Java 9 Модули. Почему не OSGi?

Никита Липский

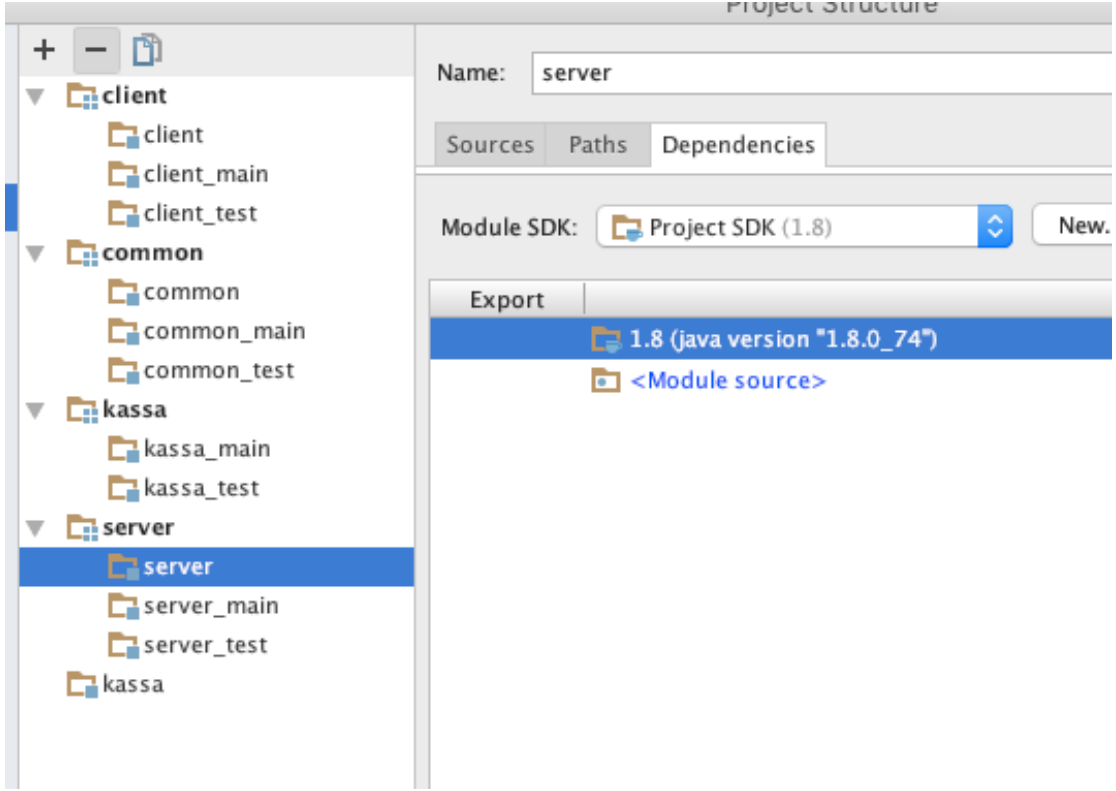
Excelsior LLC

EXCELSIOR


случаи
рассказы
дневниковые
записи
ДАНИИЛ
ХАРМС
ГОРЛО
БРЕДИТ
БРИТВОЮ
ГЛАГОЛ
4·1991

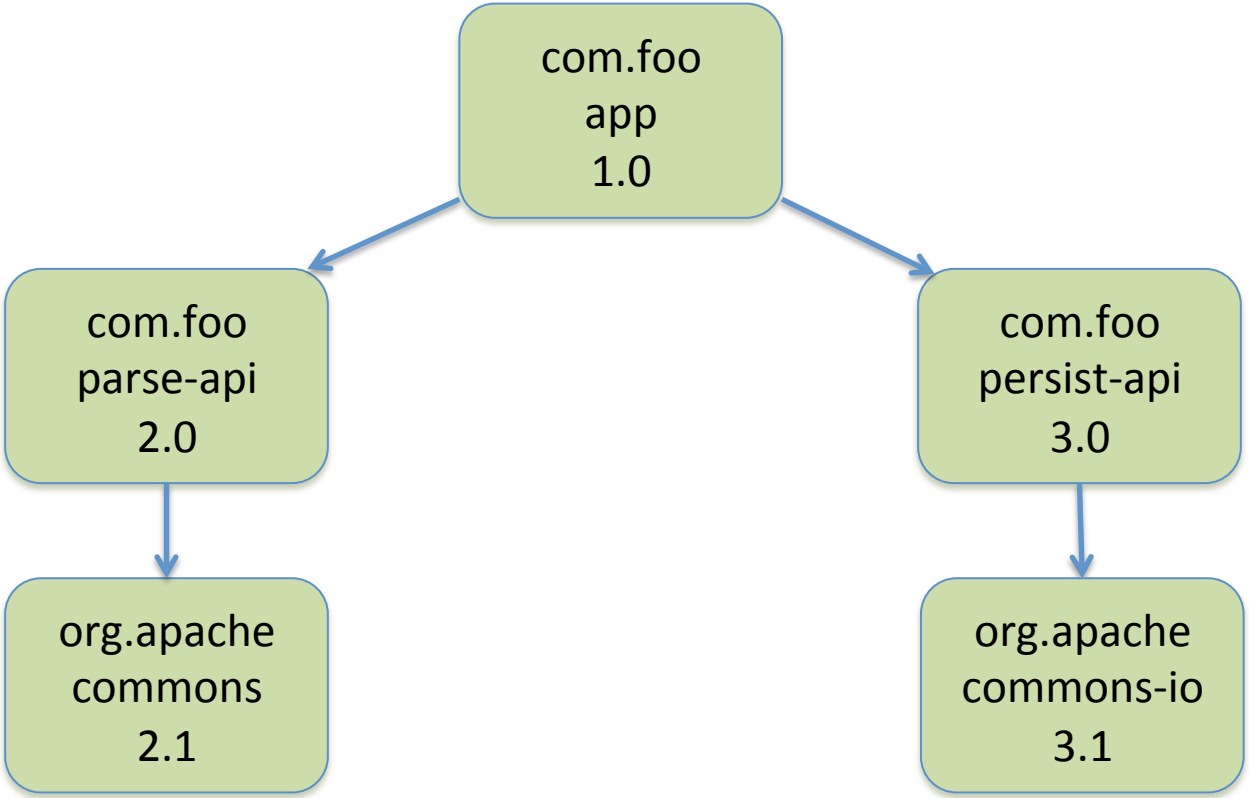
EXCELSIOR

Модули в IDE



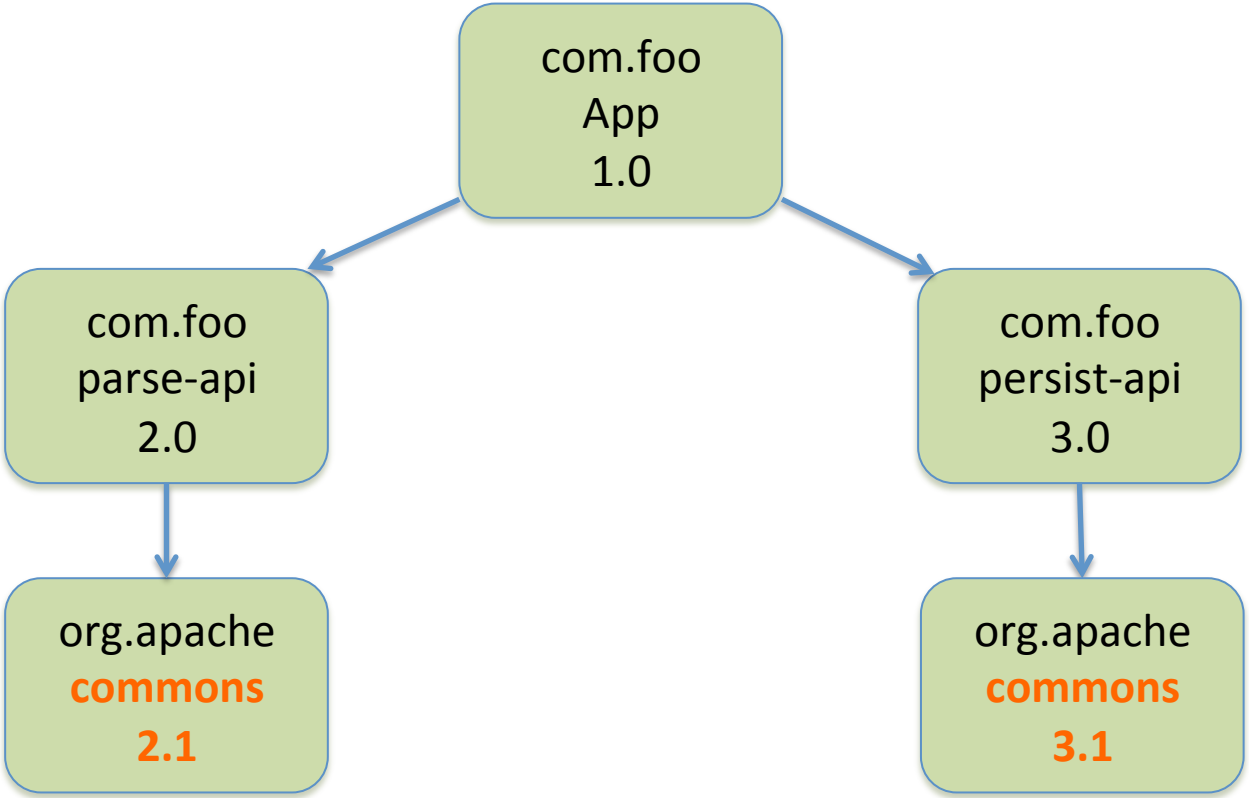
EXCELSIOR

Модули в Maven



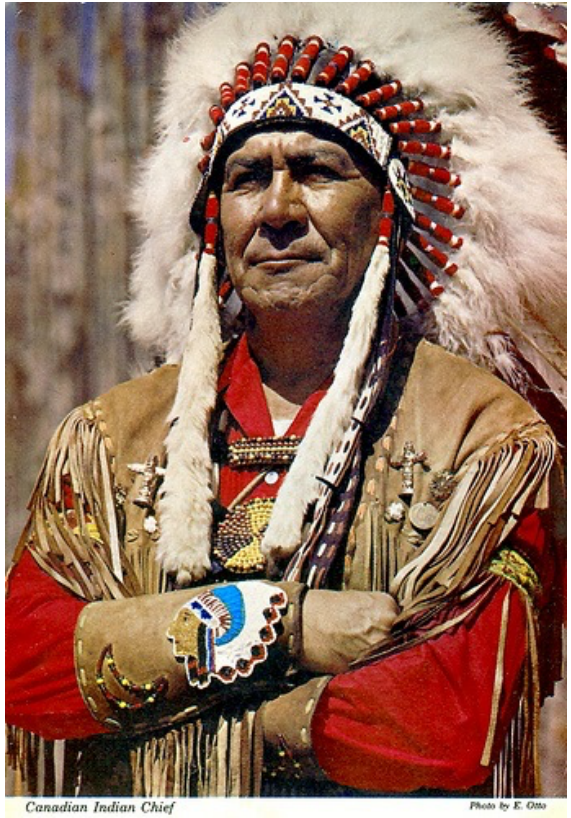
EXCELSIOR

Модули в Maven



EXCELSIOR

OSGi



EXCELSIOR

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”

EXCELSIOR

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?

EXCELSIOR

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?

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

EXCELSIOR

про себя

- Инициатор проекта Excelsior JET
  - работал над проектом более 17 лет
  - как идейный вдохновитель
  - компиляторный инженер
  - руководитель
  - и много в каких еще ролях
- twitter: @pjBooms
- team blog: https://www.excelsiorjet.com/blog

EXCELSIOR

План доклада

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

EXCELSIOR

Где OSGi?

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

Реализации:

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

EXCELSIOR

Почему OSGi?

