

Spring - Копаем до самого ядра

Евгений Борисов

bsevgeny@gmail.com

Ты кто такой?

jeka@inwhite.pro



@jekaborisov

Пишу стартап

Пишу курсы

Пишу код для JFrog-а

Синглтоны – не пишу

linkedin.com/in/evborisov

Терминология

- Апликация = приложение
- Айбернет = хибернет
- Штрудель =Собака
- Компонент – использую с любым ударением
- Параметр = Параметр
- Список пополняется...

О чём пойдёт речь: (часть первая)

- Разные дизайн патерны
- Reflection – это круто
- Spring – основные концепции:
 - Inversion of control
 - Dependency injection
 - Spring Bean
 - BeanFactory
- Разные виды контекстов

О чём пойдёт речь: (часть вторая)

- BeanPostProcessors
 - 4 уровня понимания
 - Написание BeanPostProcessora
 - Написание продвинутого BeanPostProcessora
- BeanFactoryPostProcessors
- Spring AOP
- ApplicationListener
- Что нового в Spring 4
- Spring JDBC

О чём пойдёт речь: (не уверен что успеем)

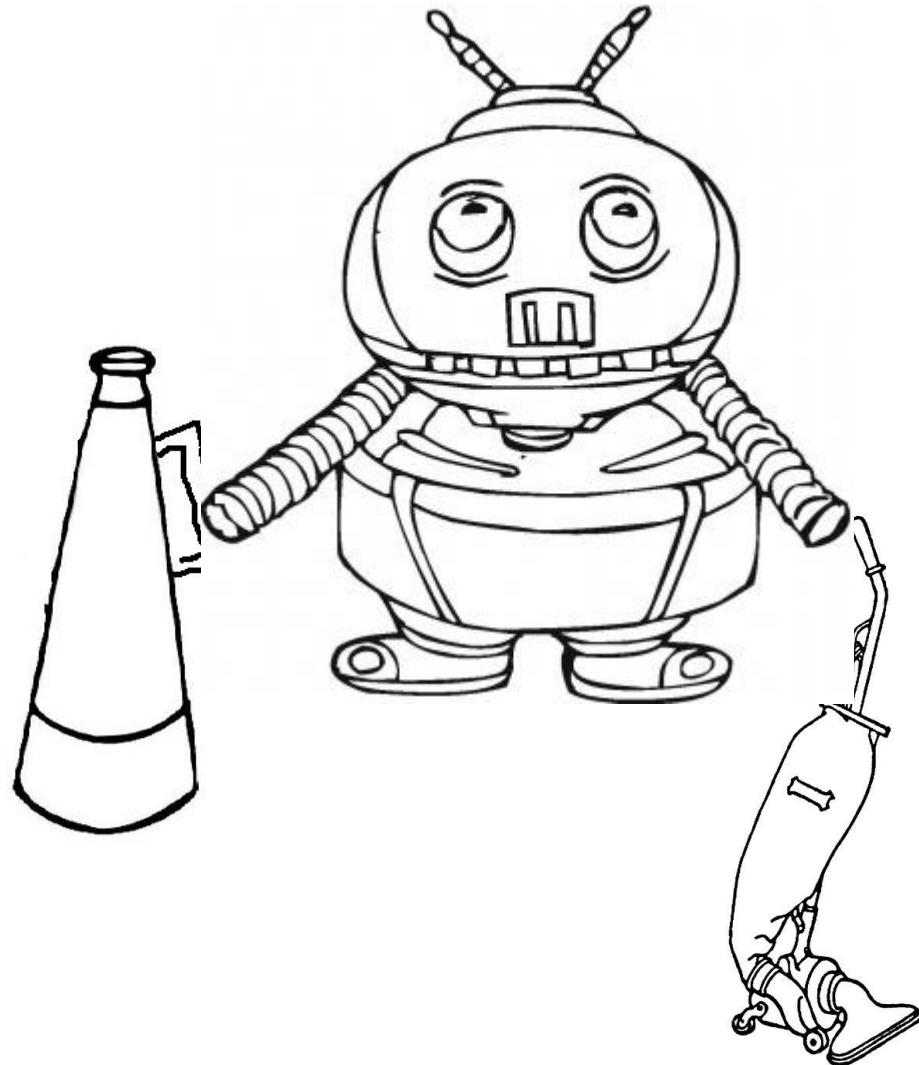
- Spring и валидации
- Spring + Hibernate
- Spring + RMI
- Spring + Quartz
- Spring MVC & Security

Как ТЫ создаешь объекты?

- Я пользуюсь: **new**
- Я очень крут, и использую только reflections
- Мне их создают
- Зачем объекты? Есть статические методы!



А чем плохо пользоваться **new?**

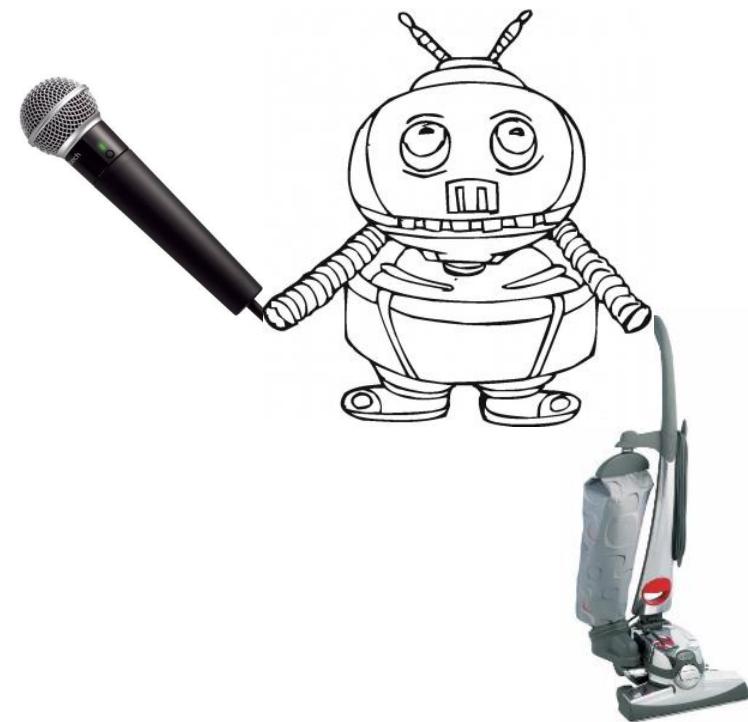


```
public class IRobot {  
    private VacuumCleaner vacuumCleaner = new VacuumCleaner();  
    private Speaker speaker = new Speaker();  
  
    public void cleanDustInTheRoom(RoomFrame room) {  
        speaker.sayJobStarted();  
        vacuumCleaner.suck();  
        room.makeRoomBrighter();  
        speaker.sayJobFinished();  
    }  
}
```

В чём тут проблема?

```
private VacuumCleaner vacuumCleaner = new VacuumCleaner();  
private Speaker speaker = new Speaker();
```

- А если надо поменять имплементацию,
надо код вскрывать, да?



Интерфейс лучше, правда же?

```
public interface Speaker {  
    void sayJobStarted();  
  
    void sayJobFinished();  
}
```

```
public class SimpleSpeaker implements Speaker {  
    public void sayJobStarted() {  
        System.out.println("Job started");  
    }  
  
    public void sayJobFinished() {  
        System.out.println("Job finished");  
    }  
}
```

```
public class AdvancedSpeaker implements Speaker {  
    public void sayJobStarted() {  
        JOptionPane.showMessageDialog(null, "Job started");  
    }  
  
    public void sayJobFinished() {  
        JOptionPane.showMessageDialog(null, "Job finished");  
    }  
}
```

А кто будет решать, какая имплементация?



А ты можешь написать сингальтон?

- Шесть фаз понимая сингальтона:



Фаза первая:
«Студент»



Фаза вторая: «Стажёр»

**А ЧТО С
МУЛЬТИРЕДИНГОМ?**

```
public class Singleton {  
    private static Singleton singleton;  
  
    private Singleton() {  
    }  
  
    public static Singleton getInstance() {  
        if (singleton==null) {  
            singleton = new Singleton();  
        }  
        return singleton;  
    }  
  
    // all business methods  
}
```

Фаза третья: «Junior Software Engineer»

**А что с
перформенсом?**

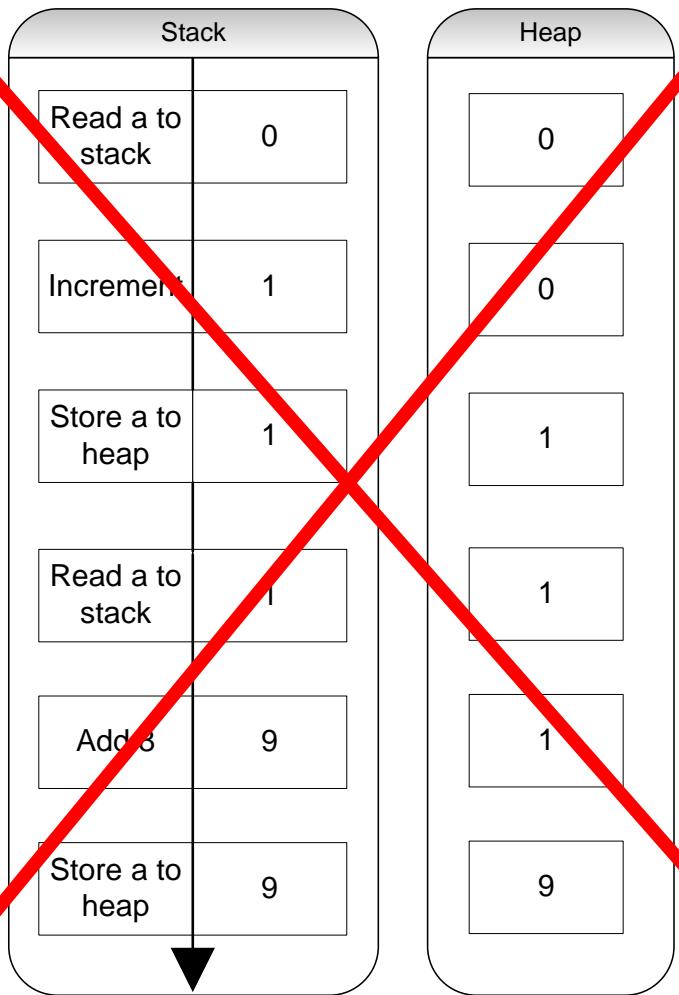
```
public class Singleton {  
    private static Singleton singleton;  
  
    private Singleton() {}  
  
    public synchronized static Singleton getInstance() {  
        if (singleton==null) {  
            singleton = new Singleton();  
        }  
        return singleton;  
    }  
  
    // all business methods  
}
```

Фаза четвертая: «Senior Software Engineer»

**А что на счёт
джава
оптимизаций?**

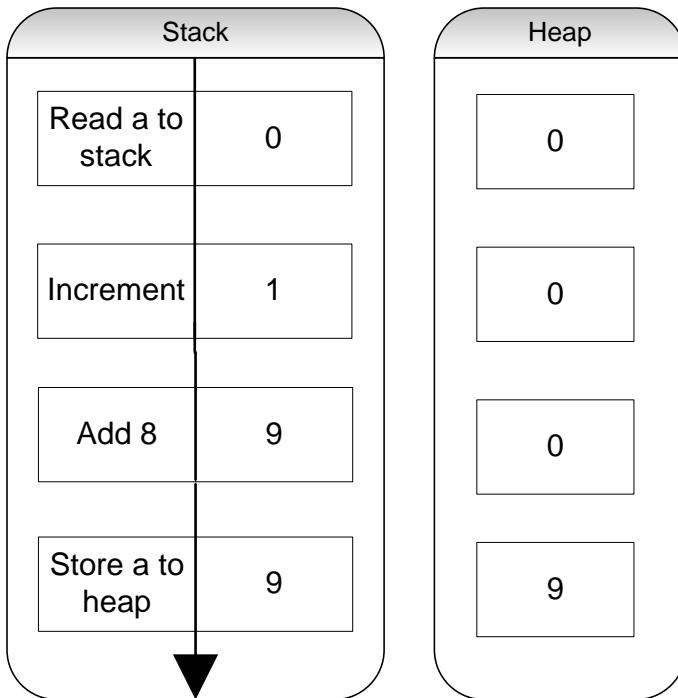
```
public class Singleton {  
    private static Singleton singleton;  
  
    private Singleton() {  
    }  
  
    public static Singleton getInstance() {  
        if (singleton == null) {  
            synchronized (Singleton.class) {  
                if (singleton == null) {  
                    singleton = new Singleton();  
                }  
            }  
        }  
        return singleton;  
    }  
  
    // all business methods  
}
```

Out-Of-Order Execution



```
1 public class OutOfOrder {  
2  
3     private int a;  
4  
5     public void foo() {  
6         a++;  
7         a+=8;  
8     }  
9  
10 }
```

Out-Of-Order Execution



```
1 public class OutOfOrder {  
2  
3     private int a;  
4  
5     public void foo() {  
6         a++;  
7         a+=8;  
8     }  
9  
10 }
```

Фаза четвертая: «Senior Software Engineer»

```
public class Singleton {  
    private static Singleton singleton;  
  
    private Singleton() {  
    }  
  
    public static Singleton getInstance() {  
        if (singleton == null) {  
            synchronized (Singleton.class) {  
                if (singleton == null) {  
                    singleton = new Singleton();  
                }  
            }  
        }  
        return singleton;  
    }  
  
    // all business methods  
}
```

Фаза пятая: «Lead Software Engineer»

```
public class Singleton {  
    private static volatile Singleton singleton;  
  
    private Singleton() {  
    }  
  
    public static Singleton getInstance() {  
        if (singleton == null) {  
            synchronized (Singleton.class) {  
                if (singleton == null) {  
                    singleton = new Singleton();  
                }  
            }  
        }  
        return singleton;  
    }  
  
    // all business methods  
}
```

Почему не надо писать синглтон

- Уверенность в завтрашнем дне
- Пишите вашу бизнес логику, а не изобретайте колесо
- А как вы будете тестировать?

Шестая Фаза «Архитектор»

- Не надо писать сингальтоны, для этого есть спринг.



Что нам даёт Spring?

- Много дизайн патернов из коробки
 - Strategy, Factory, Singleton, Proxy
- Dependency injection
- Можно тестировать без сервера приложений
- Интеграция с огромным количеством технологий
 - Hibernate, Quartz, JMS, RMI, WEB...
- AOP
- Спринг матёный

Немного истории

- В то время, когда программисты всё делали вручную и проклинали EJB 2 и J2EE 1.3
- В те далёкие годы, когда EJB ещё писались при помощи 100 500 интерфейсов и 100 500 XML
- Когда ещё в джаве не было аннотаций
- Короче в 2002 году

Немного истории

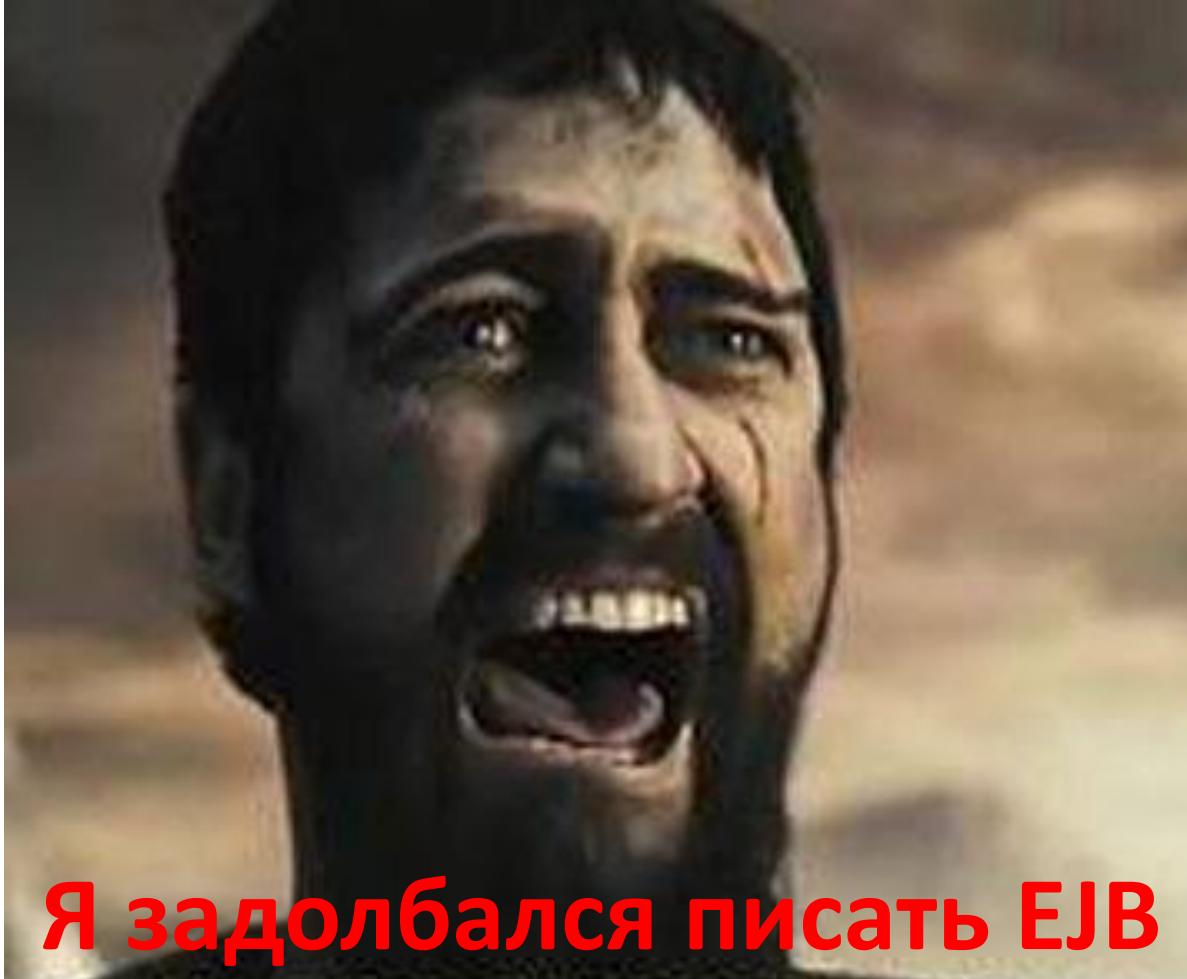
- Первый релиз спринга в 2002 году
- Spring 1.0 вышел в марте 2004 года
- В конце 2005 спринг становится лидерующей J2EE платформой
- В августе 2009 Vmware покупает спринг
- В 2013 ее передают в pivotal – и дают им возможность развернуться

Что сегодня

- Spring 4.0 – вышел в середине декабря, с опозданием на неделю
- Настоящая версия 5.0.8
- Для тех кто на 4 спринге
- Spring 4.3.18

