource().apply {  this: BasicDataSource
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://domain:3309/db"
    username = "username"
    password = "password"
}
```

```
public inline fun <T> T.apply(block: T.() → Unit): T {
    block()
    return this
}
```

```
val ds = dataSource {  this: BasicDataSource
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://domain:3309/db"
    username = "username"
    password = "password"
}
```

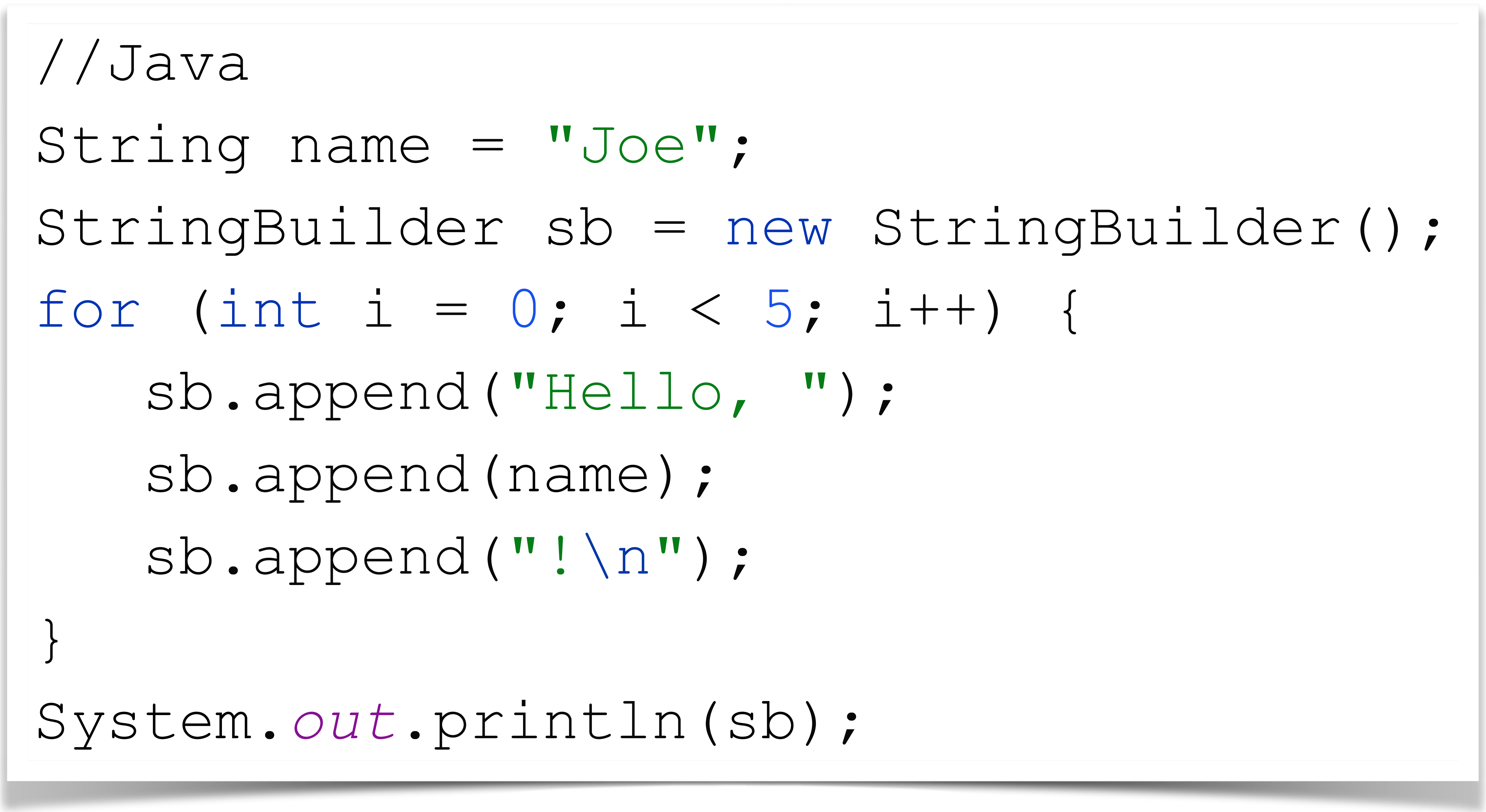
```kotlin
val ds = dataSource {  this: BasicDataSource
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://domain:3309/db"
    username = "username"
    password = "password"
}

fun dataSource(config: BasicDataSource.() -> Unit)
```

```kotlin
val ds = dataSource {  this: BasicDataSource
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://domain:3309/db"
    username = "username"
    password = "password"
}

fun dataSource(config: BasicDataSource.() -> Unit)
     = BasicDataSource().apply(config)
```

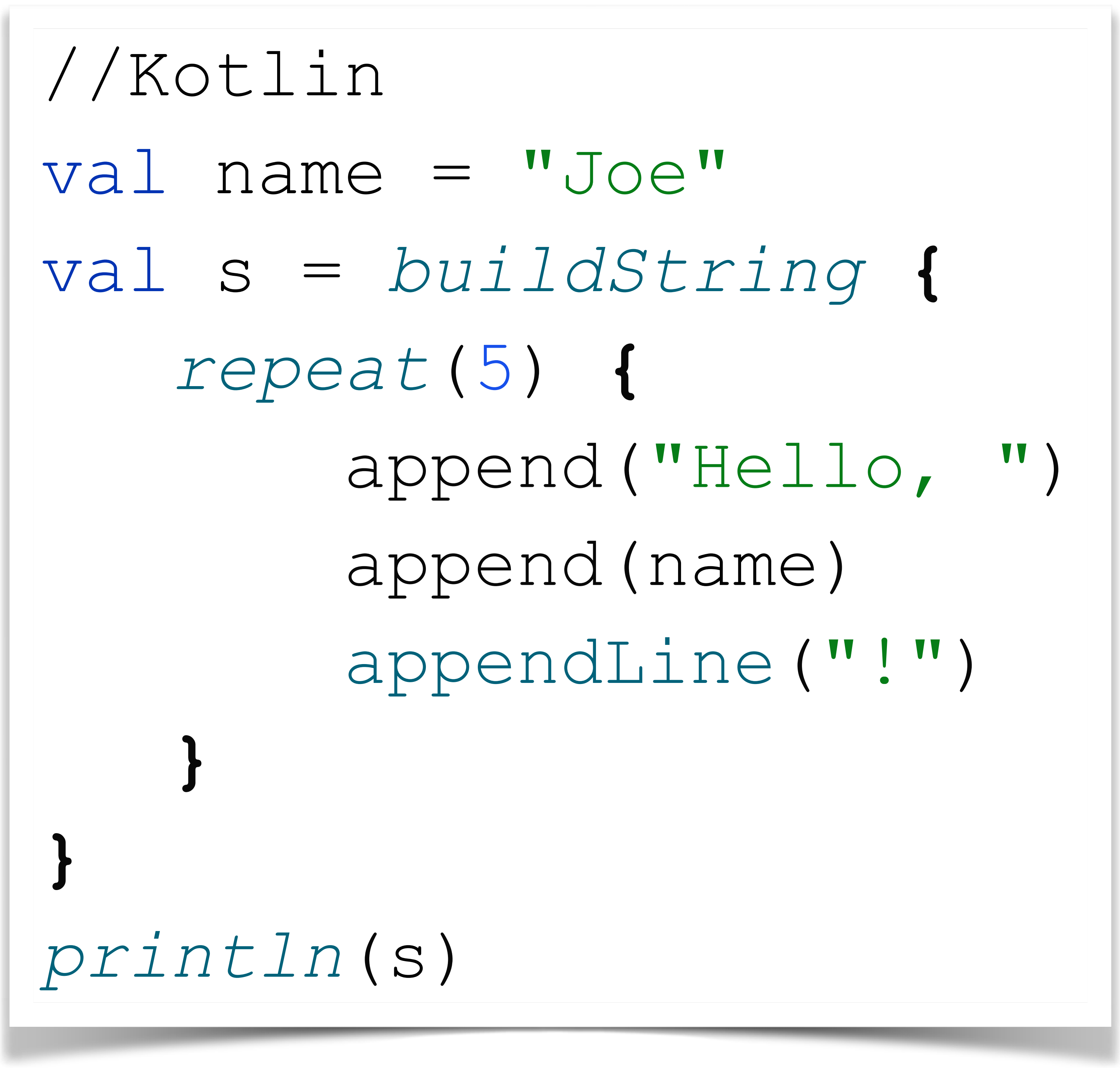
```kotlin
val ds = dataSource {  this: BasicDataSource
    driverClassName = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://domain:3309/db"
    username = "username"
    password = "password"
}

fun dataSource(config: BasicDataSource.() -> Unit)
     = BasicDataSource().apply(config)
```

Выполнить блок (лямбду) внутри экземпляра класса и вернуть этот же экземпляр

buildString



```java
//Java
String name = "Joe";
StringBuilder sb = new StringBuilder();
for (int i = 0; i < 5; i++) {
    sb.append("Hello, ");
    sb.append(name);
    sb.append("!\n");
}
System.out.println(sb);
```



```kotlin
//Kotlin
val name = "Joe"
val s = buildString {
    repeat(5) {
        append("Hello, ")
        append(name)
        appendLine("!")
    }
}
println(s)
```

kotlinx.html

```
System.out.appendHTML().html {
    body {
        div {
            a("http://kotlinlang.org") {
                target = ATarget.blank
                +"Main site"
            }
        }
    }
}
```

Ktor

```kotlin
fun main() {
    embeddedServer(Netty, port = 8080, host = "0.0.0.0") {
        routing {
            get("/html-dsl") {
                call.respondHtml {
                    body {
                        h1 { +"HTML" }
                        ul {
                            for (n in 1..10) {
                                li { +"$n" }
                            }
                        }
                    }
                }
            }
        }
    }.start(wait = true)
}
```

Ktor

Ktor’s routing

```
fun main() {
    embeddedServer(Netty, port = 8080, host = "0.0.0.0") {
        routing {
            get("/html-dsl") {
                call.respondHtml {
                    body {
                        h1 { +"HTML" }
                        ul {
                            for (n in 1..10) {
                                li { +"$n" }
                            }
                        }
                    }
                }
            }
        }
    }.start(wait = true)
}
```

Ktor

Ktor’s routing