EXCELSIOR

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/

EXCELSIOR


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)"
```

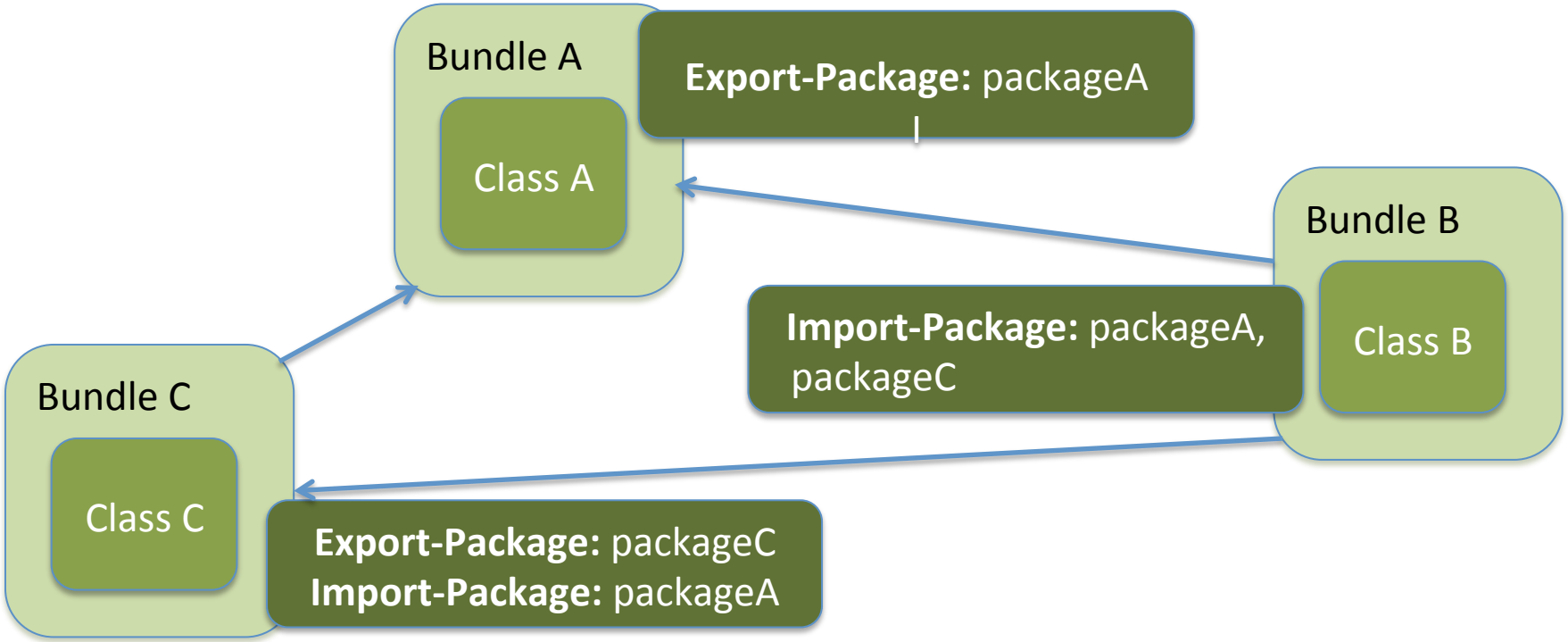
EXCELSIOR

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).

EXCELSIOR

Модульная система OSGi



EXCELSIOR

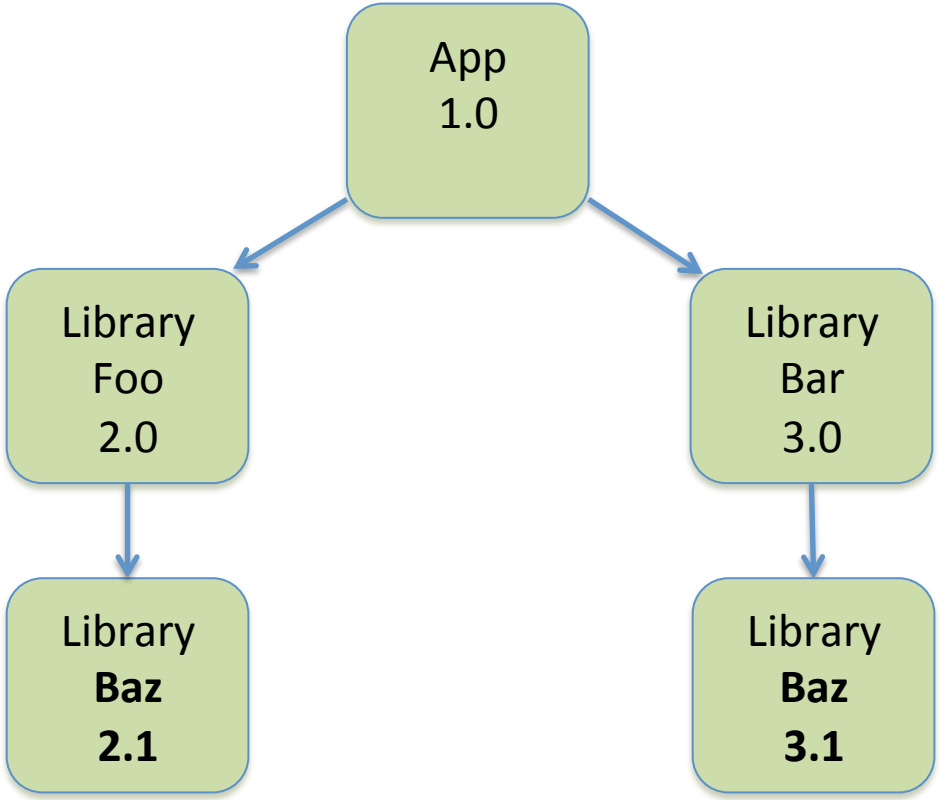
OSGi Runtime

- 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).

EXCELSIOR

Jar/Classpath Hell



EXCELSIOR

Версионирование в OSGi

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

EXCELSIOR

Почему OSGi?

Итак, OSGi обещает:

Почему OSGi?

Итак, OSGi обещает:

- Модульность
  - явные зависимости
  - инкапсуляция

EXCELSIOR

Почему OSGi?

Итак, OSGi обещает:

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

EXCELSIOR

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

EXCELSIOR

Гладко было
на бумаге,
да забыли
про овраги

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

Dslov.ru

EXCELSIOR

Как OSGi?

EXCELSIOR

Версионирование в OSGi

Вопрос: Как реализовать версионирование?

EXCELSIOR

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.

EXCELSIOR

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*.

EXCELSIOR

Версионирование в OSGi

Таким образом, каждый OSGi бандл грузится своим загрузчиком классов:

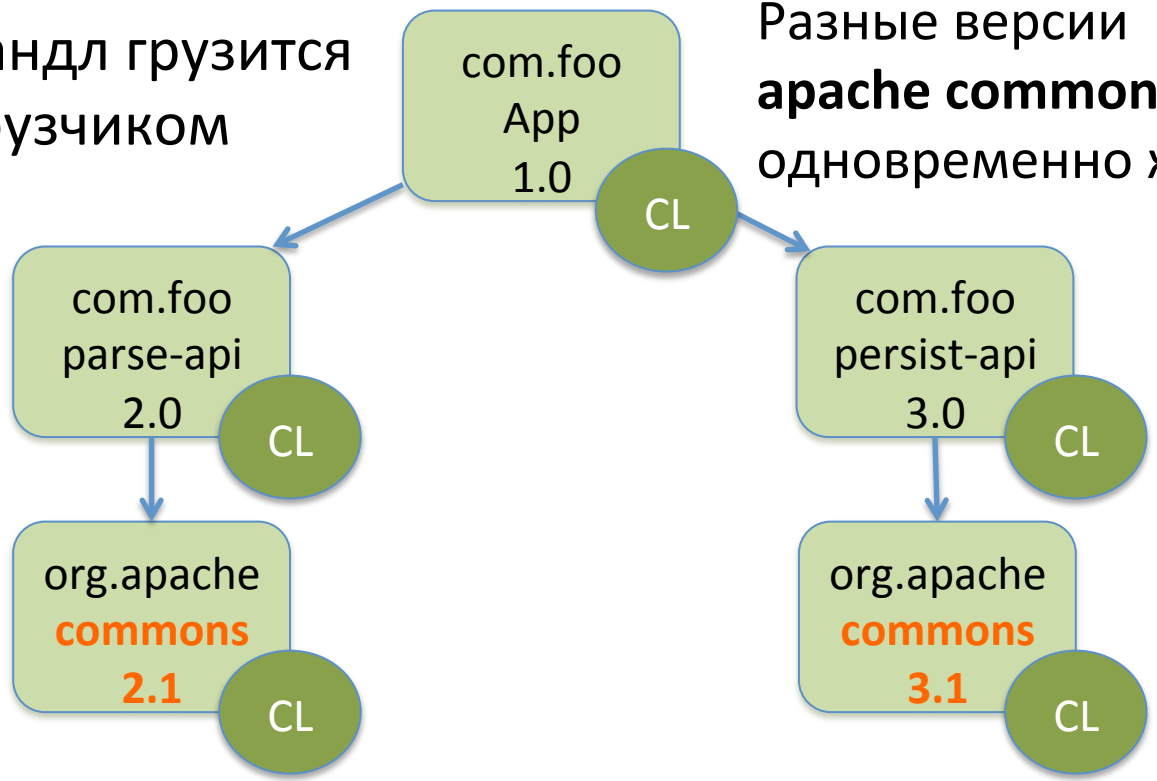
- наследник java.lang.**ClassLoader**
- ***уникальное пространство имен*** классов
- нет конфликтов с классами других бандлов

EXCELSIOR

Версионирование в OSGi

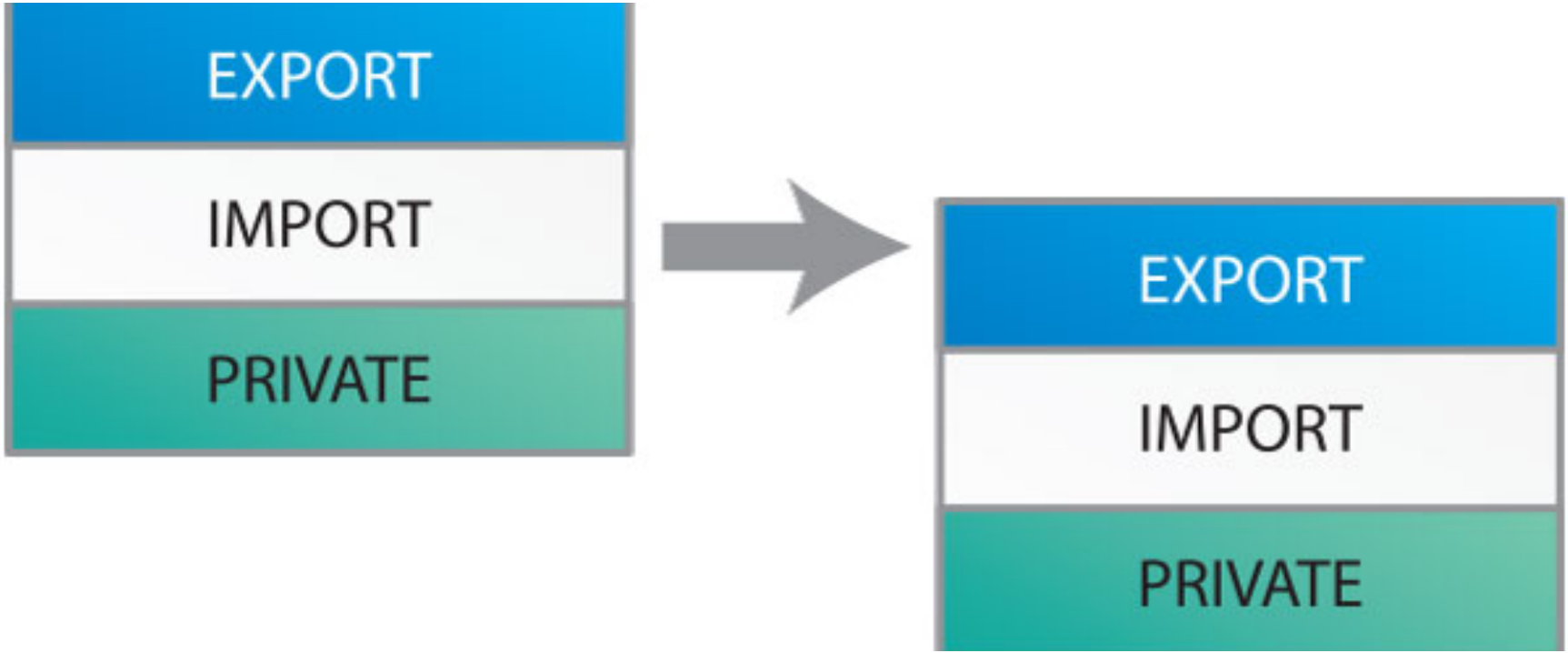
Каждый бандл грузится своим загрузчиком

Разные версии **apache commons** могут одновременно жить в JVM



EXCELSIOR

Инкапсуляция в OSGi



EXCELSIOR

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?

EXCELSIOR

Инкапсуляция в OSGi

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

EXCELSIOR

Инкапсуляция в OSGi

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

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

EXCELSIOR

Динамическое обновление

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

EXCELSIOR

Динамическое обновление

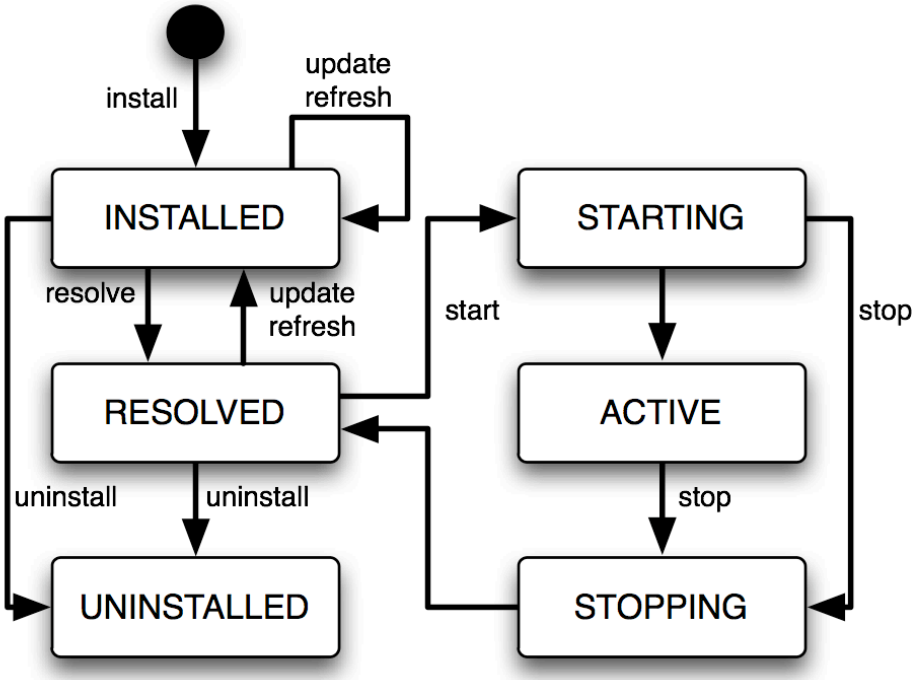
Задача: заменить один (!) изменившийся модуль в уже запущенной программе, не останавливая программу

Решение: ЗАГРУЗЧИКИ КЛАССОВ!