Философия спринга

- Или о чём думал Род Джонсон в 2002 году



Я задолбался писать ЕЖВ

Inversion of control



Dependency injection



J2EE должен быть более дружелюбным



**ДОПОЙ СНЕCKED
EXCEPTIONS**



Еще философия спринга

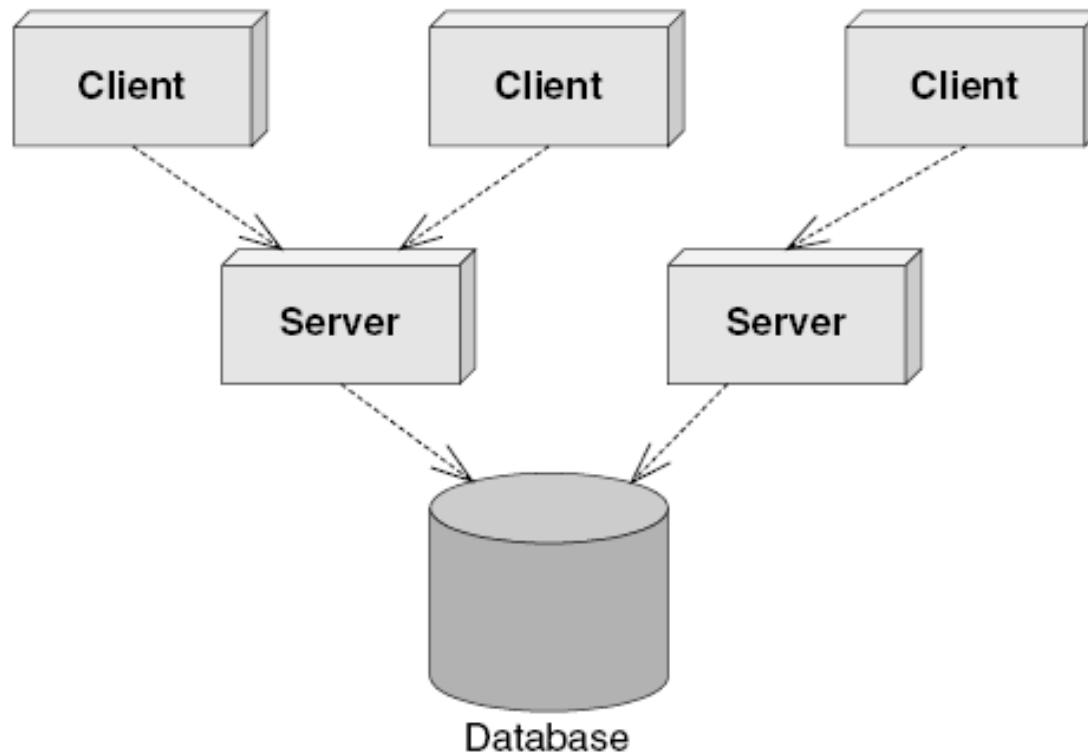
- Работать надо против интерфейсов, имплементация меняется
- В джаве есть много конвенций
- Объектно ориентированное программирование – наше всё!
- Тесты – это очень важно

Spring Framework это контейнер

Что такое контейнеры

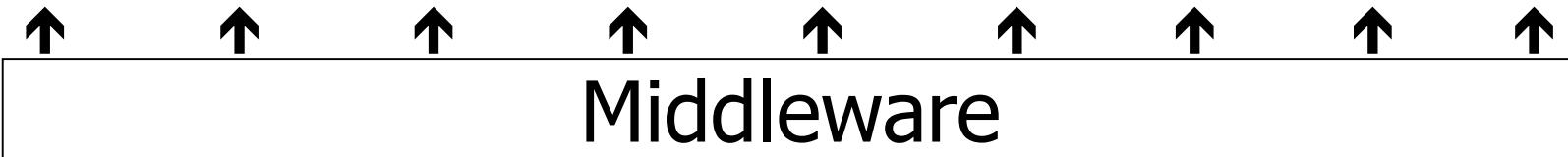
И зачем

Distributed applications



Наши заботы

- Remote method invocations
- Load balancing
- Transparent fail-over
- Back-end integration
- Transactions
- Clustering
- Dynamic redeployment
- Clean shutdown
- Logging and auditing
- Systems management
- Threading
- Message-oriented middleware
- Object life cycle
- Resource pooling
- Security
- Caching
- И т.д.



Эволюция middleware

- В прошлом организации писали свой
- Проблема в том, что он сложный
- Хорошие новости – он у всех одинаковый
- Так зачем писать, когда можно «купить»?

Application Server – Middleware из коробки

- Каждый делает то, что умеет
- Контора, которая специализируется на middleware пишет только его
- Остальные пишут только свою логику
- Разделяй и властвуй в хорошем смысле слова.

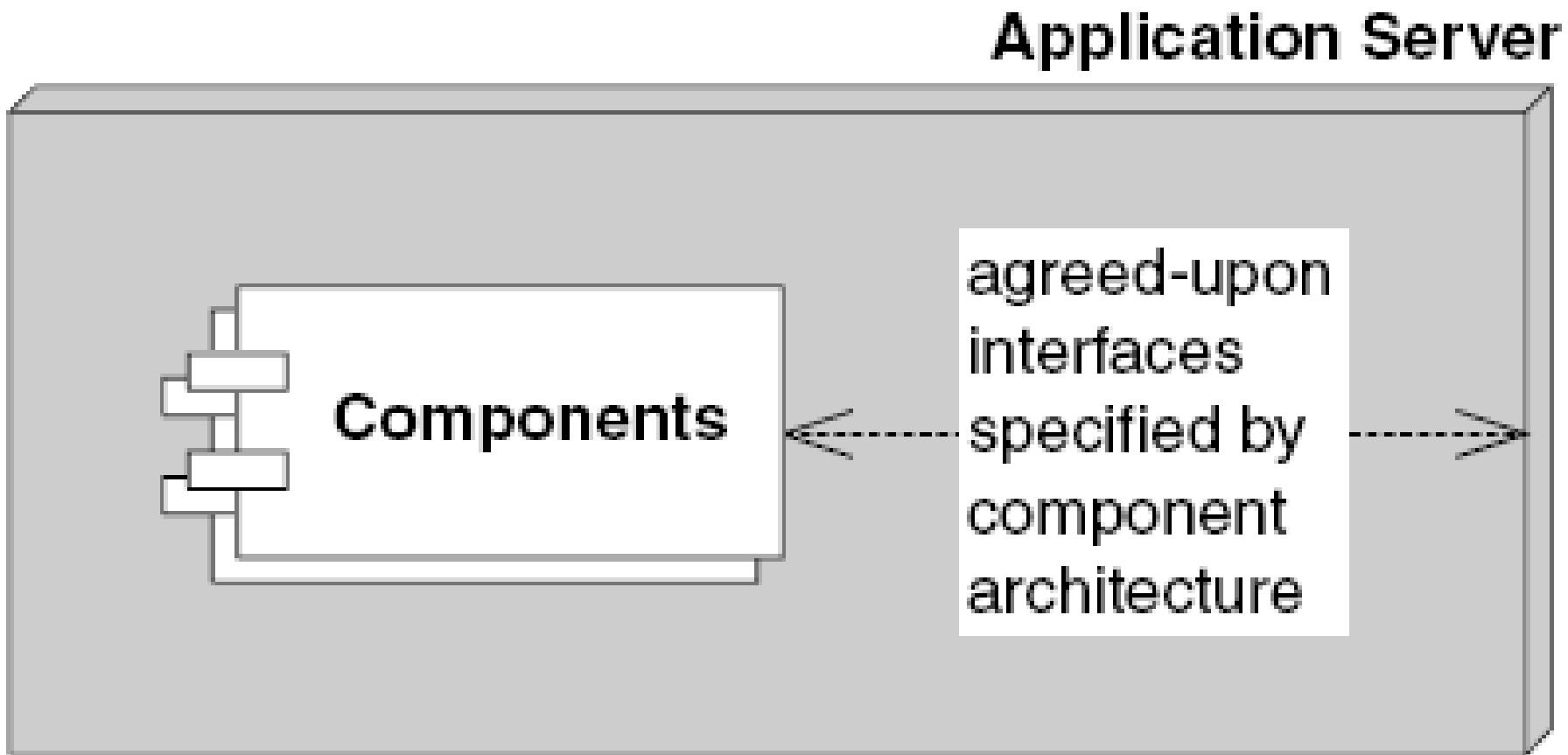
Примеры из жизни

- Это не новая концепция
- Например, DVD проигрыватель
 - Инфраструктура это проигрыватель (крутить диск, светить лазером, конвертировать сигнал)
 - Бизнес логика – кино, оно на диске
 - Много дисков используют ту же инфраструктуру
- Проигрыватель это контейнер для дисков
- Диск это компонент, который живет в конейнтер

Проблема стандартизации

- Несколько лет назад были blue-ray и hd-dvd (а до этого VHS и Betamax)
- Проигрыватели умели только одно, либо другое
- Контейнеры с проприетарными реализациями ограничивают пользователей в выборе
- Это называется lock-in
- Решение – убить конкурентов!
- Ну, или, спецификации

Спецификации контейнеров и компонентов



Явный вызов middleware

- Теперь мы знаем, что надо брать готовый middleware
- Вопрос, как его использовать?
- Явный вызов middleware делается через API:

```
transfer(Account account1, Account account2, long amount) {  
    // 1: Call middleware API to perform a security check  
    // 2: Call middleware API to start a transaction  
    // 3: Call middleware API to load rows from the database  
    // 4: Subtract the balance from one account, add to the other  
    // 5: Call middleware API to store rows in the database  
    // 6: Call middleware API to end the transaction  
}
```

Это имеет недостатки

- Тяжело писать
 - Много кода
 - Код не тот
- Тяжело содержать
 - Изменения в middleware требуют изменения в коде
- Тяжело поддерживать
 - Невозможно сконфигурировать middleware не имея доступа к исходному коду

Неявный (декларативный) вызов middleware

- Компоненты содержат только бизнес логику
- Требования к middleware описываются как metadata
 - Внешние файлы, например XML
 - Аннотации в коде
 - Соглашения
- Контейнер перехватывает вызов (*request interception*) бизнес логики, и впрыскивает (*injection*) вызовы middleware
 - До
 - После
 - Вместо

Это круто!

- Легко писать
 - Контейнер-перехватчик впрыскивает middleware прозрачно!
 - Разработка концентрируется на том, за что организации платят деньги
 - Еще больше разделяем, еще больше властствуем
- Легко содержать
 - Четкое разделение
 - Меньше и чище код
- Легко поддерживать
 - Конфигурация middleware не требует изменения (или даже знания) исходного кода
 - Обновление middleware больше не проблема

А можно примеры IoC?

- Applet, Midlet
- Servlets instantiation & life-cycle.
- EJBs life-cycle.
- Event-driven applications.
- **Dependency Injection.**

Dependency Injection

- Помогает решать проблему coupling-а
- Coupling – если модуль А связан с В то, А не может быть использован без В

```
public class IRobot {  
    private VacuumCleaner vacuumCleaner = new VacuumCleaner();  
    private Speaker speaker = new Speaker();  
  
    public void cleanDustInTheRoom(RoomFrame room) {  
        speaker.sayJobStarted();  
        vacuumCleaner.suck();  
        room.makeRoomBrighter();  
        speaker.sayJobFinished();  
    }  
}
```

Configure Spring Context with XML

```
</xml version="1.0" encoding="UTF-8">
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd">
```

```
</beans>
```

Что такое Бин?



Что такое Спринг Бин

- **SpringBeans** are reusable software components for Java.
- Короче любой класс, главное, чтобы не статические методы 😊
 - Может быть сервисом
 - Может быть POJO
 - Может быть нашим классом
 - Может быть 3-d party library

Определение бинов в xml-е

```
<!--Devices-->  
<bean id="speaker" class="devices.RussianSpeaker" scope="prototype"/>  
<bean id="vacuumCleaner" class="devices.VacuumCleaner" scope="prototype"/>
```

```
<bean class="robots.IRobot" name="iRobot">  
    <property name="speaker" ref="speaker"/>  
    <property name="vacuumCleaner" ref="vacuumCleaner"/>  
</bean>
```

Как дают название бину в xml-е

- Attribute ID
- Attribute name
- При помощи тага <alias> можно добавить ещё одно имя уже существующему бину
- `<alias name="fromName" alias="toName" />`

Важные вещи про базовые настройки бина

- Атрибут класс – получает полное имя класса (включая все пакэджи)
- По умолчанию scope = singleton (это не вам не guice)
- По умолчанию все сингальтоны создаются при поднятии контекста
- Это можно изменить в таге <beans...> default-lazy=true
- Или для конкретного бина: lazy=true

Как работает таг <property>?

- Property – подразумевает наличие сетора. (таковы конвенции)
- Зная названия property легко догодаться, как будет выглядеть settor: setPropertyName(...)
- При помощи reflection вызывается settor и в него передаётся то, что попросили

Стоп! А насколько хорошо вы знаете Reflection?



Опрос: ваш уровень знания Reflection

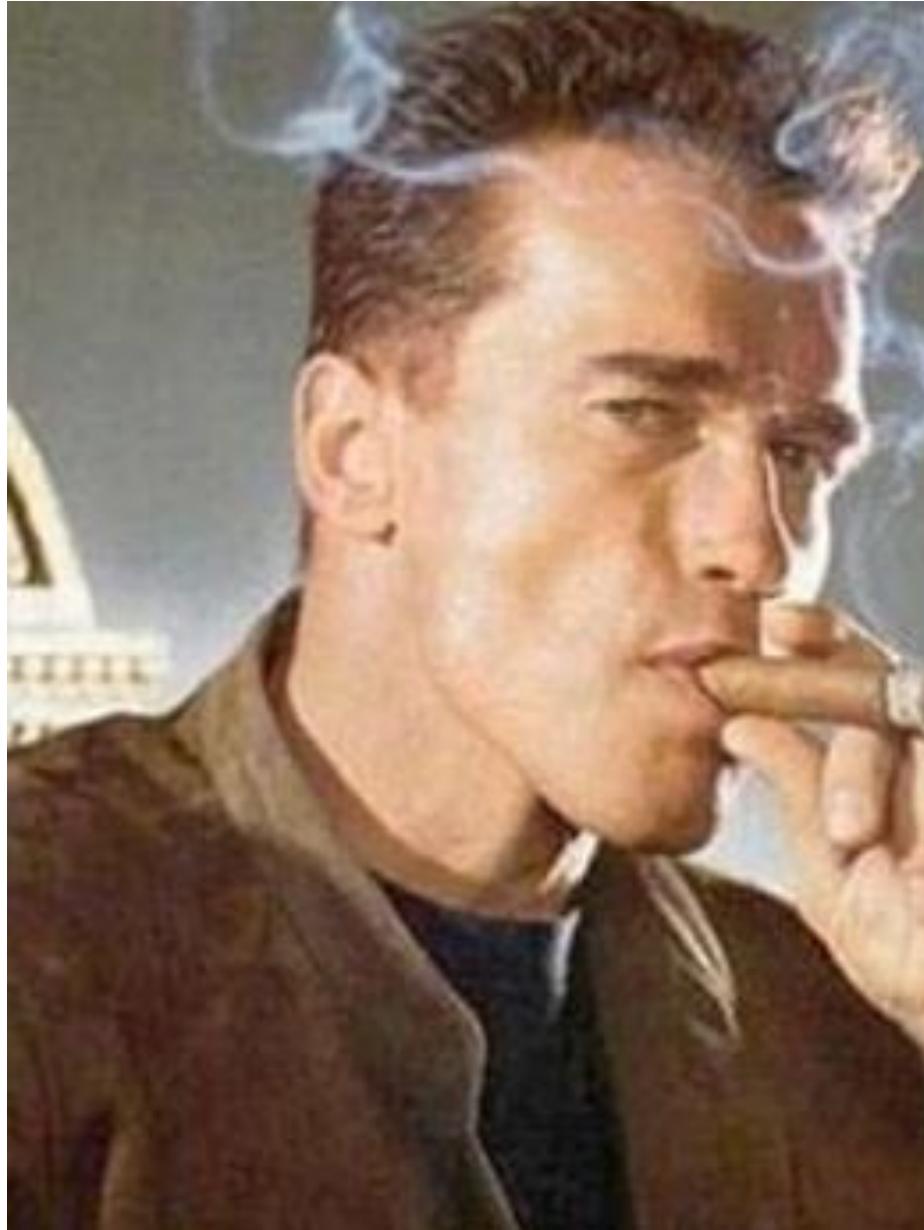


1. Я никогда не слышал, что такое reflection
2. Я знаю зачем он нужен
3. Я его использую
4. Я писал свои аннотации и считывал их в runtime
5. Я знаю разницу между RetentionPolicy.RUNTIME, RetentionPolicy.SOURCE и RetentionPolicy.CLASS

Задание

- Написать свою аннотацию, скажем `InjectRandomInt(min =4, max=9)`
- У неё должен быть параметр `repeat`
- Написать `ObjectFactory` с методом `createObject`
- Метод получает тип класса, и возвращает объект данного типа, но если в этом классе, есть филды, аннотированные `InjectRandomInt` `ObjectFactory` должна их настроить согласно `min` и `max`

Вот теперь мы знаем Reflection



Что можно впрыскивать в бины?