```
fun main() {
    embeddedServer(Netty, port = 8080, host = "0.0.0.0") {
        routing {
            get("/html-dsl") {
                call.respondHtml {
                    body {
                        h1 { +"HTML" }
                        ul {
                            for (n in 1..10) {
                                li { +"$n" }
                            }
                        }
                    }
                }
            }
        }
    }.start(wait = true)
}
```

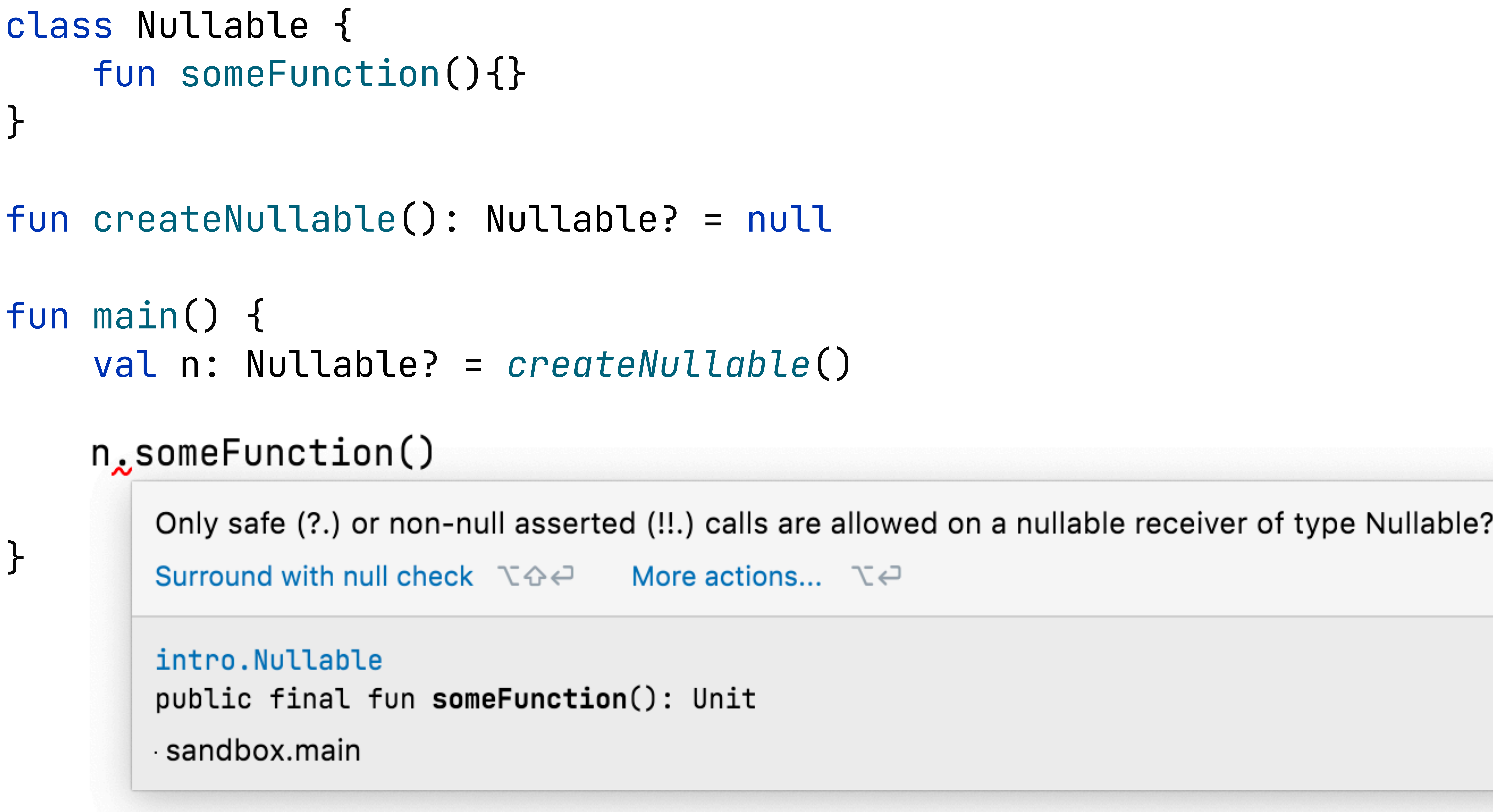
kotlinx.html

Вывод 7:

Лямбда с ресивером - это краеугольный камень для создания DSL на Kotlin

8. Null-safety

```kotlin
class Nullable {
    fun someFunction(){}
}

fun createNullable(): Nullable? = null
```



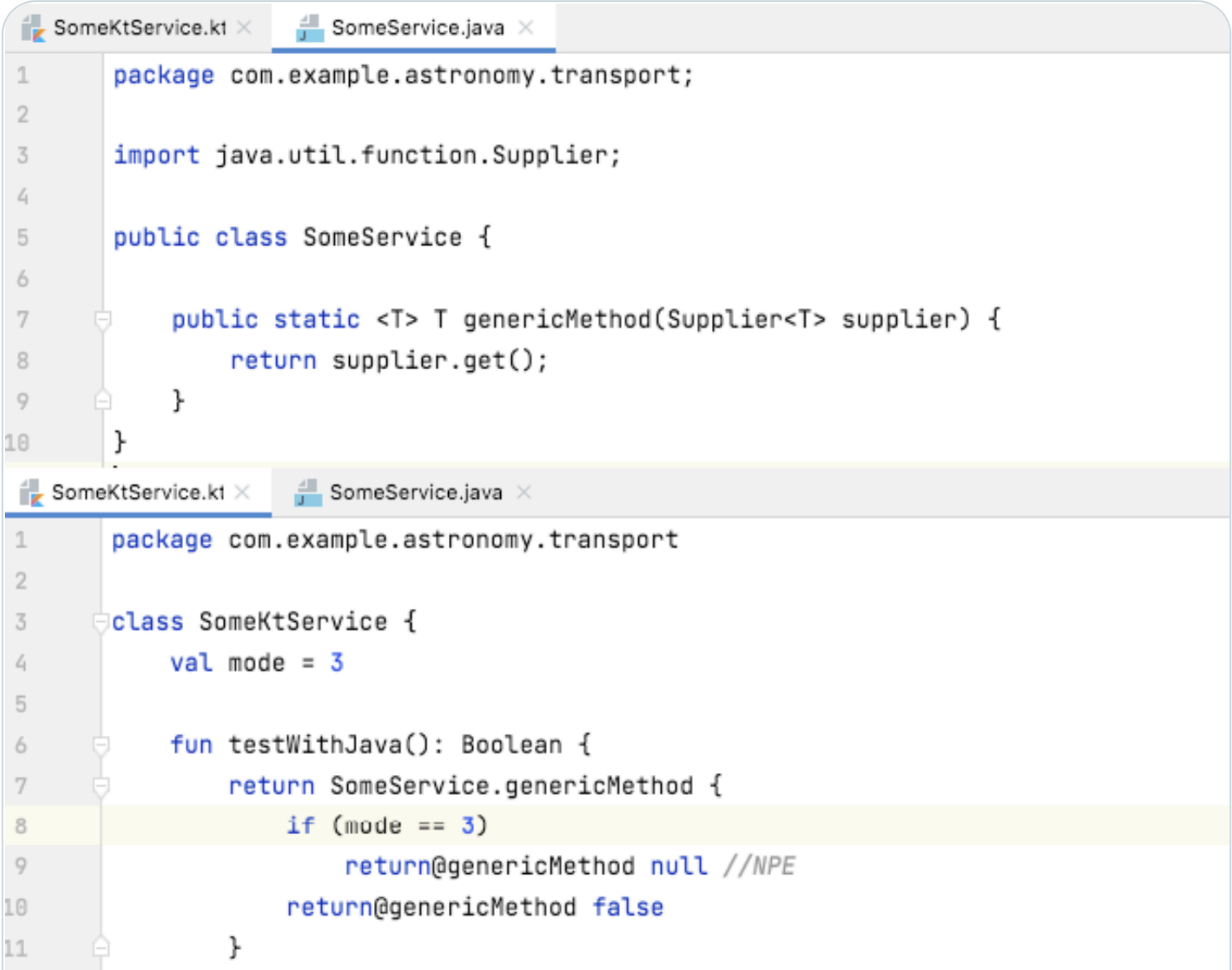
```kotlin
class Nullable {
    fun someFunction(){}
}

fun createNullable(): Nullable? = null

fun main() {
    val n: Nullable? = createNullable()

    n.someFunction()

}
```

Only safe (?.) or non-null asserted (!!.) calls are allowed on a nullable receiver of type Nullable?

```
intro.Nullable
public final fun someFunction(): Unit
```

Seems this breaks amazing @kotlin #null safety:

1- create a @Java class with the generic method accepting `Supplier<T>` as it is shown in the picture
2- call this method from a kotlin class returning `return@genericMethod null` under the if clause.
3- you made it!



```java
package com.example.astronomy.transport;

import java.util.function.Supplier;

public class SomeService {

    public static <T> T genericMethod(Supplier<T> supplier) {
        return supplier.get();
    }
}
```

```kotlin
package com.example.astronomy.transport

class SomeKtService {
    val mode = 3

    fun testWithJava(): Boolean {
        return SomeService.genericMethod {
            if (mode == 3)
                return@genericMethod null //NPE
            return@genericMethod false
        }
```

Seems this breaks amazing @kotlin #null safety:

1- create a @Java class with the generic method accepting `Supplier<T>` as it is shown in the picture
2- call this method from a kotlin class returning `return@genericMethod null` under the if clause.
3- you made it!

SomeKtService.kt
SomeService.java

```
package com.example.astronomy.transport

import java.util.function.Supplier;

public class SomeService {

    public static <T> T genericMethod(Supplier<T> supplier) {
        return supplier.get();
    }
}
```

```
package com.example.astronomy.transport

class SomeKtService {
    val mode = 3

    fun testWithJava(): Boolean {
        return SomeService.genericMethod {
            if (mode == 3)
                return@genericMethod null //NPE
            return@genericMethod false
        }
```

Yuriy Artamonov 🐧 @Yuriy_Artamonov · Se

Replying to @AlekseyStukalov @kotlin and @java

This is by design, Platform Types are not checked for nulls. So when you have interop with Java beware.

Consider using *null-safe call*

```kotlin
val order = retrieveOrder()

if (order == null || order.customer == null || order.customer.address == null){
    throw IllegalArgumentException("Invalid Order")
}
val city = order.customer.address.city
```

Consider using *null-safe call*

```
val order = retrieveOrder()

val city = order?.customer?.address?.city
```

Consider using *null-safe call*

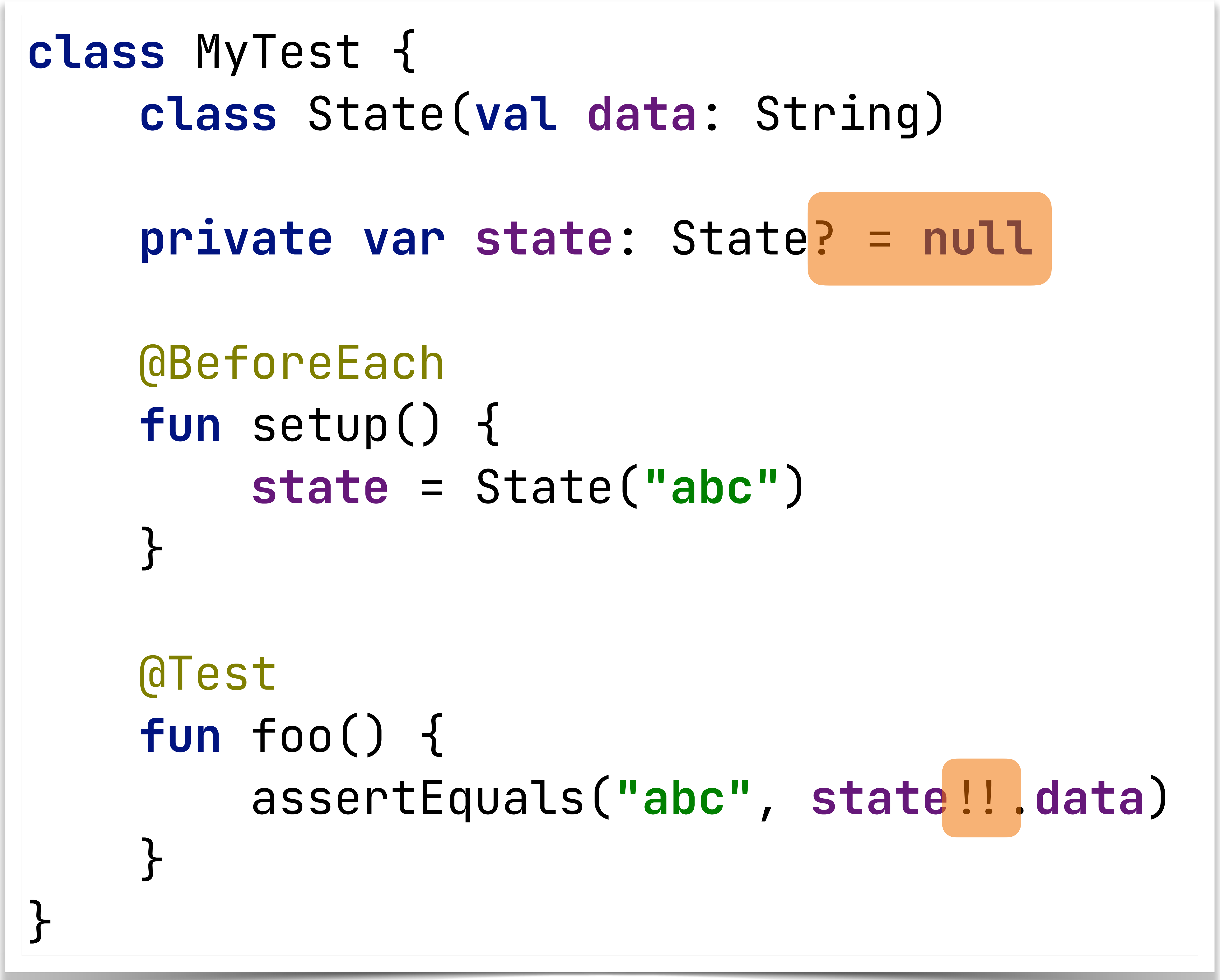
```
val order = retrieveOrder()