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

EXCELSIOR

Жизненный цикл бандла



EXCELSIOR

Ленивый старт

Старт бандлов в OSGi сделан через механизм активаторов бандлов:

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

EXCELSIOR

Ленивый старт

Старт бандлов в OSGi сделан через механизм активаторов бандлов:

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

EXCELSIOR

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

EXCELSIOR

Ленивый старт

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

Ленивый старт

Решение: ЗАГРУЗЧИК КЛАССОВ (опять!) …

EXCELSIOR

Ленивый старт

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

EXCELSIOR

Ленивый старт

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

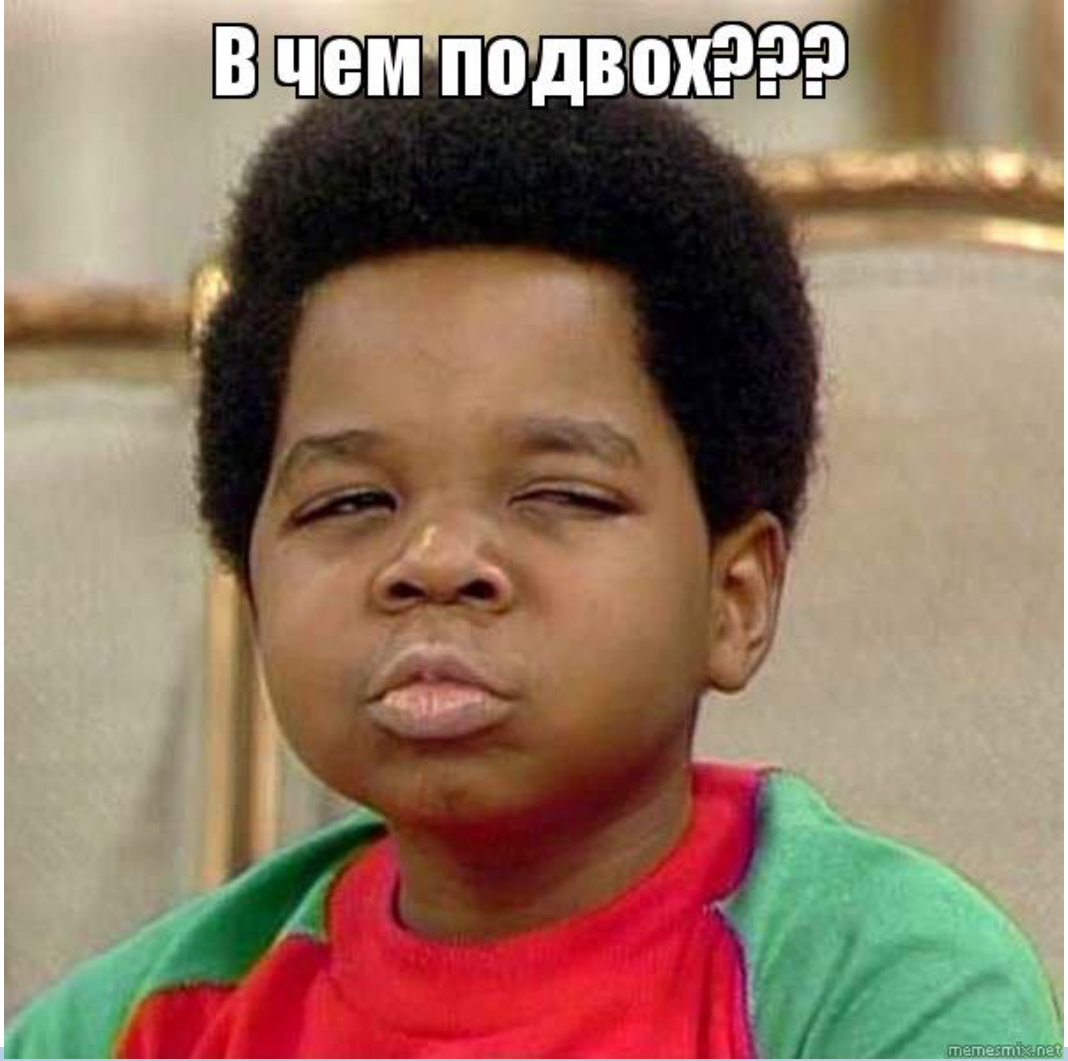
Так как загрузка классов в JVM – ленивая, то и активация бандлов становится ленивой АВТОМАТОМ

EXCELSIOR

Как OSGi?

Семь бед – один ответ!

EXCELSIOR

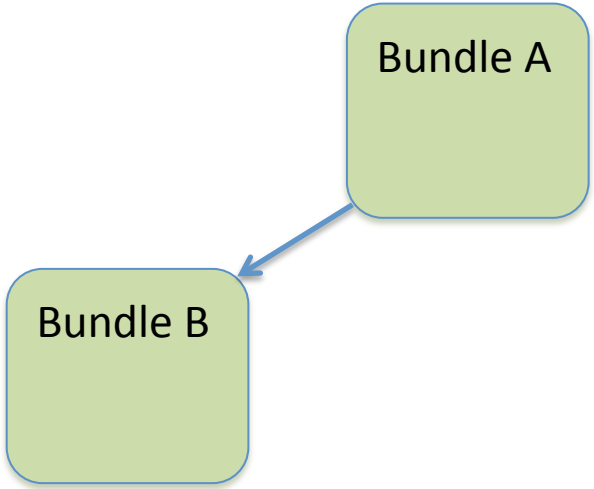


EXCELSIOR

Почему НЕ OSGi?

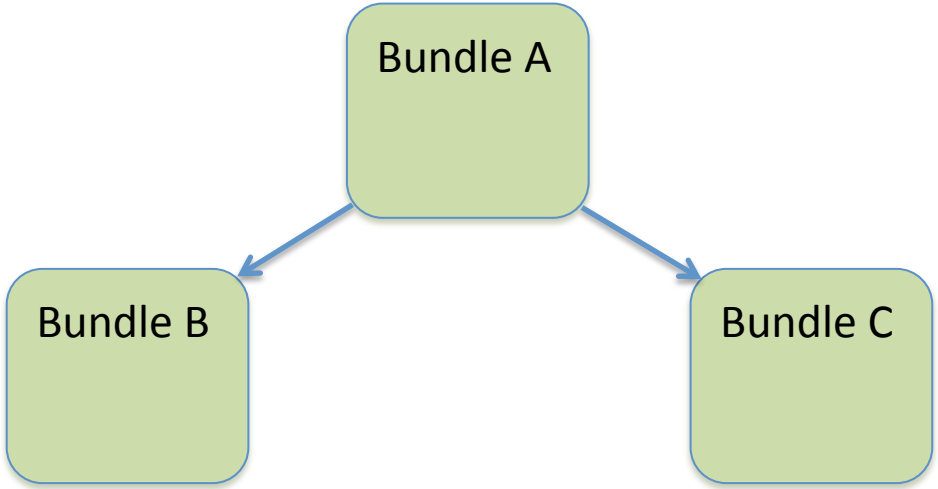
EXCELSIOR

Модульность?



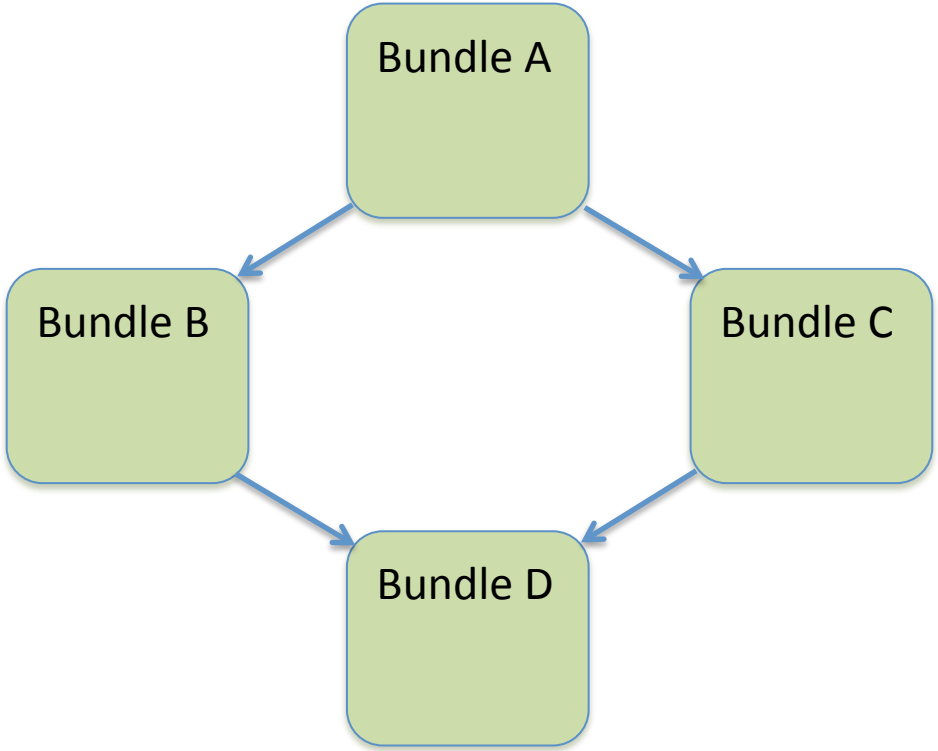
EXCELSIOR

Модульность?



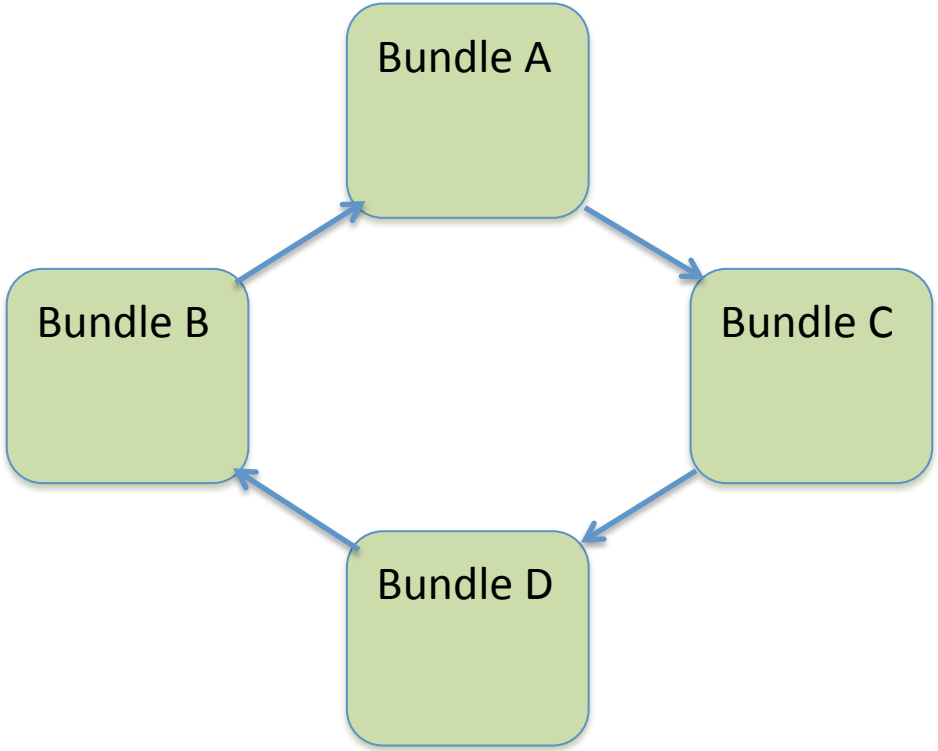
EXCELSIOR

Модульность?



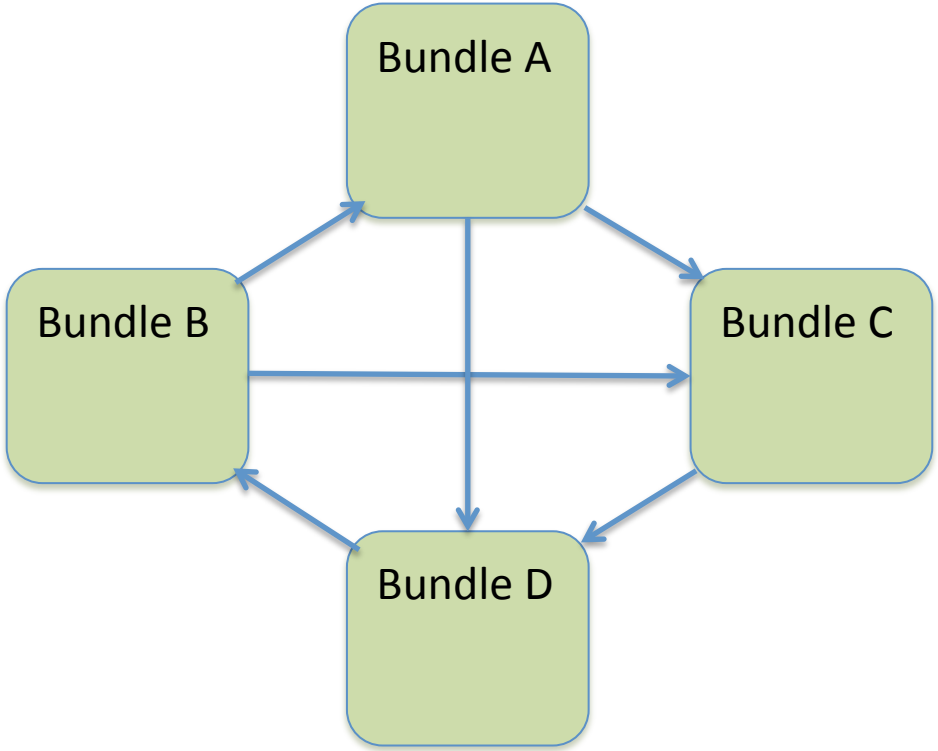
EXCELSIOR

Модульность?



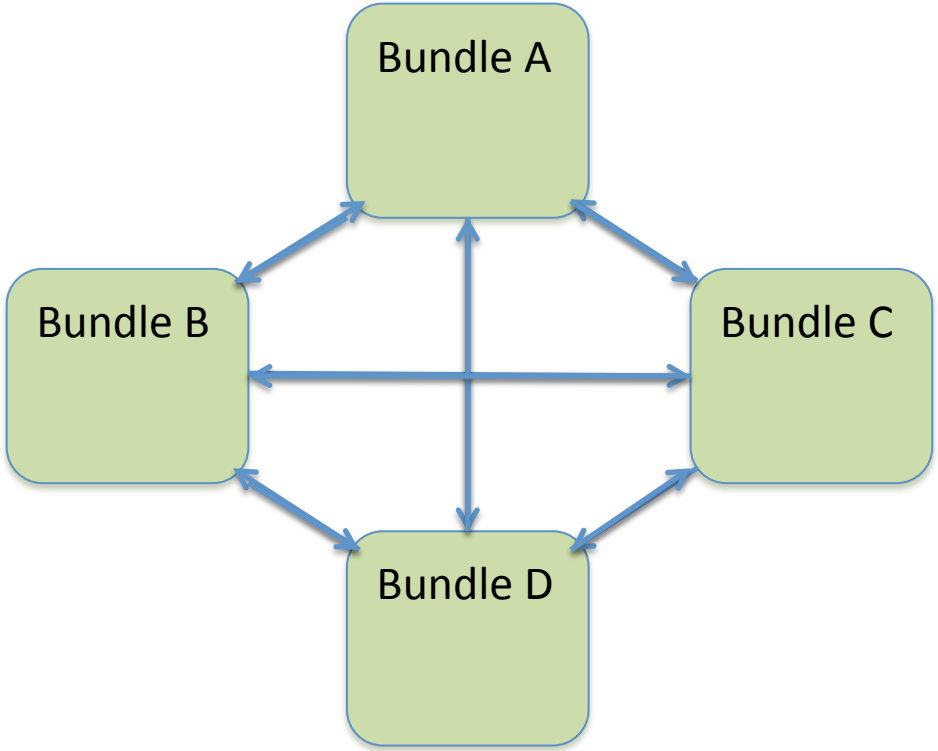
EXCELSIOR

Модульность?



EXCELSIOR

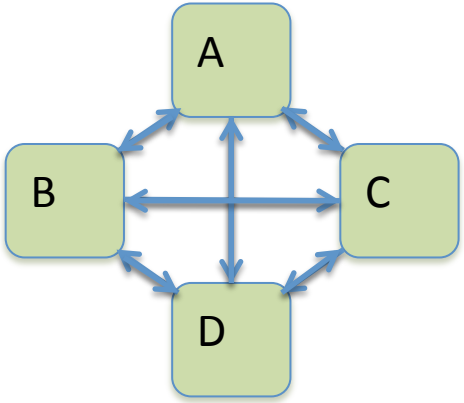
Модульность?



EXCELSIOR

Модульность?

OSGI позволяет циклы в графе зависимостей



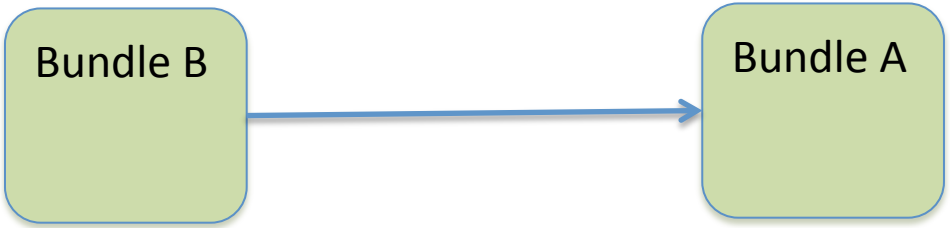
Упражнение: понять, почему это плохо

EXCELSIOR

Обновления на лету?

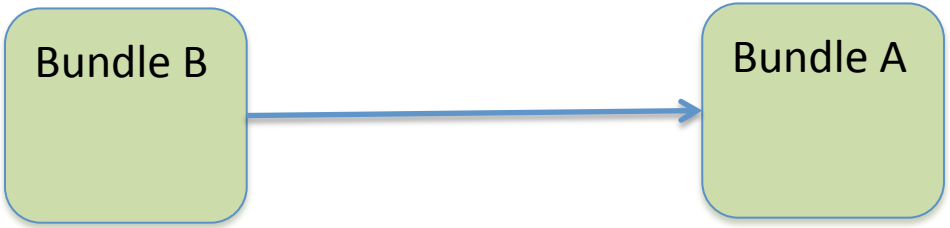
EXCELSIOR

Обновления на лету?



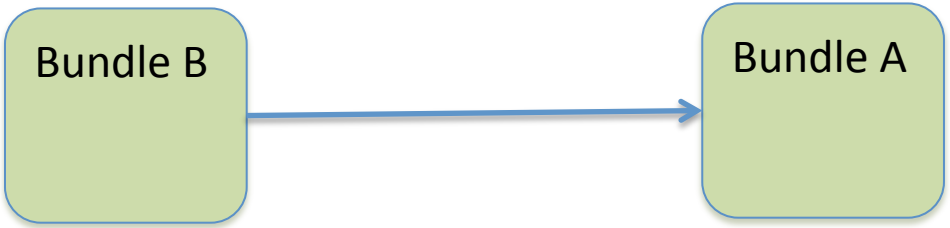
EXCELSIOR

Обновления на лету?



Обновляем бандл B

Обновления на лету?



EXCELSIOR

Обновления на лету?

Bundle A

Bundle B

EXCELSIOR

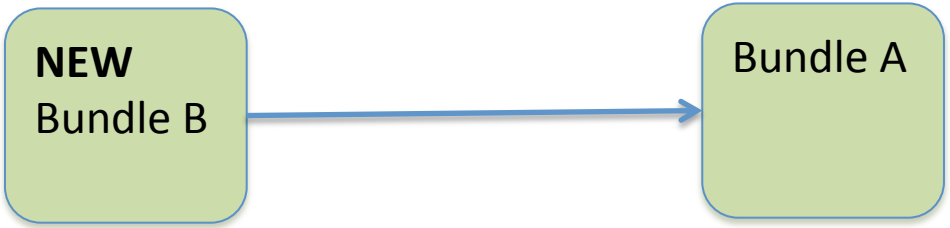
Обновления на лету?

Bundle A

NEW
Bundle B

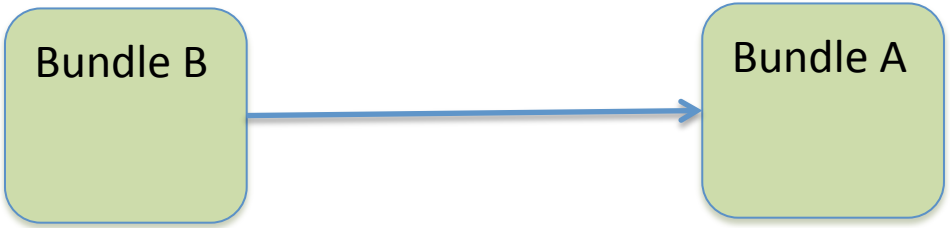
EXCELSIOR

Обновления на лету?



EXCELSIOR

Обновления на лету?



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

EXCELSIOR

Обновления на лету?

EXCELSIOR

Обновления на лету?

Если бандл B импортирует бандл A, значит есть класс из B, ссылающийся на класс из A

EXCELSIOR

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(){};
}
```

