



Java 9 Модули. Почему не OSGi?

Никита Липский

Excelsior LLC

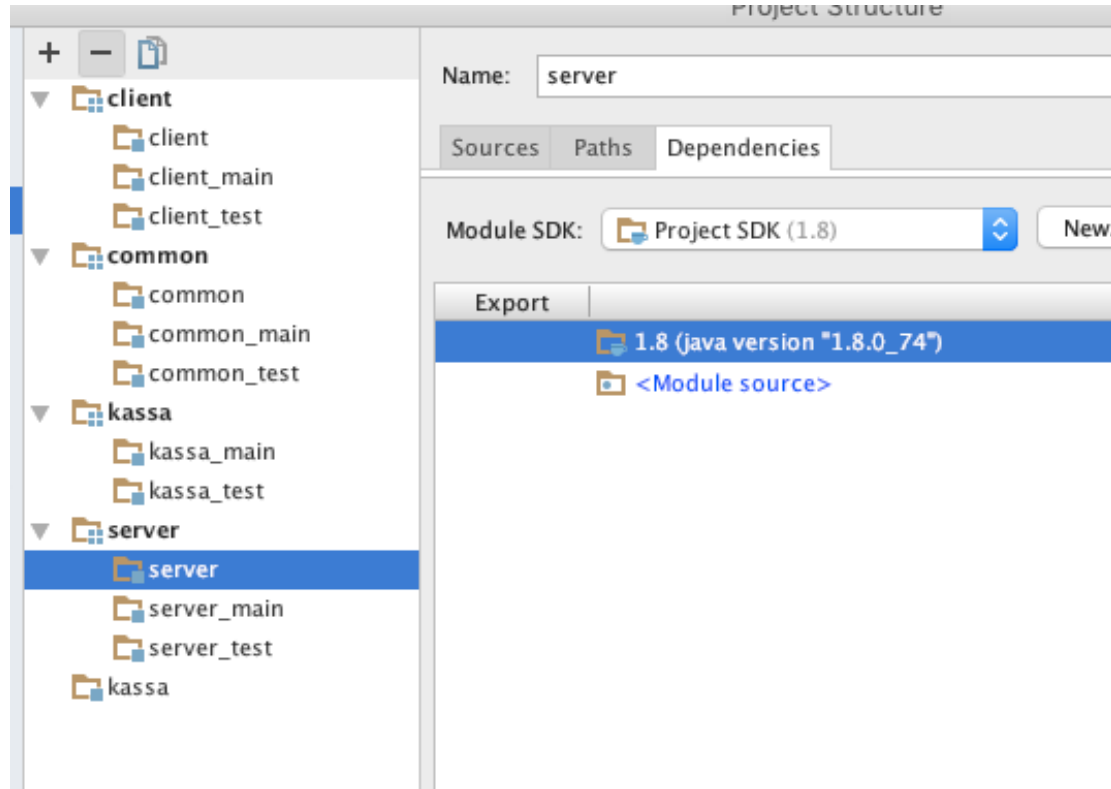
случаи
рассказы
дневниковые
записи

ДАНИИЛ ХАРМС

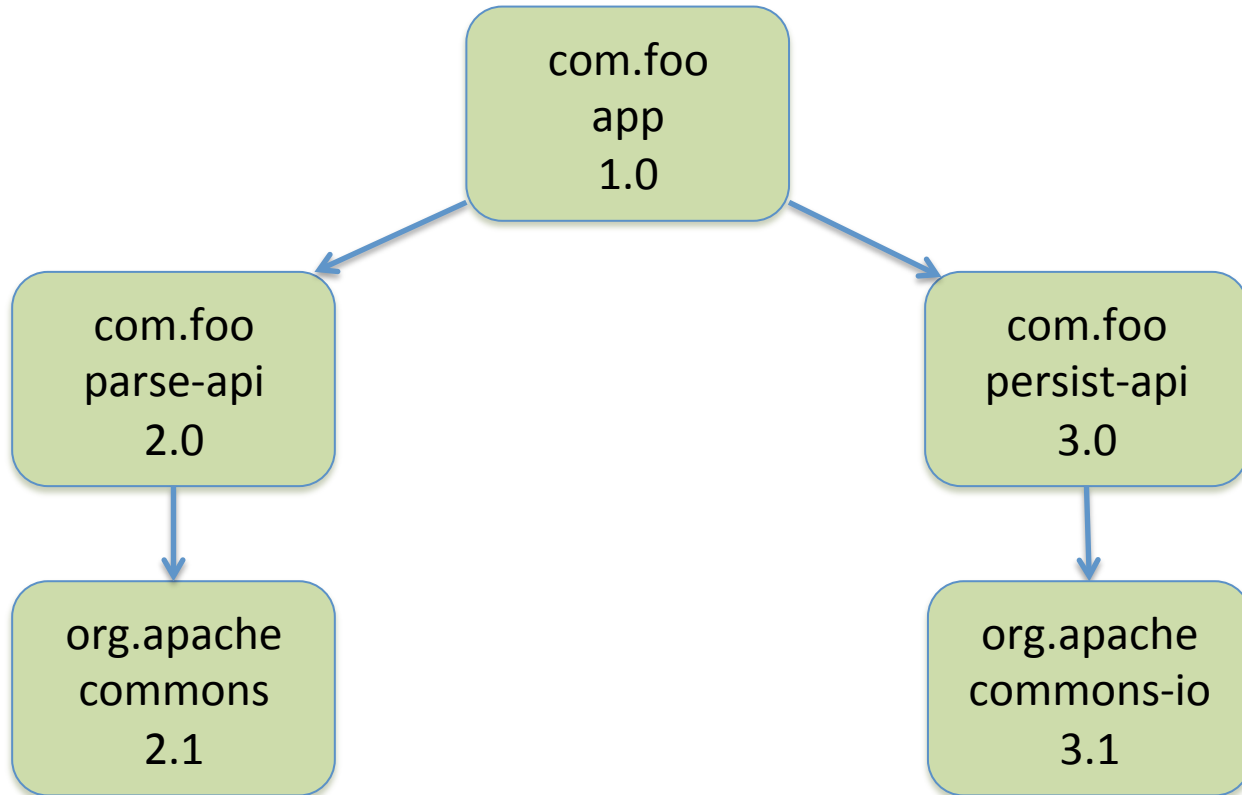
ГОРЛО
БРЕДИТ
БРИТВОЮ

ГЛАГОЛ
4-1991

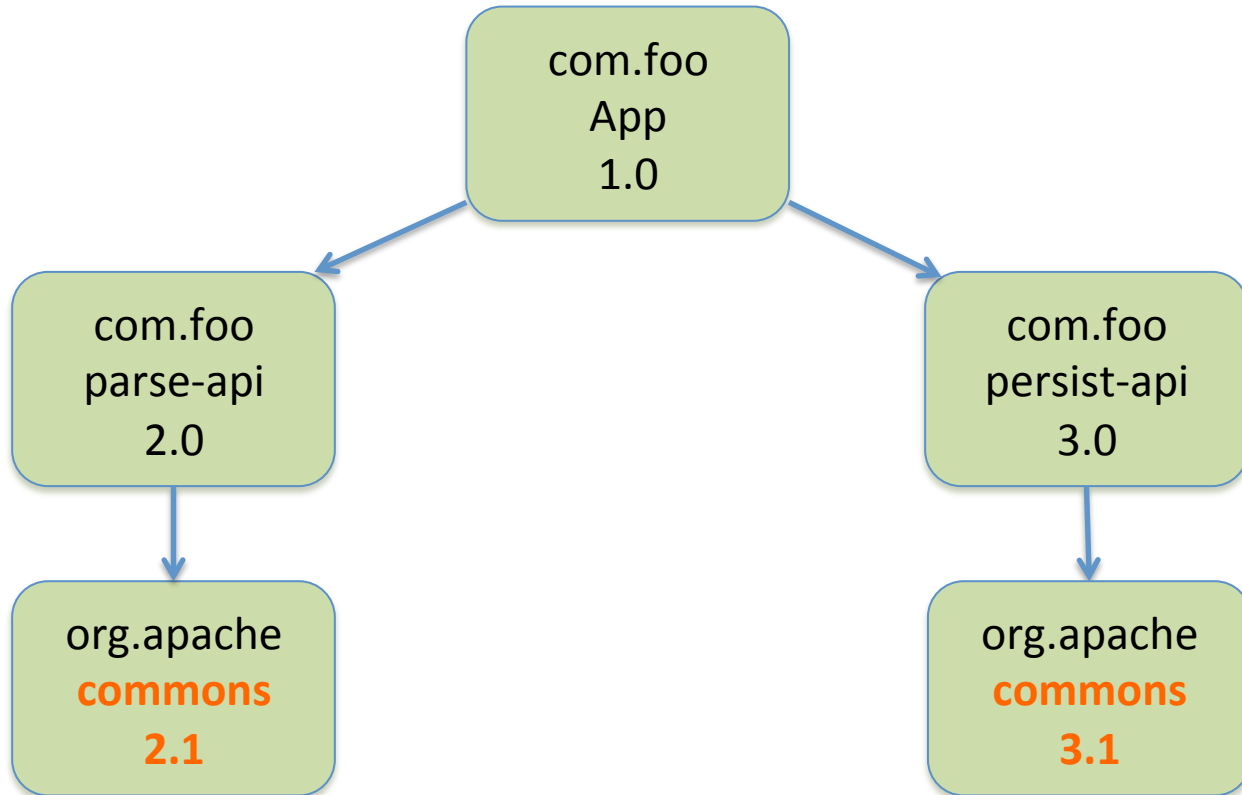
Модули в IDE



Модули в Maven



Модули в Maven



OSGi



Canadian Indian Chief

Photo by E. Oso

Почему не OSGi?

Mark Reinhold (the Chief Architect of the Java Platform Group):

“...As it (OSGi) stands, moreover, it’s useful for library and application modules but, since it’s built strictly on top of the Java SE Platform, it can’t be used to modularize the Platform itself”

Почему не OSGi?

Вопрос: почему наличие `j.l.Object` в спецификации языка Java, реализованного в свою очередь на языке Java, не приводит к bootstrap проблеме?

Почему не OSGi?

Вопрос: почему наличие `java.lang.Object` в спецификации языка Java, реализованного в свою очередь на языке Java, не приводит к bootstrap проблеме?

Факт: Существует реализация Java SE, где OSGi поддерживается на уровне JVM (на уровне платформы).

про себя

- Более 20 лет профессиональной карьеры
- Инициатор проекта Excelsior JET
 - работал над проектом более 17 лет
 - как идейный вдохновитель
 - компиляторный инженер
 - руководитель
 - и много в каких еще ролях
- twitter: @pjBooms

План доклада

- Почему OSGi
- Как OSGi
- Почему НЕ OSGi
- Почему Jigsaw не OSGi
- Jigsaw мантра
- Проблемы Jigsaw

Где OSGi?

OSGi – это стандарт OSGi alliance (IBM, Adobe, Bosch, Huawei, NTT, Oracle)

Реализации:

- Equinox
 - Eclipse IDE, Eclipse RCP, IBM Websphere
- Apache Felix
 - Oracle WebLogic, Glassfish, Netbeans

Почему OSGi?



Почему OSGi?

- Модульность
 - Снижение сложности (Reduced complexity)
 - Инкапсуляция (Hide internals)
 - Управление зависимостями и легкость развертывания (Easy deployment)
- Динамические обновления (Dynamic updates)
- Версионирование (Versioning)
- Ленивая активация модулей (Lazy)
- Simple, fast, small, secure, etc.

<https://www.osgi.org/developer/benefits-of-using-osgi/>



Export



Import

Модульная система OSGi

OSGi модуль – **Bundle** (бандл):