val city = order?.customer?.address?.city
    ?: throw IllegalArgumentException("Invalid Order")
```

Avoid not-null assertions !!

```
val order = retrieveOrder()

val city = order!!.customer!!.address!!.city
```

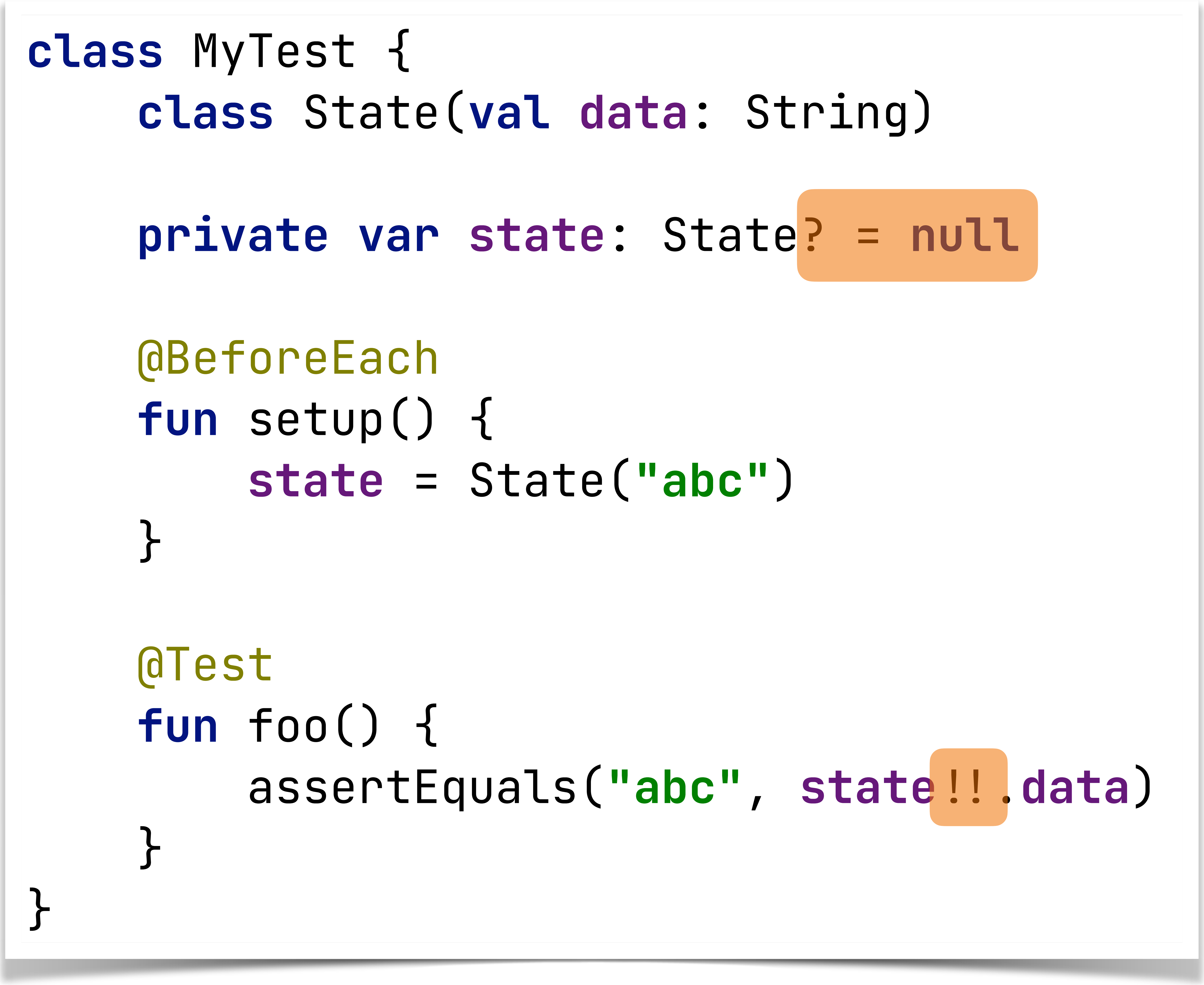
“You may notice that the double exclamation mark looks a bit rude:
it’s almost like you’re yelling at the compiler. This is intentional.” - ***Kotlin in Action***

Avoid not-null assertions !!

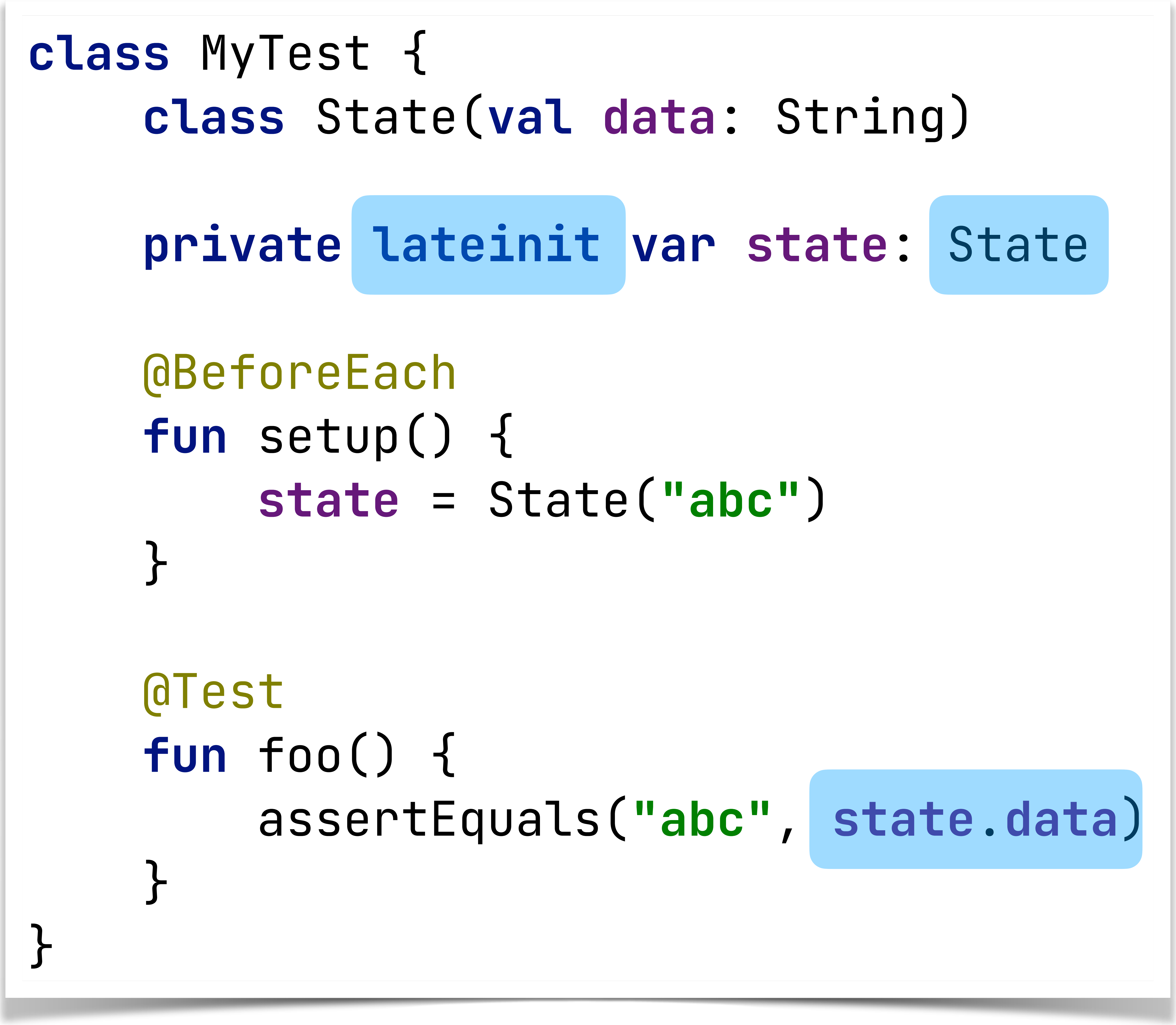


```kotlin
class MyTest {
    class State(val data: String)

    private var state: State? = null

    @BeforeEach
    fun setup() {
        state = State("abc")
    }

    @Test
    fun foo() {
        assertEquals("abc", state!!.data)
    }
}
```

Avoid not-null assertions !! - use `lateinit`