EXCELSIOR

Разрешение символьных ссылок

B.class
…
CONSTANT_Class: **A**
CONSTANT_FieldRef: **A.f@int**
CONSTANT_MethodRef: **A.foo()I**
…

A.class
…
Field: **f@int**
Method: **foo()I**
…

EXCELSIOR

Разрешение символьных ссылок

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

EXCELSIOR

Разрешение символьных ссылок

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

EXCELSIOR

Разрешение символьных ссылок

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

EXCELSIOR

Обновления на лету?

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

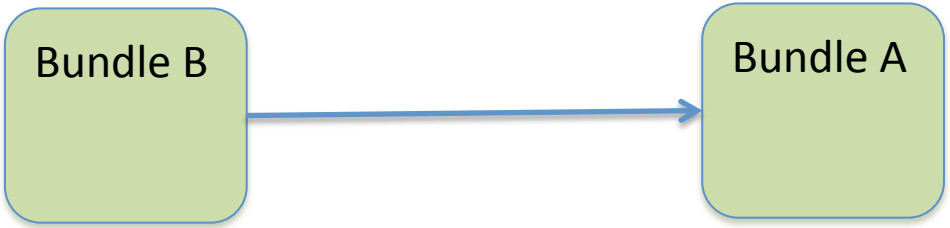
EXCELSIOR

Обновления на лету?

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

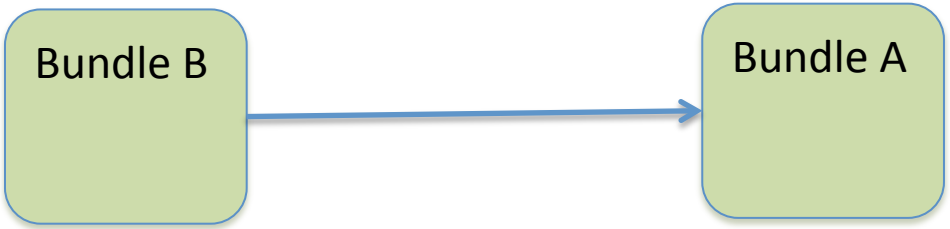
EXCELSIOR

Обновления на лету?



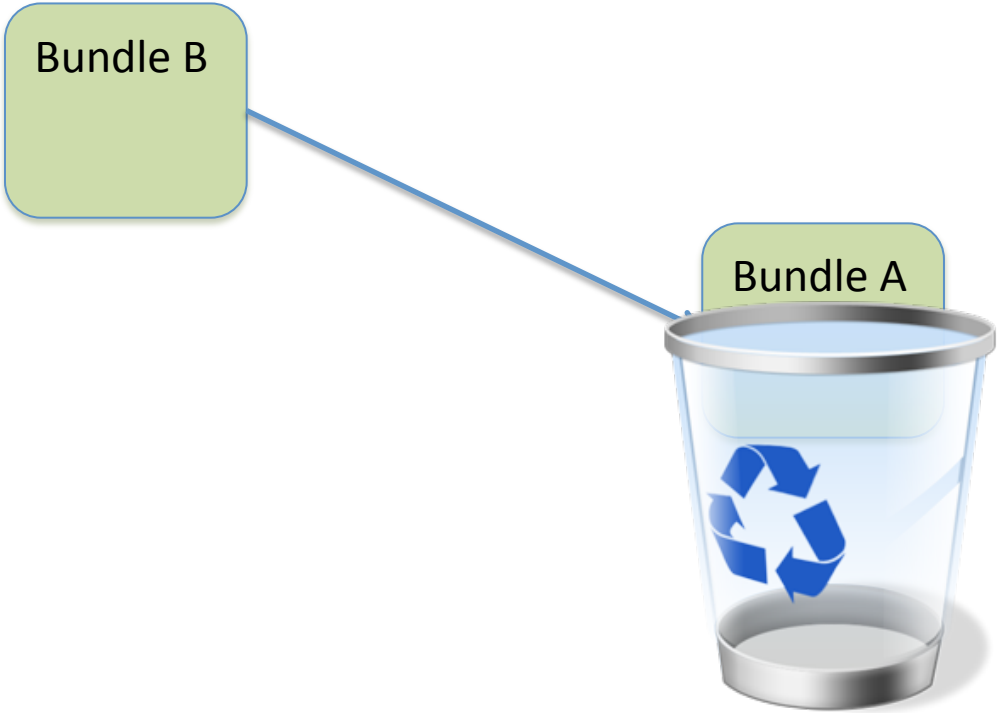
Обновляем бандл **A**

Обновления на лету?



EXCELSIOR

Обновления на лету?



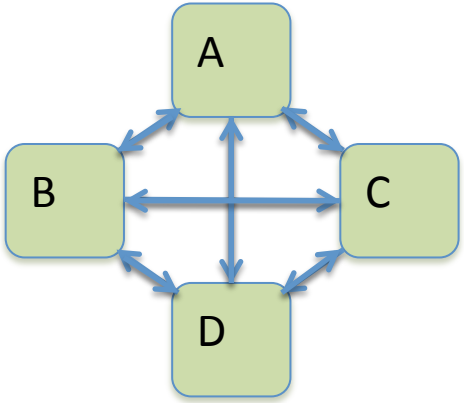
EXCELSIOR

Обновления на лету?

EXCELSIOR

Обновления на лету?

А теперь вспомним про циклические зависимости



Упражнение: попробуйте обновить SWT бандл у работающего Eclipse

EXCELSIOR


Jenga
Jenga

Обновления на лету?

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

EXCELSIOR

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

EXCELSIOR

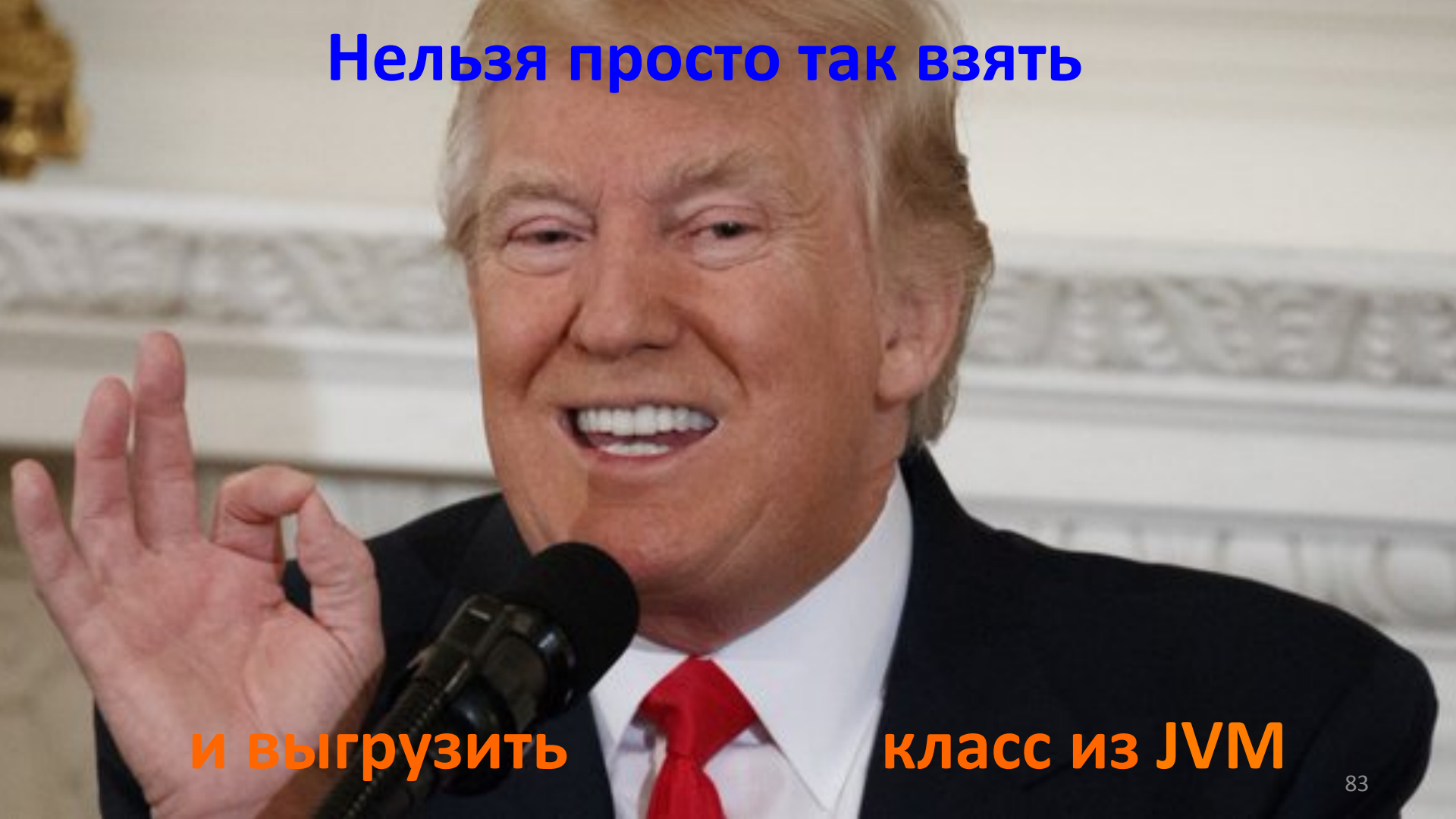
Обновления на лету?

Даже листовые бандлы не так просто выгрузить из JVM:

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

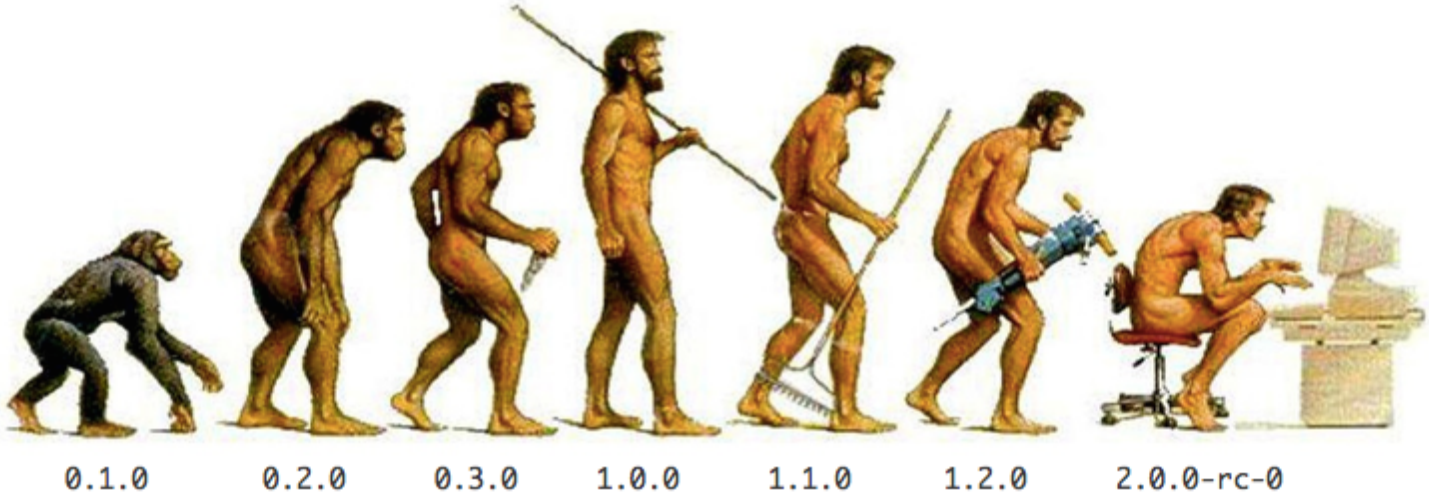
EXCELSIOR

Нельзя просто так взять



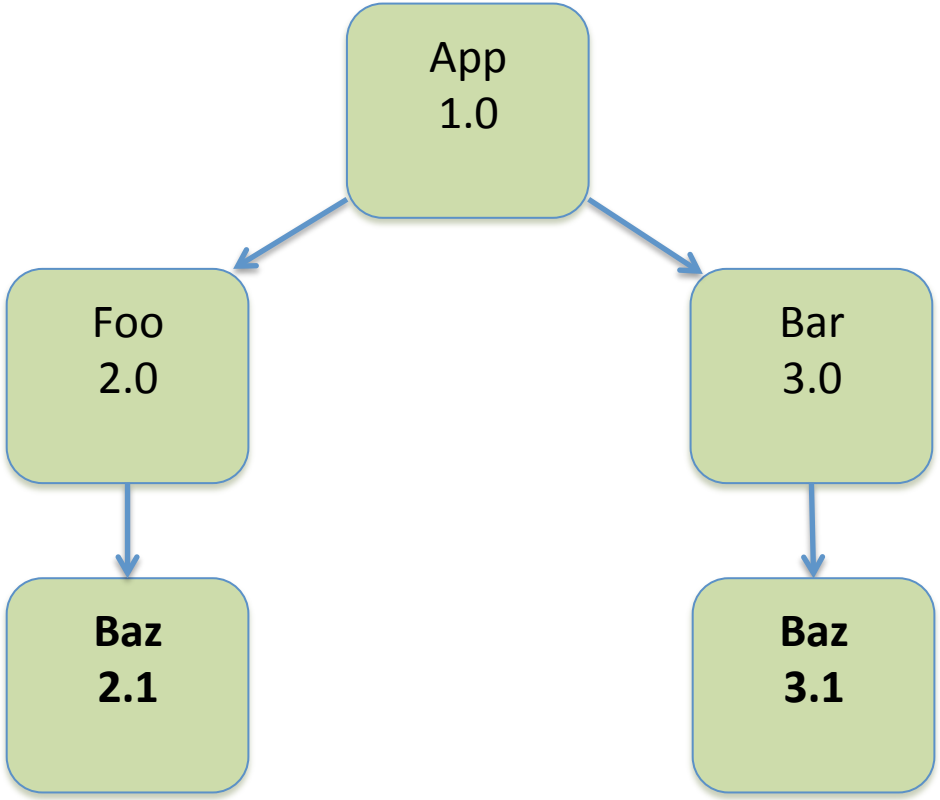
и выгрузить класс из JVM

Версионирование?



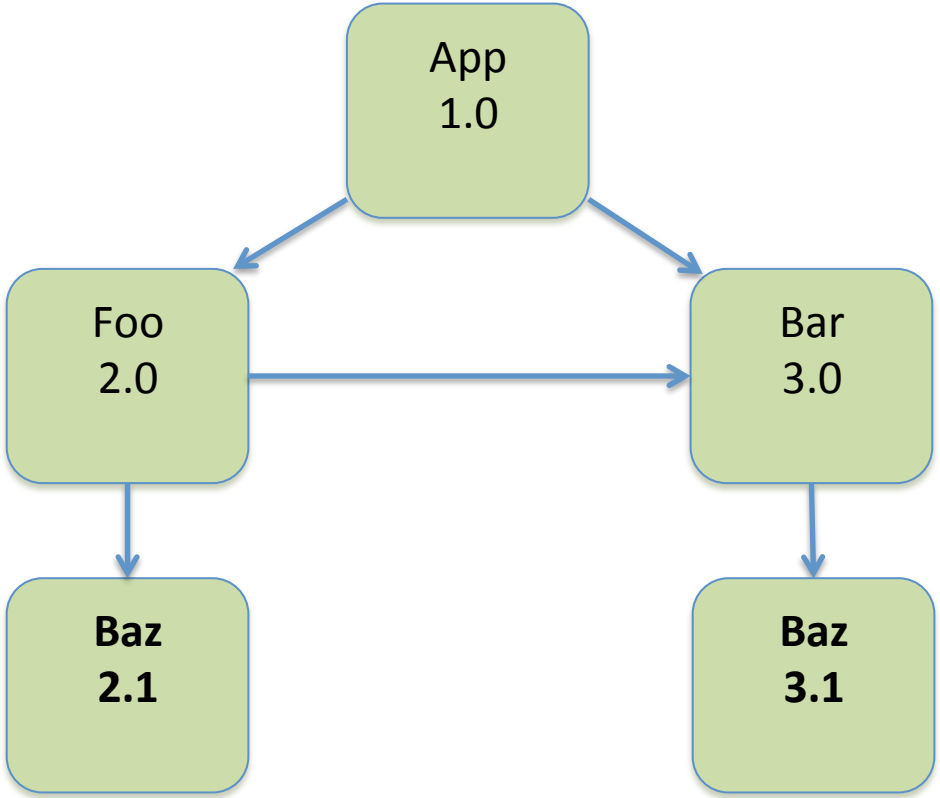
EXCELSIOR

Версионирование?



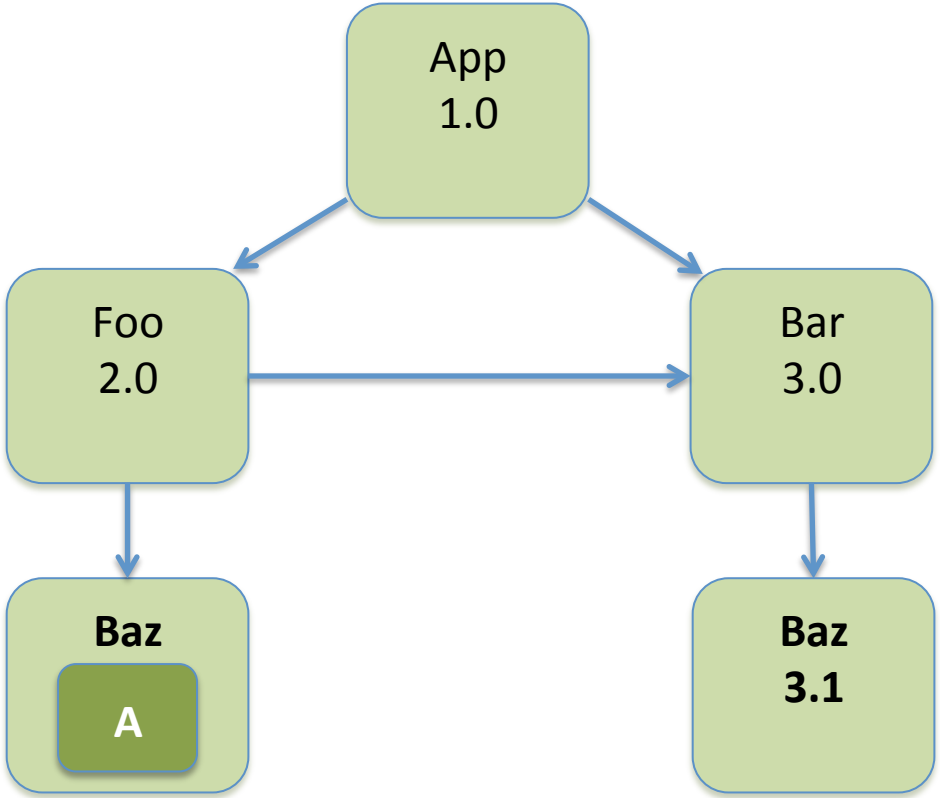
EXCELSIOR

Версионирование?



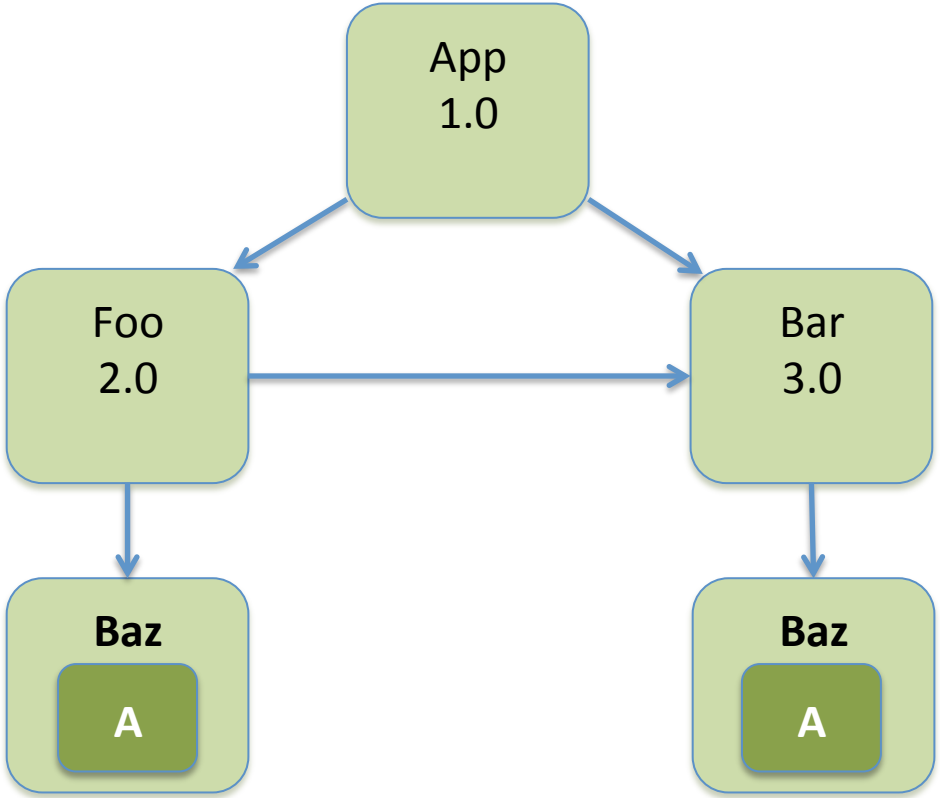
EXCELSIOR

Версионирование?



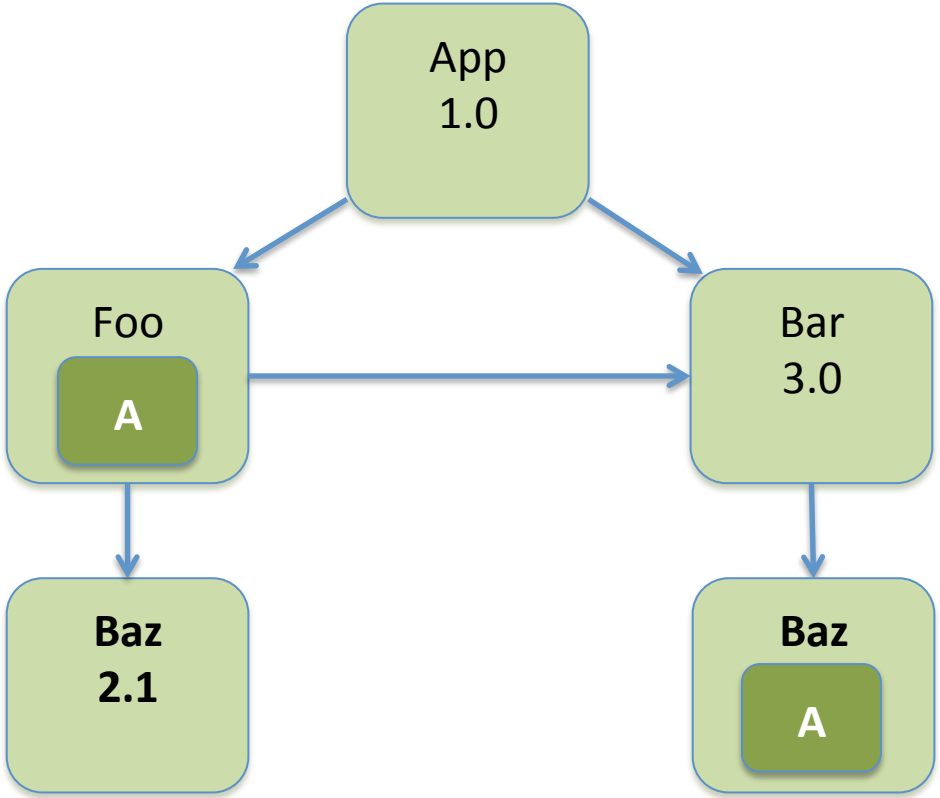
EXCELSIOR

Версионирование?



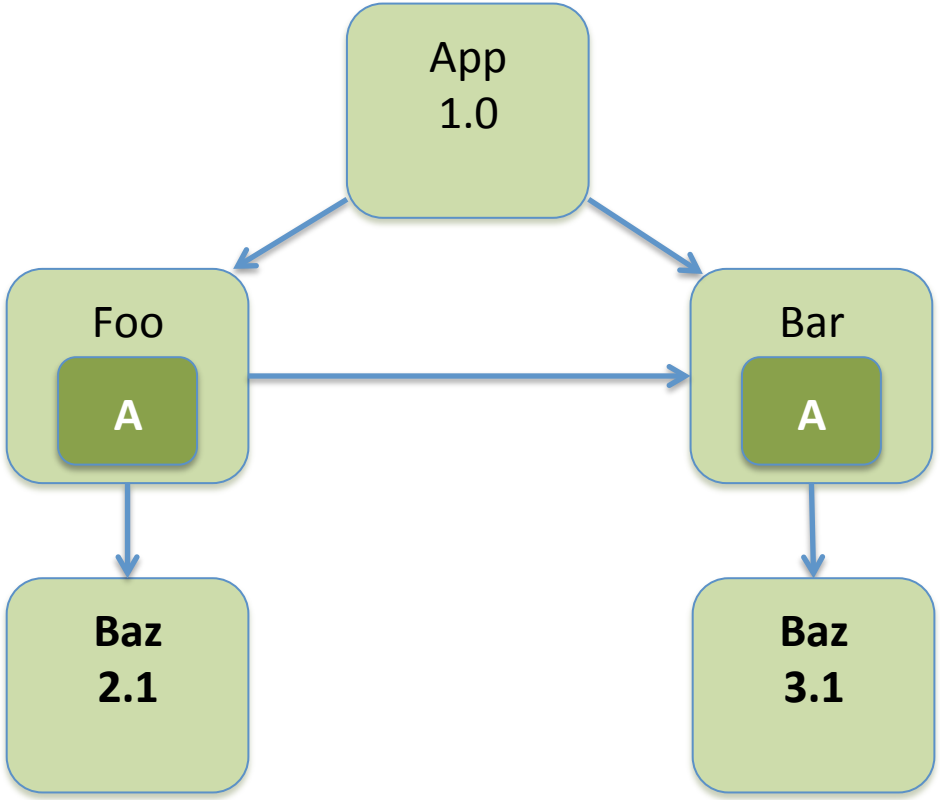
EXCELSIOR

Версионирование?