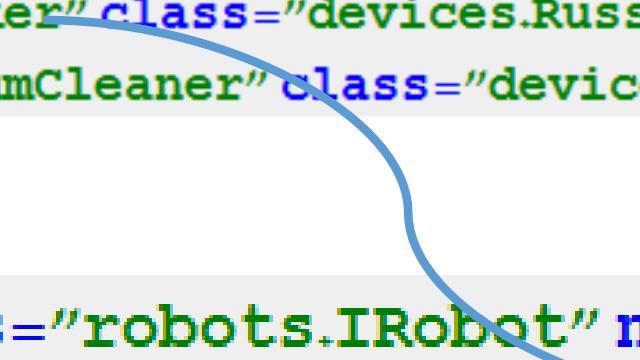
1. Другие бины при помощи атрибута **ref**
2. primitives and Strings. При помощи атрибута **value**
3. Collections.
 1. List
 2. Set
 3. Map

Впрыскивание value в бин

```
<bean class="quoters.Person">
    <property name="firstName" value="Jack"/>
    <property name="age" value="35"/>
</bean>
```

Впрыскивание бина в бин

```
<!--Devices-->  
<bean id="speaker" class="devices.RussianSpeaker" scope="prototype"/>  
<bean id="vacuumCleaner" class="devices.VacuumCleaner" scope="prototype"/>  
  
<bean class="robots.IRobot" name="iRobot">  
    <property name="speaker" ref="speaker"/>  
    <property name="vacuumCleaner" ref="vacuumCleaner"/>  
</bean>
```



Впрыскивание листа

```
<property name="messages">
    <list value-type="java.lang.String">
        <value>Shalom</value>
        <value>Privet</value>
        <value>Hi</value>
        <value>Guten Tag</value>
    </list>
</property>
```

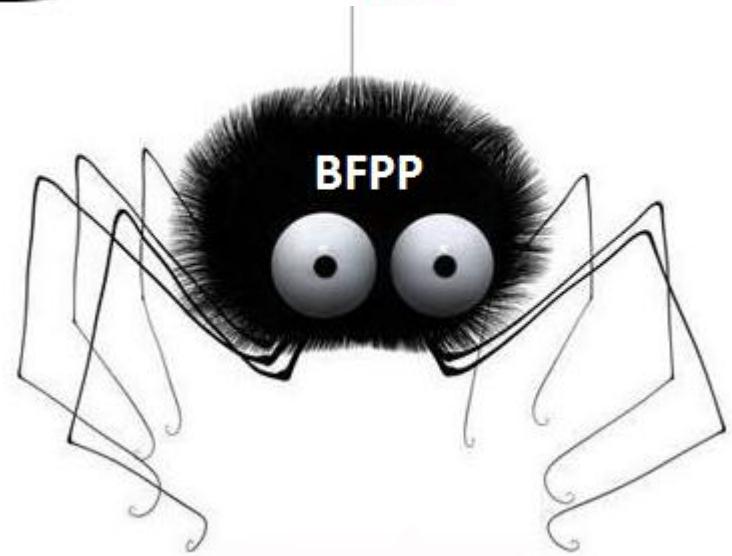
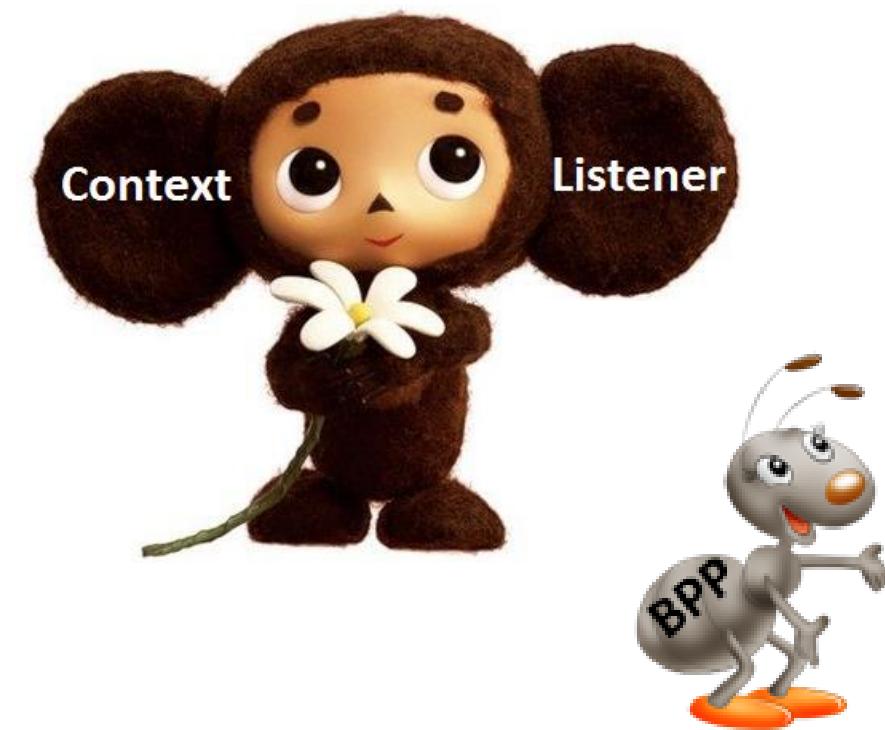
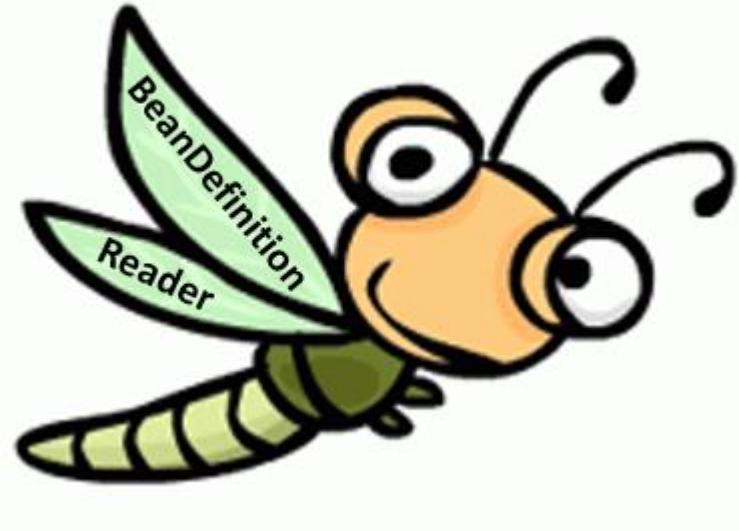
Как создать контекст Спринга

`new ClassPathXmlApplicationContext("context.xml")`

`new FileSystemXmlApplicationContext("context.xml")`

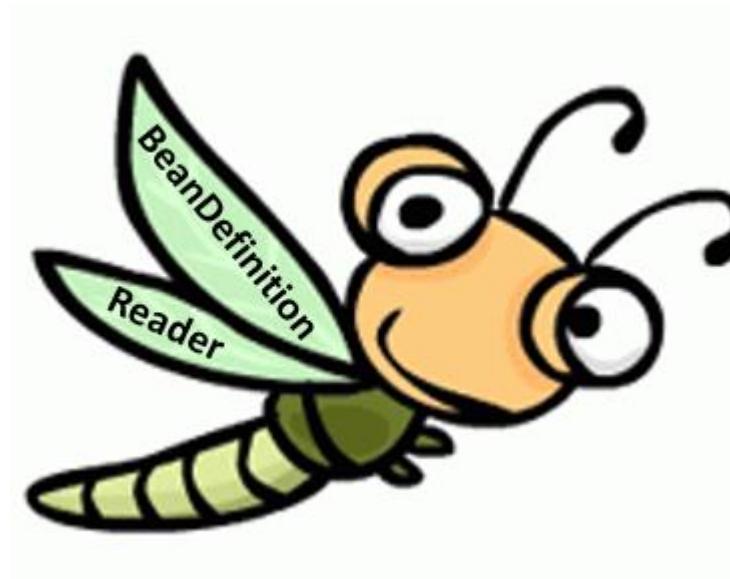
А как оно всё работает?

Спринг в картинках...

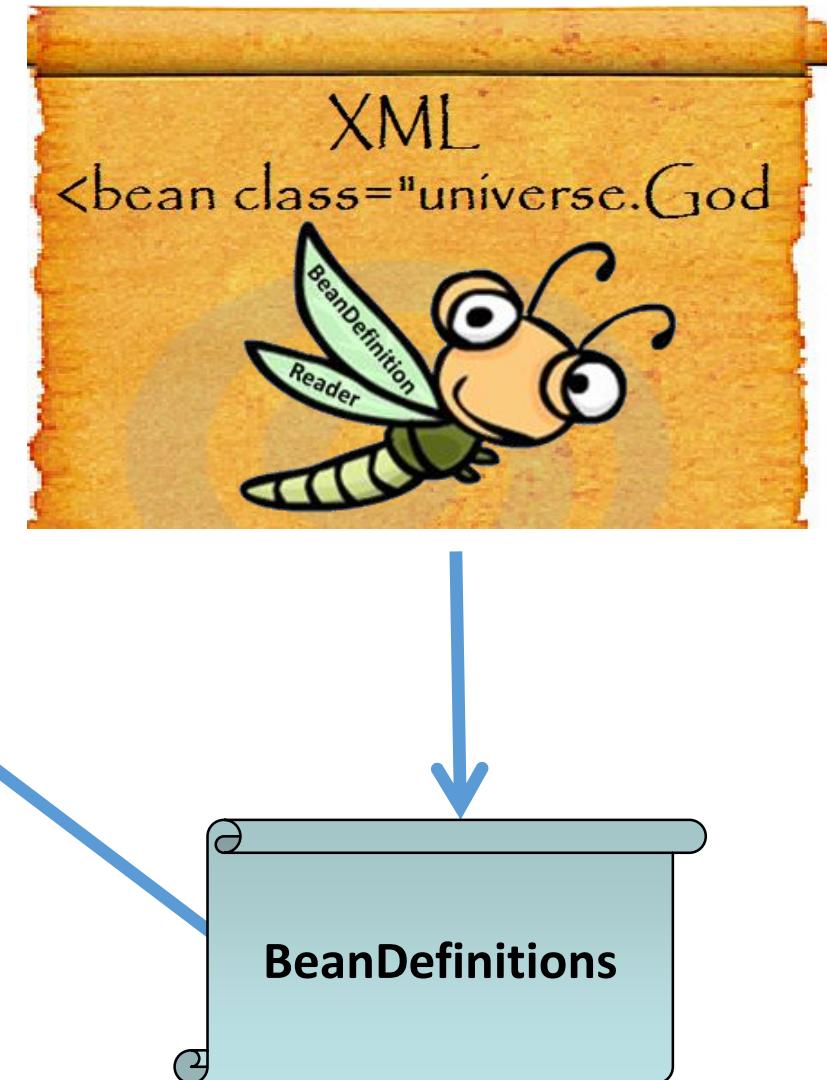
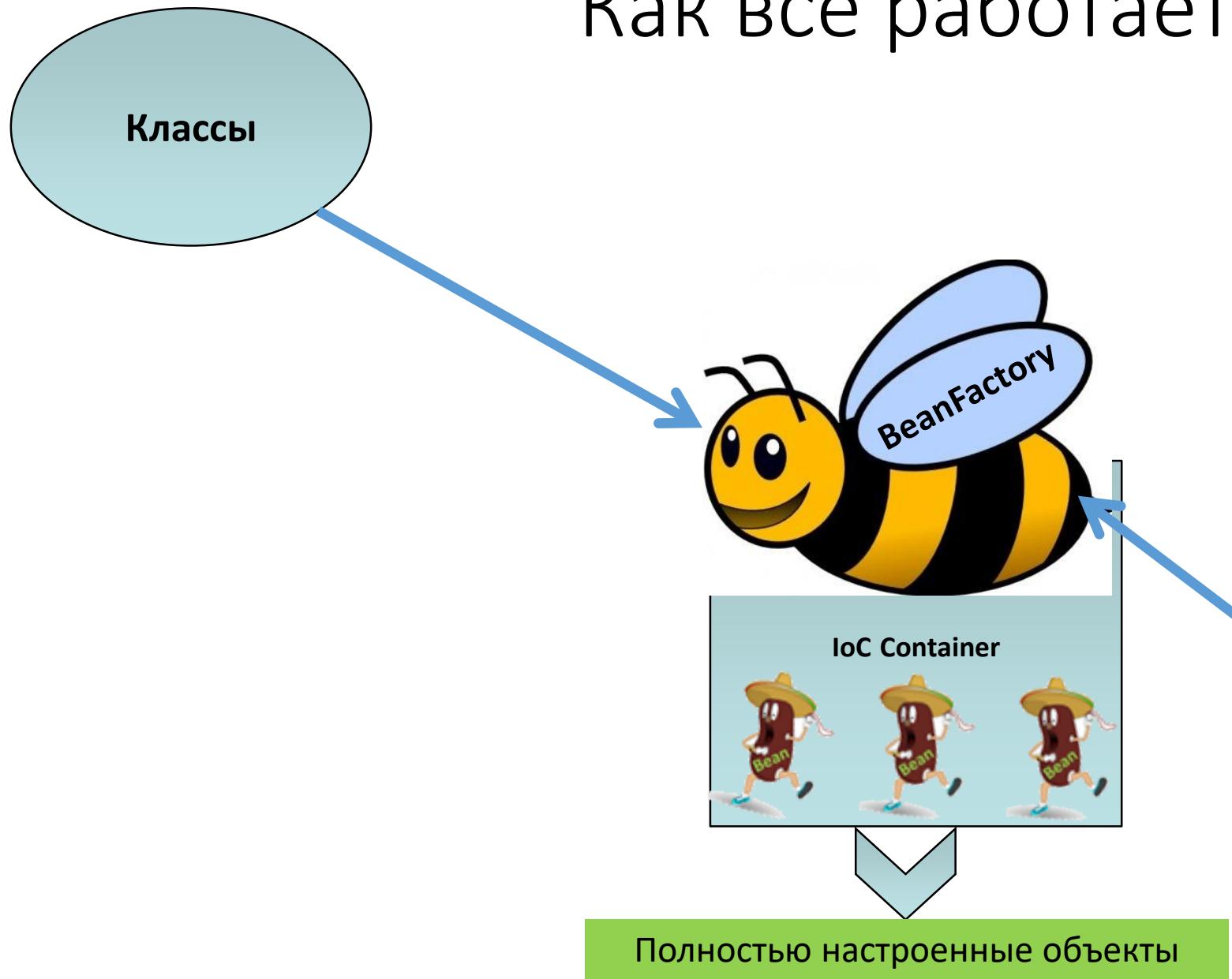


26.11.2003

XmlBeanDefinitionReader



Как всё работает



Как вытащить бин из контекста



Создание и использование контекста

```
ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext(xmlName);
IRobot iRobot = (IRobot) context.getBean("iRobot");
iRobot.cleanDustInTheRoom(room);
```

- Это называется Lookup
- Lookup можно делать по имени бина, по названию его класса, и по названию его интерфейса
- Его используют для тестирования
- Или для интеграции с легаси фреймворками, аля struts 1
- А как же в продакшоне?

Какие бывают приложения?

- Event Driven



1. Определить все бины
2. Впрыснуть все зависимости
3. Поднять контекст

- Scheduled



1. Определить все бины
2. Впрыснуть все зависимости
3. @Scheduled(cron = "1/2..")

Задание

- Написать interface Quoter с методом sayQuote и 2 имплементации
 - ShakeSpearQuoter (c property message) and TerminatorQuoter (messages)
- По вызову метода sayQuote они печатаю все меседжи
- Прописать их как бины в контексте и впрыснуть в xml-е им месседжи
- Протестировать
 - Создать контекст и сделать лукап
 - Попробовать сделать лукап не только по имени, но и по интерфэйсу.
 - Объяснить exception

Как настроить init method у бина?

- Если мы говорим про контекст xml то есть атрибут init-method
- Можно прописать его для конкретного бина в том же таге где он определяется
- Можно воспользоваться атрибутом default-init-methods для всех бинов, прописав этот метод в таге beans

Задание - Впрыснем Терминатора и Шекспира в третий бин



Задание

- Написать интерфейс с реализацией TalkingRobot у которого будет List<Quoter> и (прописать метод talk, чтобы он делегировал на все методы quote из всех Quoter-ов)
- Определить этот бин в xml-е и впрыснуть ему оба quoter-а
- Определить метод talk, как init-method
- В тесте только создавать контекст
- Изменить scope на prototype
- Почему тест перестал печатать?

Scope?!

- Scope это «срок годности» бина
- Singleton – как ясно из названия он может быть только один
- Prototype – наш XML definition используется как прототип для создания многих одинаковых бинов. Каждый запрос создает новый
- Есть ещё всякие, и можно даже создавать свои

Constructor Injection

- Таг у бина: <constructor-arg>
- Имеет атрибуты: type, index (начинается с 0), name и value

```
<bean class="heroes.Wizard">
    <constructor-arg index="1" value="Серый"/>
    <constructor-arg index="0" value="Гендальф"/>
</bean>
```

```
<constructor-arg value="12" type="int"></constructor-arg>
```

- А сейчас я покажу фокус

Задание

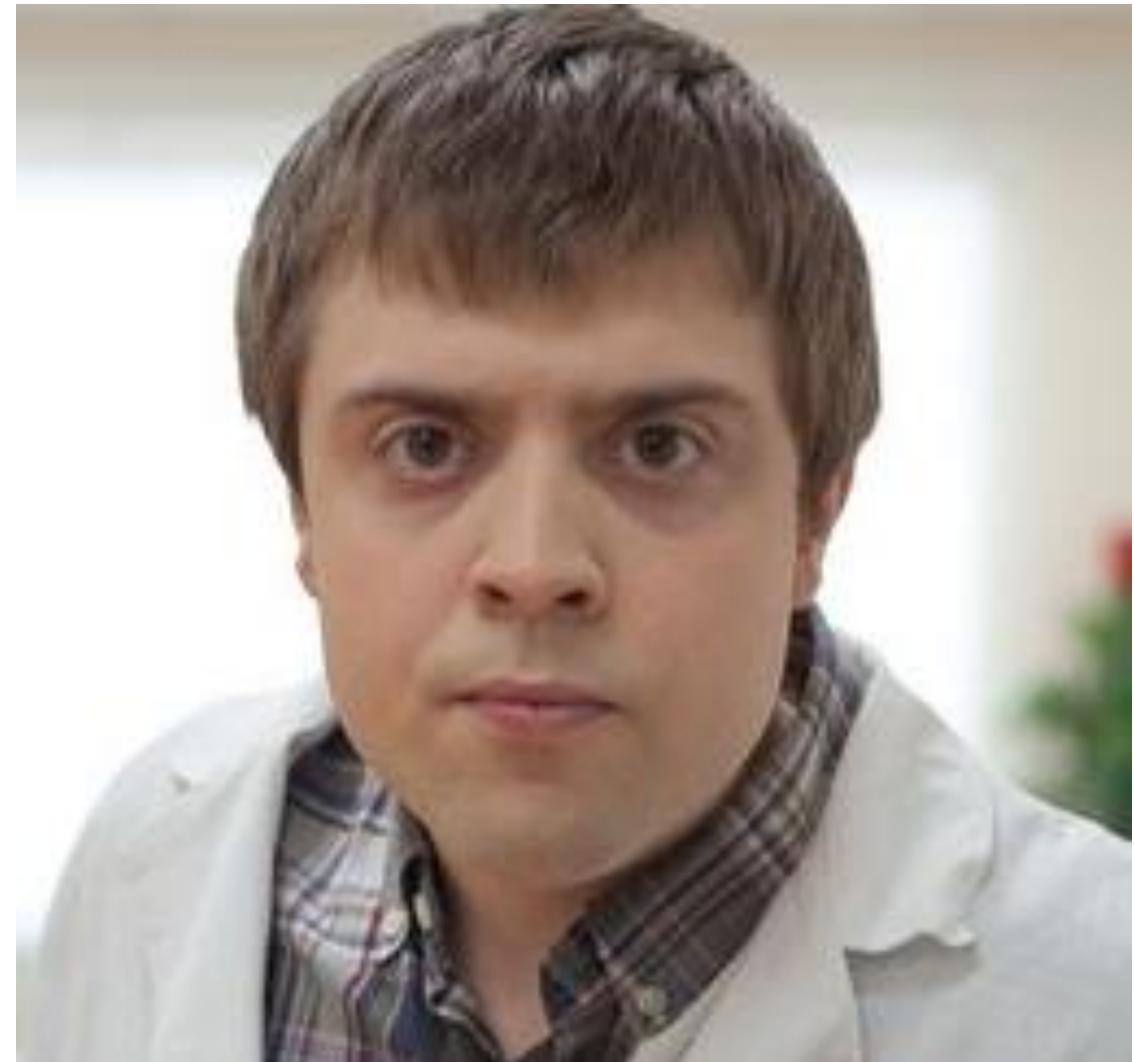
- Создать бин класса String, внести в него value (“trust me”)
- Инжектнуть его в лист цитат терминаатора

А как можно еще запускать init-method

- Init Method может определить тот, кто настраивает контекст
- А что, если это хочет сделать разработчик класса?
 1. Можно имплементировать интерфэйс InitializingBean, и прописать в нём метод: afterPropertiesSet
 - Но это было очень давно, когда ещё не было аннотаций
 2. Можно поставить @PostConstruct над нужным методом

У меня вопрос

1. А что будет если несколько @PostConstruct поставить?
2. А на хрена это вообще надо?
Ведь есть же конструктор, туда и надо фигачить инициализирующую логику



У меня загадка... Точнее две

```
public class Parent {  
    public Parent() {  
        printPi();  
    }  
}
```

```
public void printPi(){  
    System.out.println("Pi");  
}  
}
```

```
public static void main(String[] args) {  
    Son son = new Son();  
}
```

?