```kotlin
class MyTest {
    class State(val data: String)

    private var state: State? = null

    @BeforeEach
    fun setup() {
        state = State("abc")
    }

    @Test
    fun foo() {
        assertEquals("abc", state!!.data)
    }
}
```

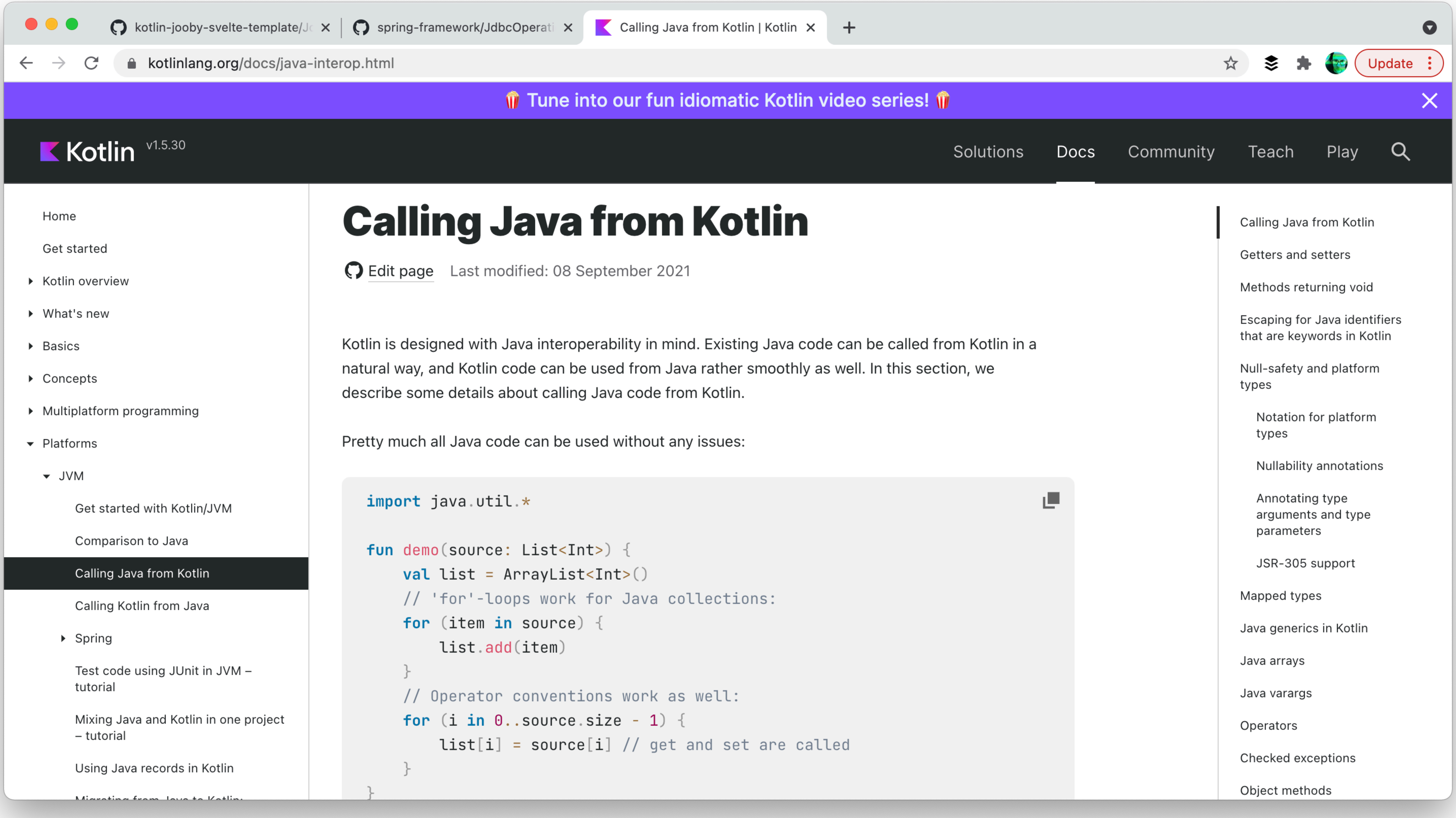


```kotlin
class MyTest {
    class State(val data: String)

    private lateinit var state: State

    @BeforeEach
    fun setup() {
        state = State("abc")
    }

    @Test
    fun foo() {
        assertEquals("abc", state.data)
    }
}
```

Вывод 8:

Null-safety - очень полезный инструмент,
но это не серебряная пуля

8. Совместимость с Java


kotlin-jooby-svelte-template/J
spring-framework/JdbcOperati
Calling Java from Kotlin | Kotlin
kotlinlang.org/docs/java-interop.html
Update
🍿 Tune into our fun idiomatic Kotlin video series! 🍿
Kotlin v1.5.30
Solutions
Docs
Community
Teach
Play

Home

Get started

Kotlin overview

What's new

Basics

Concepts

Multiplatform programming

Platforms

JVM

Get started with Kotlin/JVM

Comparison to Java

Calling Java from Kotlin

Calling Kotlin from Java

Spring

Test code using JUnit in JVM – tutorial

Mixing Java and Kotlin in one project – tutorial

Using Java records in Kotlin

Migrating from Java to Kotlin:

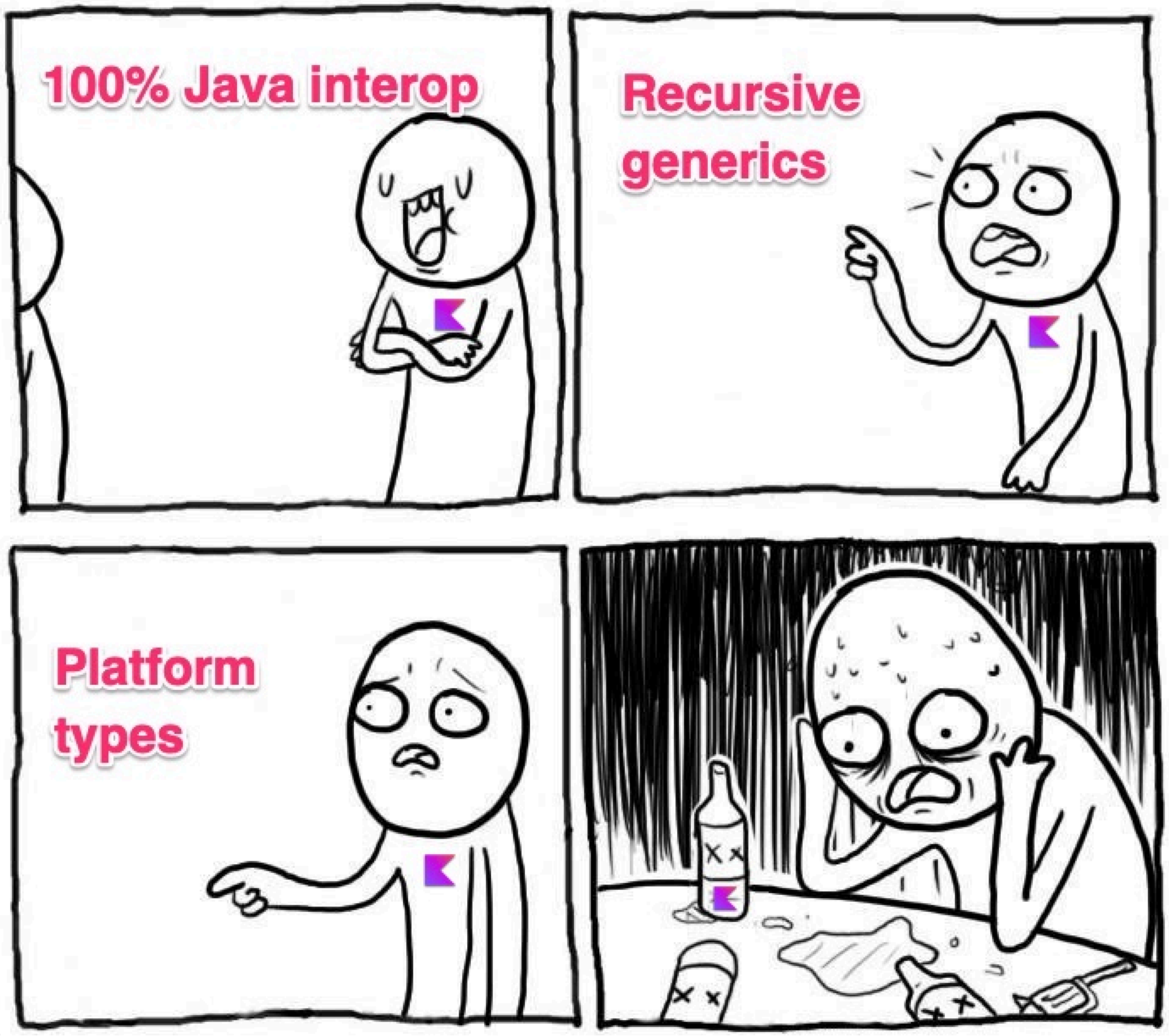
Calling Java from Kotlin

Edit page   Last modified: 08 September 2021

Kotlin is designed with Java interoperability in mind. Existing Java code can be called from Kotlin in a natural way, and Kotlin code can be used from Java rather smoothly as well. In this section, we describe some details about calling Java code from Kotlin.

Pretty much all Java code can be used without any issues:

```
import java.util.*

fun demo(source: List<Int>) {
    val list = ArrayList<Int>()
    // 'for'-loops work for Java collections:
    for (item in source) {
        list.add(item)
    }
    // Operator conventions work as well:
    for (i in 0..source.size - 1) {
        list[i] = source[i] // get and set are called
    }
}
```


100% Java interop
Recursive generics
Platform types

```java
static PostgreSQLContainer<?> container = new PostgreSQLContainer<>("postgres:13")
        .withInitScript("schema.sql")
        .withDatabaseName("database")
        .withUsername("user")
        .withPassword("password");
```

```
static PostgreSQLContainer<?> container = new PostgreSQLContainer<>("postgres:13")
        .withInitScript("schema.sql")
        .withDatabaseName("database")
        .withUsername("user")
        .withPassword("password");
```

```
class PostgreSQLContainer<SELF extends PostgreSQLContainer<SELF>> extends JdbcDatabaseContainer<SELF>
```

```java
static PostgreSQLContainer<?> container = new PostgreSQLContainer<>("postgres:13")
        .withInitScript("schema.sql")
        .withDatabaseName("database")
        .withUsername("user")
        .withPassword("password");
```

```java
class T<SELF extends T<SELF>> extends S<SELF>
```

```
static PostgreSQLContainer<?> container = new PostgreSQLContainer<>("postgres:13")
        .withInitScript("schema.sql")
        .withDatabaseName("database")
        .withUsername("user")
        .withPassword("password");
```

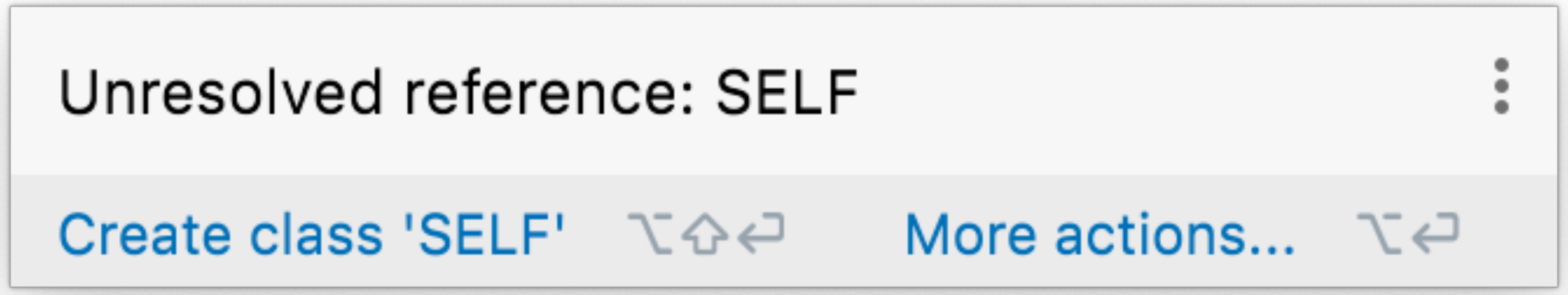
```
class PostgreSQLContainer<SELF extends PostgreSQLContainer<SELF>> extends JdbcDatabaseContainer<SELF>
```

```java
static PostgreSQLContainer<?> container = new PostgreSQLContainer<>("postgres:13")
        .withInitScript("schema.sql")
        .withDatabaseName("database")
        .withUsername("user")
        .withPassword("password");
```