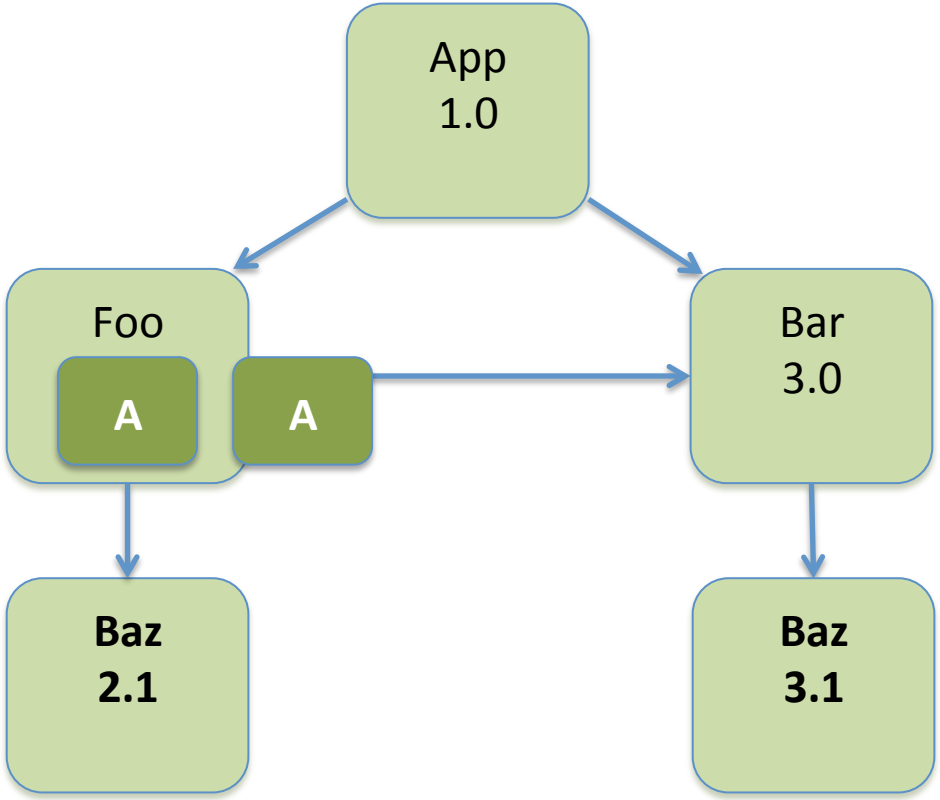
EXCELSIOR

Версионирование?



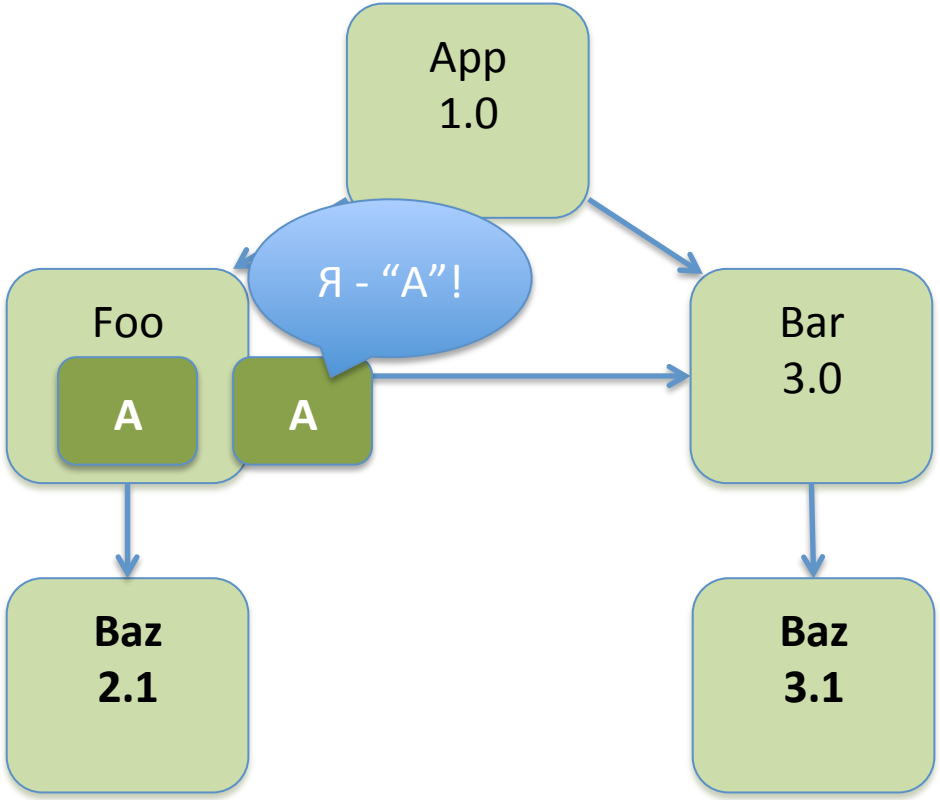
EXCELSIOR

Версионирование?



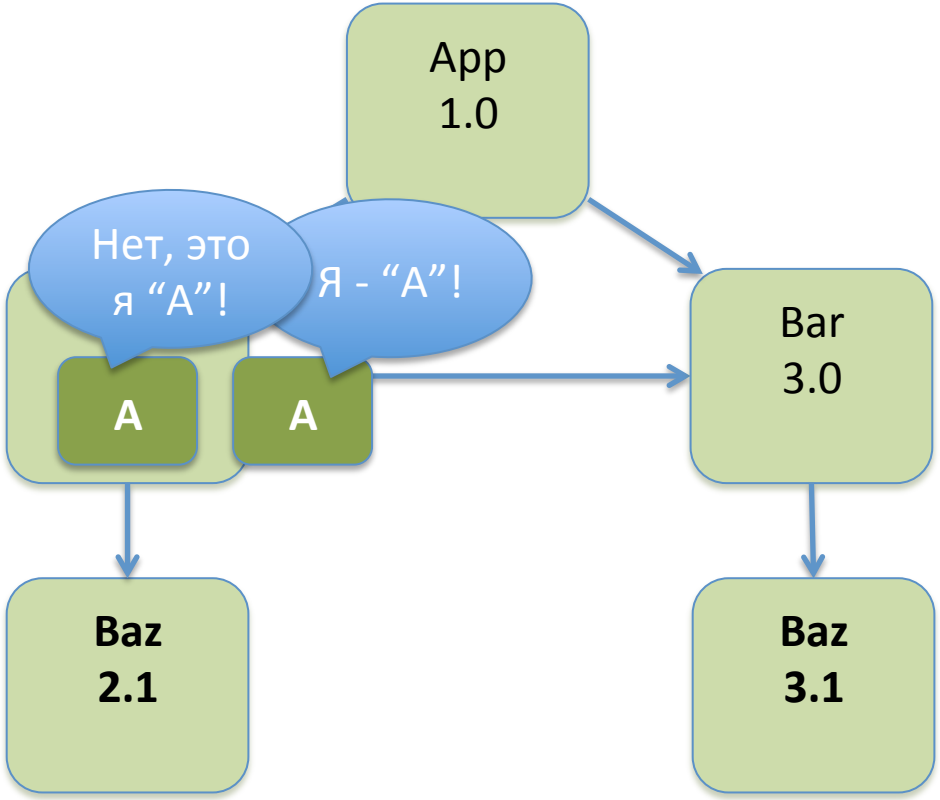
EXCELSIOR

Версионирование?



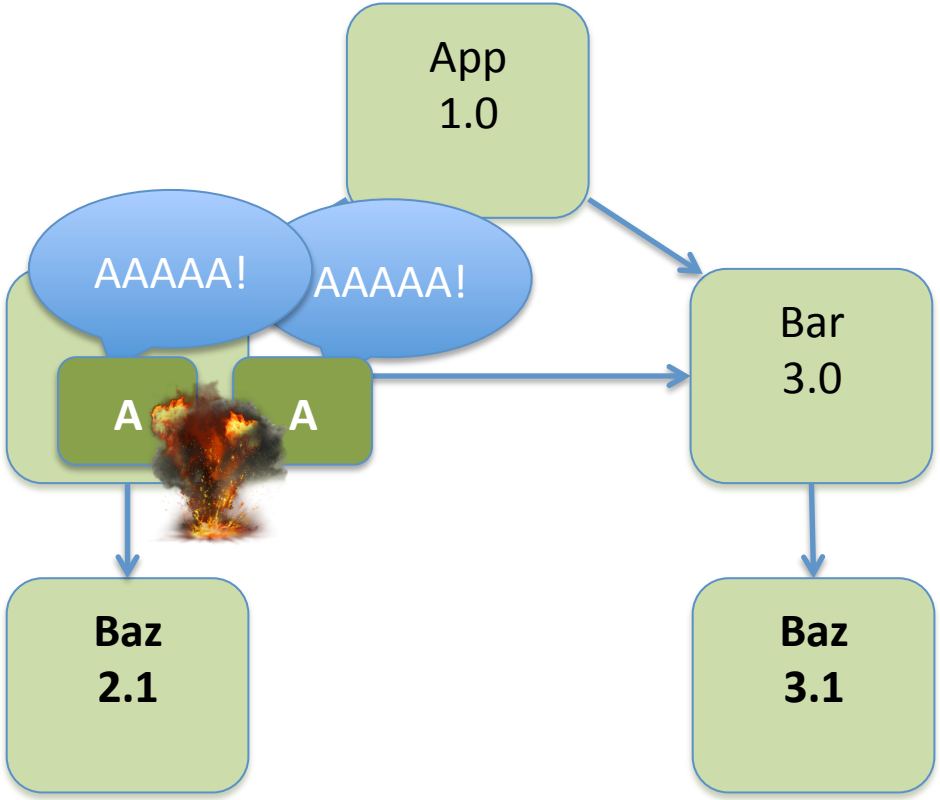
EXCELSIOR

Версионирование?



EXCELSIOR

Версионирование?



EXCELSIOR

Loading constraints

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

EXCELSIOR

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 B грузится загрузчиком L1, а A 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

- В OSGi эта проблема адресуется с помощью, т.н. *uses constraint*

EXCELSIOR

Uses constraints

- Export в OSGi может квалифицироваться директивами use
  - внешнии пакеты использующиеся экспортируемым пакетом

EXCELSIOR

Uses constraints

- Export в OSGi может квалифицироваться директивами use
  - внешнии пакеты использующиеся экспортируемым пакетом
- если в один бандл пришел по импорту какой-то use-пакет из двух разных бандлов, то этот бандл не загрузится

EXCELSIOR

Проблемы uses constraints

- Use директивы не обязательны в OSGi
- Описывать их руками выше понимания большинства разработчиков

EXCELSIOR

Проблемы uses constraints

- Use директивы не обязательны в OSGi
- Описывать их руками выше понимания большинства разработчиков
- Если ошибиться в их описании, легко получить LinkageError от JVM
  - Погуглите по словам OSGI и LinkageError, чтобы оценить масштаб проблемы

EXCELSIOR

Версионирование?

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

EXCELSIOR

Версионирование?

Даже, если вам удалось корректно задать use директивы*, то ошибки нарушения uses constraints плохо диагностируемые, лечение – неочевидное:

- http://njbartlett.name/2011/09/02/uses-constraints.html

* для этого даже есть автоматический инструмент – bnd

EXCELSIOR

Версионирование?

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

EXCELSIOR

Версионирование?

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

EXCELSIOR

Инкапсуляция?

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

EXCELSIOR

Инкапсуляция?

Reflection – универсальная отмычка

против инкапсуляции в Java

EXCELSIOR

Инкапсуляция?

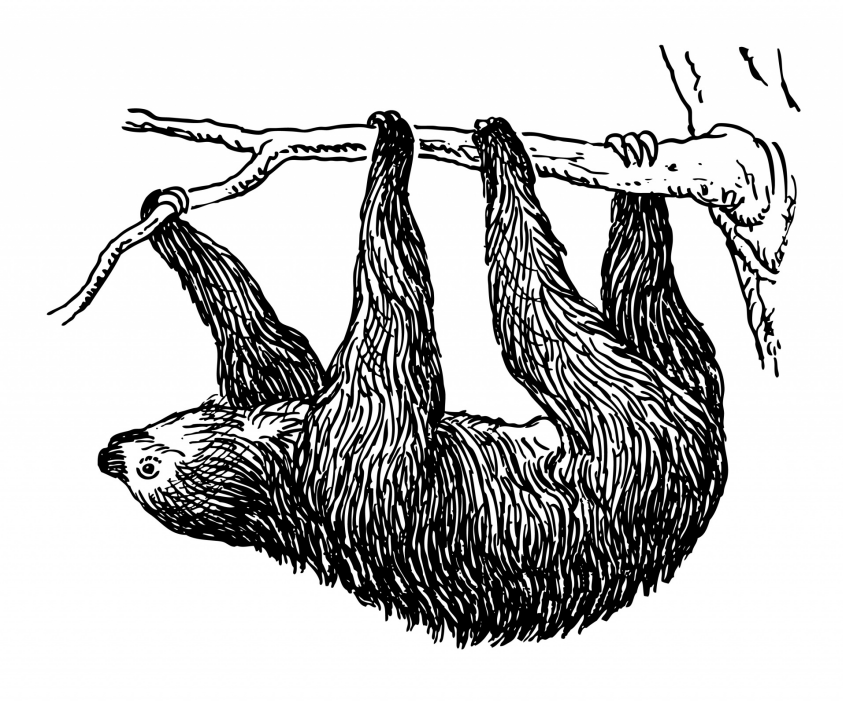
Вопрос: Ну хорошо, проблему инкапсуляции хотя бы OSGi решает?

Ответ: OSGi не защищает от несанкционированного доступа через Reflection*

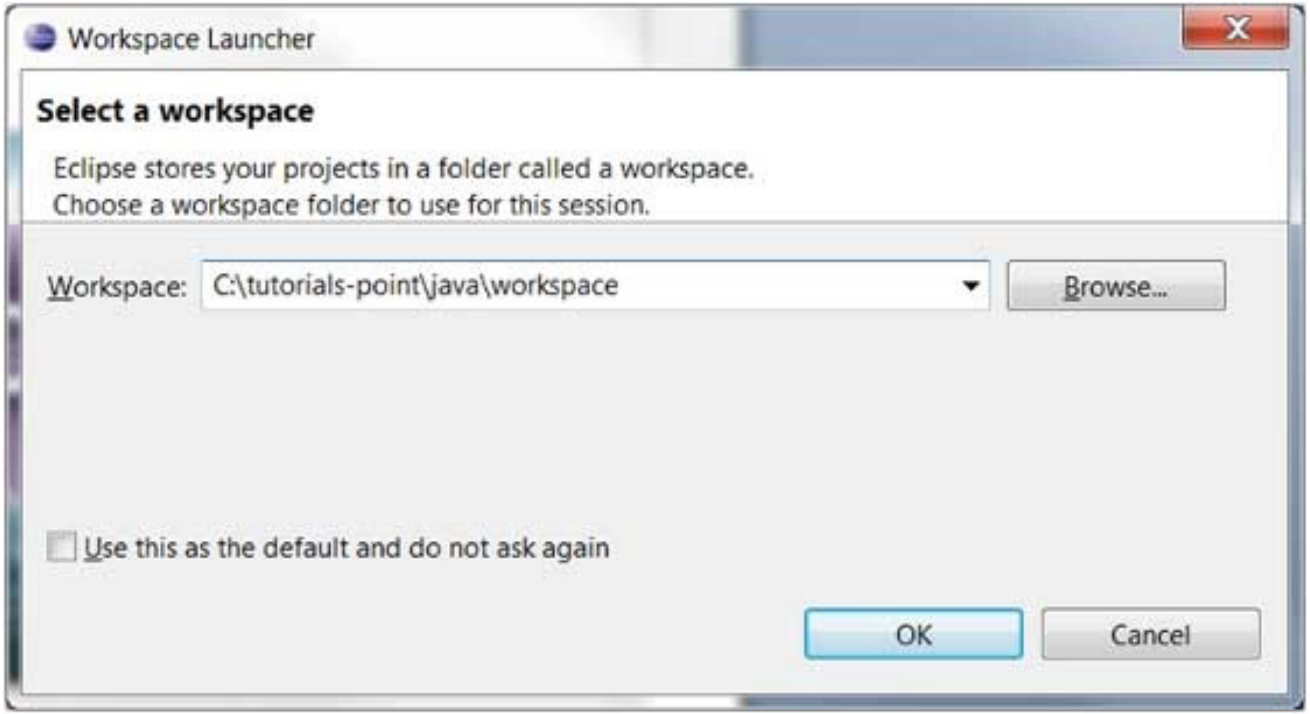
*** можно выставить Security Manager, но вендоры библиотек не могут на это расчитывать**

EXCELSIOR

Ленивый старт?