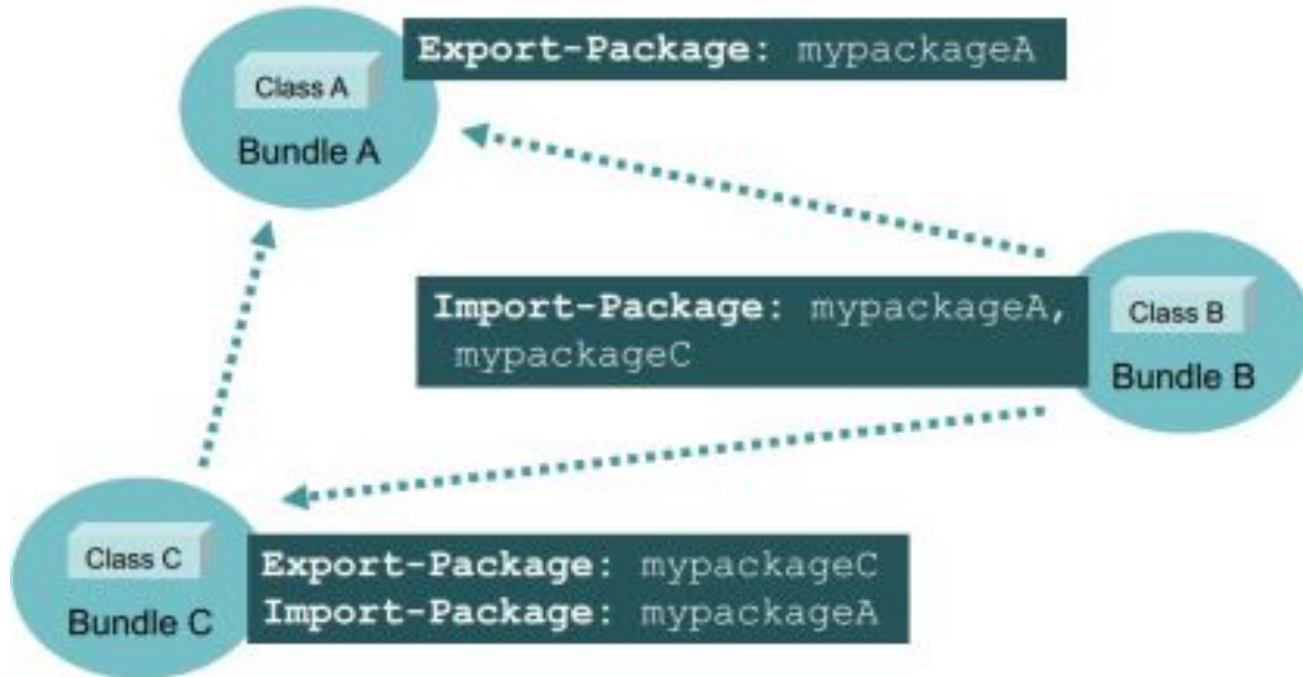
- Jar или директория
- Импорт/экспорт задается в манифесте бандла META-INF/MANIFEST.MF. Пример:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-SymbolicName: B
Bundle-Version: 1
Bundle-Name: B Bundle
Export-Package: org.bar
Import-Package: org.foo;version="[1,2)"
```


Модульная система OSGi

- OSGi бандл импортирует и экспортирует
 - пакеты (`Import-Package/Export-Package`)
 - сервисы (`Import-Service/Export-Service`).
- Может импортировать другие бандлы напрямую (`Require-Bundle`)
 - но это не приветствуется (менее гибко).

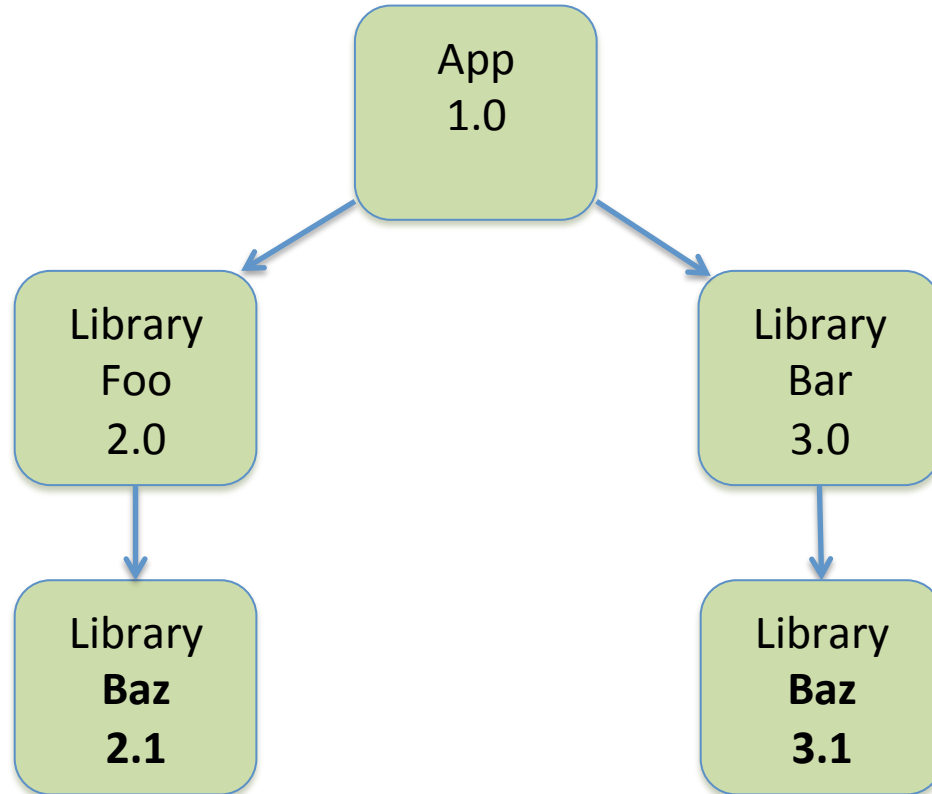
Модульная система OSGi



OSGi Runtime

- Разрешает Import/Export OSGi бандлов (wiring)
- Определяет жизненный цикл бандлов
 - может стартовать (активировать) бандлы лениво
 - позволяет обновлять бандлы без перезапуска всей системы (aka hot redeploy).

Jar/Classpath Hell



Версионирование в OSGi

- OSGi решает JAR hell:
 - импорт/экспорт квалифицируется версией
 - если два бандла требуют библиотеки разных версий, загрузятся обе версии библиотеки

Почему OSGi?

Итак, OSGi обещает:

Почему OSGi?

Итак, OSGi обещает:

- Модульность
 - явные зависимости
 - инкапсуляция

Почему OSGi?

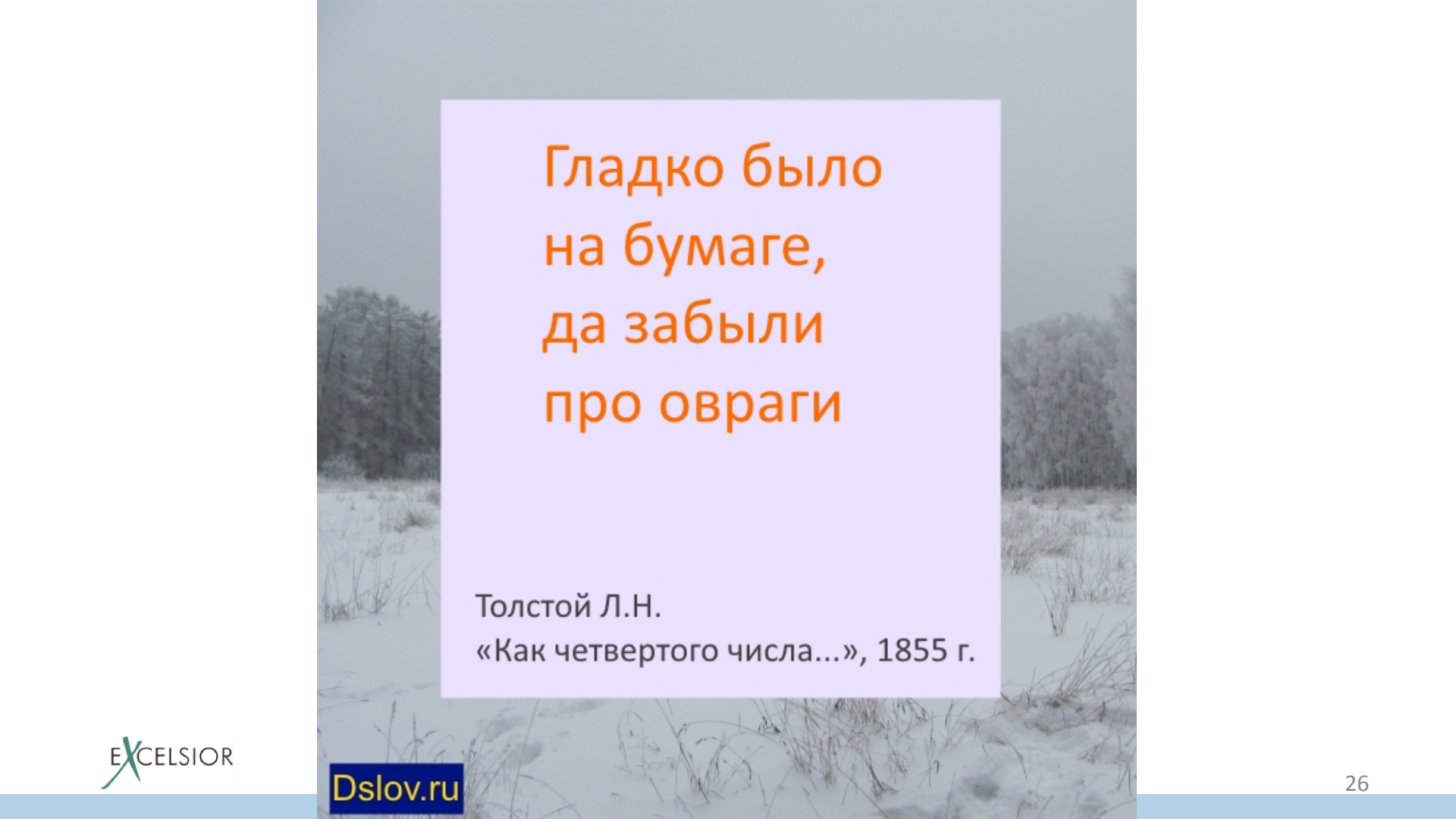
Итак, OSGi обещает:

- Модульность
 - явные зависимости
 - инкапсуляция
- Решение проблемы JAR Hell
 - через версионирование

Почему OSGi?

Итак, OSGi обещает:

- Модульность
 - явные зависимости
 - инкапсуляция
- Решение проблемы JAR Hell
 - через версионирование
- Hot ReDeploy (обновления на лету)
 - через возможность обновлять отдельный бандл динамически



Гладко было
на бумаге,
да забыли
про овраги

Толстой Л.Н.
«Как четвертого числа...», 1855 г.

Как OSGi?



Версионирование в OSGi

Вопрос: Как реализовать версионирование?

Версионирование в OSGi

Вопрос: Как реализовать версионирование?

Задача: Имеем модуль **A**, использующий библиотеку **Lib (v1)**, и модуль **B**, использующий **Lib (v2)**. **Требуется**, чтобы обе версии **Lib** работали и не конфликтовали.

Версионирование в OSGi

Вопрос: Как реализовать версионирование?

Задача: Имеем модуль **A**, использующий библиотеку **Lib (v1)**, и модуль **B**, использующий **Lib (v2)**. **Требуется**, чтобы обе версии **Lib** работали и не конфликтовали.

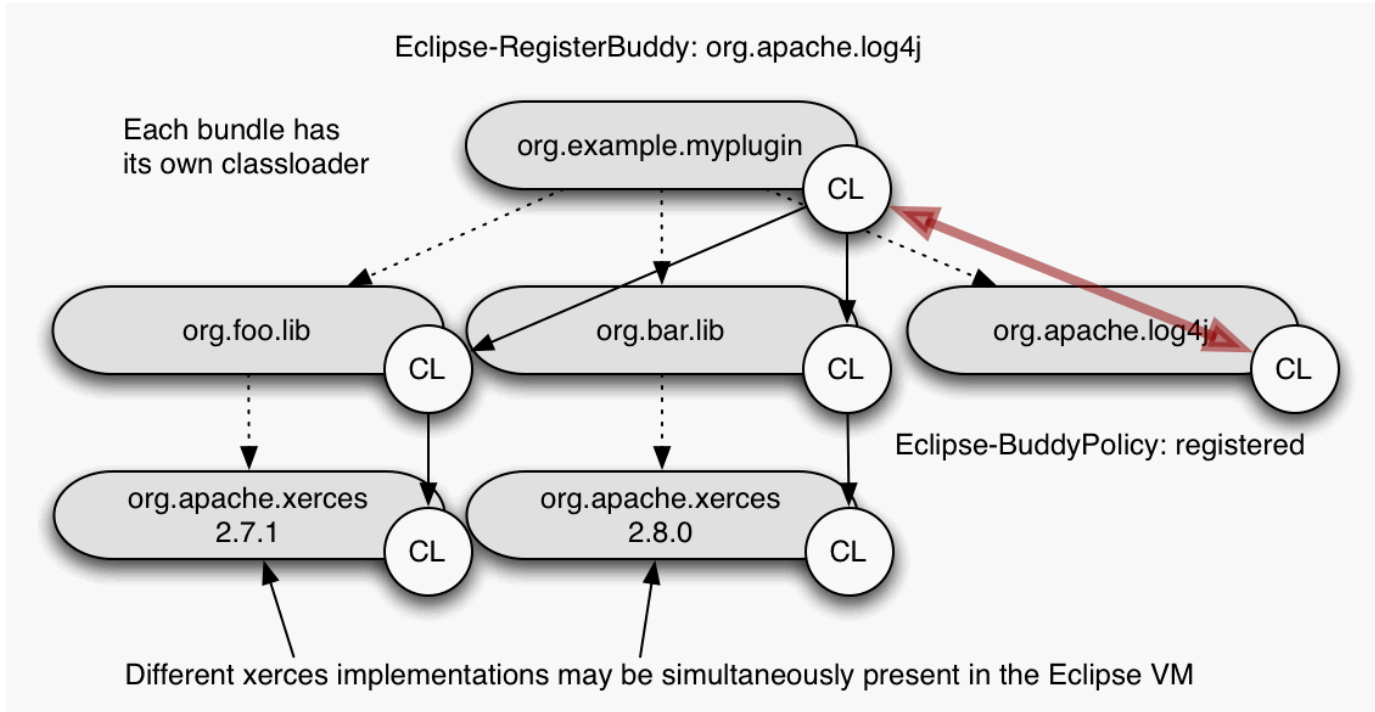
Решение: грузить обе версии библиотеки разными *загрузчиками классов*.

Версионирование в OSGi

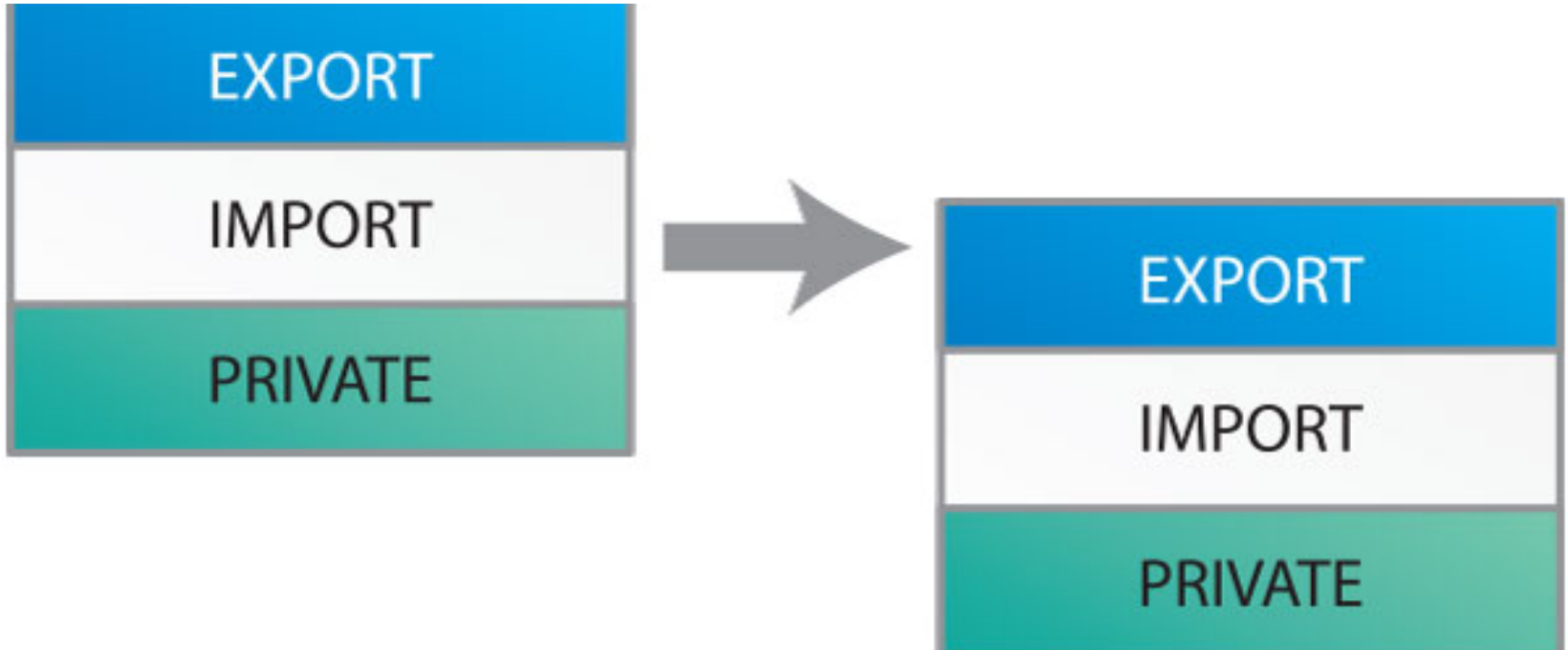
Таким образом, каждый OSGi бандл грузится своим загрузчиком классов:

- наследник `java.lang.ClassLoader`
- ***уникальное пространство имен*** классов
- нет конфликтов с классами других бандлов

Версионирование в OSGi



Инкапсуляция в OSGi



Инкапсуляция в OSGi

src/com/foo/exported/A.java:

```
package com.foo.exported;
```

```
import com.foo.internal.B;
```

```
public class A {  
    B useB;  
}
```

src/com/foo/internal/B.java:

```
package com.foo.internal;
```

```
public class B {  
  
}
```

Как сделать класс **B** *недоступным* из вне модуля?

Инкапсуляция в OSGi

Вопрос: как сделать *внутренний* класс модуля, объявленный публичным недоступным из вне?

Инкапсуляция в OSGi

Вопрос: как сделать *внутренний* класс модуля, объявленный публичным недоступным из вне?

Ответ: загрузчики классов (!) могут *прятать* внутренние классы.

Динамическое обновление

Задача: заменить один (!) изменившийся модуль в уже запущенной программе, не останавливая программу

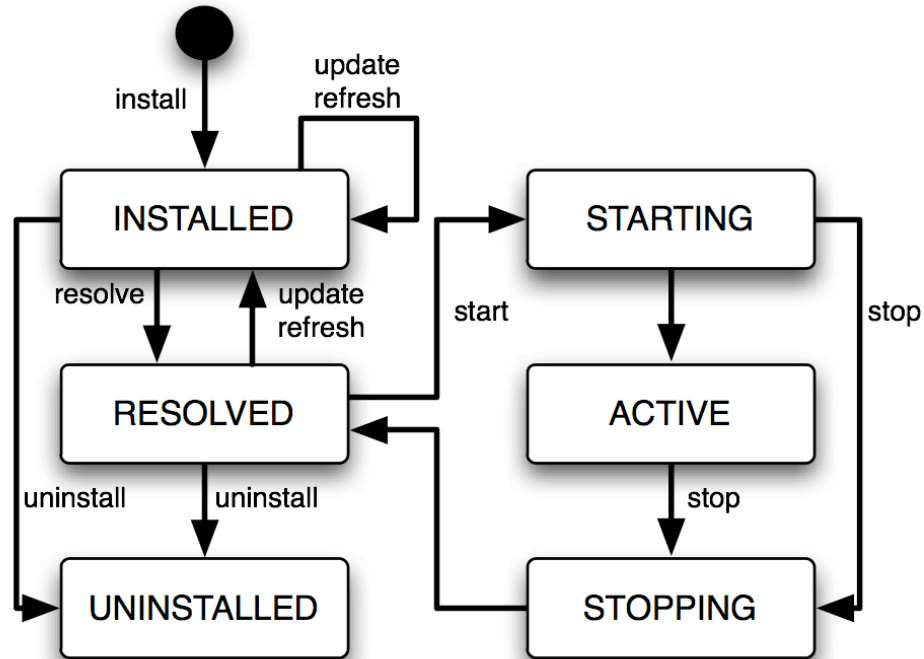
Динамическое обновление

Задача: заменить один (!) изменившийся модуль в уже запущенной программе, не останавливая программу

Решение: ЗАГРУЗЧИКИ КЛАССОВ!

(отгружаем старый модуль, грузим новый модуль новым загрузчиком классов)

Жизненный цикл бандла



Ленивый старт

Старт бандлов в OSGi сделан через концепцию активаторов бандлов:

- Каждый бандл может иметь активатор
 - директива манифеста `Bundle-Activator`
 - реализация интерфейса с методами `start()`, `stop()`
 - Аналог статического инициализатора для класса

Ленивый старт

Старт бандлов в OSGi сделан через концепцию активаторов бандлов:

```
public interface BundleActivator {  
    void start(BundleContext context) throws Exception;  
    void stop(BundleContext context) throws Exception;  
}
```

Ленивый старт

- Когда бандлы стартуют может задаваться конфигурацией OSGi окружения
- Если явно не задан момент старта бандла, то бандл стартует, когда потребуется другим запущенным бандлам

Ленивый старт

Задача: стартовать (грузить) модули только когда они потребуются в работе программы

Ленивый старт

Решение: ЗАГРУЗЧИК КЛАССОВ (опять!) ...

Ленивый старт

Решение: ЗАГРУЗЧИК КЛАССОВ (опять!) бандла реализован так, что перед тем как загрузить первый класс бандла, он вызывает метод `start()` активатора бандла

Ленивый старт

Решение: ЗАГРУЗЧИК КЛАССОВ (опять!) бандла реализован так, что перед тем как загрузить первый класс бандла, он вызывает метод `start()` активатора бандла

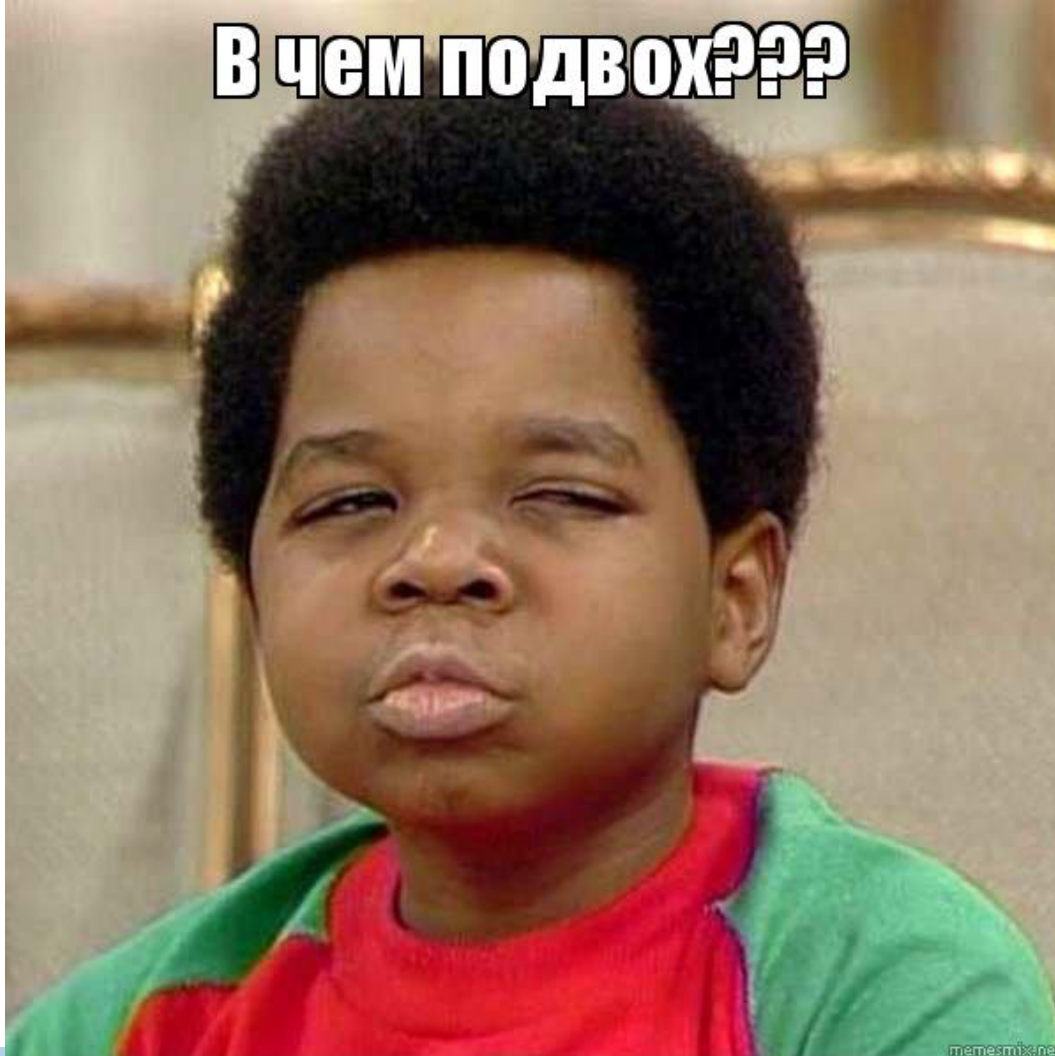
Так как загрузка классов в JVM – ленивая, то и активация бандлов становится ленивой
АВТОМАТОМ

Как OSGi?



Семь бед – один ответ!

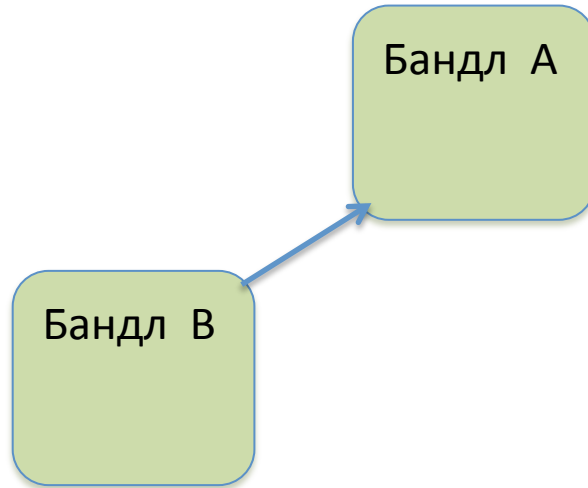
В чем подвох???



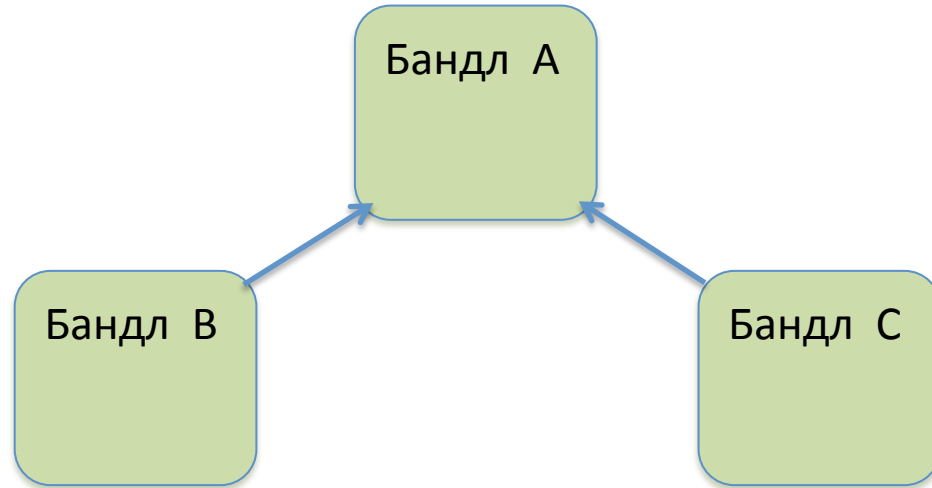
Почему НЕ OSGi?



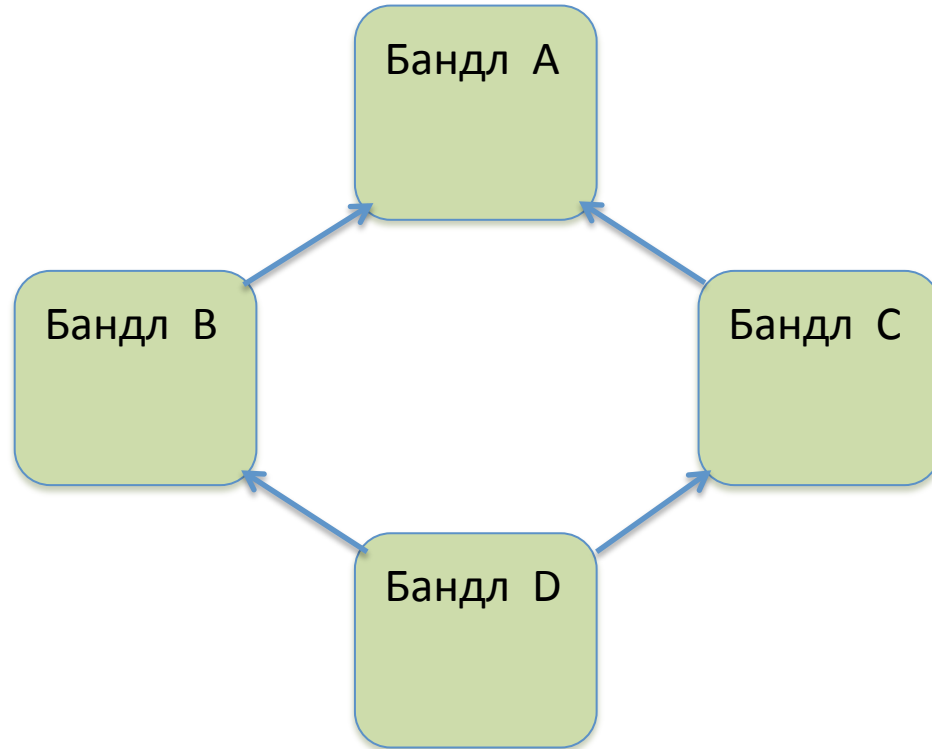
Модульность?



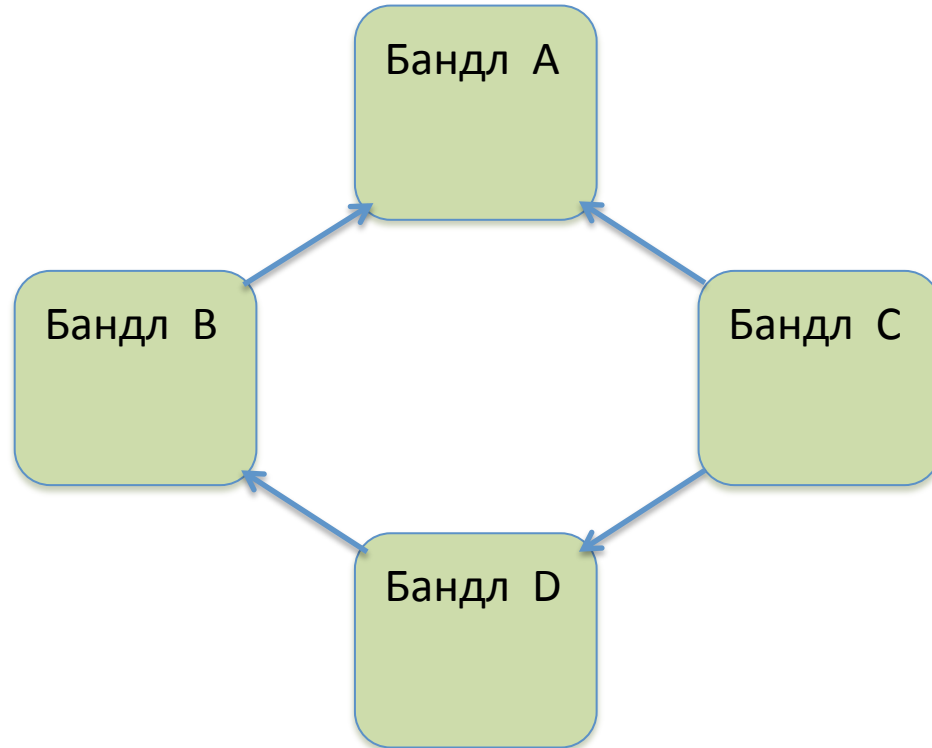
Модульность?



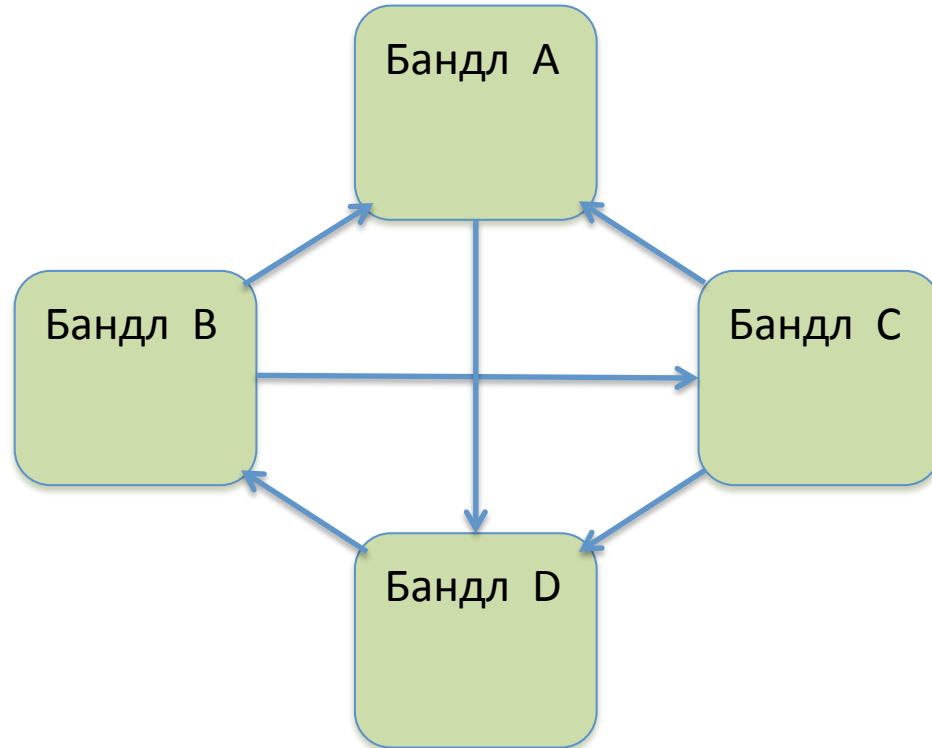
Модульность?



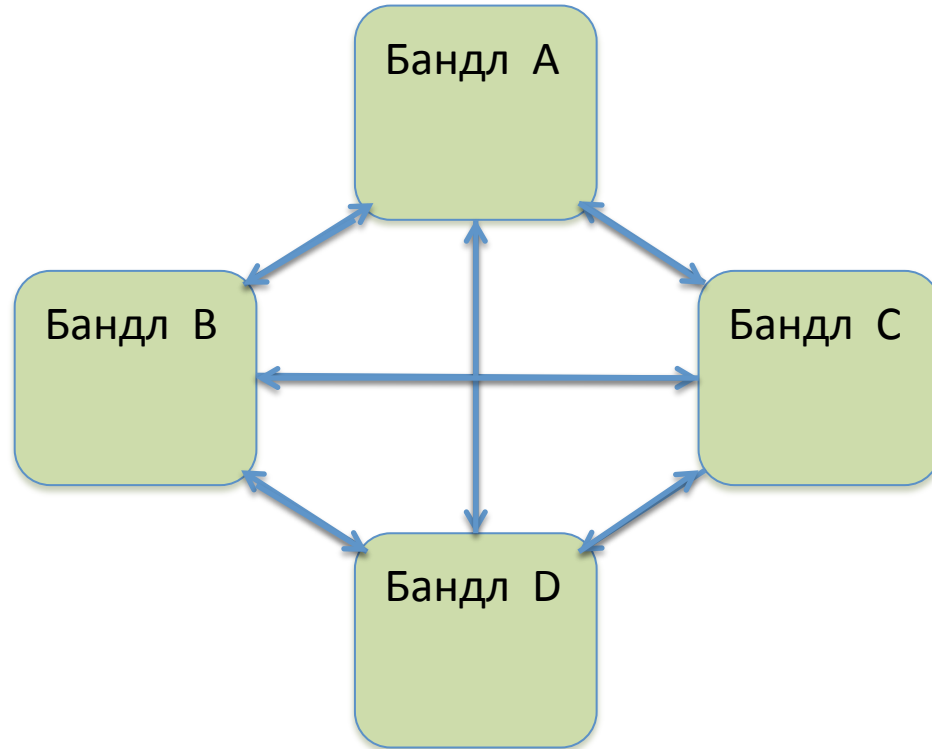
Модульность?



Модульность?



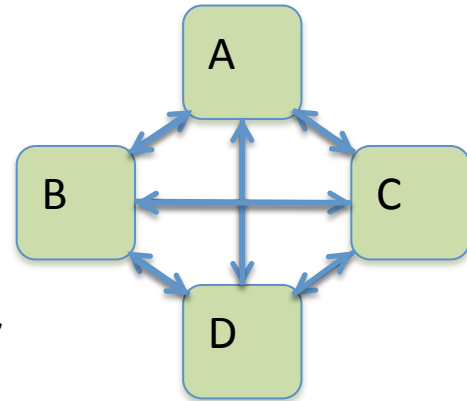
Модульность?



Модульность?

OSGI позволяет циклы в графе зависимостей

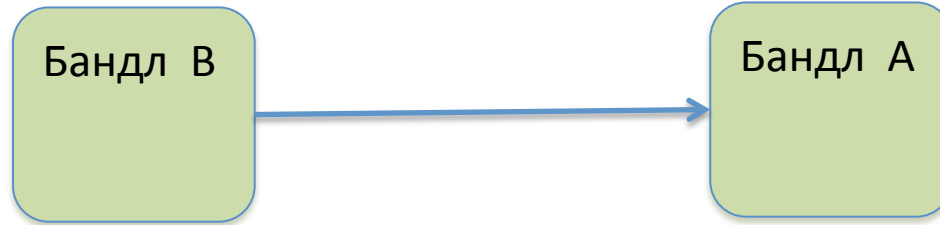
Упражнение: понять, почему
ЭТО ПЛОХО



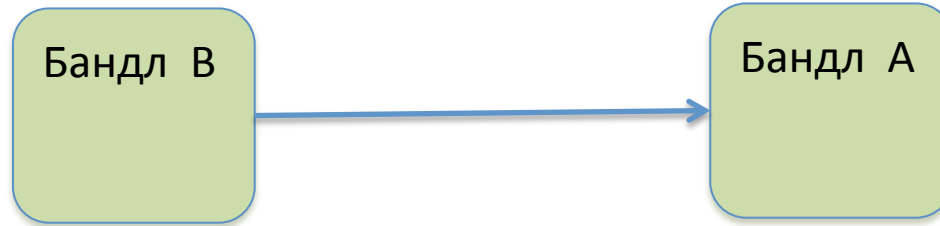
Обновления на лету?



Обновления на лету?

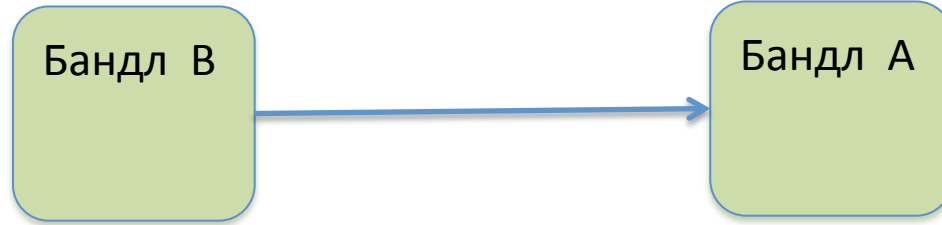


Обновления на лету?



Обновляем бандл В

Обновления на лету?



Обновления на лету?

Бандл А

Бандл В

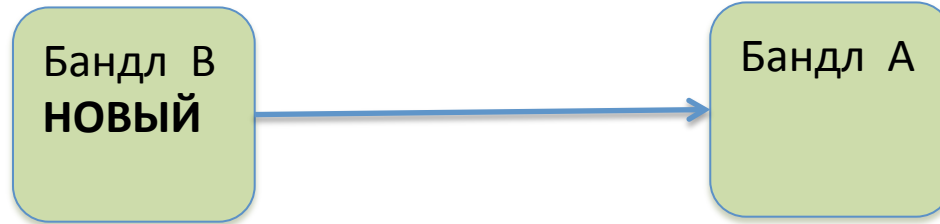


Обновления на лету?

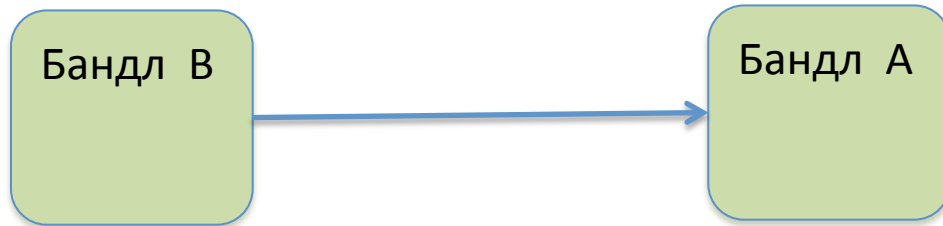
Бандл А

Бандл В
НОВЫЙ

Обновления на лету?



Обновления на лету?



Пройдет ли тот же
фокус с бандлом **А**?

Обновления на лету?



Обновления на лету?



Если бандл В импортирует бандл А, значит есть класс из В, ссылающийся на класс из А

Разрешение символьных ссылок

B.java:

```
class B {
```

```
  A useA;
```

```
  int f1 = A.f;
```

```
  int f2 = A.foo();
```

```
}
```

A.java:

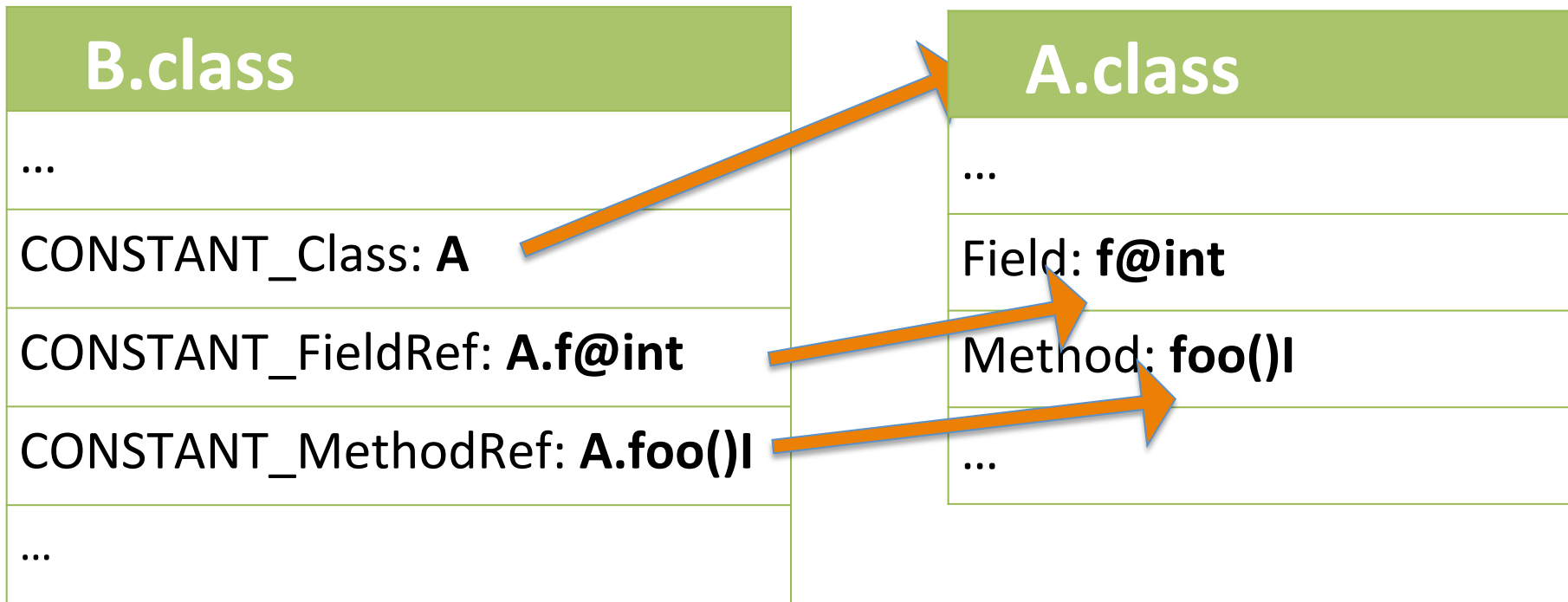
```
class A {
```

```
  static int f;
```

```
  static int foo(){};
```

```
}
```

Разрешение символьных ссылок



Разрешение символьных ссылок

- Класс ссылается на другие классы и поля, методы других классов символично

Разрешение символьных ссылок

- Класс ссылается на другие классы и поля, методы других классов символьно
- В рантайме символьные ссылки *разрешаются JVM* на реальные значения

Разрешение символьных ссылок

- Класс ссылается на другие классы и поля, методы других классов символично
- В рантайме символьные ссылки *разрешаются JVM* на реальные значения
- После первого разрешения ссылки, значение ссылки больше **не меняется!**

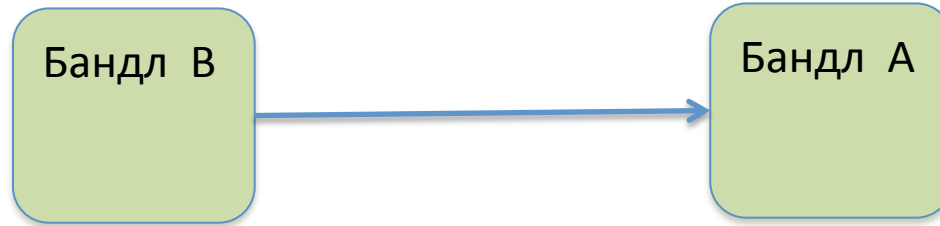
Обновления на лету?

- Если ссылка из класса В на класс А разрешилась, то класс А **нельзя выгрузить** из JVM без выгрузки класса В

Обновления на лету?

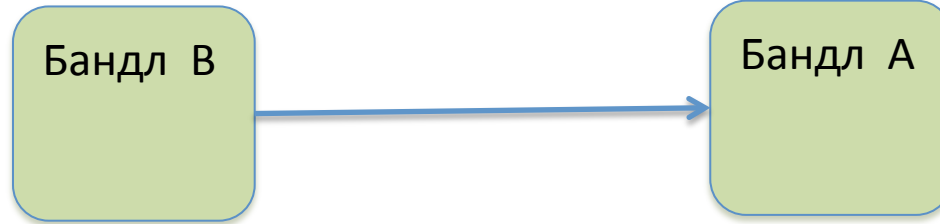
- Если ссылка из класса В на класс А разрешилась, то класс А **нельзя выгрузить** из JVM без выгрузки класса В
- Значит если бандл В импортирует бандл А, бандл А **нельзя выгрузить** без выгрузки бандла В

Обновления на лету?

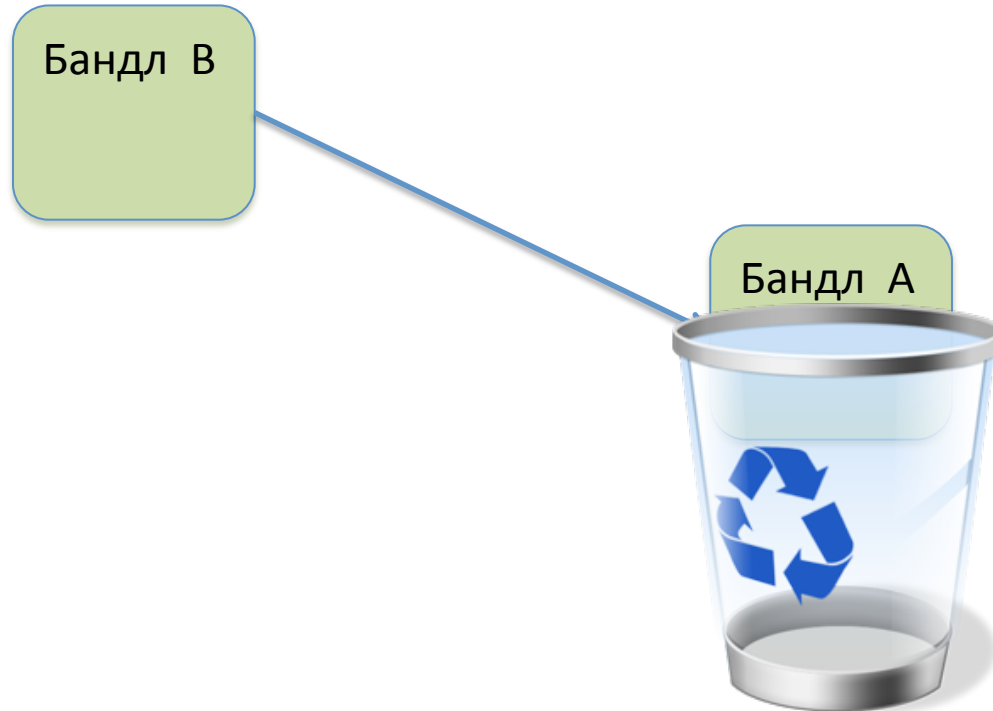


Обновляем бандл А

Обновления на лету?



Обновления на лету?



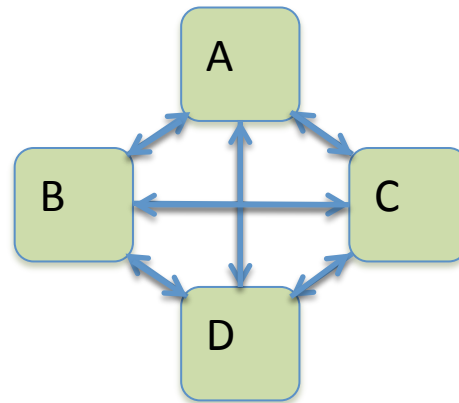
Обновления на лету?



Обновления на лету?

А теперь вспомним про
циклические зависимости

Упражнение: попробуйте
обновить SWT бандл у работающего Eclipse





Обновления на лету?

- Обновления на лету в OSGi более менее работает, только для листовых бандлов, которые никто не импортирует – *плагины*