```kotlin
val container: PostgreSQLContainer<*> = PostgreSQLContainer<SELF>("postgres:13")
    .withInitScript("schema.sql")
    .withDatabaseName("database")
    .withUsername("user")
    .withPassword("password")
```

```
static PostgreSQLContainer<?> container = new PostgreSQLContainer<>("postgres:13")
        .withInitScript("schema.sql")
        .withDatabaseName("database")
        .withUsername("user")
        .withPassword("password");
```

```
val container: PostgreSQLContainer<*> = PostgreSQLContainer<SELF>("postgres:13")
    .withInitScript("schema.sql")
    .withDatabaseName("database")
    .withUsername("user")
    .withPassword("password")
```



Unresolved reference: SELF

```java
static PostgreSQLContainer<?> container = new PostgreSQLContainer<>("postgres:13")
        .withInitScript("schema.sql")
        .withDatabaseName("database")
        .withUsername("user")
        .withPassword("password");
```

```kotlin
val container: PostgreSQLContainer<*> = PostgreSQLContainer<Nothing>("postgres:13")
    .withInitScript("schema.sql")
    .withDatabaseName("database")
    .withUsername("user")
    .withPassword("password")
```

```java
static PostgreSQLContainer<?> container = new PostgreSQLContainer<>("postgres:13")
        .withInitScript("schema.sql")
        .withDatabaseName("database")
        .withUsername("user")
        .withPassword("password");
```

```kotlin
val container: PostgreSQLContainer<*> = PostgreSQLContainer<Nothing>("postgres:13")
    .withInitScript("schema.sql") → val Nothing
    .withDatabaseName("database")
    .withUsername("user")
    .withPassword("password")
```

```java
static PostgreSQLContainer<?> container = new PostgreSQLContainer<>("postgres:13")
        .withInitScript("schema.sql")
        .withDatabaseName("database")
        .withUsername("user")
        .withPassword("password");
```

```kotlin
val container = PostgreSQLContainer<Nothing>(“postgres:13”).apply {
    withInitScript("schema.sql")
    withDatabaseName("database")
    withUsername("user")
    withPassword("password")
}
```

```java
static PostgreSQLContainer<?> container = new PostgreSQLContainer<>("postgres:13")
        .withInitScript("schema.sql")
        .withDatabaseName("database")
        .withUsername("user")
        .withPassword("password");
```

```kotlin
val container = PostgreSQLContainer(“postgres:13”)
    .withInitScript(“schema.sql")
    .withDatabaseName(“database")
    .withUsername(“user")
    .withPassword(“password")
```

1.5.30

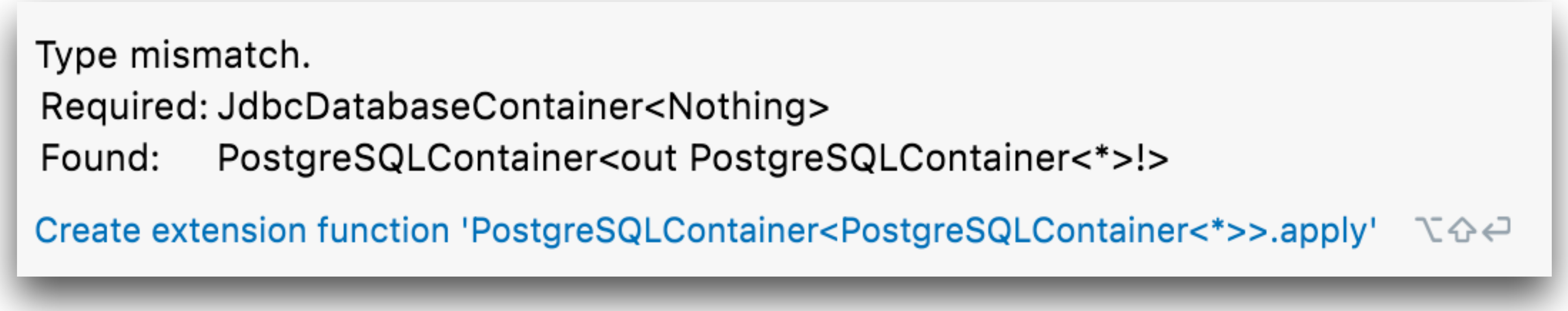
```
static PostgreSQLContainer<?> container = new PostgreSQLContainer<>("postgres:13")
        .withInitScript("schema.sql")
        .withDatabaseName("database")
        .withUsername("user")
        .withPassword("password");
```

```
val container = postgres("13-alpine") {
    withDatabaseName("db")
    withUsername("user")
    withPassword("password")
    withInitScript("sql/schema.sql")
}
```

```kotlin
fun postgres(version: String, options: JdbcDatabaseContainer.() -> Unit) =
    PostgreSQLContainer(DockerImageName.parse("postgres:$version")).apply(options)
```

```kotlin
val container = postgres("13-alpine") {
    withDatabaseName("db")
    withUsername("user")
    withPassword("password")
    withInitScript("sql/schema.sql")
}
```

```kotlin
fun postgres(version: String, options: JdbcDatabaseContainer<Nothing>.() → Unit) =
    PostgreSQLContainer(DockerImageName.parse("postgres:$version")).apply(options)
```



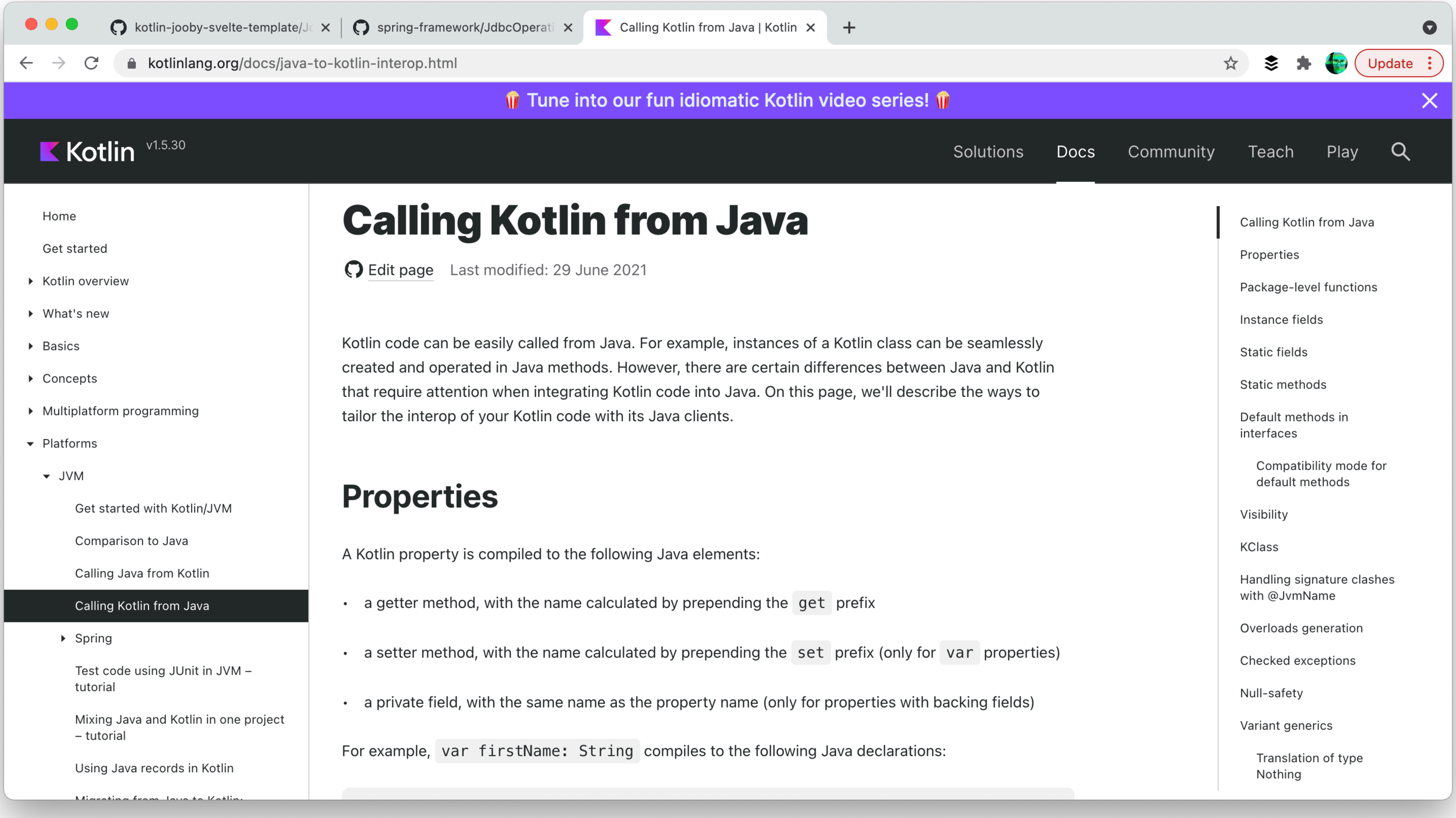
Type mismatch.
Required: JdbcDatabaseContainer<Nothing>
Found: PostgreSQLContainer<out PostgreSQLContainer<*>!>

Create extension function 'PostgreSQLContainer<PostgreSQLContainer<*>>.apply' ⌥⇧↩

```kotlin
val container = postgres("13-alpine") {
    withDatabaseName("db")
    withUsername("user")
    withPassword("password")
    withInitScript("sql/schema.sql")
}
```

```kotlin
fun postgres(version: String, options: JdbcDatabaseContainer<Nothing>.() -> Unit) =
PostgreSQLContainer<Nothing>(DockerImageName.parse(“postgres:$version")).apply(options
```