Answer:

0.0
3.141592653589793

Расположите в правильной последовательности

- @PostConstruct
- Spring setter (annotation) injection
- Son Initializer
- Parent Initializer
- Son inline
- Parent Inline
- Son Constructor
- Parent Constructor

Правильная последовательность

- Parent Inline / Parent Initializer (depends on order)
- Parent constructor
- Son Inline / Son Initializer (depends on order)
- Son constructor
- Spring setter (annotation) injection
- @PostConstruct Или init methods

Зачем это нужно знать???
Разве только для интервью!!!



Практическое применение этих знаний

```
public class BestService {  
    public BestService(){  
        chuckNorrisMethod();  
    }  
  
    public void chuckNorrisMethod() {  
        out.println("Save the world");  
    }  
}
```

```
public static void main(String[] args) {  
    new BetterThanBestService();  
}
```

```
public class BetterThanBestService extends BestService {  
    private List cache = new ArrayList();  
  
    @Override  
    public void chuckNorrisMethod() {  
        cache.add(1);  
    }  
}
```



1. Will not compile.
2. Runtime exception.
3. You can't override Chuck Norris methods.
4. Everything is ok.

А сейчас со спрингом

```
public class TerminatorQuoter implements Quoter {  
    private List<String> messages;  
  
    public TerminatorQuoter() {  
        sayQuote();  
    }  
    @Override  
    public void sayQuote() {  
        for (String message : messages) {  
            System.out.println(message);  
        }  
    }  
}
```

NullPointerException

А что будет сейчас?

```
public class TerminatorQuoter implements Quoter {  
    private List<String> messages = Arrays.asList("полюбому");  
  
    @Override  
    @PostConstruct  
    public void sayQuote() {  
        for (String message : messages) {  
            System.out.println(message);  
        }  
    }  
}
```

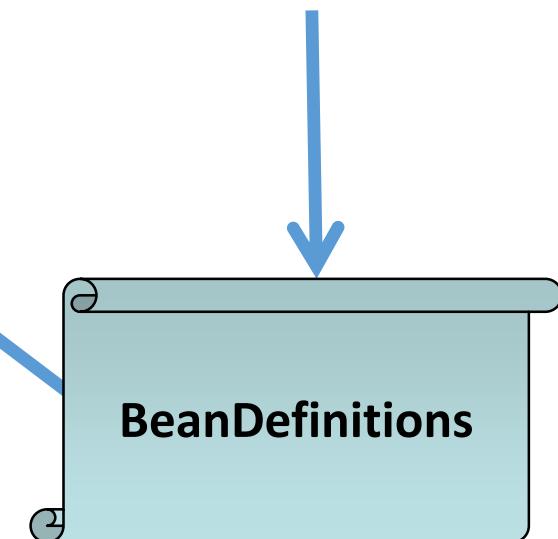
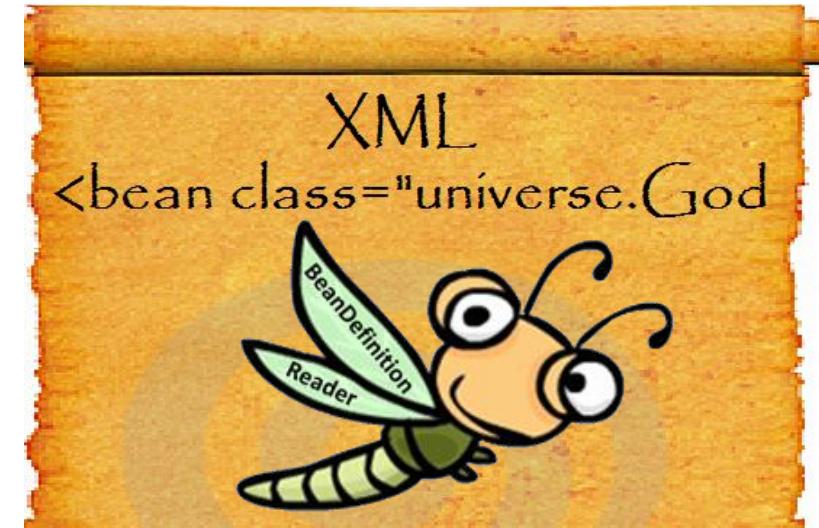
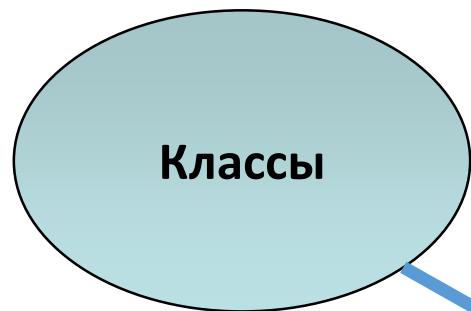
А теперь давайте попробуем @PostConstruct

- Почему не работает?
- XML Контекст по умолчанию не работает с аннотациями
- Надо вводить дополнительных игроков
- Добавте в xml бин из класса: CommonAnnotationBeanPostProcessor

4 уровня понимания BeanPostProcessor-ов

- Я могу это выговорить
- Я понимаю, как они работают
- Я могу написать свой простенький
- Я могу написать BeanPostProcessor который делает Proxy

Как всё работает



А как в это всё вписываются BeanPostProc?

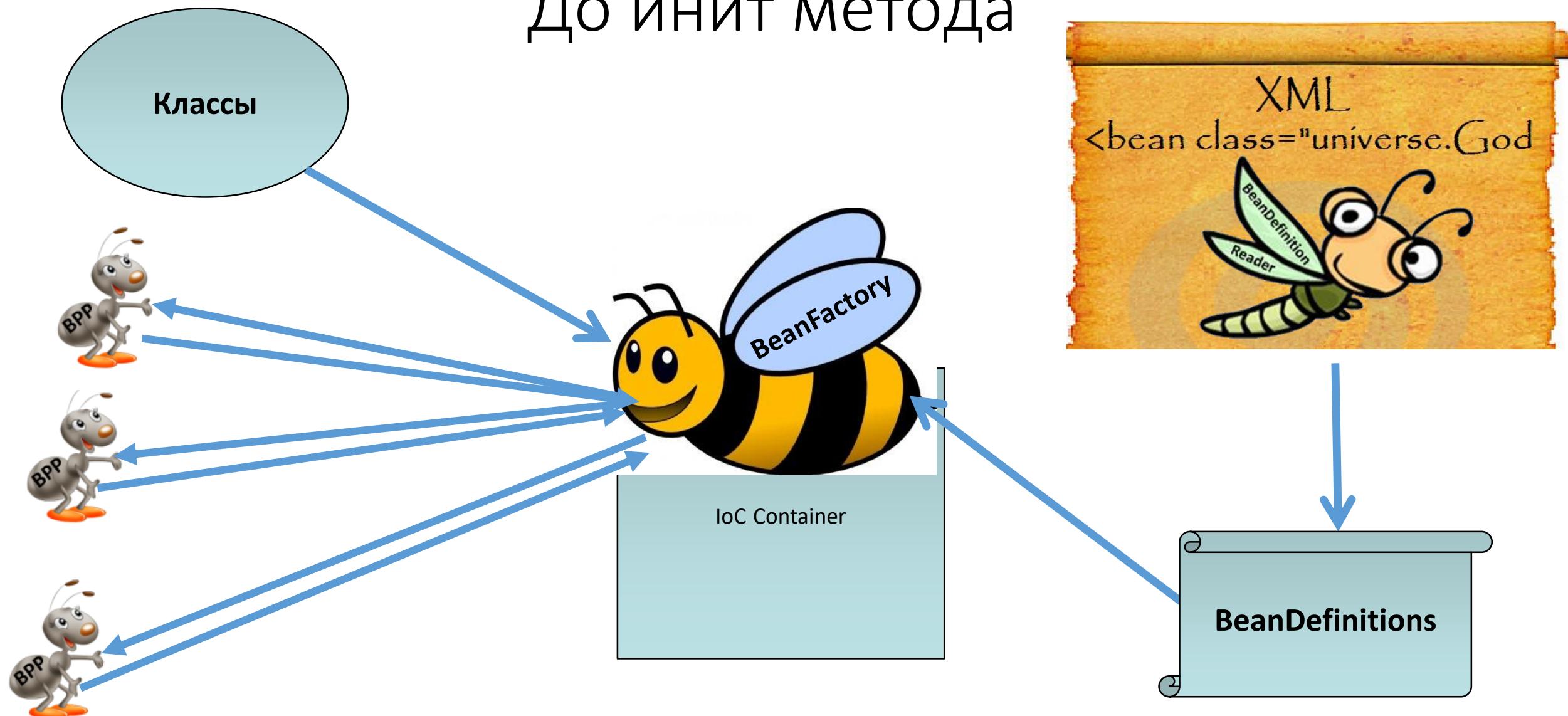
- BeanFactory создаёт объект и пихает их в контекст согласно метадате, которую предварительно парсирует BeanDefinitionDocumentReader
- Все бины, классы которых имплементируют BeanPostProcessor, будут являться не апликационными, а инфраструктурными
- Они помогают BeanFactory настраивать бины.
- Поэтому BeanFactory создаст их раньше 😊

Что за интерфейс такой BeanPostProcessor

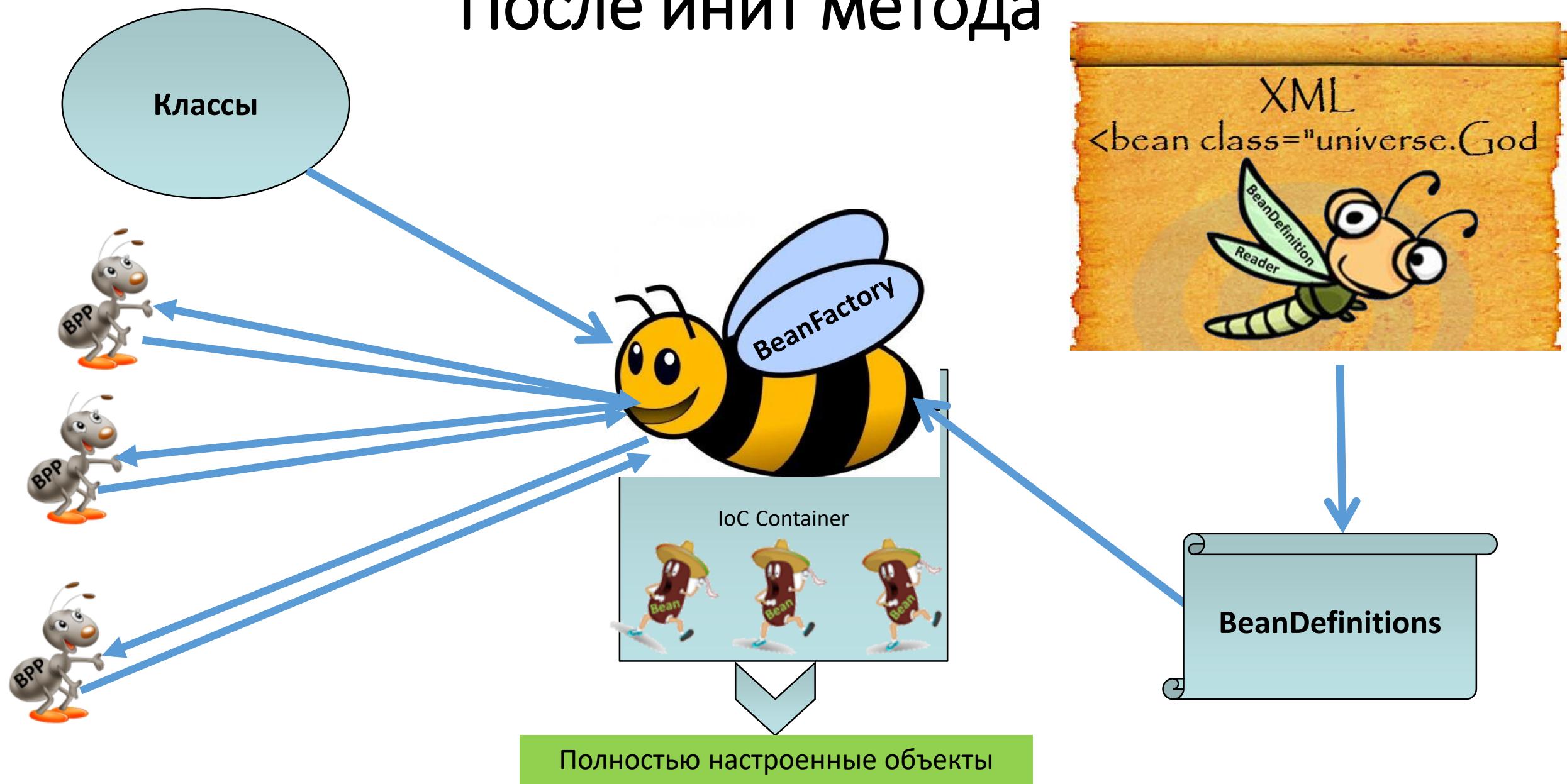
- Object postProcessBeforeInitialization(Object bean, String beanName)
- Object postProcessAfterInitialization(Object bean, String beanName)
- Первый работает до инит метода, а второй после



До инициализации



После инит метода



Задание

- Напишите свой BeanPostProcessor, который будет запускать методы аннотированные @RunThisMethod

Задание

- Напишите BeanPostProcessor который будет искать кастомную аннотацию @InjectRandomInt
- У этой аннотации будут параметры min & max

Задание

- Напишите свой BeanPostProcessor, который будет печатать сколько времени работал метода аннотированный @Benchmark
- Аннотация должна находится над классом – и все его методы должны быть завернуты в логику бенчмарка
- А теперь улучшим наш framework. Теперь аннотация должна считываться с метода, а не класса.

А как сделать динамический proxy?

- `java.lang.reflect.Proxy.newProxyInstance`
- CGLIB jar

Как работает dynamic proxy в Java

```
InvocationHandler handler = new MyInvocationHandler(...);
```

```
Foo f = (Foo) Proxy.newProxyInstance(
    Foo.class.getClassLoader(),
    new Class[] { Foo.class }, handler
);
```

java.lang.reflect.Proxy

- public static Class **getProxyClass**(ClassLoader loader, Class[] interfaces)
- public static Object **newProxyInstance**(ClassLoader loader, Class[] interfaces, InvocationHandler h)
- public static boolean **isProxyClass**(Class cl)

Interface InvocationHandler

Вызовы делегируются в *invocation handler*

public Object **invoke**(

Object proxy,

Method method,

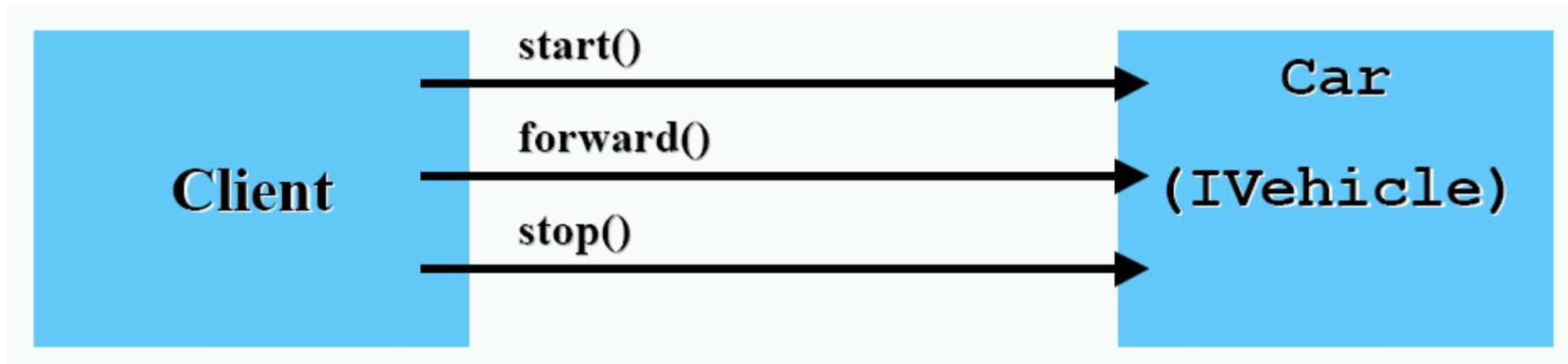
Object[] args

) **throws** Throwable

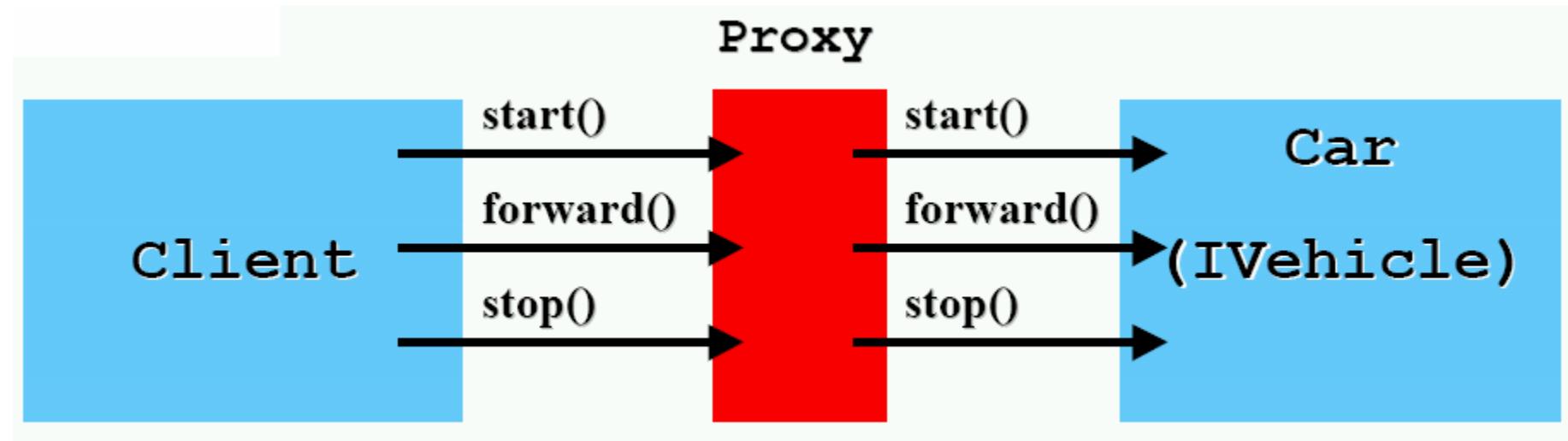
Proxy classes

- public, final, и не abstract.
- Имя неизвестно. Что-то там, начинающееся с "\$Proxy".
- Наследует от `java.lang.reflect.Proxy`.
- Реализует только те интерфейсы, которые были запрошены при создании