Обновления на лету?

- Обновления на лету в OSGi более менее работает, только для листовых бандлов, которые никто не импортирует – *плагины*
- Для реализации плагинов есть гораздо более простые способы, чем OSGi

Обновления на лету?

Даже листовые бандлы не так просто выгрузить из JVM:

- После их выгрузки, принадлежавшие им классы могут остаться жить в JVM
 - Известная проблема: Classloader Memory Leak

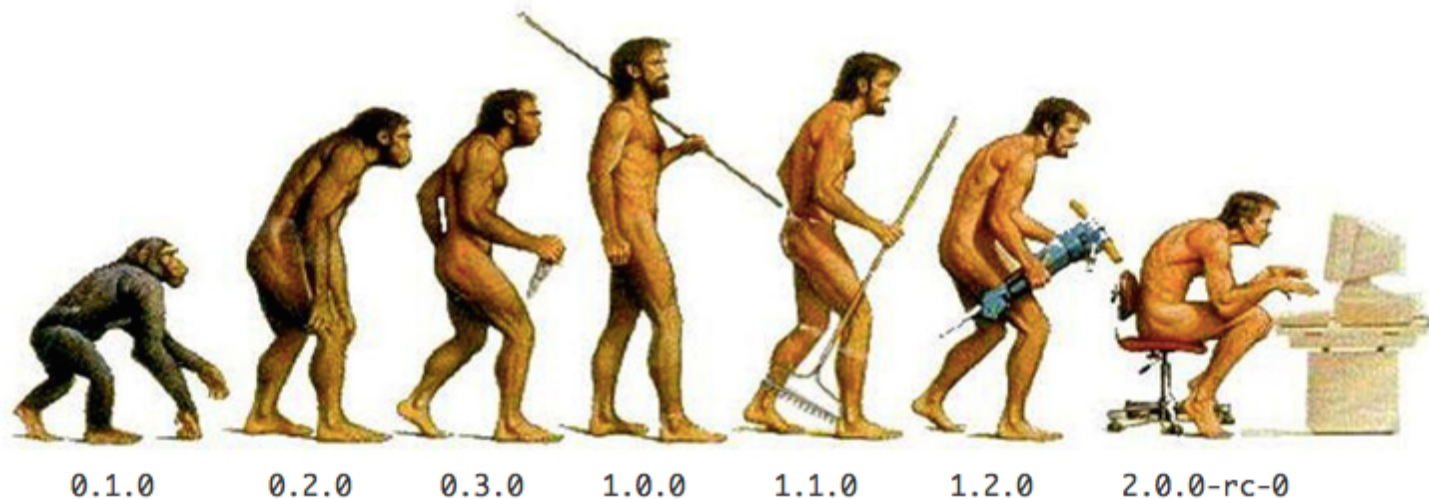
A close-up photograph of Donald Trump speaking at a podium. He is wearing a dark suit, a white shirt, and a red tie. He has a wide, toothy smile and is gesturing with his right hand, palm facing forward. A microphone is positioned in front of him. The background is a light-colored wall with decorative architectural elements.

Нельзя просто так взять

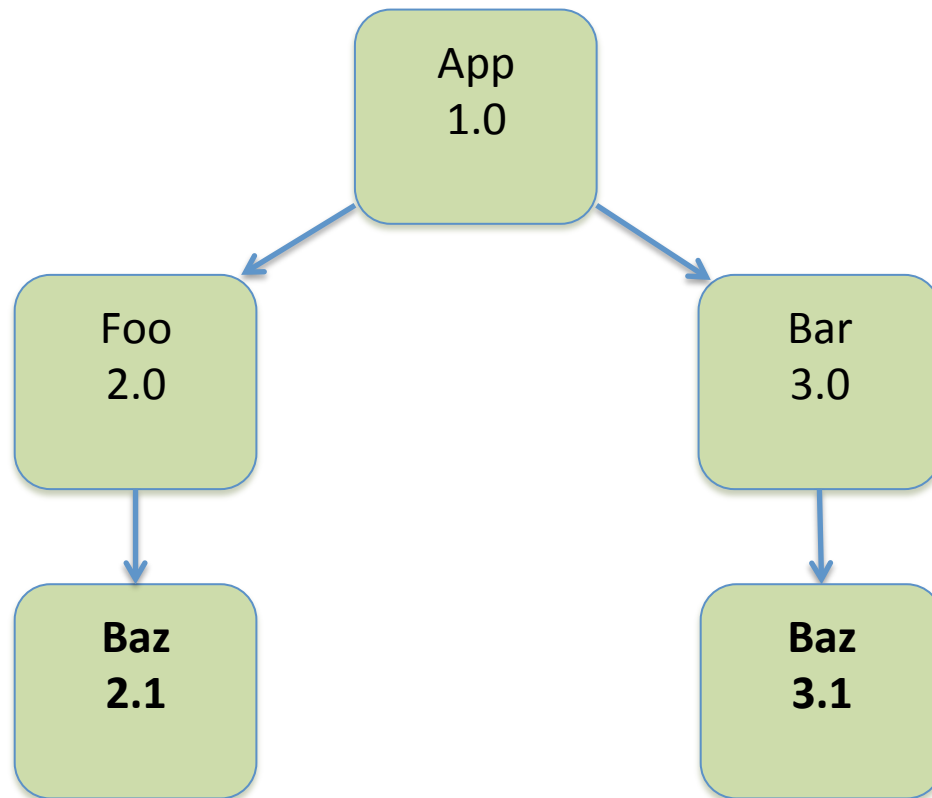
и выгрузить

класс из JVM

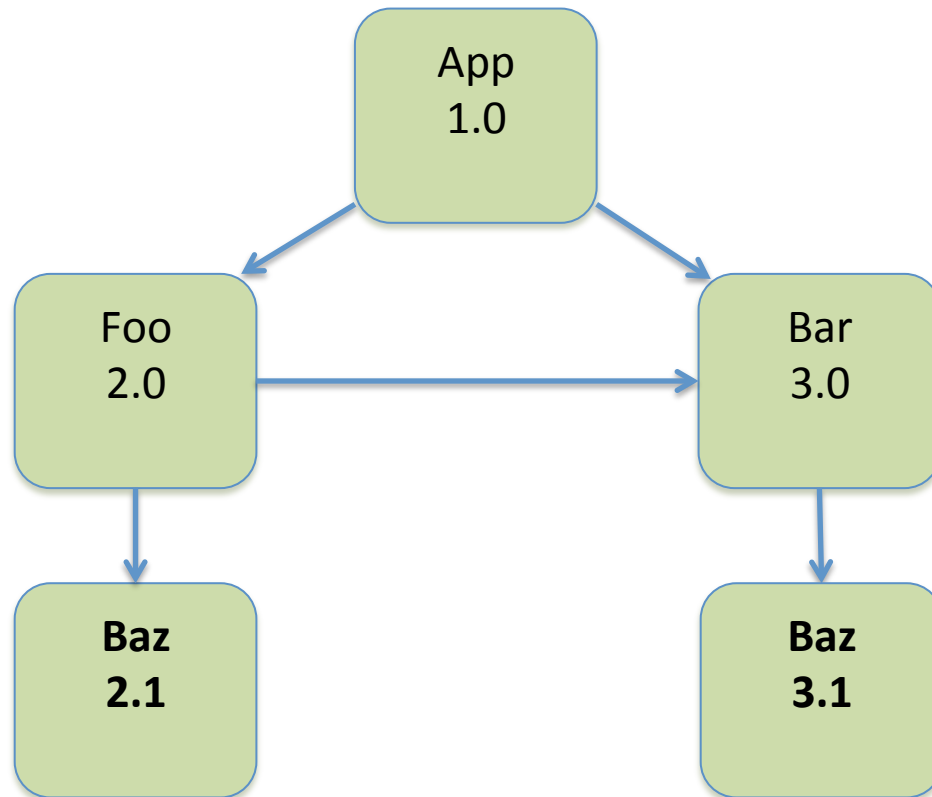
Версионирование?



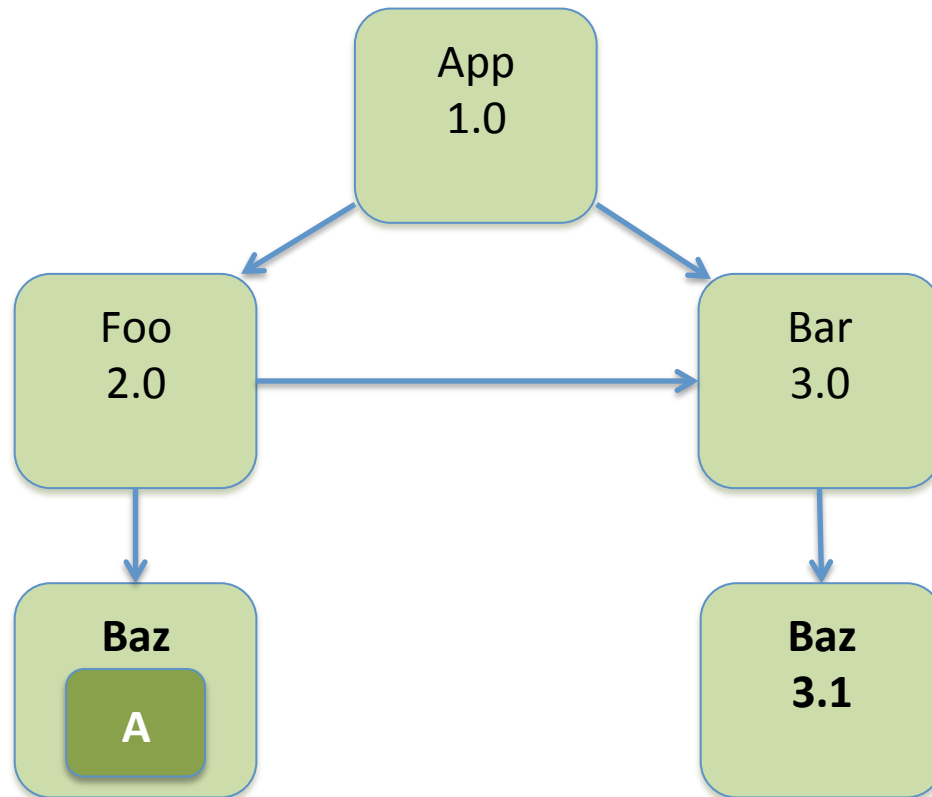
Версионирование?



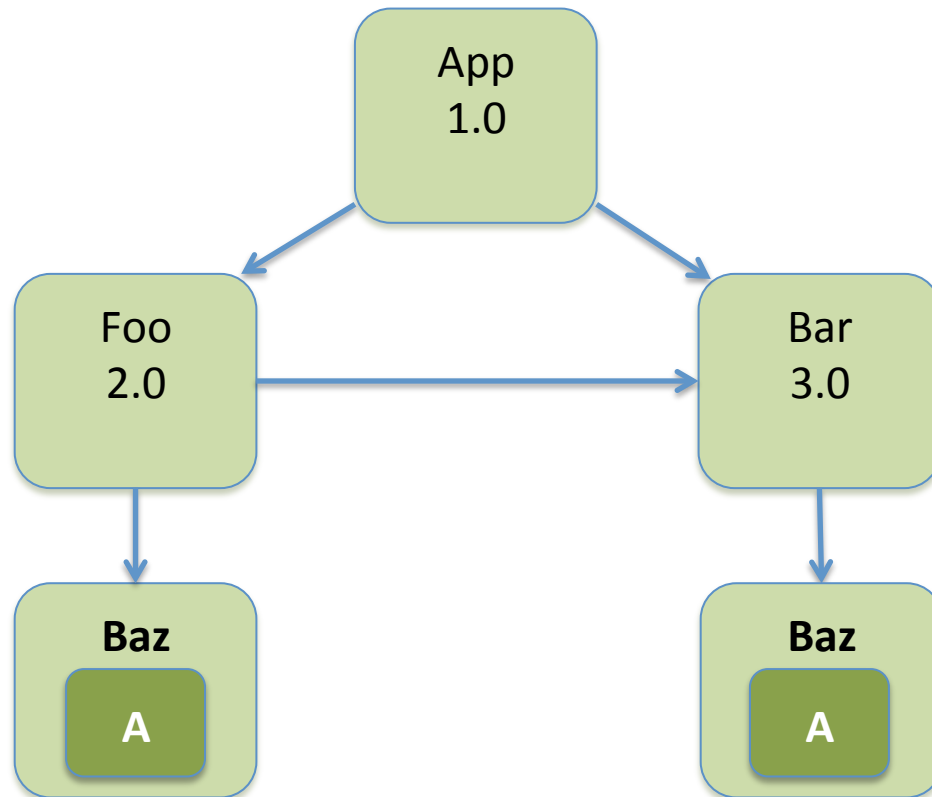
Версионирование?



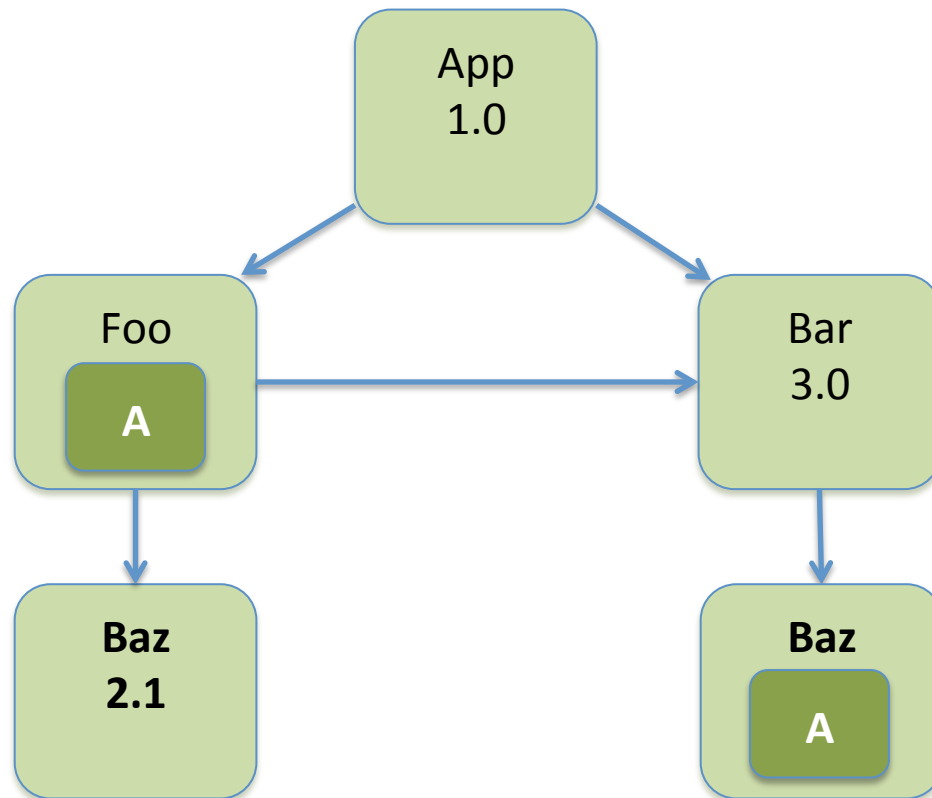
Версионирование?



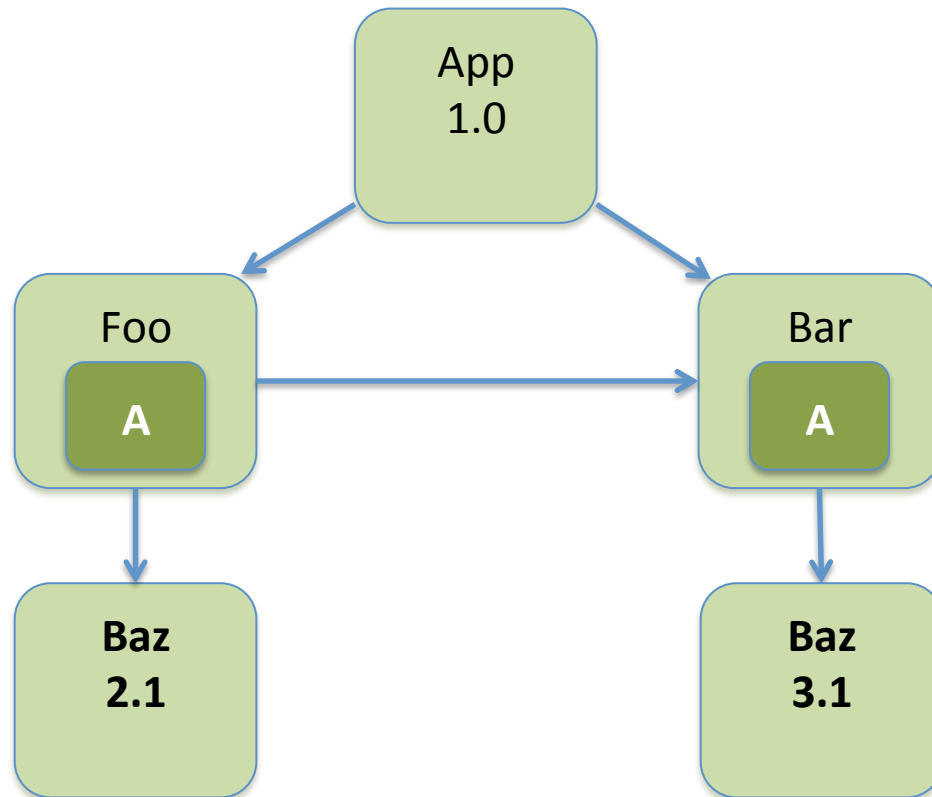
Версионирование?



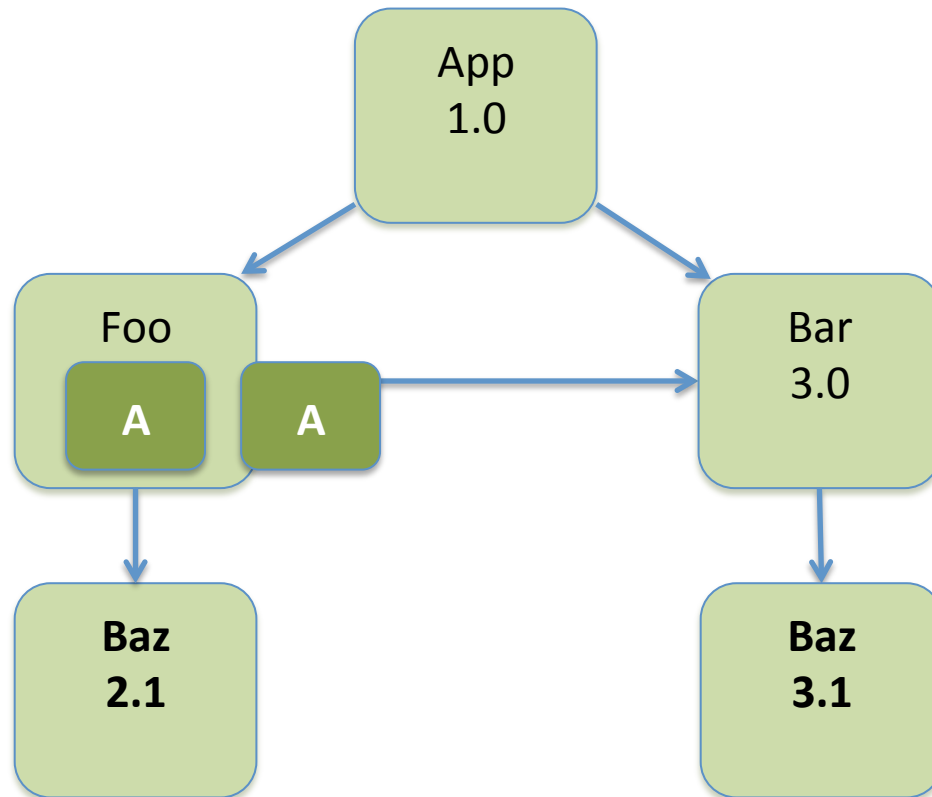
Версионирование?



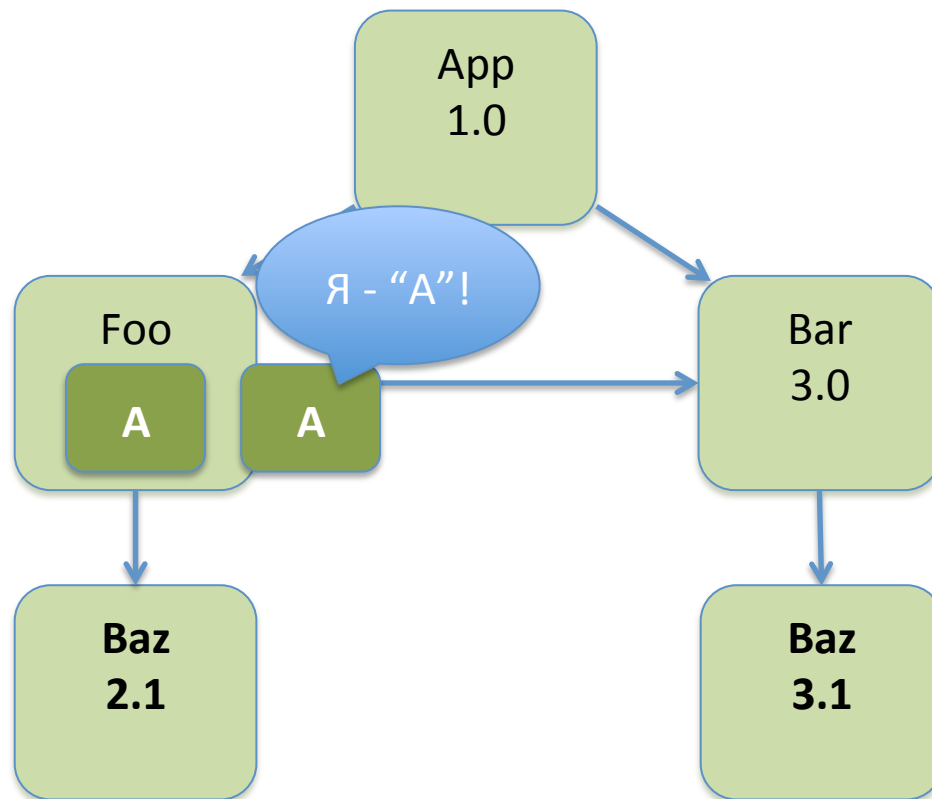
Версионирование?



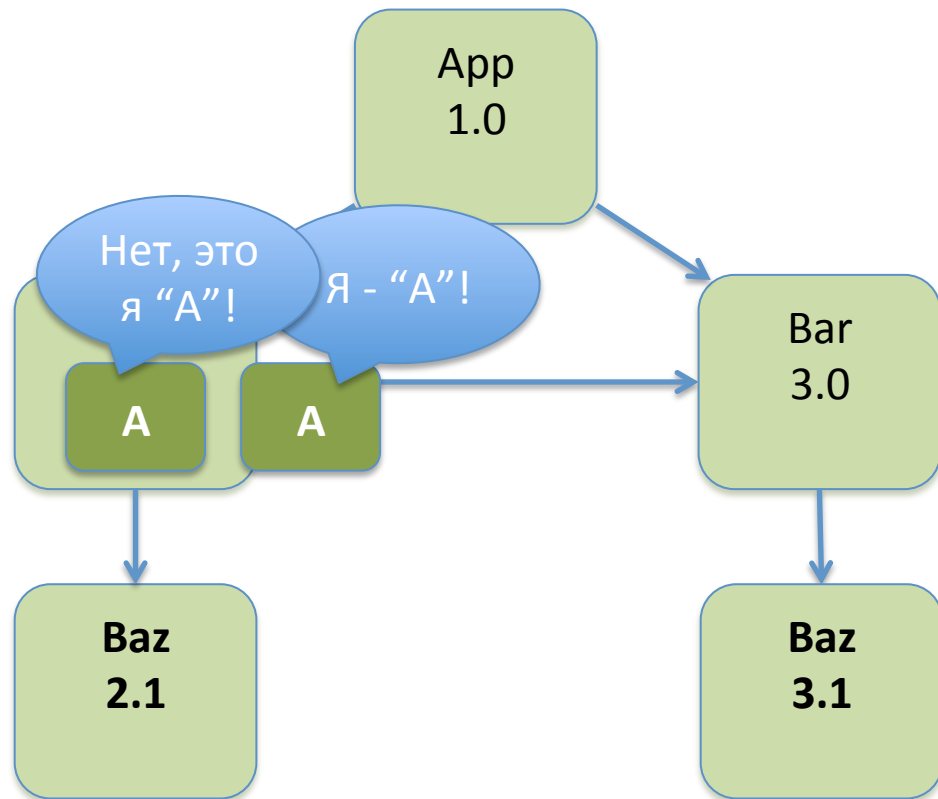
Версионирование?



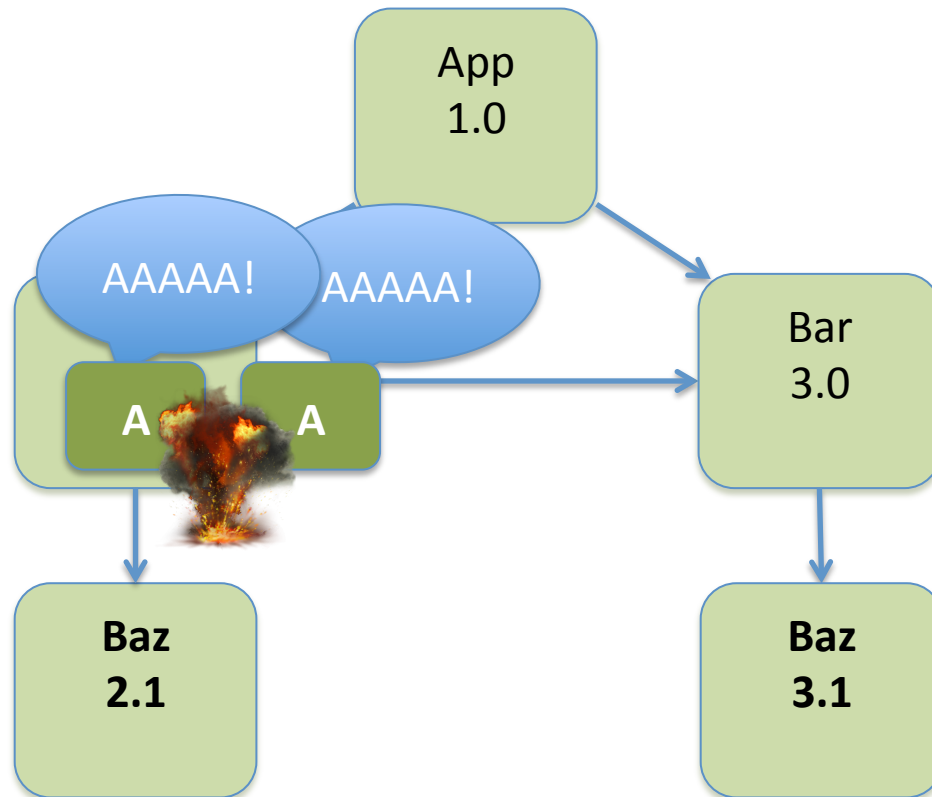
Версионирование?



Версионирование?



Версионирование?





Loading constraints

- Loading constraints запрещают двум разным классам с одинаковыми именами (fully qualified) появиться в области видимости одного класса

Loading constraints

B.java:

```
class B {  
    T1 f1 = A.f;  
    int f2 = A.m(t2);  
}
```

A.java:

```
class A {  
    static T1 f;  
    static int m(T2 t)
```

Если В грузится загрузчиком L1, а А грузится L2, то JVM проверит, что $(T1, L1) = (T1, L2)$ и $(T2, L1) = (T2, L2)$

Версионирование?

- При нарушении loading constraints, JVM кидает `java.lang.LinkageError`

Версионирование?

- При нарушении loading constraints, JVM кидает `java.lang.LinkageError`
- OSGi **НИКАК** не помогает разработчикам избежать нарушения loading constraints
 - Погуглите по словам OSGI и LinkageError, чтобы оценить масштаб проблемы

Версионирование?

Даже в поставке последних версий Eclipse есть бандлы, в которых есть потенциальные нарушения loading constraints и это никого не волнует!

Версионирование?

Вывод: OSGi не решает проблему JAR Hell, а переводит ее на новый изощренный уровень.



Версионирование?

Вывод: Не используйте в одном приложении одну и ту же библиотеку разных версий!



Инкапсуляция?

Вопрос: Ну хорошо, проблему инкапсуляции хотя бы OSGi решает?

Инкапсуляция?

Reflection – универсальная отмычка

`setAccessible(true)`

A large collection of various keys, representing a universal key for reflection. The keys are of different shapes and sizes, some with notches and others with smooth heads, all pointing towards the center. The background is a light greenish-yellow.

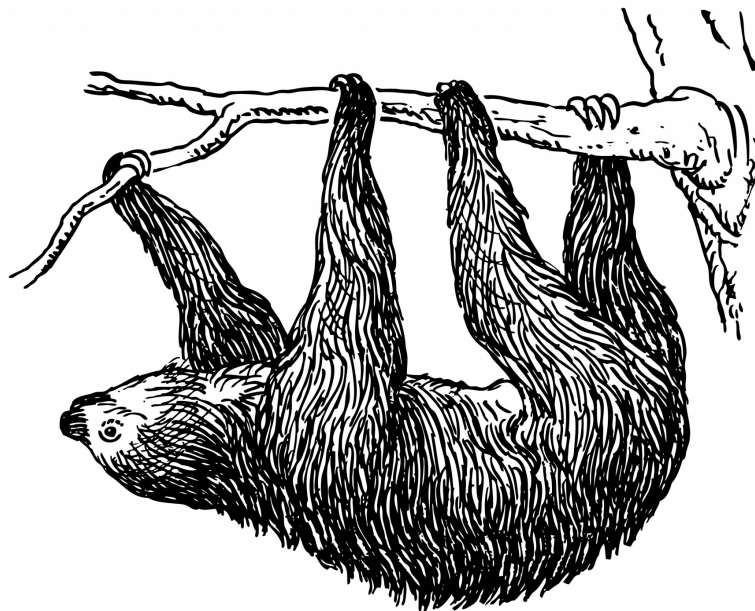
против инкапсуляции в Java

Инкапсуляция?

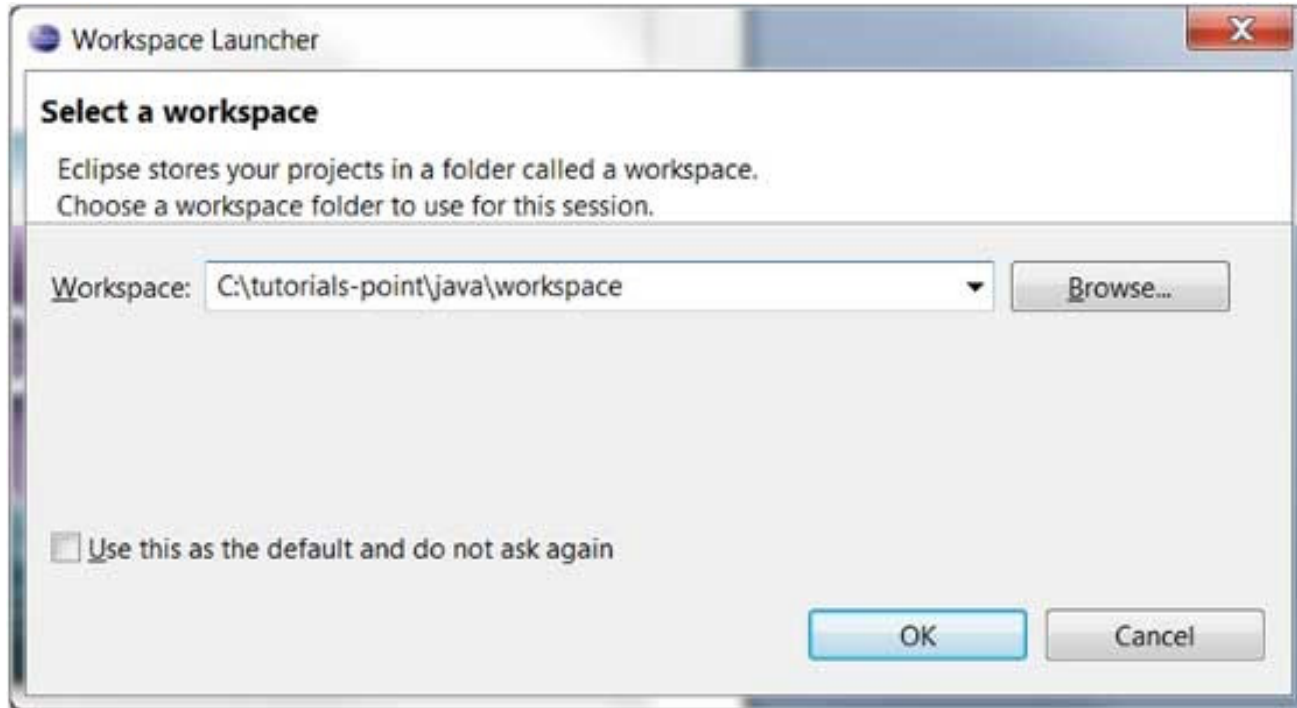
Вопрос: Ну хорошо, проблему инкапсуляции хотя бы OSGi решает?

Ответ: OSGi не защищает от несанкционированного доступа через Reflection

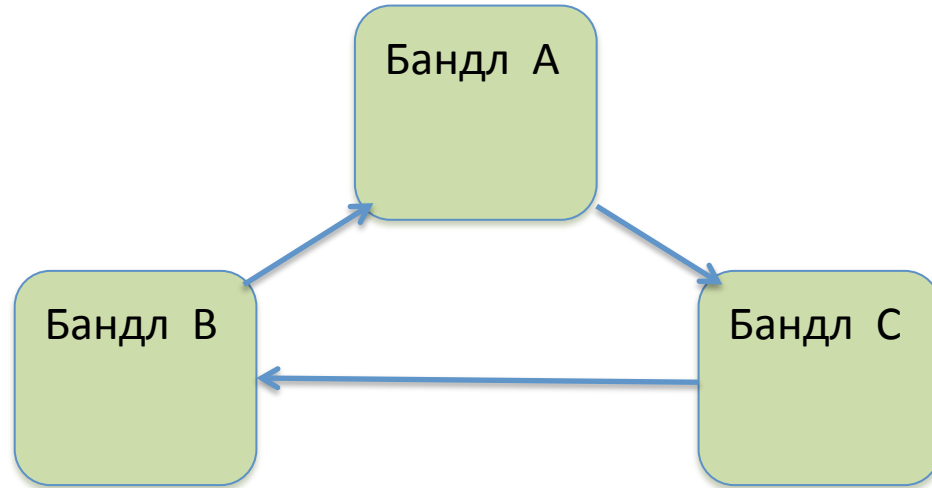
Ленивый старт?



Ленивый старт?



Ленивый старт?



Вопрос: В каком порядке активируются эти бандлы?

Ленивый старт?

Порядок активации бандлов в OSGi напрямую зависит от порядка загрузки классов в JVM

- бандл стартуют из `loadClass()` загрузчика классов бандла

Ленивый старт?

Порядок активации бандлов в OSGi напрямую зависит от порядка загрузки классов в JVM

- бандл стартуют из `loadClass()` загрузчика классов бандла

Но порядок загрузки классов в JVM не определен!

Разрешение символьных ссылок

Класс может иметь ссылки на другие классы и поля, методы других классов. JVM может разрешать ссылки:

- Лениво
 - ссылки разрешаются при первом доступе
- Энергично
 - разрешаются все ссылки какие возможно

Ленивый старт?

Порядок загрузки классов в JVM зависит от схемы разрешения ссылок между классами: ленивой, не всегда ленивой, энергичной.

Ленивый старт?

Активатор бандла B:

```
class B implements BundleActivator {  
    public void start() {  
        assert A.f != null;  
    }  
}
```

Активатор бандла A:

```
class A implements BundleActivator {  
    static T f;  
    public void start() {  
        f = new T();  
    }  
}
```

Стандартная ситуация: В думает, что A уже активирован, но по факту A может активироваться позже B

Ленивый старт?

- Схема активации бандлов в OSGi – это мина замедленного действия:
 - стоит JVM начать разрешать ссылки между классами менее лениво, практически все OSGi приложения перестанут работать

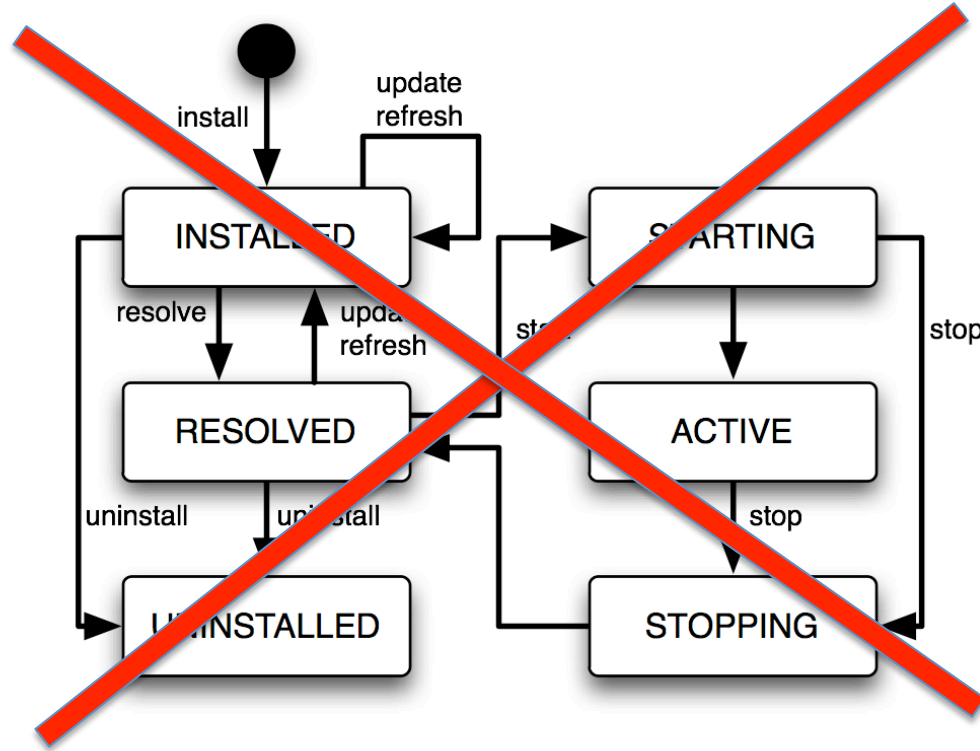
Почему не OSGi?

- Модульность с циклами
- Hot Redeploy работает только для листьев
- Нет защиты от нарушения loading constraints
- Нет защиты деталей реализации от доступа через Reflection
- Активация бандлов зависит от схемы разрешения ссылок в JVM

Jigsaw



Jigsaw



Jigsaw vs. OSGi

OSGi – динамичен по своей сути

– модули появляются только в ран-тайме

Jigsaw vs. OSGi

Jigsaw сразу задумывался статичным.

Практически весь JDK tooling знает про модули:

- javac: уважает области видимости в модулях
- jdeps: анализирует зависимости
- jar, jmod: пакует модули
- jlink: изготавливает финальный образ для развертывания
- java: в рантайме модули конечно тоже есть

Пример модуля

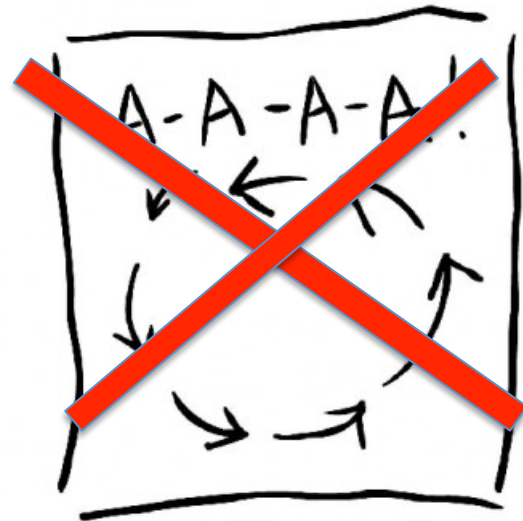
```
// src/java.sql/module-info.java  
module java.sql {  
    requires transitive java.logging;  
    requires transitive java.xml;  
  