```kotlin
val container = postgres("13-alpine") {
    withDatabaseName("db")
    withUsername("user")
    withPassword("password")
    withInitScript("sql/schema.sql")
}
```


kotlin-jooby-svelte-template/J
spring-framework/JdbcOperati
Calling Kotlin from Java | Kotlin
kotlinlang.org/docs/java-to-kotlin-interop.html
Update
🍿 Tune into our fun idiomatic Kotlin video series! 🍿
Kotlin v1.5.30
Solutions
Docs
Community
Teach
Play
Home
Get started
Kotlin overview
What's new
Basics
Concepts
Multiplatform programming
Platforms
JVM
Get started with Kotlin/JVM
Comparison to Java
Calling Java from Kotlin
Calling Kotlin from Java
Spring
Test code using JUnit in JVM – tutorial
Mixing Java and Kotlin in one project – tutorial
Using Java records in Kotlin
Migrating from Java to Kotlin:
Calling Kotlin from Java
Properties
Package-level functions
Instance fields
Static fields
Static methods
Default methods in interfaces
Compatibility mode for default methods
Visibility
KClass
Handling signature clashes with @JvmName
Overloads generation
Checked exceptions
Null-safety
Variant generics
Translation of type Nothing

Calling Kotlin from Java

Edit page   Last modified: 29 June 2021

Kotlin code can be easily called from Java. For example, instances of a Kotlin class can be seamlessly created and operated in Java methods. However, there are certain differences between Java and Kotlin that require attention when integrating Kotlin code into Java. On this page, we'll describe the ways to tailor the interop of your Kotlin code with its Java clients.

Properties

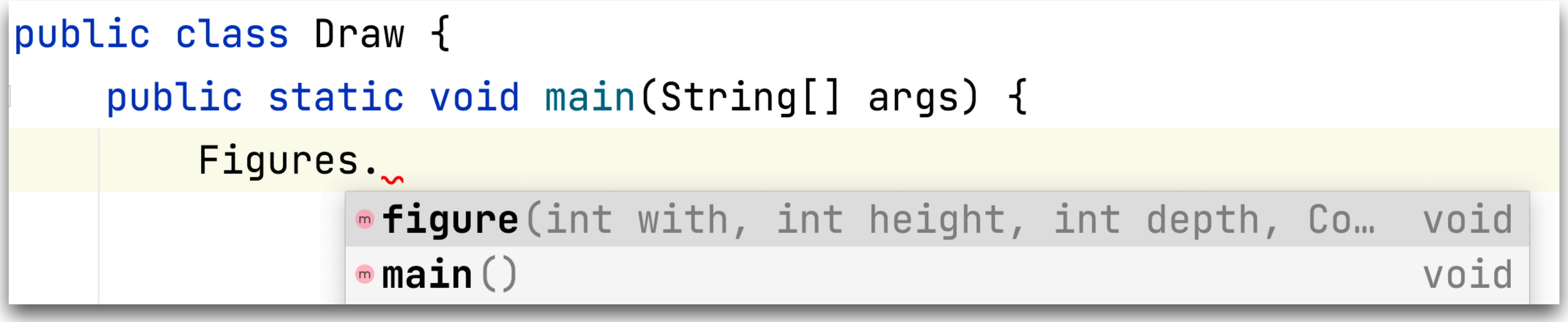
A Kotlin property is compiled to the following Java elements:

- a getter method, with the name calculated by prepending the `get` prefix
- a setter method, with the name calculated by prepending the `set` prefix (only for `var` properties)
- a private field, with the same name as the property name (only for properties with backing fields)

For example, `var firstName: String` compiles to the following Java declarations:

```kotlin
fun figure(
    with: Int = 1,
    height: Int = 1,
    depth: Int = 1,
    color: Color = Color.BLACK,
    description: String = "",
) {}
```

В Java видна только одна функция со всеми параметрами

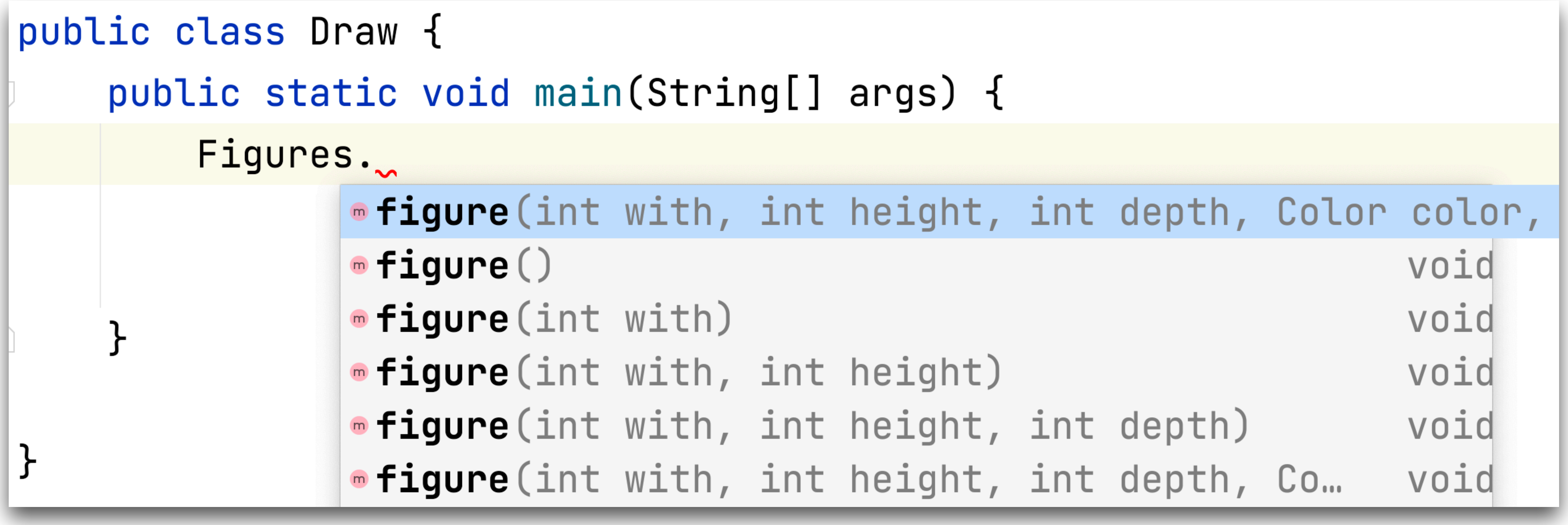


```java
public class Draw {
    public static void main(String[] args) {
        Figures.
```

```
@JvmOverloads
fun figure(
    with: Int = 1,
    height: Int = 1,
    depth: Int = 1,
    color: Color = Color.BLACK,
    description: String = "",
) {}
```

```kotlin
@JvmOverloads
fun figure(
    with: Int = 1,
    height: Int = 1,
    depth: Int = 1,
    color: Color = Color.BLACK,
    description: String = "",
) {}
```

Теперь есть все варианты

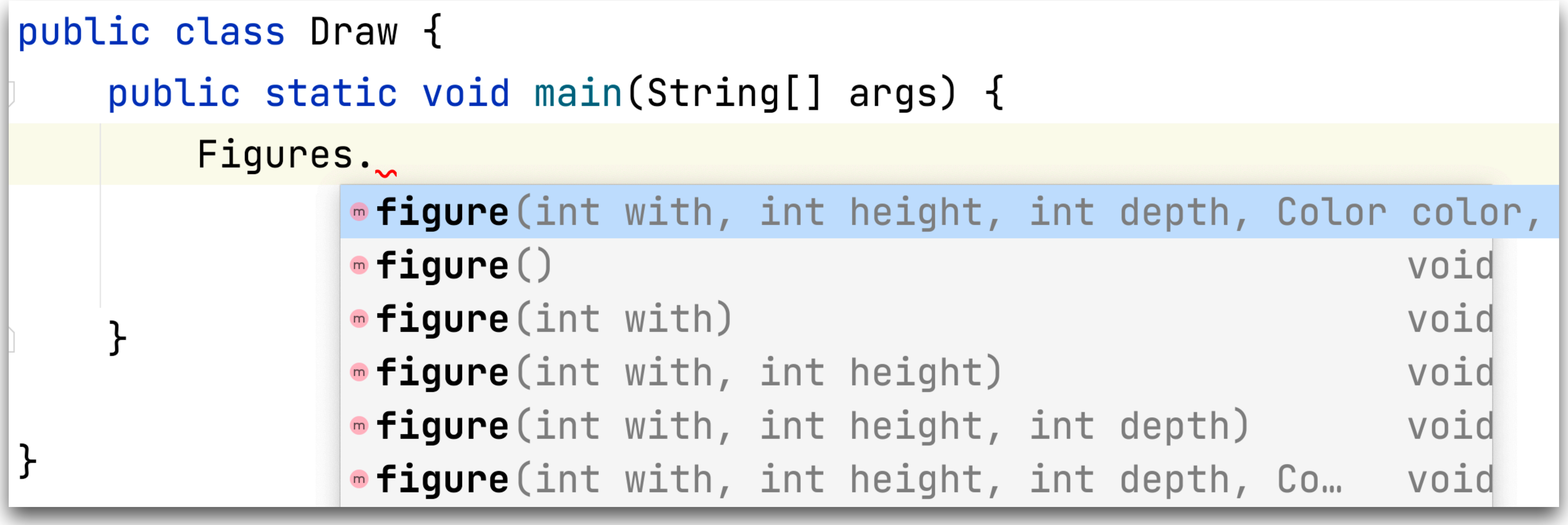


```java
public class Draw {
    public static void main(String[] args) {
        Figures.
    }
}
```

```
figure(int with, int height, int depth, Color color,
figure()                                                  void
figure(int with)                                          void
figure(int with, int height)                              void
figure(int with, int height, int depth)                   void
figure(int with, int height, int depth, Co…               void
```

```kotlin
@file:JvmName("Figures")
package me.anton

@JvmOverloads

fun figure(
    with: Int = 1,
    height: Int = 1,
    depth: Int = 1,
    color: Color = Color.BLACK,
    description: String = "",
) {}
```

Теперь есть все варианты



```java
public class Draw {
    public static void main(String[] args) {
        Figures.
    }
}
```

```
m figure(int with, int height, int depth, Color color,
m figure()                                                 void
m figure(int with)                                         void
m figure(int with, int height)                             void
m figure(int with, int height, int depth)                  void
m figure(int with, int height, int depth, Co…              void
```

Вывод 8:

а) Совместимость Kotlin с Java очень важна

б) Не на 100% бесшовная

ц) … но очень хорошая и улучшается! :)

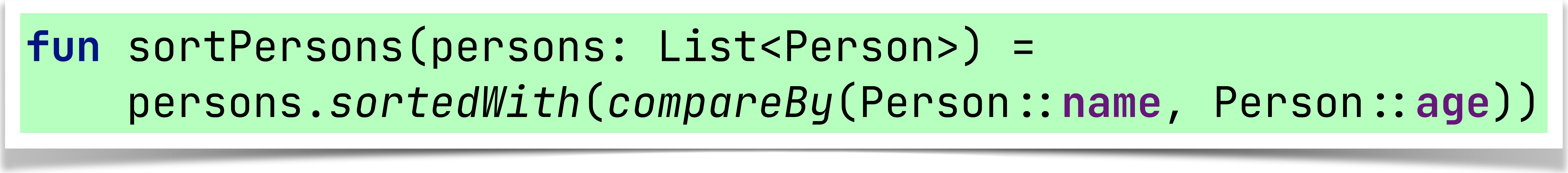
9. Стандартная библиотека

compareBy

```
class Person(
    val name: String,
    val age: Int
)

fun sortPersons(persons: List<Person>) =
    persons.sortedWith(Comparator<Person> { person1, person2 →
        val rc = person1.name.compareTo(person2.name)
        if (rc ≠ 0)
            rc
        else
            person1.age - person2.age
    })
```

compareBy

```
class Person(
    val name: String,
    val age: Int
)

fun sortPersons(persons: List<Person>) =
    persons.sortedWith(Comparator<Person> { person1, person2 →
        val rc = person1.name.compareTo(person2.name)
        if (rc ≠ 0)
            rc
        else
            person1.age - person2.age
    })
```



```
fun sortPersons(persons: List<Person>) =
    persons.sortedWith(compareBy(Person::name, Person::age))
```

groupBy

```
class Request(
    val url: String,
    val remoteIP: String,
    val timestamp: Long
)

fun analyzeLog(log: List<Request>) {
    val map = mutableMapOf<String, MutableList<Request>>()
    for (request in log) {
        map.getOrPut(request.url) { mutableListOf() }
            .add(request)
    }
}
```

groupBy

```
class Request(
    val url: String,
    val remoteIP: String,
    val timestamp: Long
)

fun analyzeLog(log: List<Request>) {
    val map = mutableMapOf<String, MutableList<Request>>()
    for (request in log) {
        map.getOrPut(request.url) { mutableListOf() }
            .add(request)
    }
}
```

```
fun analyzeLog(log: List<Request>) {
    val map = log.groupBy(Request::url)
}
```