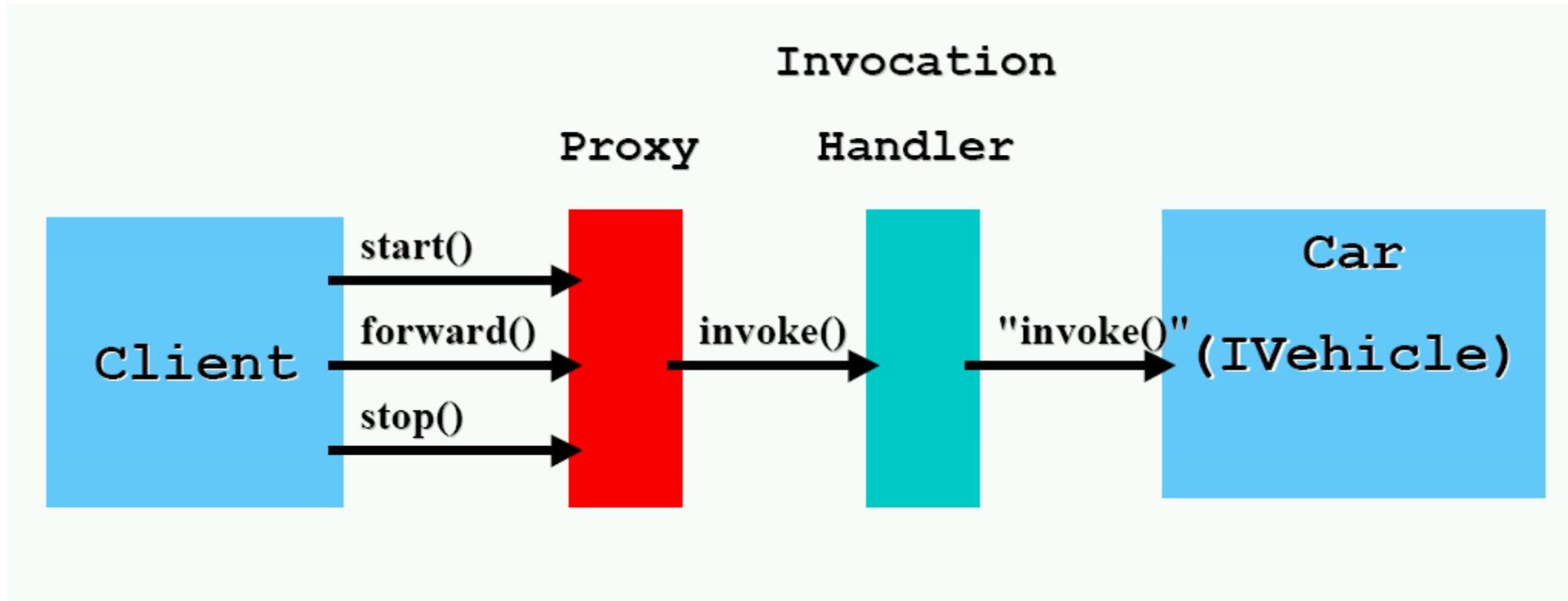
Example – Vehicle



Example – VehicleProxy



Example – Dynamic proxy



Очень сложное задание

- Напишите еще один BeanPostProcessor который будет делать прокси.
- Например которые обворачивает транзакцией все методы бинов, класс которых аннотирован @Transaction
- А теперь тестируем...



МОЖЕТ ОСТАТЬСЯ ТОЛЬКО



ОДИН БИНПОСТПРОЦЕССОР

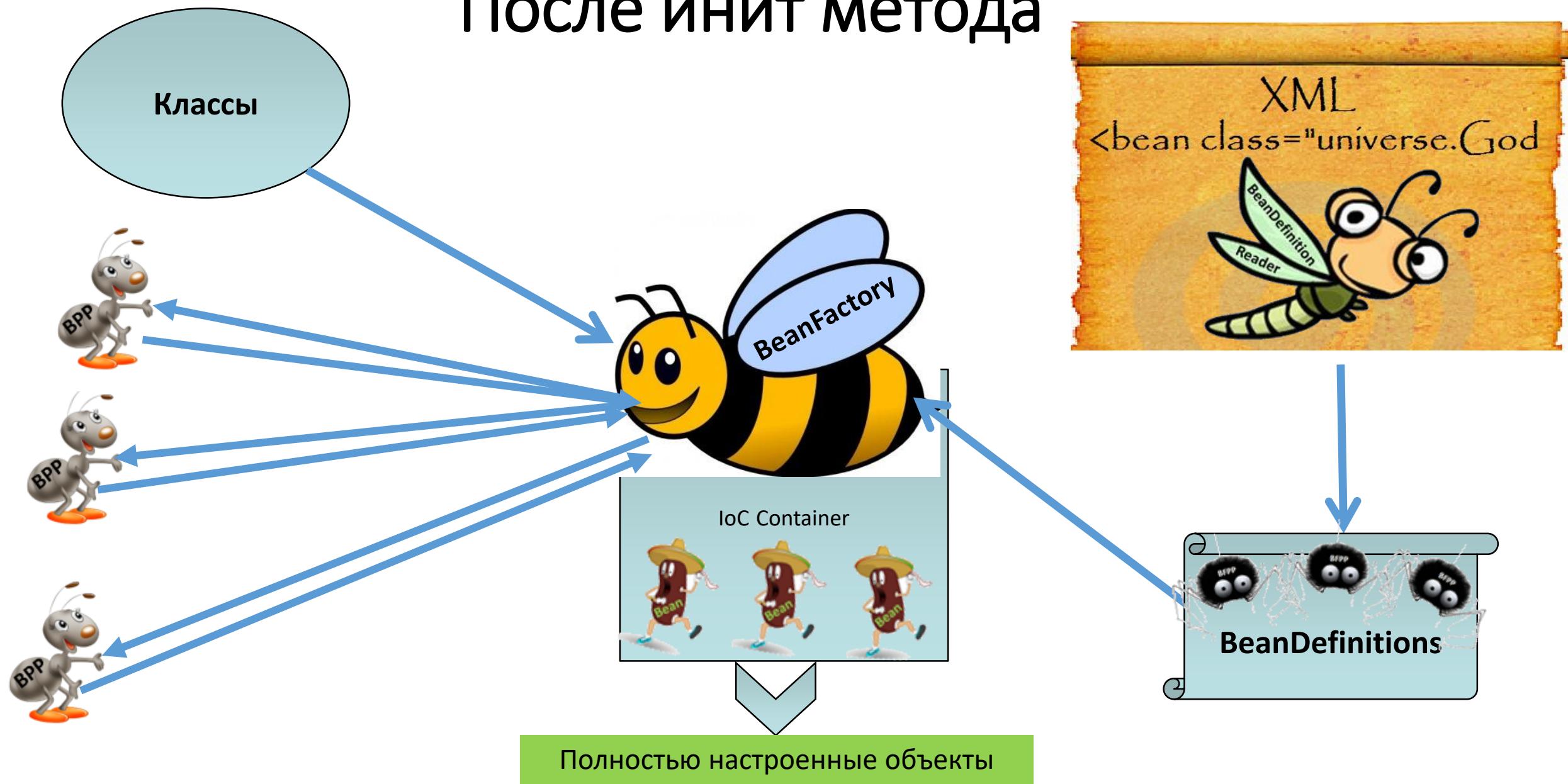
Destroy Method

- Finalize не для использования
- Где помещать логику которая закрывает все ресурсы?
- У каждого бина можно определить destroy method
- В xml-е у тага bin есть атрибут destroy-method
- В таге beans есть аттрибу default-destroy-methods
- Когда работают все destroy methods? Когда закрывается контекст
- Destroy method никогда не будет работать для прототайпов

Задание

- Прикрутите дополнительный компонент, который будет выводить лог названия всех `destroy` методов определенных для прототипов, с предупреждением, что они работать не будет

После инит метода



BeanFactoryPostProcessor

- Этот интерфэйс имеет один единственный метод:
- `postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory)`
- Этот метод запуститься на этапе, когда другие бины ещё не созданы, и есть только BeanDefinitions
- Зато их можно поменять или считать
- Теперь можно выполнить предыдущее задание

Задание

- Напишите собственную аннотацию, `@DeprecatedClass`.
- У этой аннотации будет параметр `newClass` указывающий на новую имплементацию
- Напишите `BeanFactoryPostProcessor`, который будет заменять все устаревшие классы на новые

Существующие BeanFactoryPostProcessor

- Если вы хотите впрыскивать проперти из проперти файлов:
- **<context:property-placeholder location="classpath:application.properties,**
- Где может находится считываемый ресурс:
 - classpath
 - file
 - http
- А как впрыскивать сами значения?

```
<bean class="properties.example.Login">
    <constructor-arg name="name" value="${name}" />
    <constructor-arg name="password" value="${password}" />
</bean>
```

Factory Beans

- Как сделать так, чтобы спринг создавал бины не при помощи конструктора класса, а при помощи кастомной логики?

FactoryBean interface

```
public interface FactoryBean<T> {  
  
    T getObject() throws Exception;  
  
    Class<?> getObjectType();  
  
    boolean isSingleton();  
}
```

Что вы думаете по этому поводу?

```
<bean id="guestroom" class="ui.RoomFrame" scope="prototype">
<property name="scuba" ref="scuba"></property>
<property name="IRobot" ref="iRobot"></property>
<property name="duster" ref="duster"></property>
<constructor-arg value="Guest Room"></constructor-arg>
</bean>
```

```
<bean id="hall" class="ui.RoomFrame">
<property name="scuba" ref="scuba"></property>
<property name="IRobot" ref="iRobot"></property>
<property name="duster" ref="duster"></property>
<constructor-arg value="Hall"></constructor-arg>
</bean>
```



Beans inheritance

```
<bean id="room" class="ui.RoomFrame" abstract="true">
  <property name="scuba" ref="scuba">
  </property> <property name="IRobot" ref="iRobot">
  </property> <property name="duster" ref="duster">
  </property>
</bean>

<bean id="hall" parent="room">
  <constructor-arg value="Hall"></constructor-arg>
</bean>

<bean id="guestroomb" parent="room">
  <constructor-arg value="guestroomb"/>
</bean>
```

XML inheritance

Inside your xml context you can just...

```
<import resource="english-application-context.xml"/>
```

XML Namespaces

- Более высокий «язык» конфигураций, чем базовый `<beans/>`
 - Абстракции и более конкретные валидации схемы
 - Упрощает конфигурацию
- Каждый «язык» представлен своим namespace-ом, который можно импортировать
 - Можно так-же написать свой

Importing a Custom XML Namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans           Assign namespace a prefix
        http://www.springframework.org/schema/beans/spring-context-2.5.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-2.5.xsd">

    <!-- You can now use <context:> tags -->
</beans>
```

Assign namespace a prefix

Associate namespace with its XSD

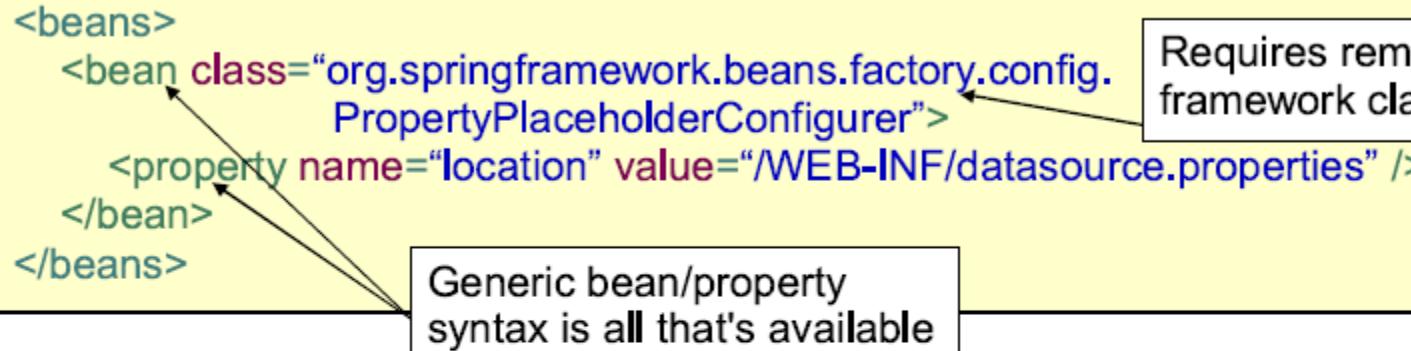
Преимущества

- Before

```
<beans>
  <bean class="org.springframework.beans.factory.config.
    PropertyPlaceholderConfigurer">
    <property name="location" value="/WEB-INF/datasource.properties" />
  </bean>
</beans>
```

Generic bean/property syntax is all that's available

Requires remembering framework classnames



- After

```
<beans ... >
  <context:property-placeholder location="/WEB-INF/datasource.properties" />
</beans>
```

Creates same bean as above, but hides framework details

Attributes are specific to the task, rather than generic name/value pairs

Примеры Namespace-ов

- Неполный список:^{*}
 - aop, beans, context, jee, jms, tx, util

* Полный список тут: <http://www.springframework.org/schema/>

Util Namespace

- <util:/> удобный namespace!
- Как ясно из названия, всякие удобства:
 - Наполнение коллекций
 - Доступ к статическим полям и константам
 - Загрузка пропертий
- Но ничего, что нельзя было бы сделать без него
 - С помощью всяких низко-уровневых кошмаров, типа FieldRetrievingFactoryBean

Конфигурируем список

```
<bean id="inventoryManager" class="foo.MyInventoryManager">
    <property name="warehouses">
        <util:list> ←
            <ref bean="primaryWarehouse" /> List is an inner bean
            <ref bean="secondaryWarehouse" />
            <bean class="foo.Warehouse" .../>
        </util:list>
    </property>           ← Elements are beans (one inner)
</bean>

<bean id="primaryWarehouse" .../>
<bean id="secondaryWarehouse" .../>
```

Конфигурируем Сет

```
<bean id="notificationService" class="foo.MyNotificationService">
    <property name="subscribers" ref="subscribers"/>
</bean>

<util:set id="subscribers">
    <value>larry@foo.com</value>
    <value>curly@foo.com</value>
    <value>moe@foo.com</value>
</util:set>
```

Set is a top-level bean

Elements are literal values

Конфигурируем Map

```
<bean id="inventoryManager" class="foo.MyInventoryManager">
    <property name="warehouses">
        <util:map>
            <entry key="primary" value-ref="primaryWarehouse" />
            <entry key="secondary">
                <bean class="foo.Warehouse" .../>
            </entry>
        </util:map>
    </property>
</bean>

<bean id="primaryWarehouse" .../>
```

Value is a top-level bean

Value is an inner bean

Конфигурируем Мап

```
<bean id="inventoryManager" class="foo.MyInventoryManager">
    <property name="warehouses">
        <util:map>
            <entry>
                <key>
                    <bean class="foo.WarehouseKey">
                        <constructor-arg value="123456789" />
                    </bean>
                </key>
                <bean class="foo.Warehouse" .../>
            </entry>
        </util:map>
    </property>
</bean>
```

Key is an inner bean

Обращаемся к константе

```
<bean id="notificationService"
      class="example.MyNotificationService">
    <property name="hostname">
      <util:constant static-field="example.MyConstants.HOST_NAME" />
    </property>
</bean>
```

Accesses static field in the MyConstants class

No more XML, no more
bloodshed

Аннотации

- @Component / @Service / @Repository
- @Controller
- @PostConstruct / @Predestroy
- @Autowired / @Inject / @Resource
- @Required
- @Qualifier
- @Scope
- @Lazy

А как сделать так, чтобы эти аннотации считывались?

Мне что, все
БинПостПроцессоры, которые
за них отвечают,
наизусть помнить?



Два способа активизировать аннотации

```
<context:component-scan base-package="lesson4"></context:component-scan>
</beans>
```

Всего одна строчка

```
<context:component-scan base-package="com.inwhite.services">
    <context:include-filter type="regex"
        expression="com\.\inehite\.\services\..*Course"/>
    <context:exclude-filter type="annotation"
        expression="com.inwhite.annotations.Deprecated"/>
</context:component-scan>
```

Или:

```
AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext("lesson4");
```

Как декларировать бины через аннотации

- @Component

```
public class ShakeSpearQuoter {
```

- @Component("tQ")

```
public class TerminatorQuoter {
```

- @Service

```
public class MyService {
```

- @Repository

```
public class MyDao {
```

Аннотации для впрыскивания зависимостей

- @Resource – javax
 - @Inject - javax
 - @Autowired - spring
 - @Value – spring
-
- Впрыскивание в статические поля не поддерживается

@Resource - Впрыскивает зависимости по имени бина

- `@Resource`
`private DbService dbService;`
- Будет впрыснут бин с названием dbService
- А если название бина не совпадает с названием проперти?
- `@Resource(name = "databaseService")`
`private DbService dbService;`
- `@Resource` можно ставить над филдом или над сетером
- Обрабатывается CommonAnnotationBeanPostProcessor
- `@Resource` лучше не использовать

Трудный выбор...

@Autowired



@Inject



@Autowired

- Спринговая аннотация
- Можно ставить над филдом, конструктором и методом
- Впрыскивает по типу. Если больше чем один бин является кандидатом на впрыскивание, контекст падает
- По умолчанию @Autowired(required=true), но можно поменять
- При помощи @Qualifier можно уточнить какой именно бин должен вспрыснуться

@Qualifier

- Это мета аннотация
- Можно использовать вместе с @Autowired и указывать имя бина

`@Autowired`

`@Qualifier("terminatorQuote")`

`private String message;`

- Но правильней определить свою аннотацию и пользоваться ей

- `@Retention(RetentionPolicy.RUNTIME)`

`@Qualifier`

`public @interface TerminatorQuote { }`

Теперь можно
пользоваться так:



- `@Autowired
@TerminatorQuote
private String message;`

А можно вот так:

```
@Retention(RetentionPolicy.RUNTIME)  
@Qualifier  
public @interface Quote {  
    QuoteType value();  
}
```

```
@Autowired  
@Quote(QuoteType.)  
private String SHAKE_SPEAR  
        TERMINATOR
```

Задание

- Напишите 2 аннотации @Oracle и @Derby
- Напишите 2 класса implementирующие интерфейс Dao с методом crud()
- Один из классов является реализацией оракла, второй дерби
- Укажите это при помощи собственных аннотаций
- Напишите сервис, в с проперти Dao и инжекните в него один раз реализацию оракла, а второй дерби, при помощи собственных аннотаций

@Inject

- Стандарт джавы – это плюс
- У него нет, как в @Autowired параметра required
- @Autowired(required=true) = @Autowired = @Inject
- @Autowired(required=false) = @Autowired(required=false)

Трудный выбор...

@Autowired



Вкусно

@Inject



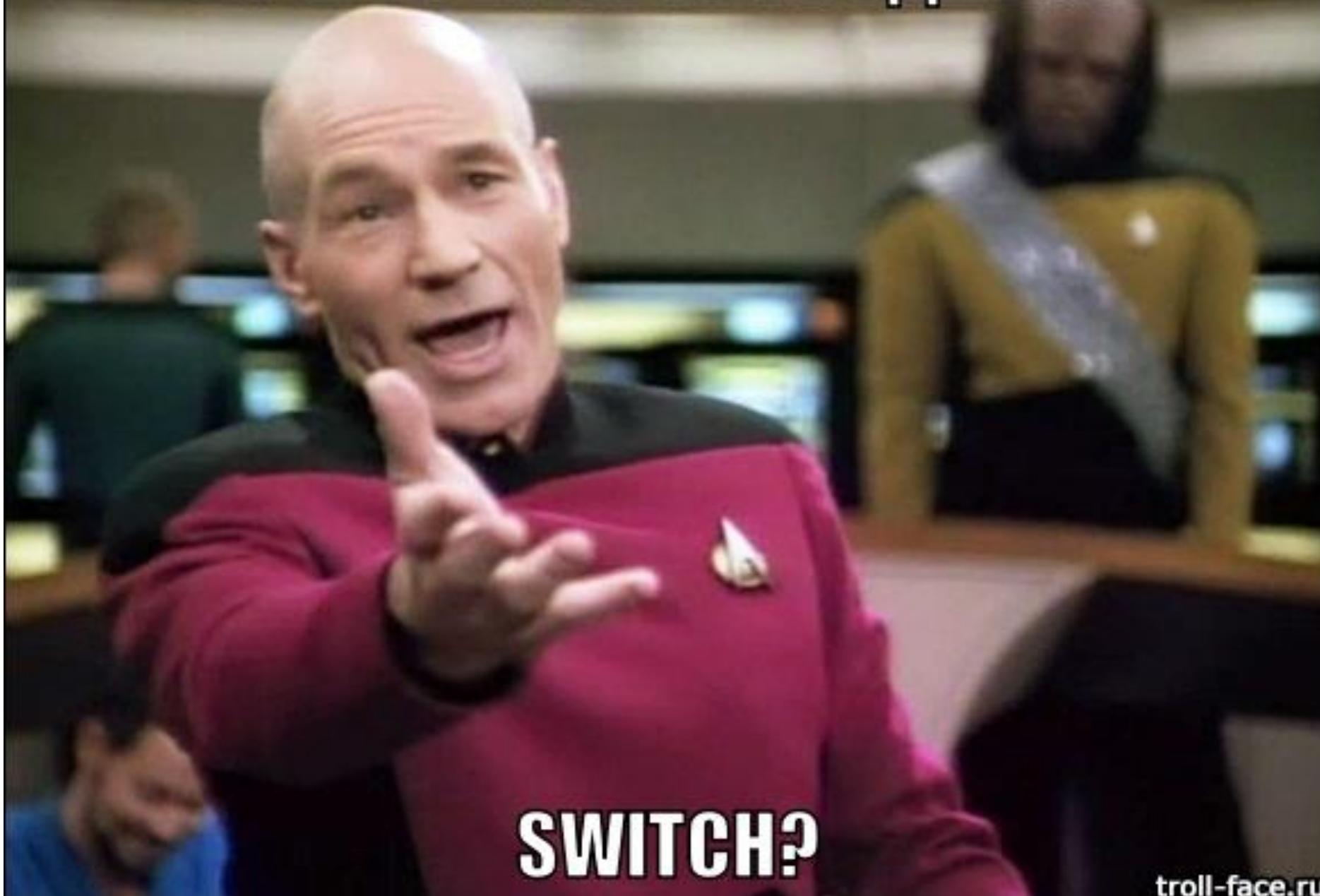
Стандарт

Напишите инфраструктуру ...

- Программа получает интеджер и запускает соответствующую бизнес логику.
- Допустим для начала есть только 2 варианта (1 и 2)
- Код, который вы сейчас напишите будет основой для инфраструктуры.
- Все ваши последователи будут делать как вы



ПОЧЕМУ ПРОСТО НЕ СДЕЛАТЬ



SWITCH?

Мы любим тебя, Switch ...

```
public DistribHandler resolveHandler(Integer.valueOf(documentObject.getDocumen
switch (documentObject.getDocumentType()) {
    case PDF_STORAGE:
        fileContainer = new PdfRecordFileContainer();
        getPdfFromStorage(fileContainer);
        break;
    case PDF_SRC:
        fileContainer = new PdfRecordFileContainer();
        fillObjectsForPdf(fileContainer, documentObject);
        break;
    case PDF_WS:
        fileContainer = new WsPdfRecordFileContainer();
        getPdfFromPdfWs(fileContainer, documentObject);
        break;
    case LIS:
        fileContainer = new PdfRecordFileContainer();
        getLisDocument(j, fileContainer);
        break;
    case IMAGE:
        fileContainer = new PdfRecordFileContainer();
        getPdfFromImageDocument(j, fileContainer);
        break;
    case FORM:
        fileContainer = new PdfRecordFileContainer();
        getFormDocument(fileContainer);
        break;
}

switch (value.getNumericValue()) {
    case 1:
        text = MessageFormat.format("{0} - {1}", emailRequest.getSubject(), "הצעה לפולשת");
        break;
    case 2:
        text = MessageFormat.format("{0} - {1}", emailRequest.getSubject(), "רביישת לפולשת");
        break;
    case 3:
        text = MessageFormat.format("{0} - {1}", emailRequest.getSubject(), "חידוש לפולשת");
        break;
    case 4:
        text = MessageFormat.format("{0} - {1}", emailRequest.getSubject(), "שינויים בפולשת");
        break;
    case 5:
        break;
    case 8:
        text = MessageFormat.format("{0} - {1}", brandHebName);
        break;
    case 6:
        break;
    case 7:
        text = MessageFormat.format("{0} - {1}", brandHebName);
        break;
    case 9:
        break;
    case 17:
        text = MessageFormat.format("{0} - {1}", brandHebName);
        break;
    case 10:
        break;
    case 13:
        text = MessageFormat.format("{0} - {1}", emailRequest.getSubject(), brandHebName);
        break;
    case 11:
        text = emailRequest.getSubject();
        break;
    case 12:
        if (!resources.getBrandKey().isBituhYashir()) {
            text = format("5% מהתו", brandHebName);
        } else {
            text = format("5% מהתו", brandHebName);
        }
        break;
    case 14:
        text = MessageFormat.format("{0} - {1}", brandHebName);
        break;
    case 15:
        text = MessageFormat.format("{0} - {1}", emailRequest.getSubject());
        break;
    case 16:
        text = MessageFormat.format("{0} - {1}", emailRequest.getSubject());
        break;
}

if (item.getAddresseeCode().length() >= 1) {
    item.setAddresseeCode(item.getAddresseeCode().trim());
    if (item.getAddresseeCode().trim().equals("0")) {
        item.setAddresseeCode("");
    }
}

IcyDatFileConsumer.class);
Ic(yDatFileConsumer);
Ic(yDetailsConfConsumer);
Ic(yLetterConsumer);
Ic(yCompulsoryConsumer);
setSeqNr(item.getCompSeqNr());
currentPath, basket, dataConsumer);

IcyDatFileConsumer.class);
Ic(yDatFileConsumer);
Ic(yDetailsConfConsumer);
Ic(yLetterConsumer);
Ic(yCompulsoryConsumer);
setSeqNr(item.getCompSeqNr());
currentPath, basket, dataConsumer);

IcyDatFileConsumer.class);
Ic(yDatFileConsumer);
Ic(yDetailsConfConsumer);
Ic(yLetterConsumer);
Ic(yCompulsoryConsumer);
setSeqNr(item.getCompSeqNr());
currentPath, basket, dataConsumer);
```

А вот и альтернатива

```
switch (personalMailBusinessModel.getFtlType()) {
    case WELCOME:
        ftlService.fillWelcomeFtl(data, personalMailBusinessModel);
        break;
    case EMAIL_CALLBACK:
        ftlService.fillEmailCallbackFtl(data, personalMailBusinessModel);
        break;
    case PROTECTION_APPROVAL_RemINDER:
        ftlService.fillProtectApprovalReminderFtl(data, personalMailBusinessModel);
        break;
    case CALL_ROUTER:
        ftlService.fillCallRouterFtl(data, personalMailBusinessModel);
        break;
    case PREV_YEAR_INTERNET:
        ftlService.fillPrevYearInternet(data, personalMailBusinessModel);
        break;
    case PREV_YEAR_SALES:
        ftlService.fillPrevYearSales(data, personalMailBusinessModel);
        break;
    case PROPOSAL_PREMIUM_RemINDER:
        ftlService.fillProposalPremiumReminder(data, personalMailBusinessModel);
        break;
    case MORTGAGE:
        ftlService.fillMortgageFtl(data, personalMailBusinessModel);
        break;
    case MORTGAGE_ENSLAVED_STRUCTUR:
        ftlService.fillMortgageEnslavedStructurFtl(data, personalMailBusinessModel);
        break;
    case RENEWAL_WAIT_TO_THINK:
        ftlService.fillRenewalWantToThinkFtl(data, personalMailBusinessModel);
        break;
    case PREV_YEAR_OUT_SALES_APARTMENTS:
        ftlService.fillPrevYearOutSalesApartmentsFtl(data, personalMailBusinessMode
        break;
    case LIFE_OUT_SALES_FOLLOW_UP:
        ftlService.fillLifeOutSalesFollowUpFtl(data, personalMailBusinessModel);
        break;
    case APPS_PROPOSAL_FOLLOW_UP:
        ftlService.fillAppsProposalFollowUpFtl(data, personalMailBusinessModel);
        break;
}
```

ftlMainService.generateHtml(data, personalMailBusinessModel);

Что нельзя делать при помощи аннотаций

- Нельзя декларировать бины из 3-d party libraries
- Нельзя прописать несколько бинов из одного и того же класса
- Нельзя работать с проперти файлами (не верно со 4.3)

Что нельзя делать при помощи XML

- В нём нельзя писать код. Если настройка бина требуют какой-то логики, то через XML это делать очень неудобно

Задание

- Определите два бина: JFrame(Singleton) и Color(Prototype)
- Color должен впрыскиваться в frame и отображаться на его фоне
- Цвет всегда должен быть случайным
`new Color(random,random,random)`

Java Config

- Нужно аннотировать класс @Configuration
- Чтобы прописывать в нём бины надо пользоваться @Bean

- ```
@Bean
@Scope(BeanDefinition.SCOPE_PROTOTYPE)
public Color randomColor() {
 Random random = new Random();
 return new Color(random.nextInt(255), random.nextInt(255), random.nextInt(255));
}
```

- Можно создавать контекст из джава конфига
  - @ComponentScan(basePackages = "com.servies")
  - @Configuration
- ```
public class AppConfig {
```

Как впрыскивать проперти без XML-а

- ```
@PropertySource("classpath:mails.properties")
@ComponentScan(basePackages = "com.services")
@Configuration
public class AppConfig { ... }
```
- ```
@Bean
public PropertySourcesPlaceholderConfigurer configurer() {
    return new PropertySourcesPlaceholderConfigurer();
}
```
- ```
public class ShakespeareQuoter implements Quoter {
 @Value("${shakeSpearQuote}")
 private String message;
```
- ```
new AnnotationConfigApplicationContext(AppConfig.class)
```

Зависимости бинов друг от друга

- Напишите 2 бина. Один в постконракте создает файл, а другой в своем постконракте печатает его длину
- Путь к файлу впрыскивается через @Value из проперти файла

@DependsOn

- Должен использоваться когда нет прямой зависимости между бинами, но есть косвенная

Как комбинировать несколько контекстов

```
@Configuration  
public class DatabaseConfig {  
    @Bean  
    public DataSource dataSource() {  
        // instantiate, configure and return DataSource  
    }  
}  
  
@Configuration  
@Import(DatabaseConfig.class)  
public class AppConfig {  
    @Autowired DatabaseConfig dataConfig;  
  
    @Bean  
    public DBService myBean() {  
        // reference the dataSource() bean method  
        return new DBService(dataConfig.dataSource());  
    }  
}
```

Конгломерат: Xml + JavaConfig

```
@Configuration  
@ImportResource ("classpath:/com/acme/database-config.xml")  
public class AppConfig {  
    @Autowired  
    DataSource dataSource; // from XML  
  
    @Bean  
    public DBService myBean() {  
        // inject the XML-defined dataSource bean  
        return new DBService(this.dataSource);  
    }  
}
```

А вот теперь сложно задание...

- Вы знаете анекдот про хирурга?

А теперь вернёмся к нашему цветному фрейму

- В тесте вызываем 50 раз лукап нашего фрейма, и запускаем его метод `showOnRandomPlace`
- Первый вариант: и фрейм и цвет – сингальтоны
- Второй вариант: оба прототайпа
- Третий вариант: Фрейм прототайп, цвет – сингальтон
- Четвёртый вариант: Фрейм – сингальтон, цвет – прототайп
- А вот теперь сделайте, чтобы цвет мигал

Как можно создавать контекст

Source/ Type	XML	Annotations
Classpath	ClassPathXmlApplicationContext	AnnotationConfigApplicationContext
File system	FileSystemXmlApplicationContext	
Web container	XMLConfigWebApplicationContext	AnnotationConfigWebApplicationContext

Сравнение стратегий

Bean Definition	XML	Annotations	@Configuration
How to declare?	<bean>	@Component class	@Bean method
What?	“class” attribute	Annotated class	You can't know!
Name	“id” or “name” attr	Decap. class name	Method name
Instantiation	Reflection	Reflection	Java code
Dependencies	<constructor-arg>, <property>	@Autowired	Constructor params, setter methods
Scope	“scope” attr	@Scope	@Scope
Init & destroy methods	<init-method>, <destroy-method>	@PostConstruct, @PreDestroy	Method invocation, “initMethod”, “destroyMethod” attributes

Когда что подходит

Что важно для проекта	XML	Annotations	@Config
Центральная конфигурация в одном месте	✓	✗	✓
Чтобы не было спринга в сорсах	✓	✗	✓
Не перекомпилировать при изменении конфигурации	✓	✗	✗
Поддержка IDE + refactoring	✗	✓	✓
Минимум конфигурации	✗	✓	✗

Когда чем пользоваться

Sample Usage	XML / Groovy Script	Annotations	@Config
My source code		✓	
3 rd -party utils and frameworks*			✓
Configuration by non-devs	✓		
Modular configuration requirements	✓		

* Предположим, что мы любим программировать на джаве больше, чем на xml-е

Еще раз, разница между XML Config и Java Config

- XML – metadata для создания bean definitions
 - Java config – код для создания beans
-
- Когда создавался java config казалось, что bean definitions – ненужный костыль
 - Но нет!

Кто царь?!

- Groovy Configuration: код для bean definitions!

```
import spring.battle.groovy.Cluster
import spring.battle.groovy.CreationCallbackBeanFactoryPostProcessor
import spring.battle.groovy.ImportController
import spring.battle.groovy.XmlParser

import static spring.battle.groovy.Cluster.Builder.ConstantReconnectionPolicy
import static spring.battle.groovy.Cluster.Builder.PoolingOptions.Options.LOCAL
import static spring.battle.groovy.Cluster.DowngradingConsistencyRetryPolicy.INSTANCE

beans {

    xmlns([mvc: 'http://www.springframework.org/schema/mvc'])
    xmlns([aop: 'http://www.springframework.org/schema/aop'])
    xmlns([context: 'http://www.springframework.org/schema/context'])

    mvc.'annotation-driven'()

    context.'property-placeholder'(location: 'file:cassandra.properties')

    xmlParser XmlParser

    cluster (ClusterFactoryBean) {
        contactPoint = '${contactPoint}'
        connectionsPerHost = '${connectionsPerHost}'
        reconnectionPolicy = '${reconnectionPolicy}'
    }

    importController ImportController, xmlParser, cluster
}
```

JUnit 4 и аннотация @Test

```
public class TransferServiceTest {  
  
    @Test public void correctTransfer() {  
        // do tests  
        Assert.assertNotNull(1234);  
    }  
}
```

- Удобно, но причем тут Спринг?

System Testing with JUnit 4

- `@ContextConfiguration`
- Значение по умолчанию: config.xml из того же package-а.
- А теперь можно `@Autowired` все зависимости, которые мы хотим оттестить

```
@ContextConfiguration(locations={"/transfer-config.xml"})
@RunWith(SpringJUnit4ClassRunner.class)
public class TransferServiceTest {
    @Autowired TransferService serviceToTest;
}
```

`@DirtiesContext(classMode = DirtiesContext.ClassMode.BEFORE_EACH_TEST_METHOD)`

Тестирование транзакционного кода

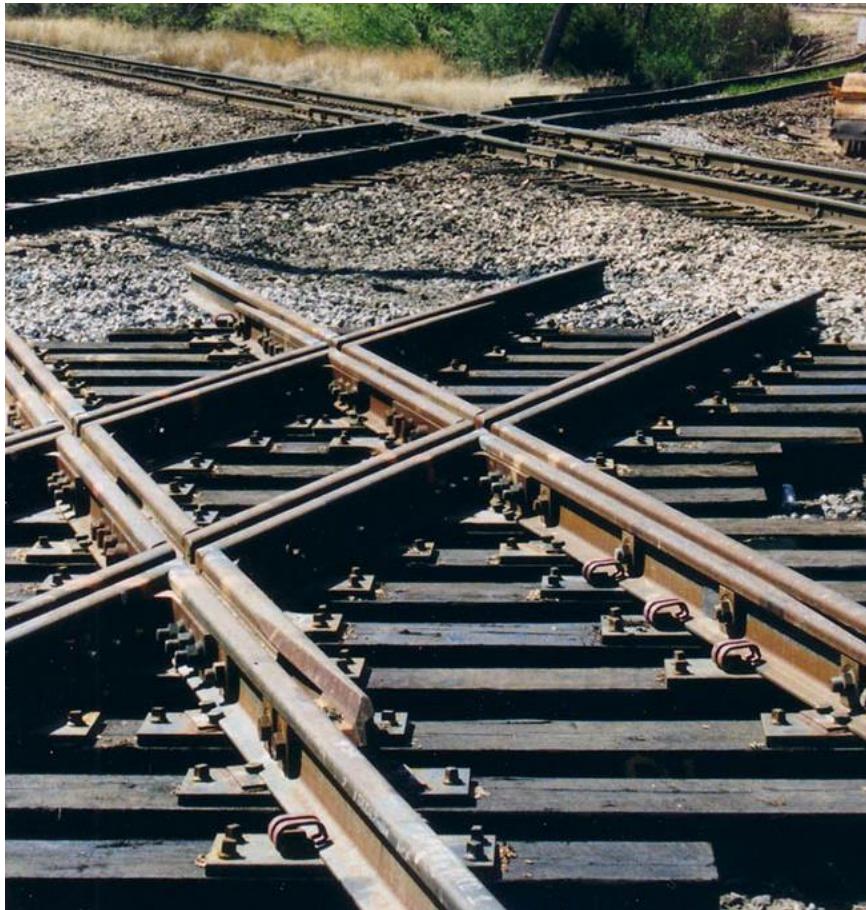
- А что с транзакциями? Изменения из тестов не должны попадать в базу!
- Ставим `@Transactional` и получаем транзакцию, и откат после теста!