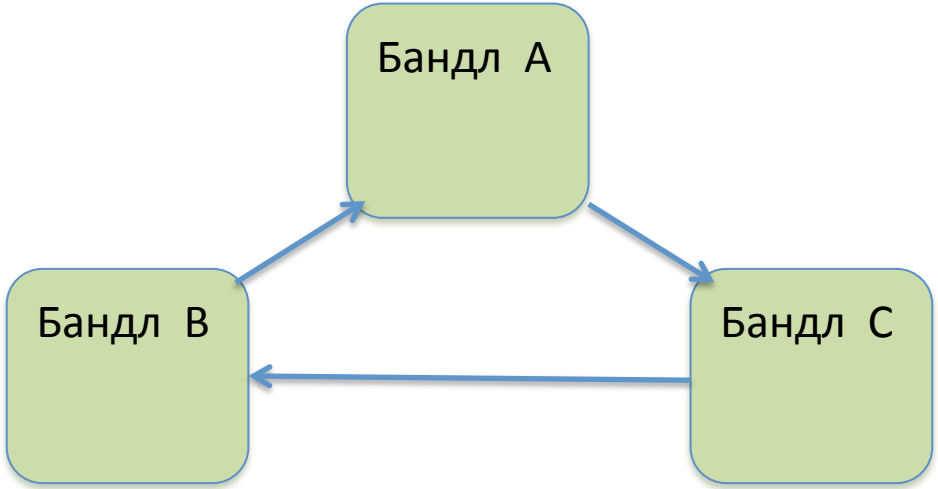


Ленивый старт?



EXCELSIOR

Ленивый старт?



Вопрос: В каком порядке активируются эти бандлы?

EXCELSIOR

Ленивый старт?

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

- при ленивой активации бандл стартуют из loadClass() загрузчика классов бандла

EXCELSIOR

Ленивый старт?

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

- при ленивой активации бандл стартуют из loadClass() загрузчика классов бандла

Но порядок загрузки классов в JVM не определен!

EXCELSIOR

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

EXCELSIOR

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

EXCELSIOR

Ленивый старт?

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

EXCELSIOR

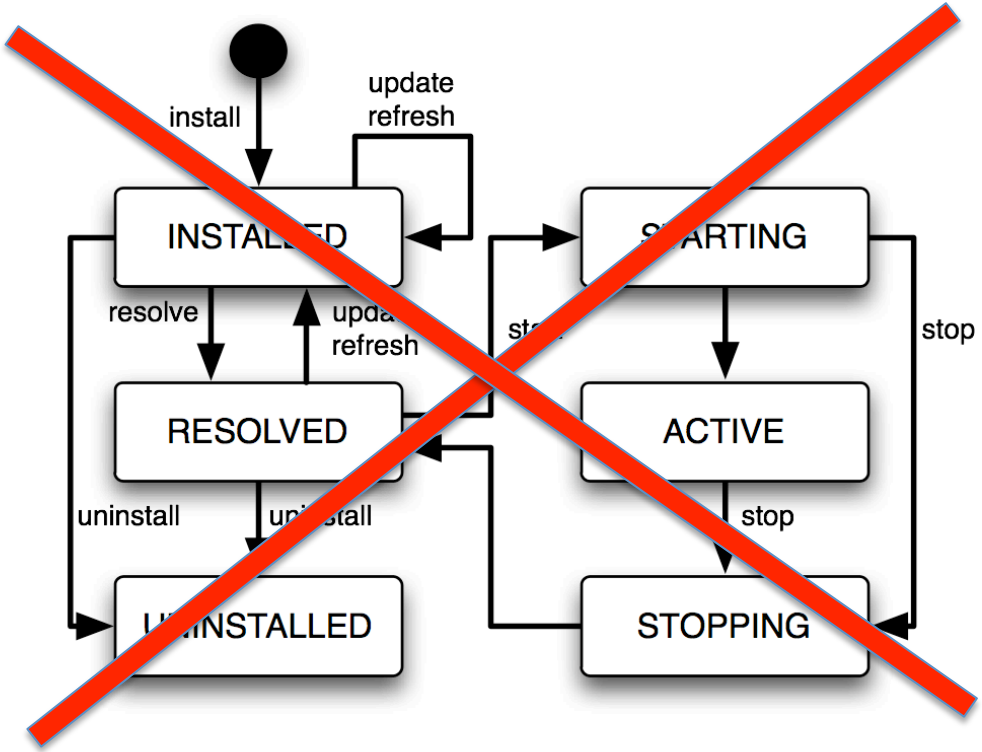
Почему не OSGi?

- Модульность с циклами
- Hot Redeploy работает только для листьев
- Несколько версий одной библиотеки все еще приводит к проблемам
- Нет защиты деталей реализации библиотек от доступа через Reflection
- Ленивая активация бандлов зависит от схемы разрешения ссылок в JVM

EXCELSIOR

Jigsaw

Jigsaw



EXCELSIOR

Jigsaw vs. OSGi

OSGi – динамичен по своей сути

– модули появляются только в ран-тайме

EXCELSIOR

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: пакуют модули
- jlink: изготавливает финальный образ для развертывания
- java: в рантайме модули конечно тоже есть