```kotlin
"Ja49va1K7otl54in7".partition { "123456789".contains(it)}
//(4917547, JavaKotlin)

"L7ea3rn 1Kotl8in 12".filter { it.isDigit() } // 731812

"L7ea3rn 1Kotl8in 12F".findLast { it.isDigit() } // 2

"Learn Kotlin".substringAfter("a") // rn Kotlin

"Learning Kotlin".padStart(20, '.') // ...Learning Kotlin

"Learn Kotlin".all { it.isLowerCase() } // false
```

```
listOf(1, 2, 3, 4, 5).zip(listOf(6, 7, 8, 9))
//Output: [(1, 6), (2, 7), (3, 8), (4, 9)]

listOf(1, 2, 3, 4, 5).zipWithNext()
//Output: [(1, 2), (2, 3), (3, 4), (4, 5)]

listOf(1 to 2, 3 to 4, 5 to 6).unzip()
//Output: ([1, 3, 5], [2, 4, 6])

listOf(1, 2, 3, 4, 5, 6, 7, 8, 9).chunked(4)
//Output: [[1, 2, 3, 4], [5, 6, 7, 8], [9]]

listOf(1, 2, 3, 4, 5, 6, 7, 8, 9).windowed(4, 2, true)
//Output: [[1, 2, 3, 4], [3, 4, 5, 6], [5, 6, 7, 8], [7, 8, 9], [9]]

listOf(1, 2, 3, 4, 5, 6, 7, 8, 9).partition { it % 2 == 0 }
//Output: ([2, 4, 6, 8], [1, 3, 5, 7, 9])
```

filterIsInstance

```
fun findAllStrings(objects: List<Any>) =
    objects.filter { it is String }
```

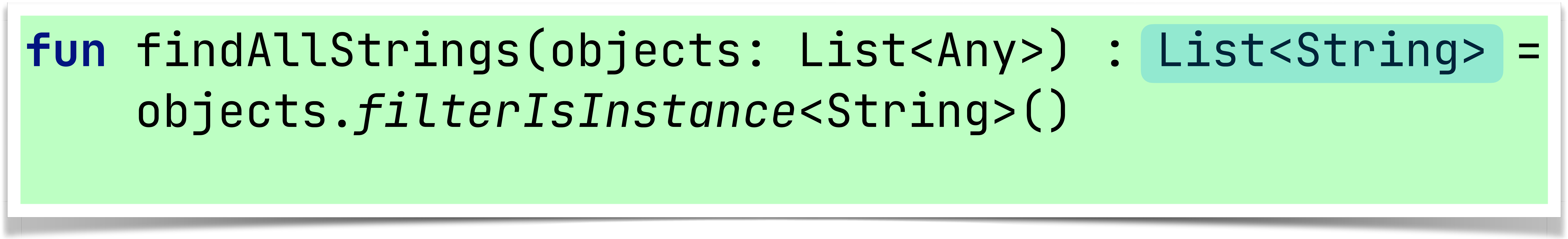
filterIsInstance

```
fun findAllStrings(objects: List<Any>) =
    objects.filter { it is String }
```

```
fun findAllStrings(objects: List<Any>) =
    objects.filterIsInstance<String>()
```

filterIsInstance

```kotlin
fun findAllStrings(objects: List<Any>) : List<Any> =
    objects.filter { it is String }
```



```kotlin
fun findAllStrings(objects: List<Any>) : List<String> =
    objects.filterIsInstance<String>()
```

```kotlin
File("/my/file.txt").readText()

File("/my/file.txt").writeText("Hello!")

File("/my/file.txt").forEachLine { println(it) }
```

Result<T>

```
fun someFunction(): Result<String> =
    try {
        Result.success("Yes")
    } catch (e: Exception){
        Result.failure(e)
    }
```

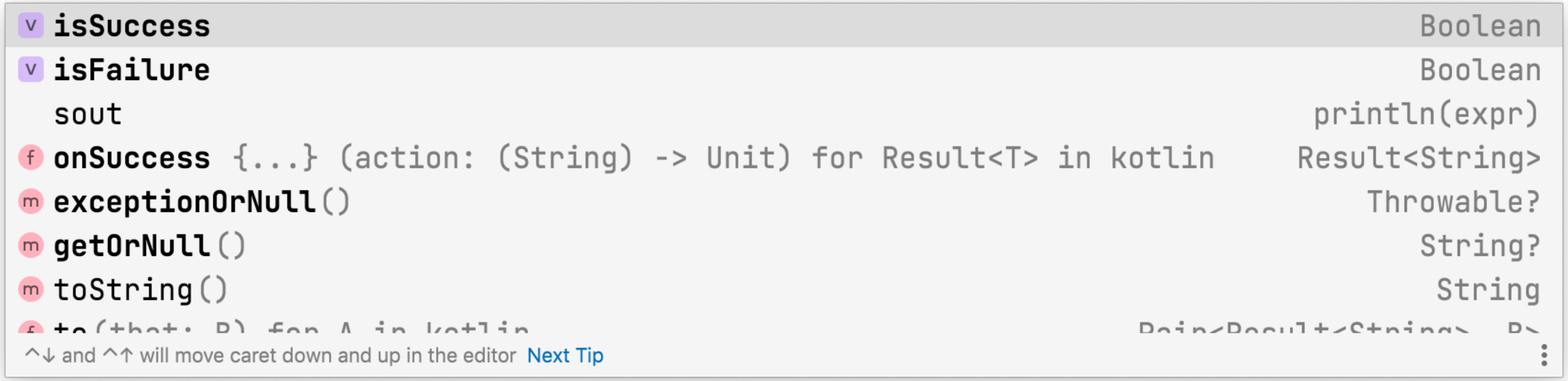
Result<T>

```
fun someFunction(): Result<String> =
    try {
        Result.success("Yes")
    } catch (e: Exception){
        Result.failure(e)
    }
```

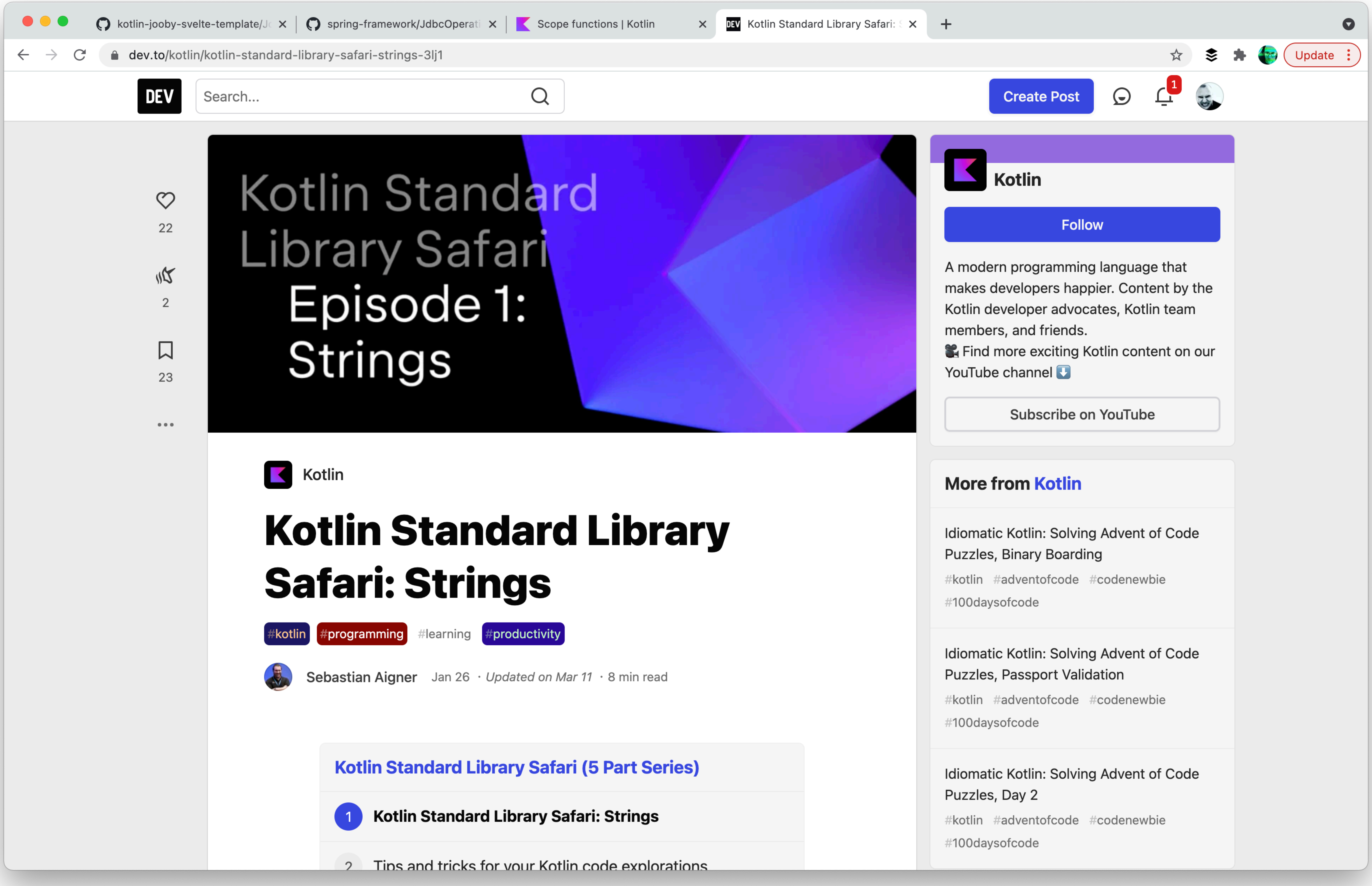
```
fun someOtherFunction() {
    someFunction().
}
```

Result<T>

```
fun someFunction(): Result<String> =
    try {
        Result.success("Yes")
    } catch (e: Exception){
        Result.failure(e)
    }