```
@Transactional @Test public void correctTransfer() {  
    // do tests  
}
```

Тесты

Важно, а со Спрингом и просто

Aspect Oriented Programming (AOP)

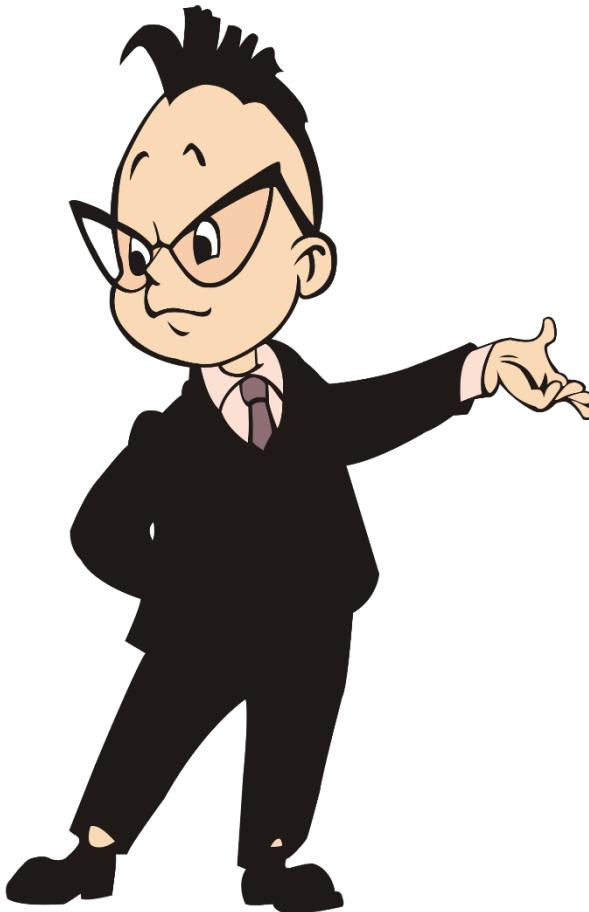


А нафига оно нам надо?

- Сделайте трэйсинг всей вашей апликации
- Перед началом и концом работы метода должно выводится в лог его название с соответствующим сообщением
- Допустим у нас 50,000 классов по 5 методов в среднем
- Значит надо в 500,000 мест скопипастить тот же код
- Не сложно?



А давайте замутим reflection



- Смешать разные аспекты?
Бизнес код + tracing
- А что делать с CVS
- Как потом всё это поддерживать



Это проблему решает AOP

- Аспектно ориентированные языки:
AspectJ, Spring AOP, JBoss AOP, и.т.д.
- Что можно с их помощью делать:
tracing, benchmarking, logging, validations, security, transactions, caching...

Терминология

- Aspect – аналог класса
- Advice – аналог метода
- Joinpoint – когда должны работать advice-ы
- Pointcut – для каких методов должны работать advice-ы
- Weaving – как всё это работает

Spring & AspectJ

- В первой версии спринга они пытались написать свой синтакс для AOP
- Потом перешли на синтакс AspectJ
- Чтобы активизировать BeanPostProcessor, который отвечает за сканирование аннотаций связанных с аспектами надо:
 - Для XML `<aop:aspectj-autoproxy/>`
 - Для java config:
 - `@Configuration
@EnableAspectJAutoProxy
public class AppConfig {`

@Before

- @Aspect

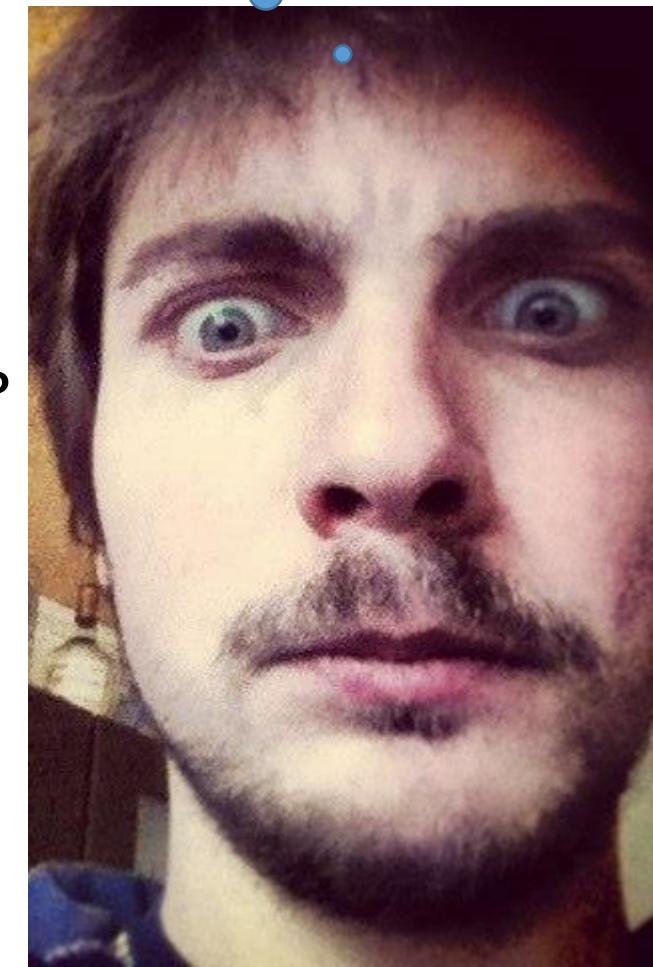
```
public class MyAspect {  
    @Before("execution(*  
aop.example.application..*.foo(..))")  
    public void beforeFooMethods() {  
        System.out.println("Фyyyyyyyyy");  
    }  
}
```
- @Before("execution(*
aop.example.application..*.foo(..))")

```
public void beforeFooMethods(JoinPoint jp) {  
    String methodName = jp.getSignature().getName();  
    System.out.println(methodName);  
}
```

Troubleshooting

Не работает...

- Ты часом не aspect вместо класса создал, Есть у Intelij такая опция.
- А ты @EnableAsptectJAutoProxy поставил?
- А ты @Aspect поставил
- А ты свой аспект объявил бином спринга? (@Component)
- А ты сканируешь пэкэдж где находится твой аспект, ведь если он @Component этот пэкэдж надо сканировать
- А те методы которые ты хочешь перехватывать запускаются у бинов спринга. Ты ничего не создаешь через new?
- Ты ведь не надеешься, что аспект перехватит метод который вызывается @PostConstruct-ом?



Какие джойнпоинты есть?

- @Before
- @AfterReturning
- @AfterThrowing
- @After
- @Around

@After

- @AfterReturning(pointcut = "execution(* com.inwhite..*.foo())",
 returning = "RetVal")
public void afterFoo(Double retVal) {
 System.out.println("AFTER foo() " + retVal);
}
- @AfterThrowing(pointcut = "execution(*
 aop.example.application..*.*(..))", throwing = "ex")
public void dbException(DatabaseRuntimeException ex) {
 System.out.println(ex);
}

Можно давать названия для pointcut-ов

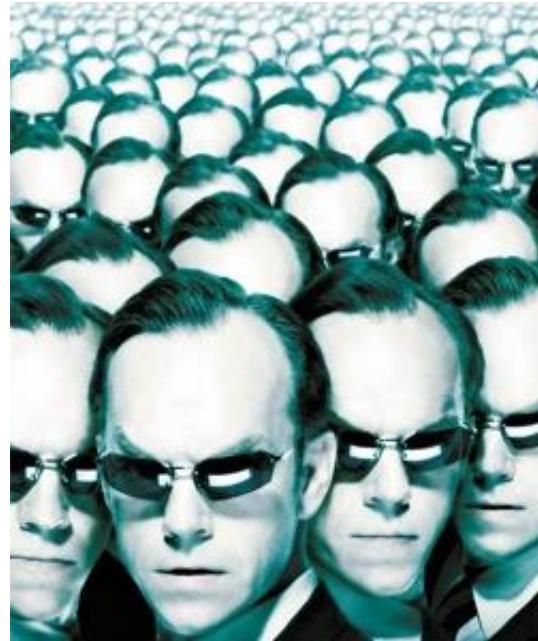
- ```
@Pointcut("execution(* aop.example.application..*.foo(..))")
public void fooMethods() {}
```
- ```
@After("fooMethods()")
public void beforeFooMethods(JoinPoint jp) {
    String methodName = jp.getSignature().getName();
    System.out.println(methodName);
}
```

Еще примеры поинткатов

- `execution(* send(int, ..))`
- `execution(@Transactional void *(..))`
- `execution((@privacy.Sensitive *) *(..))`

Instantiation Model

- By default, a single instance of the aspect is created.
- You can alter this behavior by using a *perthis* or *pertarget* expression in the @Aspect annotation.



Instantiation Model

- By using a *perthis* expression, an instance of the aspect is created for every different bean (proxy).
- E.g.:

```
@Aspect("perthis(within(com.inwhite..*))")
public class TracingAspect {
}
```

- By the same meaning is: *pertarget* (only not for each proxy, but for the target bean).
- Note, that in both cases the aspect bean itself must not be in a singleton scope!

Advice Ordering

- When two or more advices advice the same *Joinpoint* and declared in different *Aspects*, you can define order:
 - By implementing *Ordered* interface.
 - Or by annotate the aspect with `@Order`.
- The aspect with the lower level has higher precedence.
 - Runs first before.
 - Runs last after.
- When such two or more advices are in the same Aspect, there is no way to determine order.

Задание

- Напишите свой DatabaseRuntimeException
- Напишите Аспект, который будет посылать мэйл с меседжом этого эксшепшона всем работникам DBA
- Мэйлы всех работников DBA должны впрыскиваться в аспект
- Попробуйте чтобы stack trace эксепшона был глубже чем один
- Сколько раз посылается каждый мэйл?

Ещё задание

- Напишите аспект, который будет перехватывать все методы возвращающий объекты аннотированные @MyDeprecated и будет заменять то, что должно вернуться, на объект из нового класса, указанный в аннотации.
- Используйте:

```
@Around("execution( (@quoters.annotations.MyDeprecated *) *(..) )")
public void handleDeprecated(ProceedingJoinPoint pjp) {
}
```

Все это хорошо, но почему аспекты не работают в @Postconstruct?

- Те бин постпроцессоры, которые создают прокси, работают после того, как init methods закончились.
- А что же делать если мне нужно чтобы какая то логика побежала в самом начале апликации, но уже после того, как все бин пост процессоры отработали?

ApplicationListener

- У него есть метод: onApplicationEvent
- Есть 4 event-а
- ContextRefreshedEvent
- ContextStartedEvent,
- ContextStoppedEvent
- ContextClosedEvent

Задание

- Напишите свою аннотацию `@PostProxy`
- Пропишите свой `ContextListener`, чтобы он по `contextRefreshed`, запускал все методы, аннотированные этой аннотацией

Data Access with Spring

Правильные приложения работающие с базами данных

Роль Спринга

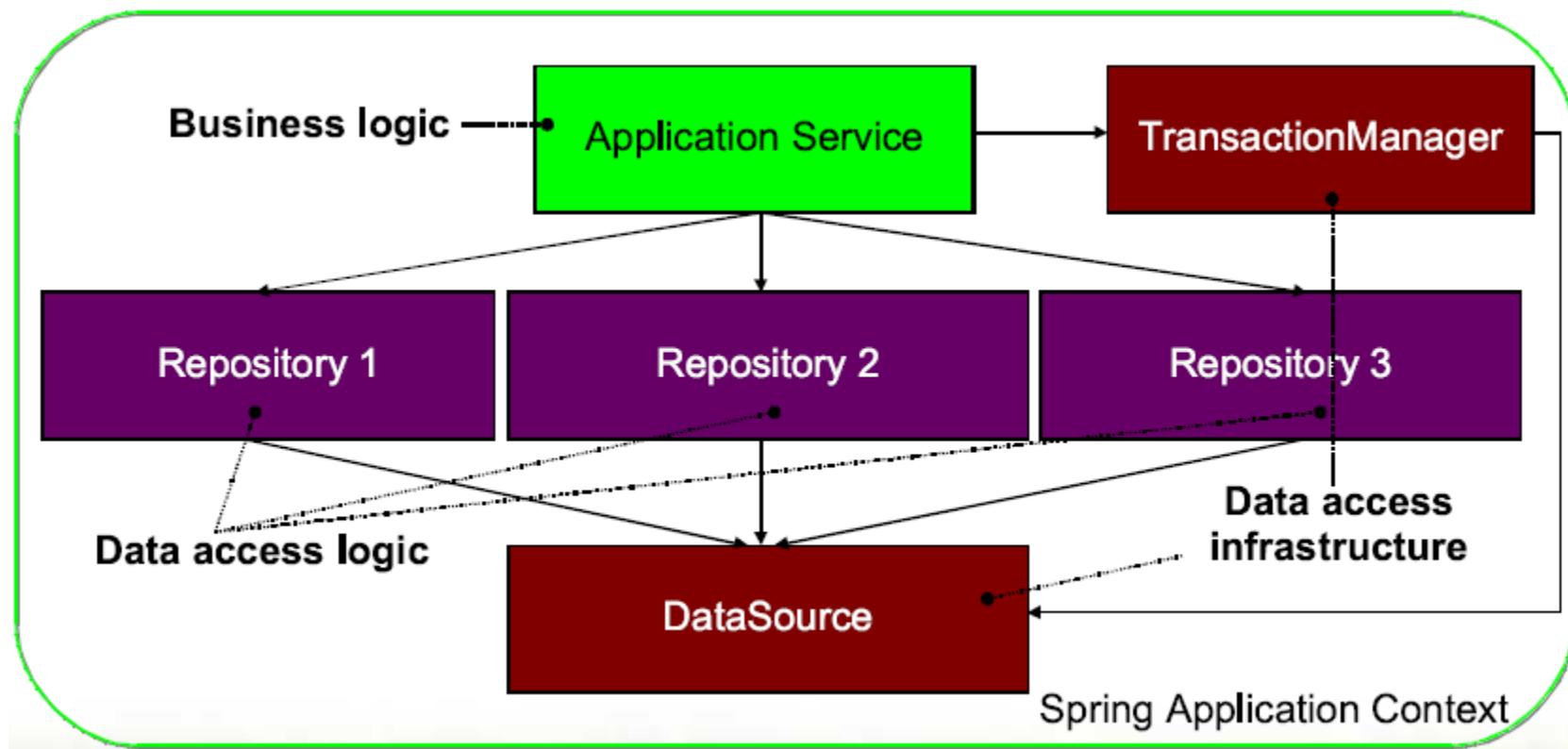
- Поддержка доступа к данным
 - Легко, эффективно
- Продвижение правильной архитектуры
 - Изолирование бизнес логики от сложностей доступа к данным

Легко!