EXCELSIOR

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;
}
```

EXCELSIOR

Jigsaw vs. OSGi

Jigsaw запрещает циклы

в графе зависимостей (явных)

Модули импортируют

модули, а не пакеты.

Версионирование

В первых драфтах Jigsaw версионирование было (аналогичное OSGi).

Версионирование

В первых драфтах Jigsaw версионирование было (аналогичное OSGi).

Но потом от него отказались …

EXCELSIOR

Версионирование

В первых драфтах Jigsaw версионирование было (аналогичное OSGi).

Но потом от него отказались …

Почему?

EXCELSIOR

Версионирование

Версионирование – немедленное означает:

1 модуль ←→ 1 загрузчик классов

Версионирование

Версионирование – немедленное означает:

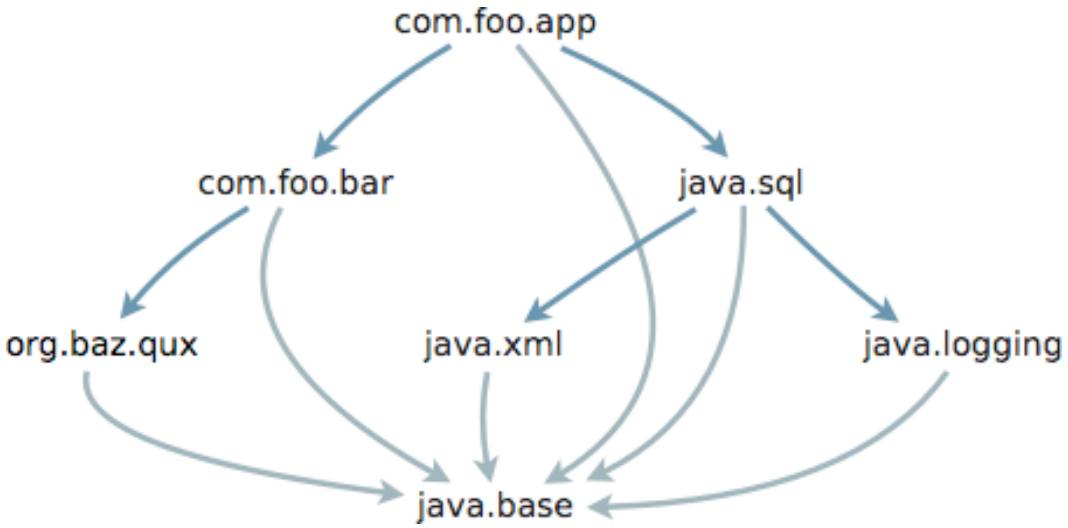
1 модуль ←→ 1 загрузчик классов

И в первых версиях Jigsaw так и было!

EXCELSIOR

Jigsaw и загрузчики

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



EXCELSIOR

Jigsaw и загрузчики

Проблема обратной совместимости:

По спецификации

getClassloader() == null

для всех классов платформы.

EXCELSIOR

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

EXCELSIOR

Jigsaw и загрузчики

Проблема 2: Как защитить разработчика от нарушения loading constraints?

EXCELSIOR

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

EXCELSIOR

Версионирование

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

- Сводится к 3-SAT

EXCELSIOR

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.

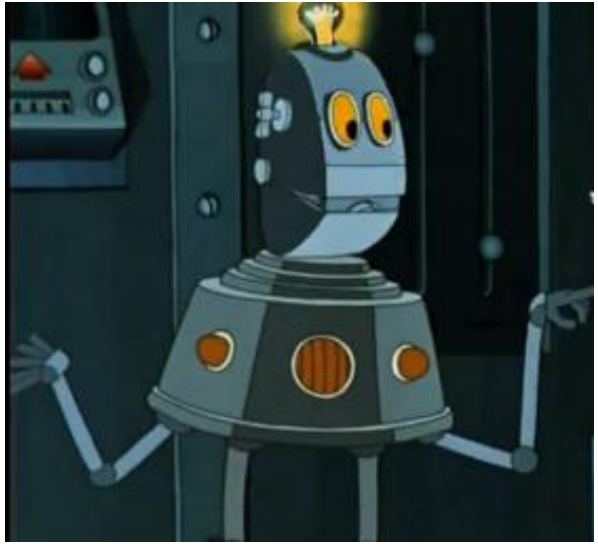
Jigsaw

- Динамических обновлений нет
- Версионирования нет

Jigsaw

- Динамических обновлений нет
- Версионирования нет

А что тогда есть?



EXCELSIOR

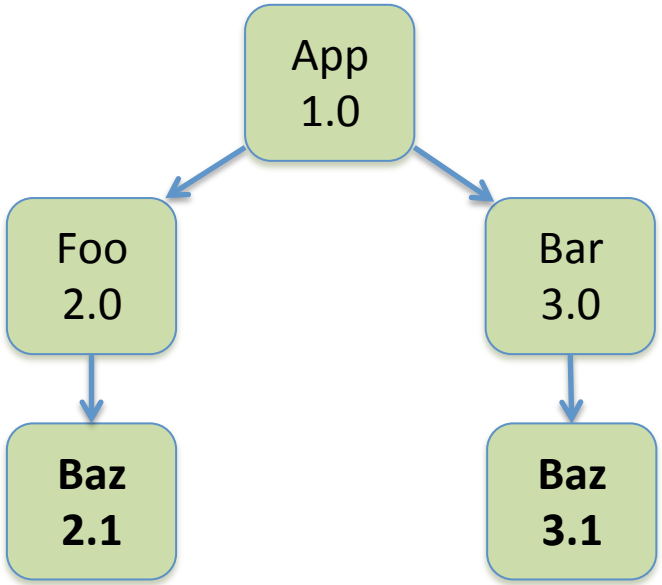
Jigsaw Mantra

Reliable Configuration

Strong Encapsulation

EXCELSIOR

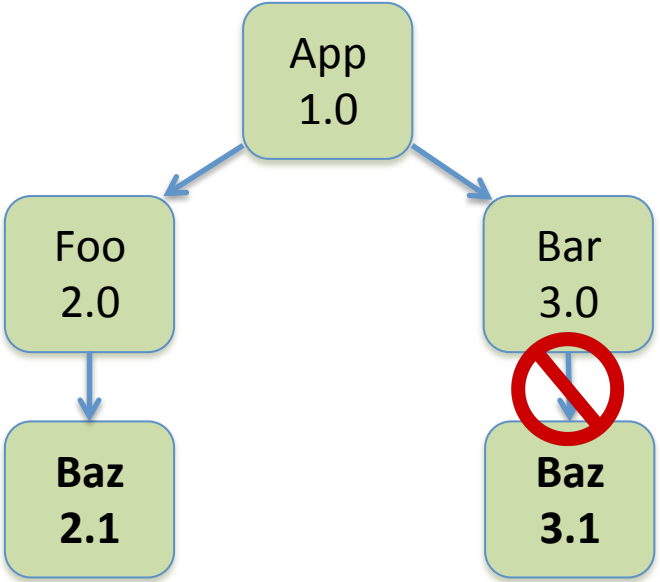
Reliable Configuration



EXCELSIOR

Reliable Configuration

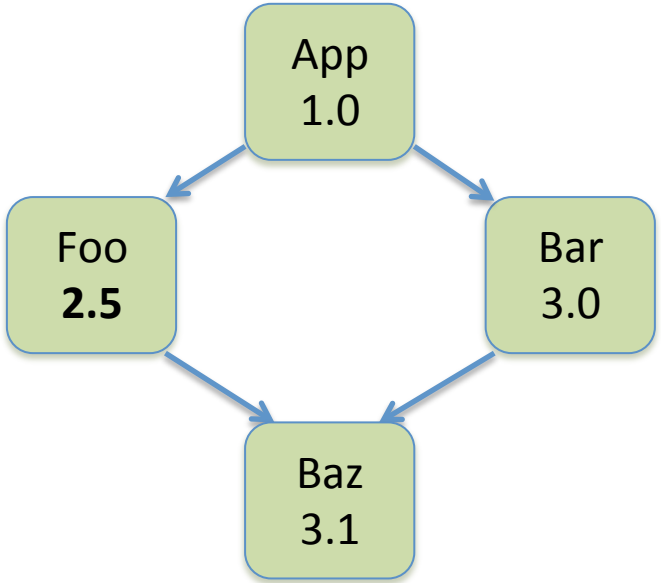
Такая ситуация в Jigsaw просто **запрещена**!



EXCELSIOR

Reliable Configuration

Правильная (**reliable**) конфигурация:



EXCELSIOR

Reliable Configuration

Reliable configuration это:

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

Свойства проверяются на старте (wiring)

EXCELSIOR

Strong Encapsulation

EXCELSIOR

Strong Encapsulation

Модули в Java 9 – это first class citizens

EXCELSIOR

Strong Encapsulation

Модули в Java 9 – это first class citizens

- Определяют области видимости
  - Через декларируемый экспорт

EXCELSIOR

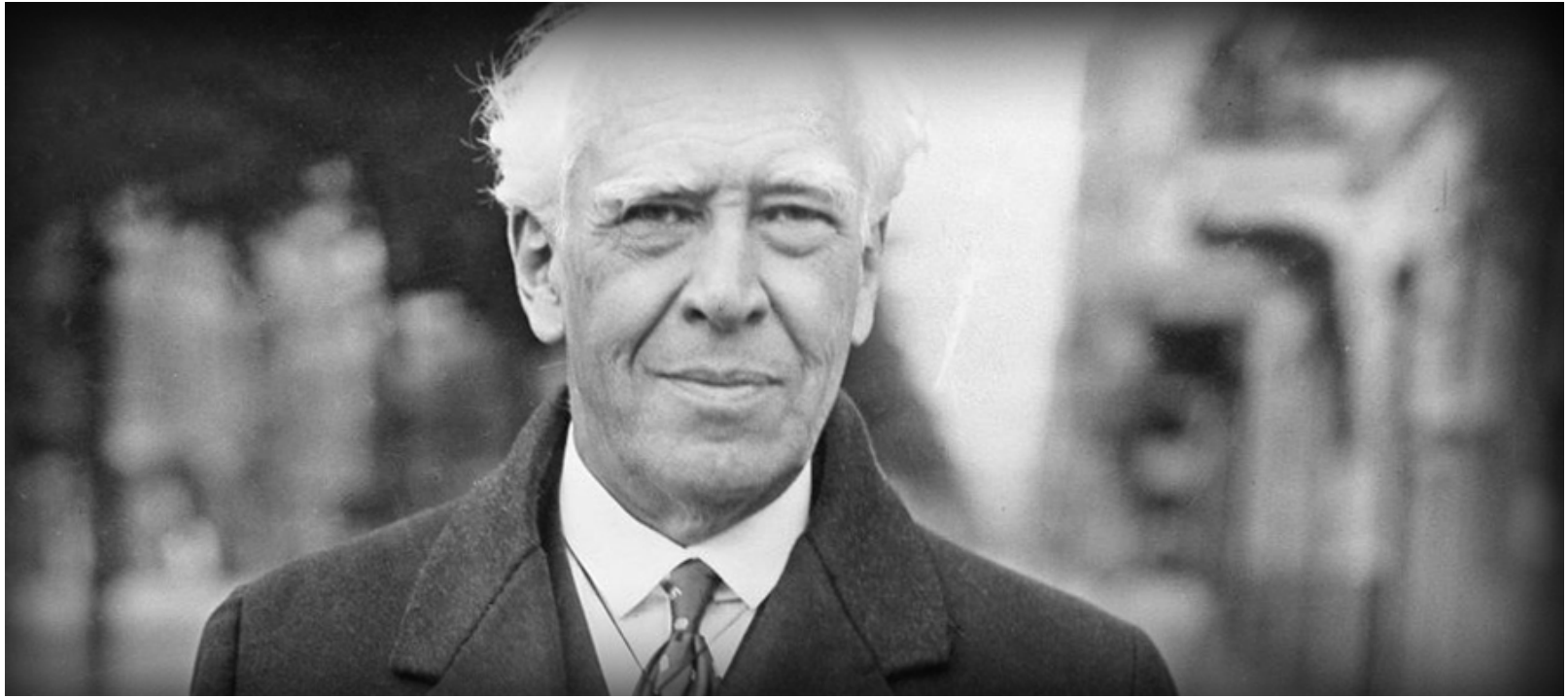
Strong Encapsulation

Модули в Java 9 – это first class citizens

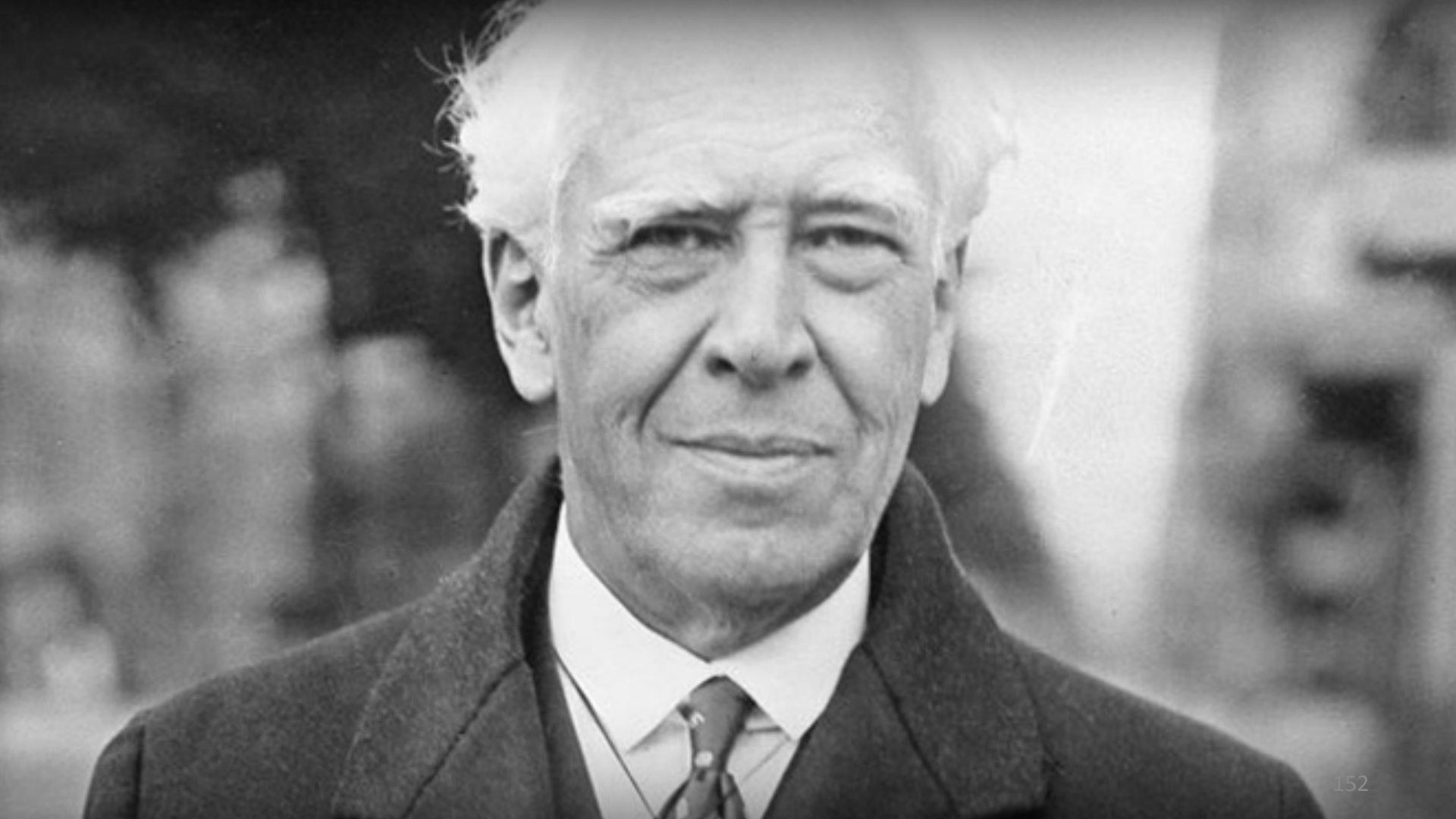
- Определяют области видимости
  - Через декларируемый экспорт
- Доступа к не экспортируемой функциональности из вне модуля нет даже через reflection
  - Не работает даже **setAccessible(true)**

EXCELSIOR

И что, у Jigsaw нет проблем?



EXCELSIOR



Reliable Configuration?

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

EXCELSIOR

Reliable Configuration?

- В ранних версиях Jigsaw рефлективный 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

EXCELSIOR

Reliable Configuration?

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

EXCELSIOR

Reliable Configuration?

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

EXCELSIOR

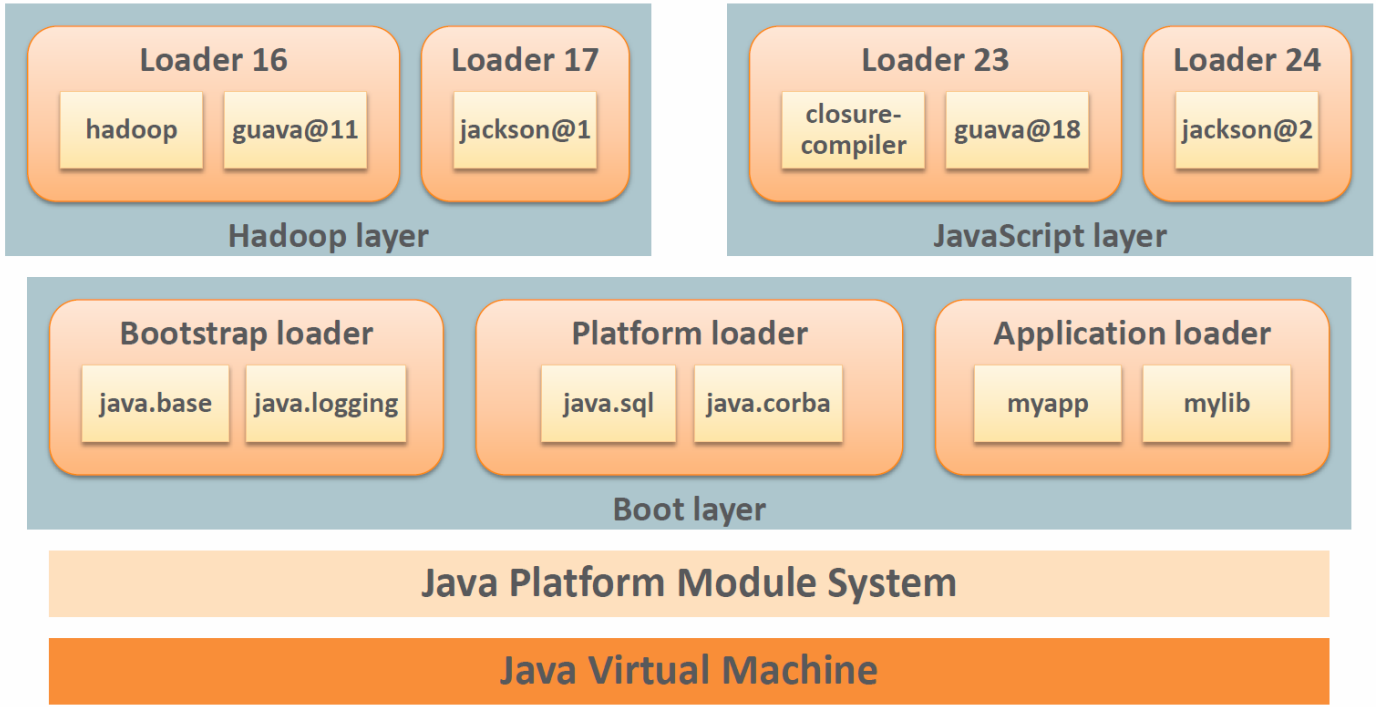
Jigsaw Layers

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

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

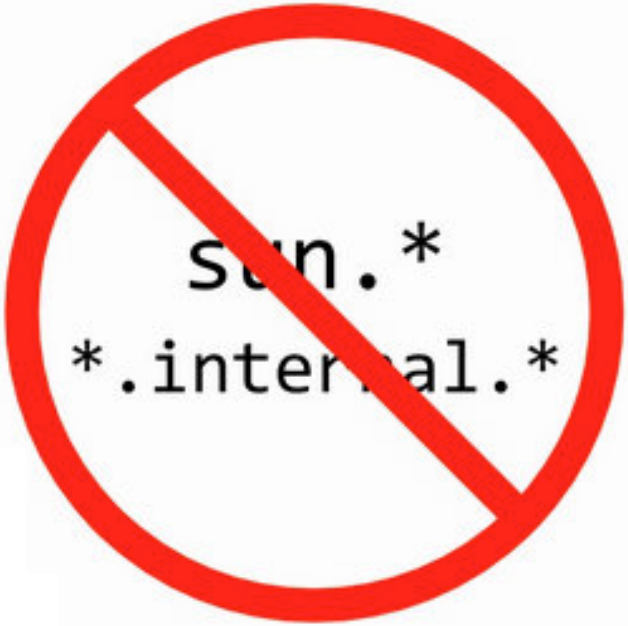
EXCELSIOR

Jigsaw Layers *



* Картинка позаимствована из презентации Alex Buckley: Project Jigsaw Under the hood

Strong Encapsulation?



EXCELSIOR

Strong Encapsulation?

Платформа разбита на модули:

- Что означает, что приватные API действительно стали приватными

EXCELSIOR

Strong Encapsulation?

Платформа разбита на модули:

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

sun.misc.Unsafe ?

EXCELSIOR

Strong Encapsulation?

Ok, **sun.misc.Unsafe** Java сообщество

(в неравной борьбе) **отстояло**!

(не надолго)

EXCELSIOR

Strong Encapsulation?

А как же Dependency Injection?

Strong Encapsulation?

DI frameworks существенно зависят:

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

EXCELSIOR

Strong Encapsulation?

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

EXCELSIOR

Strong Encapsulation?

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

EXCELSIOR

Strong Encapsulation?

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

EXCELSIOR

Strong Encapsulation?

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

EXCELSIOR

Strong Encapsulation?

Для DI frameworks введены ***open modules***:

- open module позволяет рефлективный доступ к своей неэкспортированной функциональности

EXCELSIOR

Strong Encapsulation?

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

EXCELSIOR

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

EXCELSIOR

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)

EXCELSIOR

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

EXCELSIOR

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

EXCELSIOR

Польза Jigsaw

Для миграции на modulepath в Jigsaw придумана целая система:

EXCELSIOR

Польза Jigsaw

Для миграции на modulepath в Jigsaw придумана целая система:

- Old Classpath образует **Unnamed Module**

EXCELSIOR

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**

EXCELSIOR

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

EXCELSIOR

Польза Jigsaw

Но, к сожалению, для большинства enterprise Java разработчиков польза от Jigsaw сомнительна:

- ~~Java EE~~ EE4J стандарты пока не определяют как они собираются взаимодействовать с модульной системой

EXCELSIOR

Польза Jigsaw

Но, к сожалению, для большинства enterprise Java разработчиков польза от Jigsaw сомнительна:

- ~~Java EE~~ EE4J 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!

EXCELSIOR

Jigsaw в enterprise

Тем не менее, для enterprise разработки Jigsaw можно уже сейчас использовать:

- В Tomcat и Jetty embedded и других фреймворках живущих на classpath (Play)
  - если все зависимости живут в classpath, то можно мигрировать на module path
- Можно форкнуть ваш любимый app server, добавив модульный слой в его загрузчик приложений
  - пример: https://github.com/pjBooms/tomcat

EXCELSIOR


Don't throw the baby out
with the bathwater!

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

EXCELSIOR

Вопросы и ответы

Никита Липский,
Excelsior

nlipsky@excelsior-usa.com
twitter: @pjBooms

Team blog: https://www.excelsiorjet.com/blog

EXCELSIOR