fun someOtherFunction() {
    someFunction().
}
```



v isSuccess Boolean
v isFailure Boolean
sout println(expr)
f onSuccess {...} (action: (String) -> Unit) for Result<T> in kotlin Result<String>
m exceptionOrNull() Throwable?
m getOrNull() String?
m toString() String
f to(that: B) for A in kotlin Pair<Result<String>, B>
^↓ and ^↑ will move caret down and up in the editor Next Tip


kotlin-jooby-svelte-template/J
spring-framework/JdbcOperati
Scope functions | Kotlin
Kotlin Standard Library Safari:
dev.to/kotlin/kotlin-standard-library-safari-strings-3lj1
Update
DEV
Search...
Create Post
1
22
2
23
Kotlin Standard Library Safari
Episode 1: Strings

Kotlin

Kotlin Standard Library Safari: Strings

#kotlin #programming #learning #productivity

Sebastian Aigner Jan 26 · *Updated on Mar 11* · 8 min read

Kotlin Standard Library Safari (5 Part Series)

1 **Kotlin Standard Library Safari: Strings**

2 Tips and tricks for your Kotlin code explorations

Kotlin

Follow

A modern programming language that makes developers happier. Content by the Kotlin developer advocates, Kotlin team members, and friends.
🎥 Find more exciting Kotlin content on our YouTube channel ⬇️

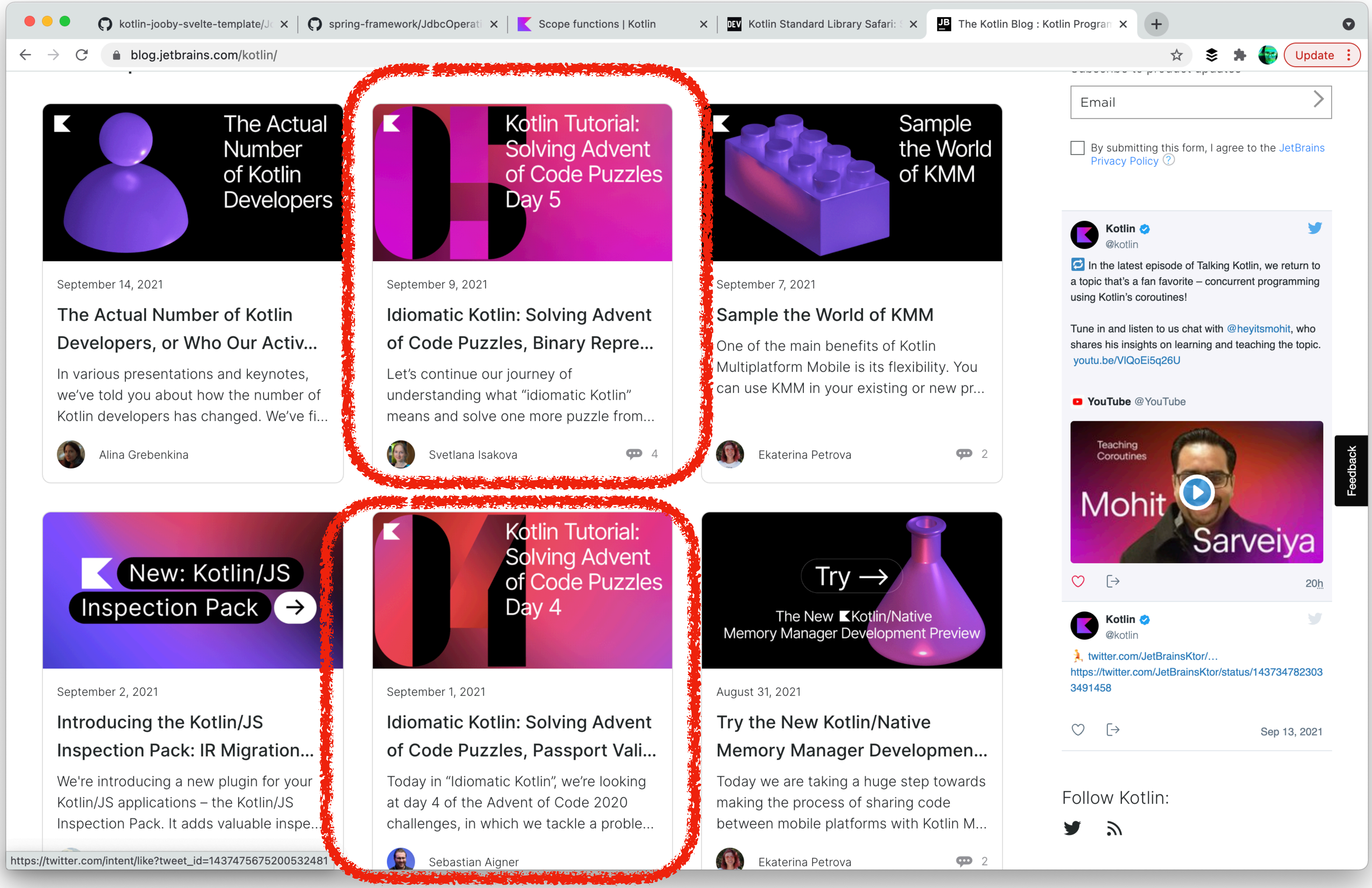
Subscribe on YouTube

More from Kotlin

Idiomatic Kotlin: Solving Advent of Code Puzzles, Binary Boarding
#kotlin #adventofcode #codenewbie #100daysofcode

Idiomatic Kotlin: Solving Advent of Code Puzzles, Passport Validation
#kotlin #adventofcode #codenewbie #100daysofcode

Idiomatic Kotlin: Solving Advent of Code Puzzles, Day 2
#kotlin #adventofcode #codenewbie #100daysofcode


kotlin-jooby-svelte-template/J
spring-framework/JdbcOperati
Scope functions | Kotlin
Kotlin Standard Library Safari:
The Kotlin Blog : Kotlin Progran
blog.jetbrains.com/kotlin/
Update
The Actual Number of Kotlin Developers
Kotlin Tutorial: Solving Advent of Code Puzzles Day 5
Sample the World of KMM
New: Kotlin/JS Inspection Pack
Kotlin Tutorial: Solving Advent of Code Puzzles Day 4
Try →
The New Kotlin/Native Memory Manager Development Preview
Teaching Coroutines
Mohit Sarveiya
Feedback

Email

□ By submitting this form, I agree to the JetBrains Privacy Policy

September 14, 2021

The Actual Number of Kotlin Developers, or Who Our Activ...

In various presentations and keynotes, we've told you about how the number of Kotlin developers has changed. We've fi...

Alina Grebenkina

September 9, 2021

Idiomatic Kotlin: Solving Advent of Code Puzzles, Binary Repre...

Let's continue our journey of understanding what "idiomatic Kotlin" means and solve one more puzzle from...

Svetlana Isakova 4

September 7, 2021

Sample the World of KMM

One of the main benefits of Kotlin Multiplatform Mobile is its flexibility. You can use KMM in your existing or new pr...

Ekaterina Petrova 2

September 2, 2021

Introducing the Kotlin/JS Inspection Pack: IR Migration...

We're introducing a new plugin for your Kotlin/JS applications – the Kotlin/JS Inspection Pack. It adds valuable inspe...

September 1, 2021

Idiomatic Kotlin: Solving Advent of Code Puzzles, Passport Vali...

Today in "Idiomatic Kotlin", we're looking at day 4 of the Advent of Code 2020 challenges, in which we tackle a proble...

Sebastian Aigner

August 31, 2021

Try the New Kotlin/Native Memory Manager Developmen...

Today we are taking a huge step towards making the process of sharing code between mobile platforms with Kotlin M...

Ekaterina Petrova 2

https://twitter.com/intent/like?tweet_id=1437475675200532481

Kotlin @kotlin

In the latest episode of Talking Kotlin, we return to a topic that's a fan favorite – concurrent programming using Kotlin's coroutines!

Tune in and listen to us chat with @heyitsmohit, who shares his insights on learning and teaching the topic. youtu.be/VlQoEi5q26U

YouTube @YouTube

20h

Kotlin @kotlin

twitter.com/JetBrainsKtor/... https://twitter.com/JetBrainsKtor/status/1437347823033491458

Sep 13, 2021

Follow Kotlin:

Home
Get started
Kotlin overview
What's new
Basics
Concepts
Multiplatform programming
Platforms
Releases and roadmap
Standard library
Collections
Scope functions
Opt-in requirements
Official libraries
API reference
Language reference
Tools
Learning materials
Other resources

Standard library / Opt-in requirements

Opt-in requirements

Edit page Last modified: 08 September 2021

The opt-in requirement annotations `@RequiresOptIn` and `@OptIn` are Experimental. They may be dropped or changed at any time. Opt-in is required (see details below). Use them only for evaluation purposes. We appreciate your feedback on it in YouTrack ↗.

`@RequireOptIn` and `@OptIn` annotations were introduced in 1.3.70 to replace previously used `@Experimental` and `@UseExperimental`; at the same time, `-opt-in` compiler option replaced `-Xuse-experimental`.

The Kotlin standard library provides a mechanism for requiring and giving explicit consent for using certain elements of APIs. This mechanism lets library developers inform users of their APIs about specific conditions that require opt-in, for example, if an API is in the experimental state and is likely to change in the future.

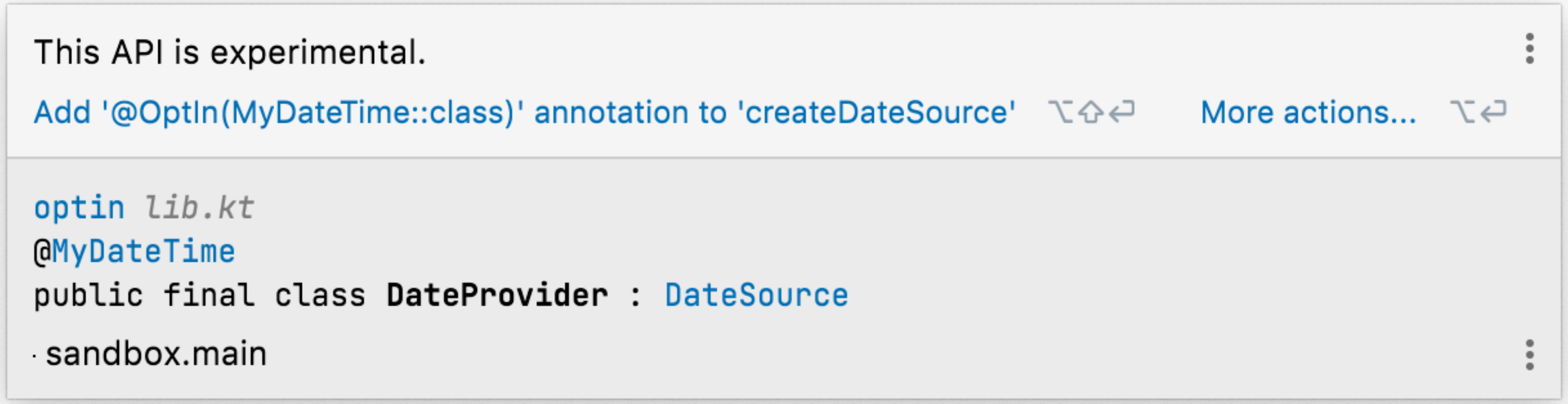
To prevent potential issues, the compiler warns users of such APIs about these conditions and requires them to opt in before using the API.

Opt in to using API

If a library author marks a declaration from a library's API as requiring opt-in, you should give an explicit consent for using it in your code. There are several ways to opt in to such APIs, all applicable without technical limitations. You are free to choose the way that you find best for your situation.

```kotlin
@RequiresOptIn(message = "This API is experimental.")
@Retention(AnnotationRetention.BINARY)
@Target(AnnotationTarget.FUNCTION, AnnotationTarget.CLASS)
annotation class MyDateTime // Opt-in requirement annotation

@MyDateTime
class DateProvider
```

```
@RequiresOptIn(message = "This API is experimental.")
@Retention(AnnotationRetention.BINARY)
@Target(AnnotationTarget.FUNCTION, AnnotationTarget.CLASS)
annotation class MyDateTime // Opt-in requirement annotation

@MyDateTime
class DateProvider



fun createDateSource() = DateProvider()
```



This API is experimental.

Add '@OptIn(MyDateTime::class)' annotation to 'createDateSource' ⌥⇧↩
More actions... ⌥↩

```
optin lib.kt
@MyDateTime
public final class DateProvider : DateSource
```

sandbox.main

```kotlin
@RequiresOptIn(message = "This API is experimental.")
@Retention(AnnotationRetention.BINARY)
@Target(AnnotationTarget.FUNCTION, AnnotationTarget.CLASS)
annotation class MyDateTime // Opt-in requirement annotation

@MyDateTime
class DateProvider



@OptIn(MyDateTime::class)
fun createDateSource() = DateProvider()
```

Вывод 9:

В стандартной библиотеке очень много полезного

@antonarhipov