    exports java.sql;  
    exports javax.sql;  
    exports javax.transaction.xa;  
    uses java.sql.Driver;  
}
```


Jigsaw vs. OSGi

Jigsaw запрещает циклы
в графе зависимостей (явных)

Модули импортируют
модули, а не пакеты.

ПЛАН эвакуации



Версионирование

В первых дrafтах Jigsaw версионирование было (аналогичное OSGi).

Версионирование

В первых дrafтах Jigsaw версионирование было (аналогичное OSGi).

Но потом от него отказались ...

Версионирование

В первых дrafтах Jigsaw версионирование было (аналогичное OSGi).

Но потом от него отказались ...

Почему?

Версионирование

Версионирование – немедленное означает:

1 модуль \leftrightarrow 1 загрузчик классов

Версионирование

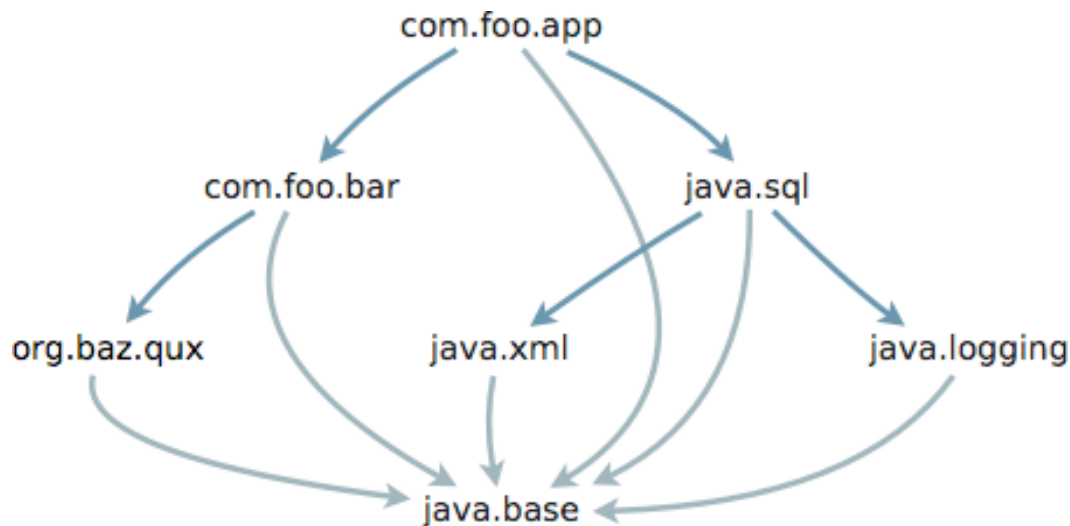
Версионирование – немедленное означает:

1 модуль \leftrightarrow 1 загрузчик классов

И в первых версиях Jigsaw так и было!

Jigsaw и загрузчики

Jigsaw – это не только модули, но и разбиение Java платформы на модули.



Jigsaw и загрузчики

Проблема обратной совместимости:

По спецификации

`getClassloader() == null`

для всех классов платформы.

Jigsaw и загрузчики

Проблема обратной совместимости:

По спецификации

`getClassloader() == null`

для всех классов платформы.

Что противоречит разбиению платформы на модули, где каждый модуль грузится своим загрузчиком

Jigsaw и загрузчики

Проблема 2: Как защитить разработчика от нарушения loading constraints?

Версионирование

Еще деталь: импорт в ранних версиях Jigsaw (как и в OSGi) специфицировался не просто версией, а **диапазоном версий**:

- Модуль мог декларировать, что может работать с зависимостями версий от и до

Версионирование

Проблема 3: Оказалось, что разрешение зависимостей (wiring модулей) от диапазонов версий – это **NP** полная проблема!

- Сводится к 3-SAT

Версионирование

... после чего версионирование в модульной системе Java приказало долго жить.

Версионирование

... после чего версионирование в модульной системе Java приказало долго жить.

Нет версионирования – не нужны загрузчики классов для модулей.

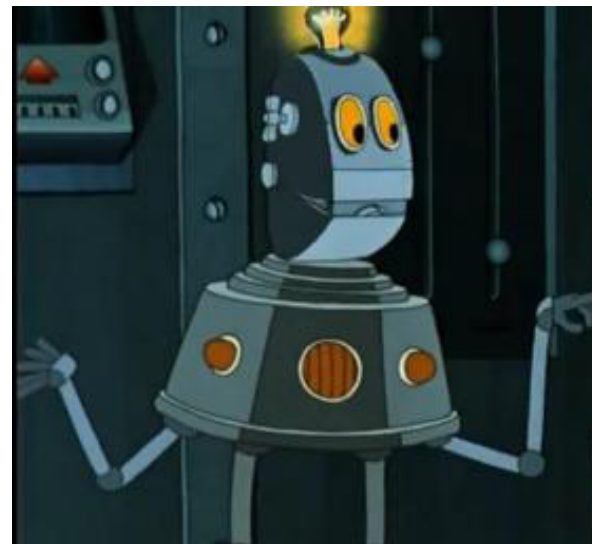
Jigsaw

- Динамических обновлений нет
- Версионирования нет

Jigsaw

- Динамических обновлений нет
- Версионирования нет

А что тогда есть?



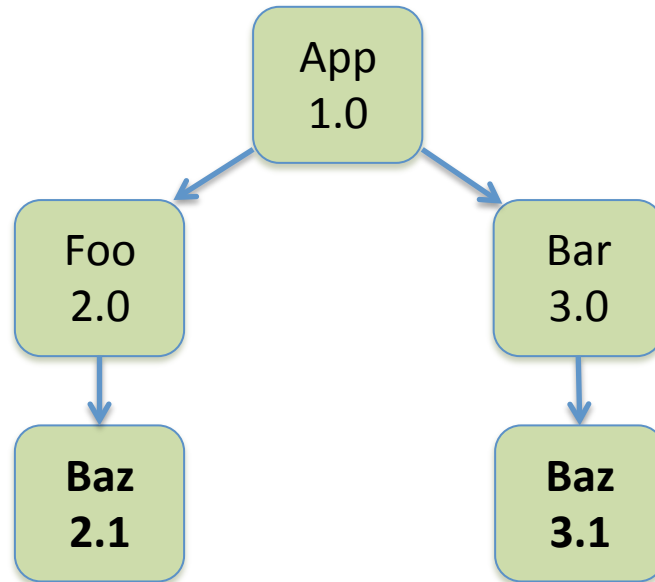
Jigsaw Mantra

**Reliable
Configuration**



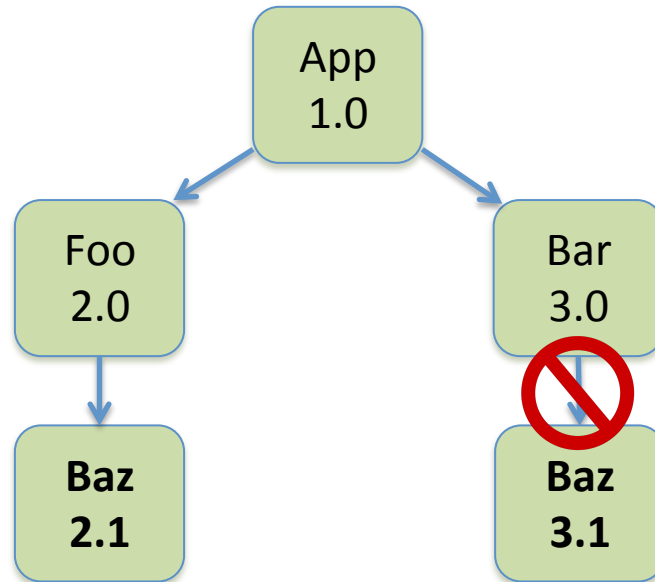
**Strong
Encapsulation**

Reliable Configuration



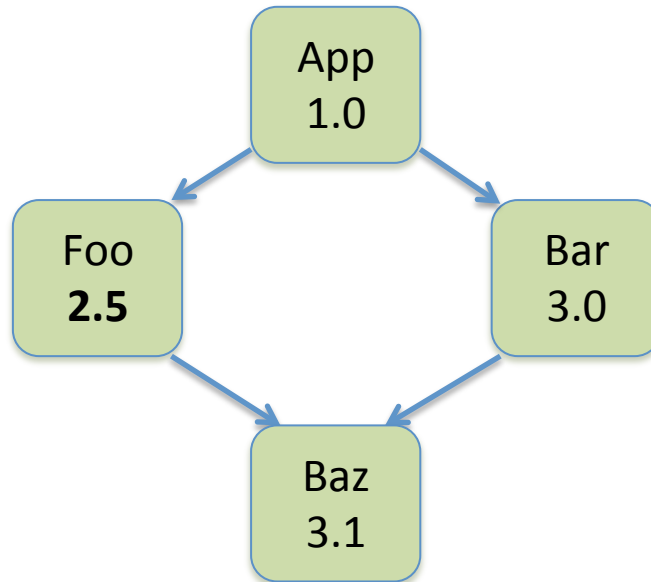
Reliable Configuration

Такая ситуация в Jigsaw просто запрещена!



Reliable Configuration

Правильная (**reliable**) конфигурация:



Reliable Configuration

Reliable configuration это:

- Все зависимости модулей удовлетворены
- Нет циклических зависимостей
- Нет модулей экспортирующих одинаковые пакеты (split packages)

Свойства проверяются на старте (wiring)

Strong Encapsulation



Strong Encapsulation

Модули в Java 9 – это first class citizens

Strong Encapsulation

Модули в Java 9 – это first class citizens

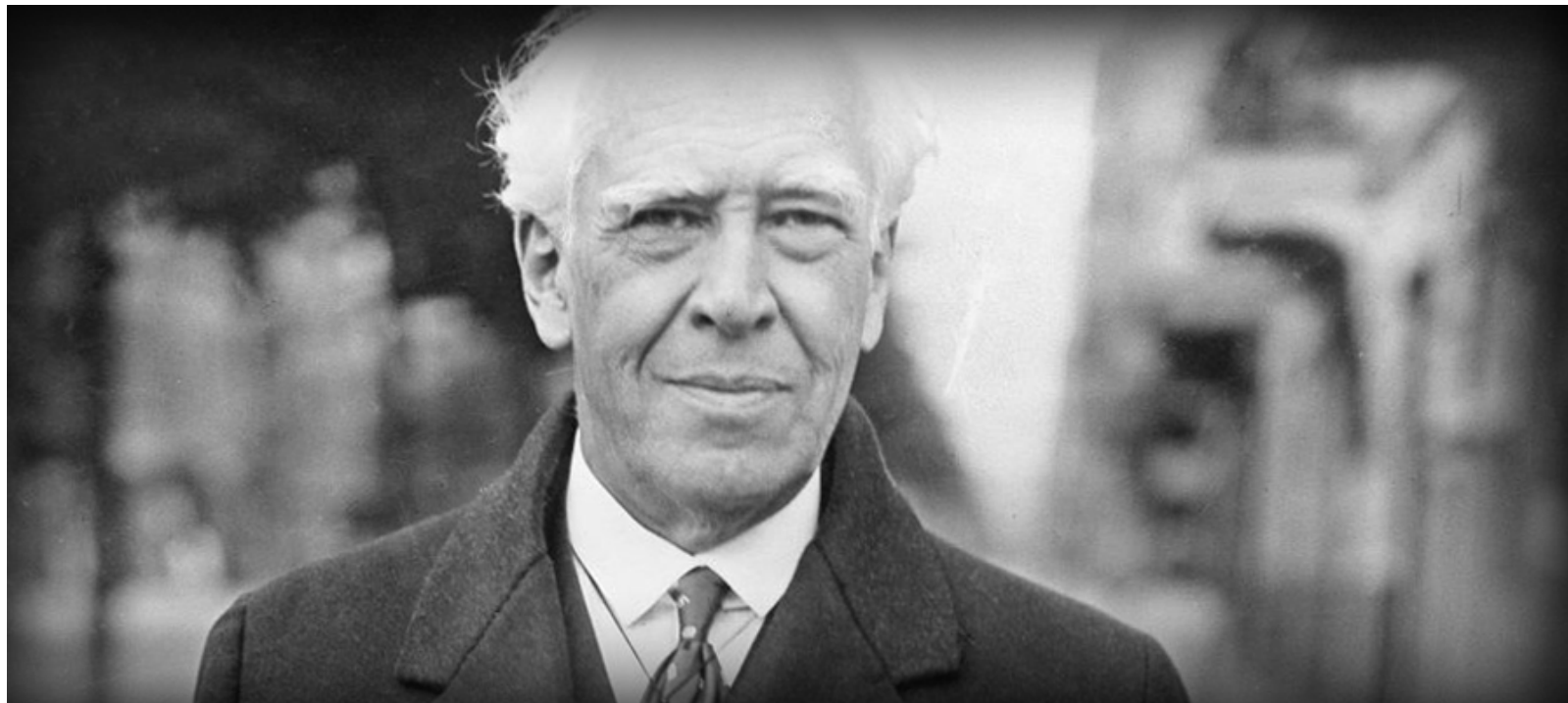
- Определяют области видимости
 - Через декларируемый экспорт

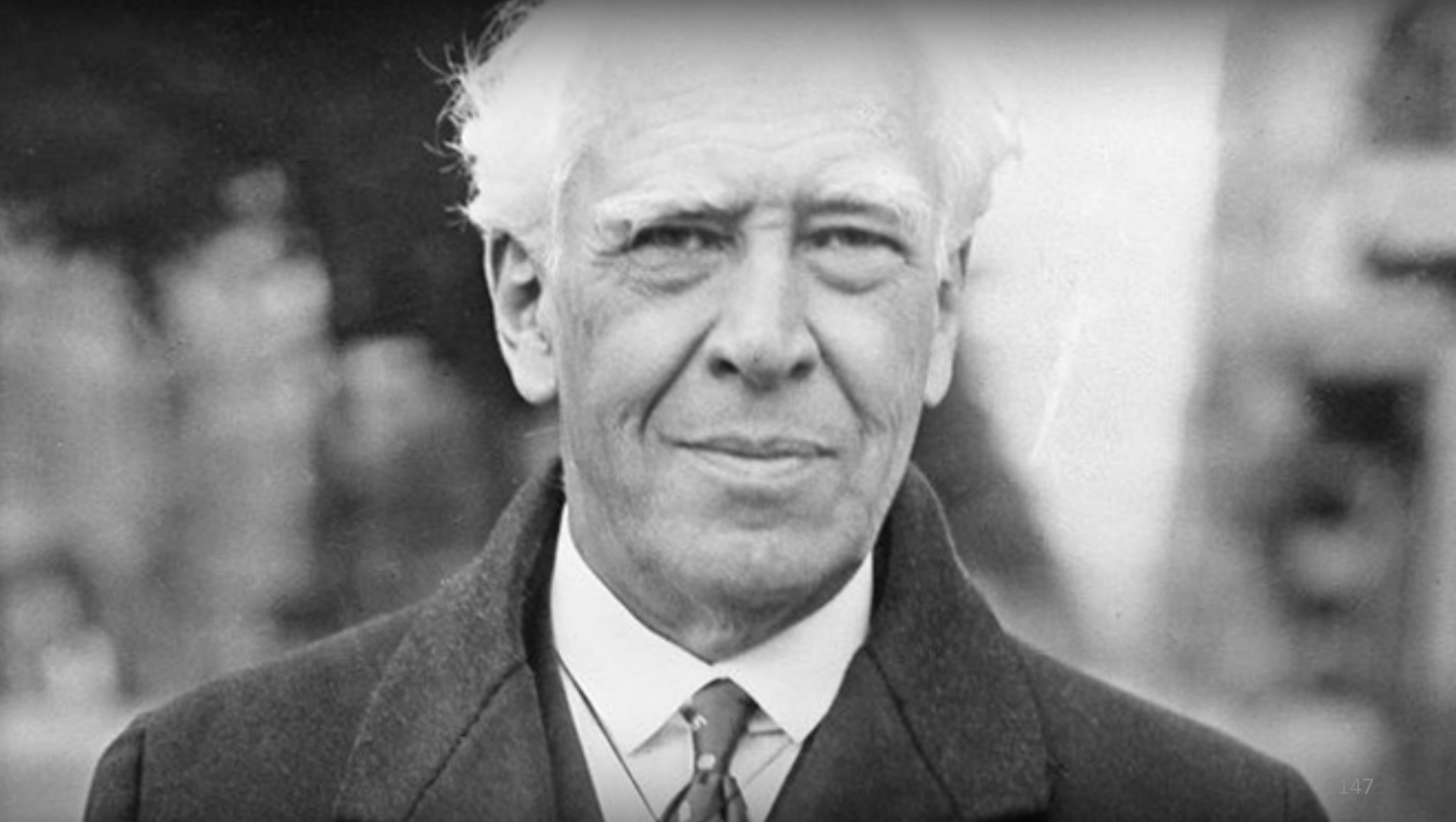
Strong Encapsulation

Модули в Java 9 – это first class citizens

- Определяют области видимости
 - Через декларируемый экспорт
- Доступа к не экспортируемой функциональности из вне модуля нет даже через reflection
 - Не работает даже **setAccessible(true)**

И что, у Jigsaw нет проблем?





Reliable Configuration?

- В ранних версиях Jigsaw рефлексивный доступ между модулями запрещался, если между ними нет явных зависимостей

Reliable Configuration?

- В ранних версиях Jigsaw рефлексивный доступ между модулями запрещался, если между ними нет явных зависимостей
- Но с уходом загрузчиков из модульной системы от этого пришлось отказаться
 - так как `Class.forName()` должен работать по старому

Reliable Configuration?

- Однако, если у вас возможны рефлексивные зависимости, не прописанные явно, где гарантия, что результирующая конфигурация **reliable**?

Reliable Configuration?

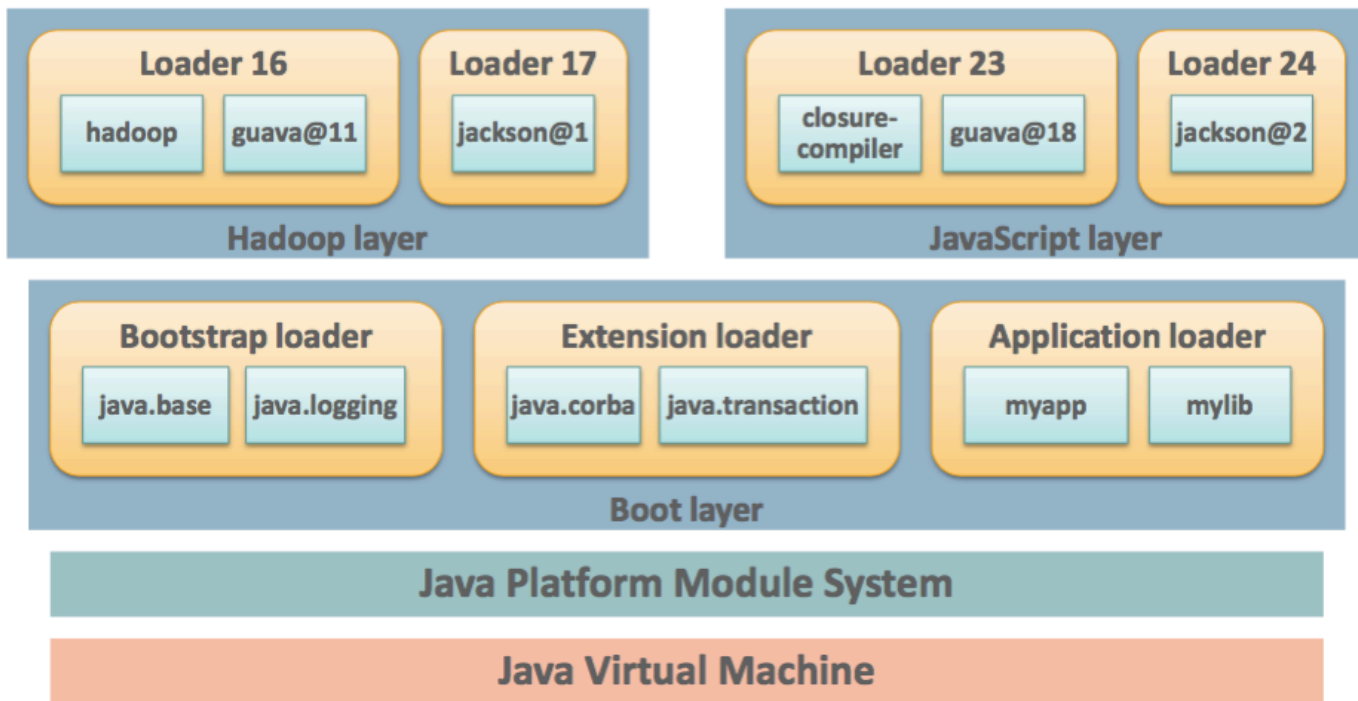
Split packages запрещены, но как же контейнеры приложений (Tomcat, Java EE)?

Jigsaw Layers

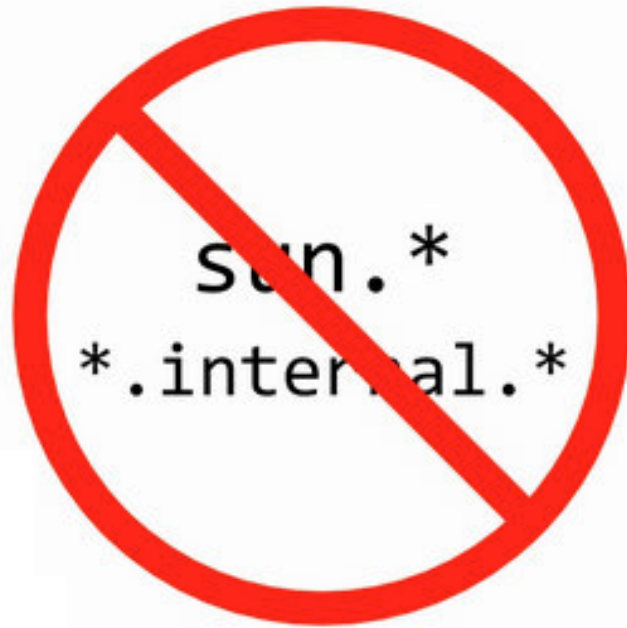
Для контейнеров приложений введен механизм слоев (*Layers*):

- Локальная модульная система для каждого приложения в контейнере
- Разные модули с одинаковыми пакетами должны принадлежать разным слоям

Jigsaw Layers *



Strong Encapsulation?



Strong Encapsulation?

Платформа разбита на модули:

- Что означает, что приватные API действительно стали приватными

Strong Encapsulation?

Платформа разбита на модули:

- Что означает, что приватные API действительно стали приватными
- Но а как же

sun.misc.Unsafe ?

Strong Encapsulation?

Ok, **sun.misc.Unsafe** Java сообщество
(в неравной борьбе) **отстояло!**
(не надолго)

Strong Encapsulation?

А как же Dependency Injection?

Strong Encapsulation?

DI frameworks существенно зависят:

- от возможности рефлексивного доступа в код, куда они внедряют зависимости
- в том числе в неэкспортируемый код

Strong Encapsulation?

```
module my.module {  
  exports my.module.pack;  
}
```


Strong Encapsulation?

```
module my.module {  
  exports my.module.pack;  
  exports my.module.internal.public.morozov;  
}
```

Strong Encapsulation?

```
module my.module {  
  exports my.module.pack;  
}
```

Strong Encapsulation?

```
open module my.module {  
  exports my.module.pack;  
}
```

Strong Encapsulation?

Для DI frameworks введены *open modules*:

- open module позволяет рефлексивный доступ к своей неэкспортированной функциональности

Strong Encapsulation?

Open module – это уже не совсем strong encapsulation, но это лучше чем ничего.

Jigsaw

Хорошо, ну а какая мне в конце концов от Jigsaw польза?

Польза Jigsaw

Если все ваши зависимости сейчас перечислены в `classpath`, то постепенно мигрируя на `modulepath` вы улучшите архитектуру своего приложения:

Польза Jigsaw

Если все ваши зависимости сейчас перечислены в `classpath`, то постепенно мигрируя на `modulepath` вы улучшите архитектуру своего приложения:

- разберетесь с циклами в зависимостях

Польза Jigsaw

Если все ваши зависимости сейчас перечислены в `classpath`, то постепенно мигрируя на `modulepath` вы улучшите архитектуру своего приложения:

- разберетесь с циклами в зависимостях
- `split` пакетами (уберете `jar hell`)

Польза Jigsaw

Если все ваши зависимости сейчас перечислены в `classpath`, то постепенно мигрируя на `modulepath` вы улучшите архитектуру своего приложения:

- разберетесь с циклами в зависимостях
- `split` пакетами (уберете `jar hell`)
- с необоснованным доступом в детали реализации других модулей

Польза Jigsaw

Если все ваши зависимости сейчас перечислены в classpath, то постепенно мигрируя на modulepath вы улучшите архитектуру своего приложения:

- разберетесь с циклами в зависимостях
- split пакетами (уберете jar hell)
- с необоснованным доступом в детали реализации других модулей
- с зависимостями на JDK private API

Польза Jigsaw

Для миграции на `modulepath` в Jigsaw придумана целая система:

Польза Jigsaw

Для миграции на `modulepath` в Jigsaw придумана целая система:

- Old Classpath образует **Unnamed Module**

Польза Jigsaw

Для миграции на `modulepath` в Jigsaw придумана целая система:

- Old Classpath образует **Unnamed Module**
- Жары из `classpath` можно временно переносить в `modulepath` как **Auto Module**

Польза Jigsaw

Для миграции на `modulepath` в Jigsaw придумана целая система:

- Old Classpath образует **Unnamed Module**
- Жары из `classpath` можно временно переносить в `modulepath` как **Auto Module**
- автомодулям можно постепенно добавлять модульную декларацию

Польза Jigsaw

Но к сожалению, для большинства enterprise Java разработчиков пользы от Jigsaw пока не будет:

- Java EE стандарты пока не определяют как они собираются взаимодействовать с модульной системой

Польза Jigsaw

Но к сожалению, для большинства enterprise Java разработчиков пользы от Jigsaw пока не будет:

- Java EE стандарты пока не определяют как они собираются взаимодействовать с модульной системой
- Даже стандарт servlet контейнеров не знает пока про модули
 - Зависимости в war файлах – это по сути тот же самый classpath!

Роды неизбежны!



Заключение

- OSGi – милая попытка дать разработчикам модули
 - но к сожалению в OSGi есть много проблем
 - в том числе фундаментальных
- Jigsaw – тщательно продуманная система без видимых фундаментальных проблем
 - но с системой сдержек и противовесов
 - есть проблемы принятия сообществом

Вопросы и ответы

Никита Липский,
Excelsior

nlipsky@excelsior-usa.com
twitter: @pjBooms