```
int count = jdbcHelper.queryForInt(  
    "SELECT COUNT(*) FROM CUSTOMER");
```

- Раздобыть connection
- Открыть транзакцию
- Выполнить statement
- Обработать resultSet
- Обработать все исключения
- Закрыть connection

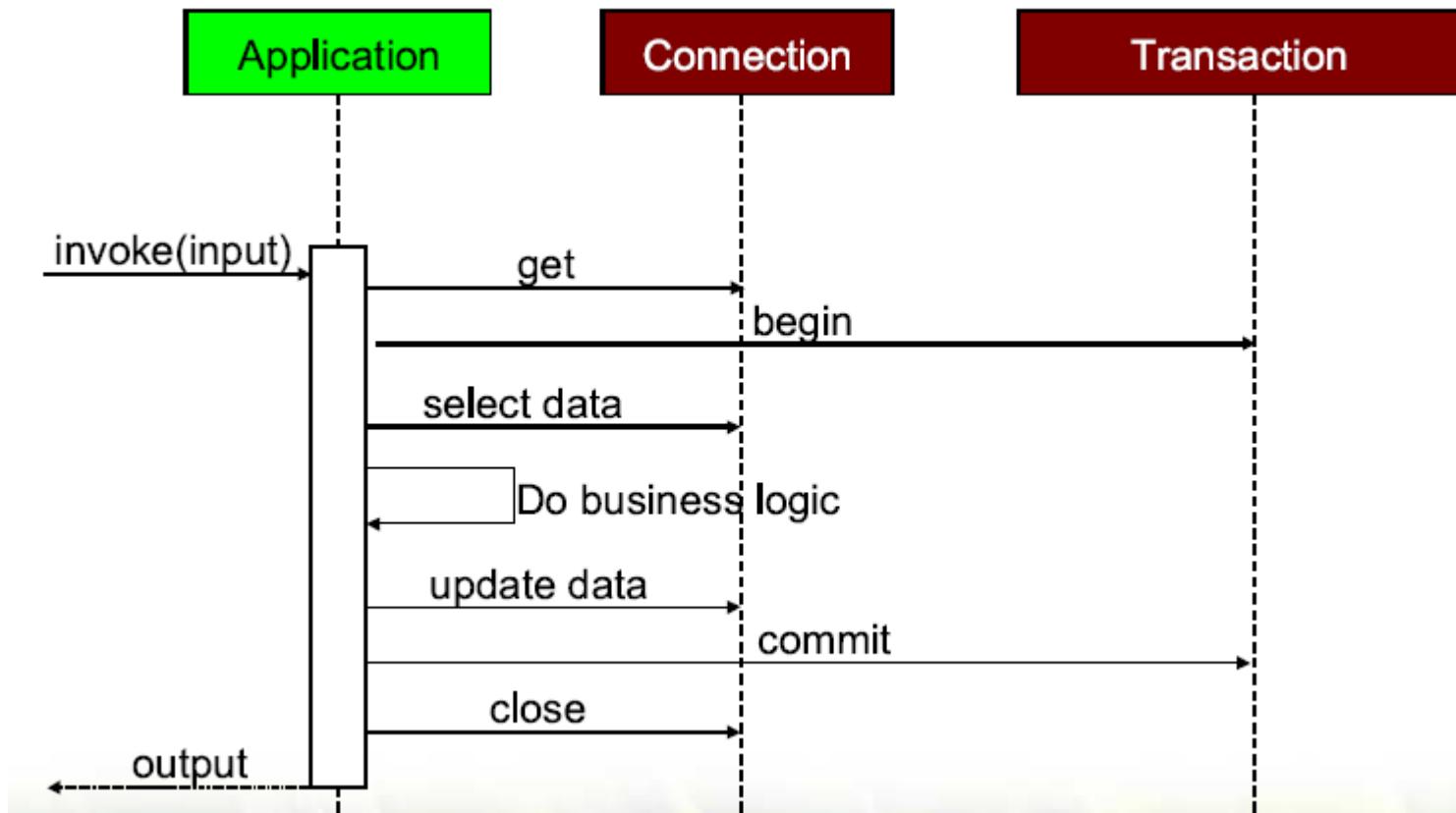
Правильная архитектура



Управление ресурсами в Спринге

- Доступ к удаленным системам (базам данных) требует ресурсов
 - Ими надо правильно управлять
- Разработчики не умеют, не хотят, да и не должны этим заниматься
 - Для этого есть middleware, то есть Spring

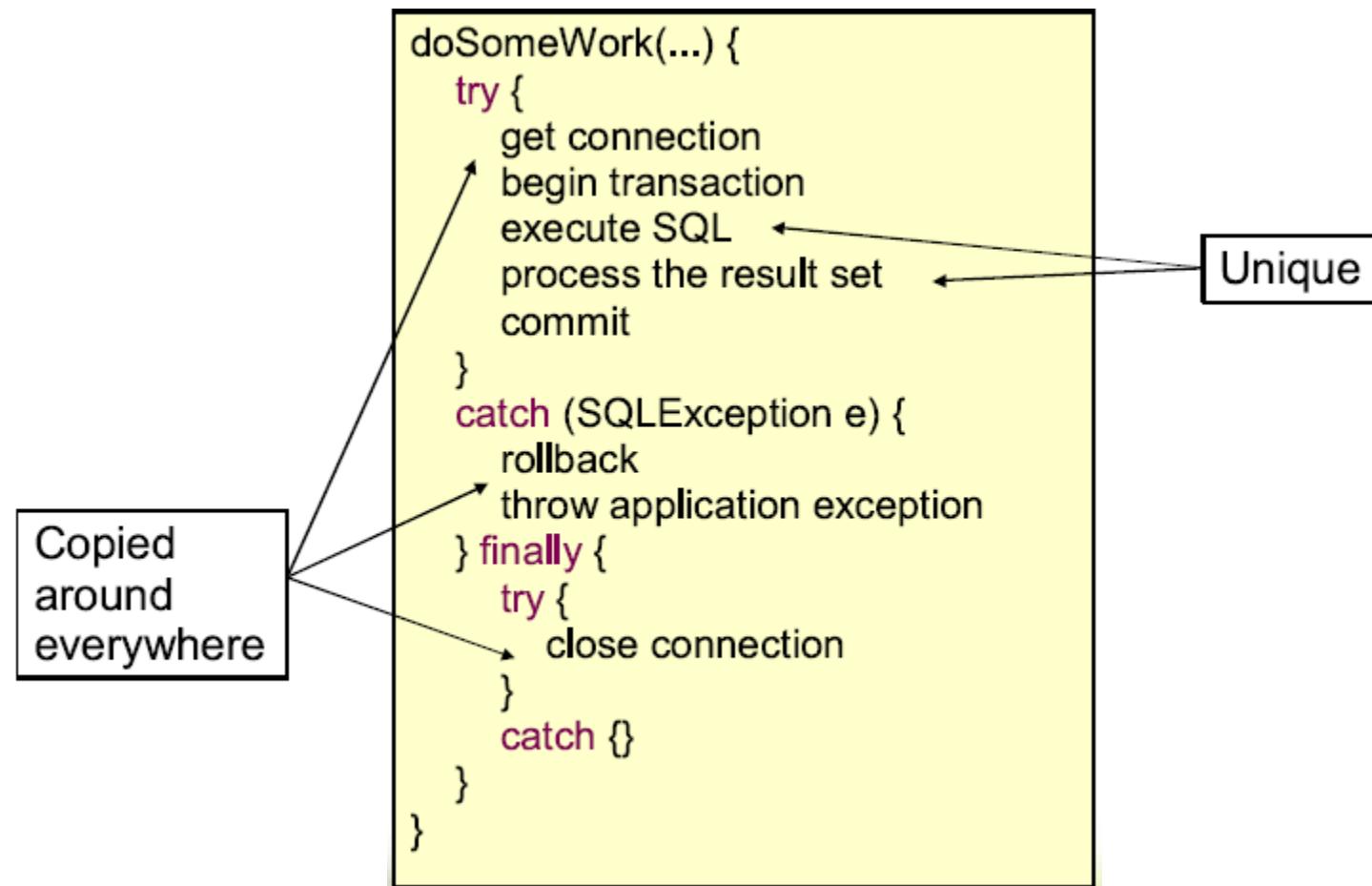
Эффективное управление ресурсами



Ручками?

- Сплошной копи-пейст...
багов.

Копи-пейст



Баги!

```
doSomeWork(...) {
    try {
        get connection
        begin transaction
        execute SQL
        execute more SQL
        commit
        close connection
    }
    catch (SQLException) {
        rollback
        log exception
    }
}
```

FAILS

close never called

Управление ресурсов в Спринге

- Спринг управляет ресурсами за нас
 - Не пишем лишний код
 - Скорее всего нет багов
- А мы пишем то, за что нам платят деньги

Ключевые преимущества

- Декларативное управление транзакциями
 - Границы транзакций устанавливаются через meta-data
 - transaction manager Spring-а делает всю работу
- Управление connection-ами
 - Connections открываются и закрываются автоматически
 - Никаких утечек
- Умная обработка исключений
 - Настоящая причина всегда видна
 - Ресурсы всегда освобождаются корректно

Управление ресурсами

 **BEFORE**

 **AFTER**

```
doSomeWork(..) {  
    try {  
        establish connection  
        begin transaction  
        execute SQL  
        process the result set
```

```
@Transactional  
doSomeWork(..) {  
    execute SQL  
    process the result set  
}
```

```
try {  
    close connection  
}  
catch {}  
}
```

Работает везде!

- Работает с любыми базами данных
 - Java Database Connectivity (JDBC)
 - JBoss Hibernate
 - Java Persistence API (JPA)
 - Oracle Toplink
 - Apache iBatis
 - А если уж подключить Spring Data...
- В любой среде
 - Локально
 - В контейнере JEE

Поддержка доступа к данным

- Для каждой технологии:
 - Удобная конфигурация бинов (при необходимости - FactoryBeans)
 - Базовые классы DAO
 - data access template для упрощения работы с API

JDBC DAO Support

```
public class JdbcCustomerRepository extends SimpleJdbcDaoSupport  
implements CustomerRepository {  
  
    public int getCountOfCustomersOlderThan(int age) {  
        String sql = "select count(*) from customer where age > ?";  
        return getSimpleJdbcTemplate().queryForInt(sql, age);  
    }  
}
```

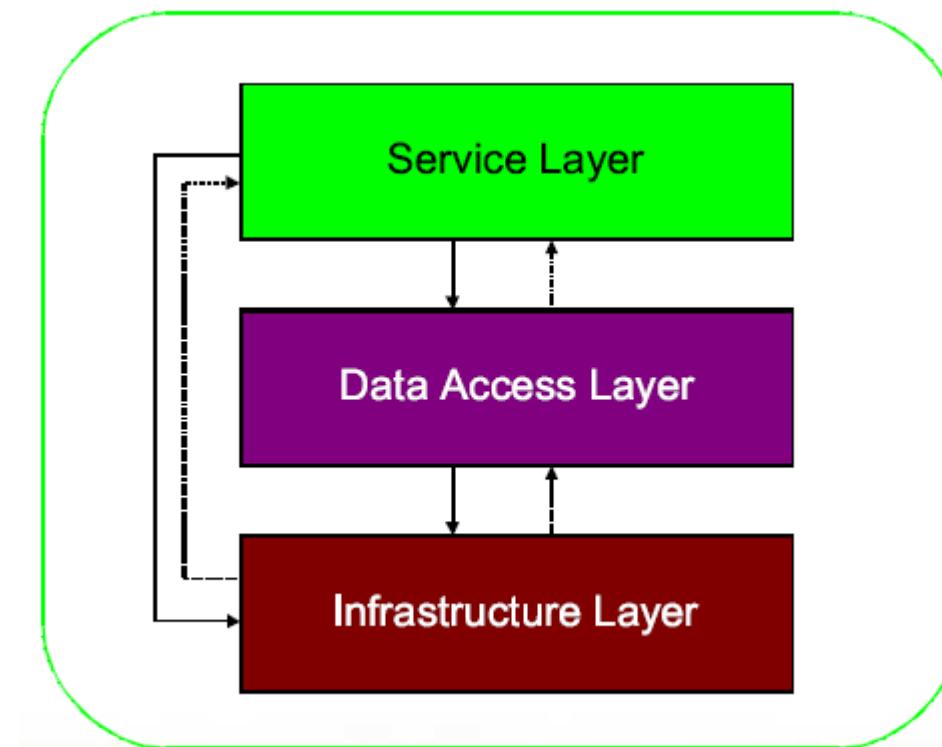
Support class provides DAO configuration assistance

Data access helper simplifies use of API

Доступ к данным в многоуровневой системе

- Спринг продвигает разделение на уровни
- Чаще всего их три
 - Service
 - Высокий уровень доступа к функциональности системы
 - Data access
 - Интерфейс доступа к данным (например, базе данных)
 - Infrastructure
 - Низкоуровневые сервисы для других уровней

Многоуровневая система



Service (Зеленым)

- Публичные функции приложения
 - Клиенты обращаются через этот уровень
- Инкапсулирует логику, необходимую для этих функций
 - Делегирует к уровню инфраструктуры для работы с транзакциями
 - Делегирует к уровню доступа для мапирования информации из базы данных

Data Access (Фиолетовый)

- Используется сервисом для работы с нужной информацией
- Инкапсулирует сложность доступа к информации
 - API доступа
 - JDBC, Hibernate, etc
 - Работа с информацией
 - SQL, HQL, etc
 - Маппирование информации в форму, подходящую для сервиса
 - Например JDBC ResultSet to в граф объектов

Infrastructure (Красный)

- Низкоуровневые сервисы, потребляемые другими уровнями
 - Натуральный middleware
 - Лучше пользоваться готовым, например Spring
- Скорее всего будет разный для разных сред - Production vs. test

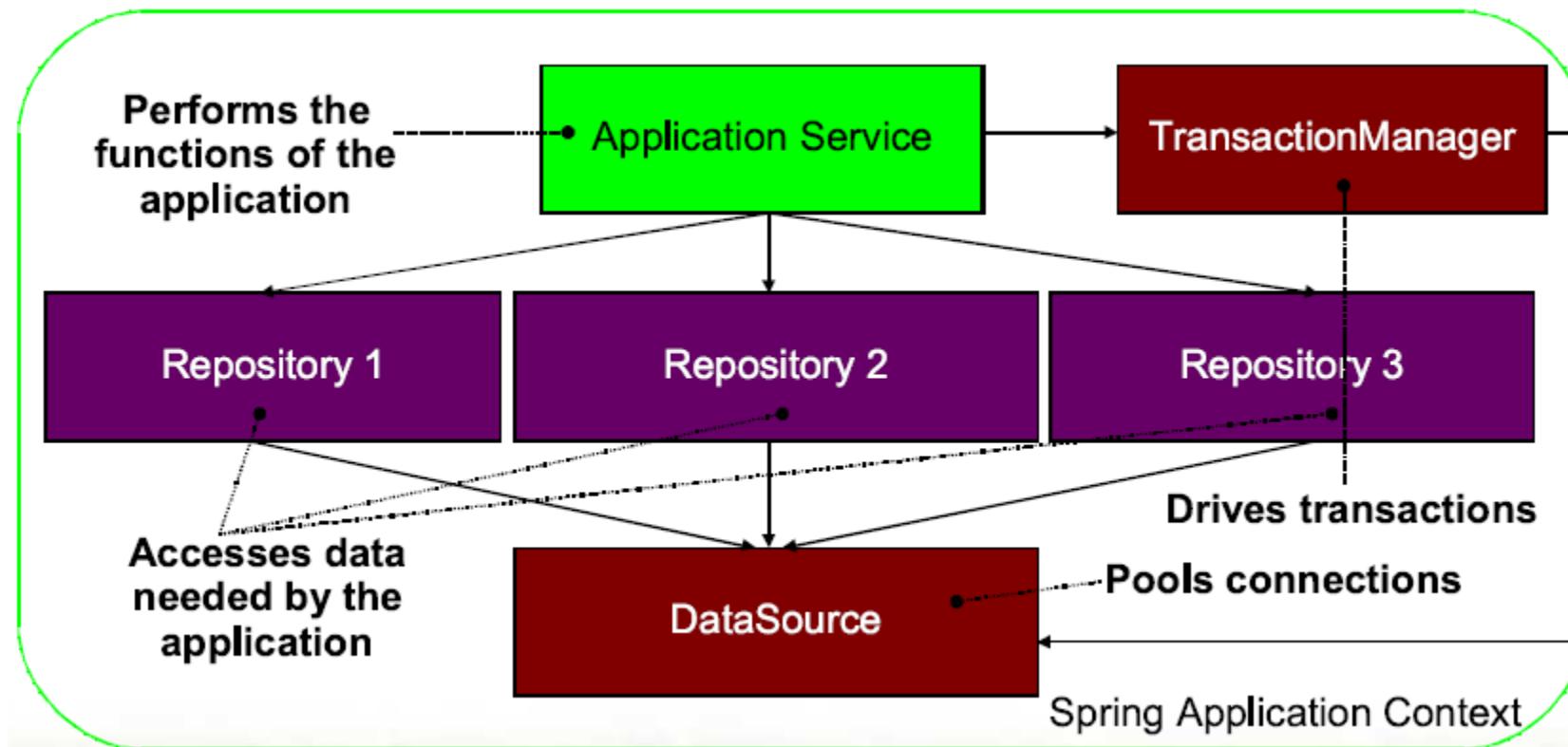
Инфраструктура: Transaction Manager

- У Спринга собственная абстракция
- Покрывает широкий выбор провайдеров
 - JDBC
 - Java Persistence API (JPA)
 - Hibernate
 - Toplink
- И стандартный Java Transaction API (JTA)

Инфраструктура - Data Source

- Спринг использует стандартный JDBC data source для подключения
 - Идет с несколькими реализациями
 - Интегрируется с популярными
 - Apache DBCP, c3p0
 - Может интегрироваться с теми, что в JEE container-ах

Связка уровней во время конфигурации



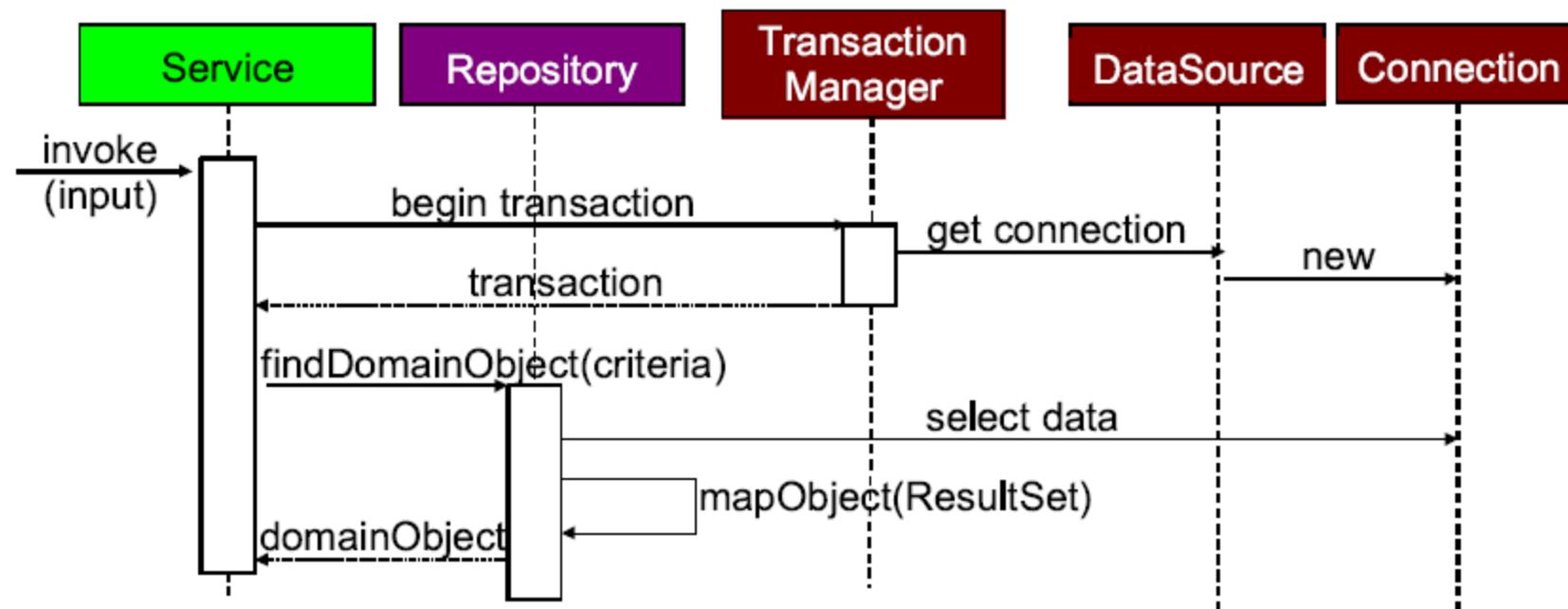
Использование уровней вместе

- Сервис инициализирует бизнес процесс
 - Делегирует в transaction manager начать транзакцию
 - Делегирует в репозитории для загрузки необходимых данных
 - Все вызовы происходят в транзакции
 - Репозитории часто возвращают domain objects

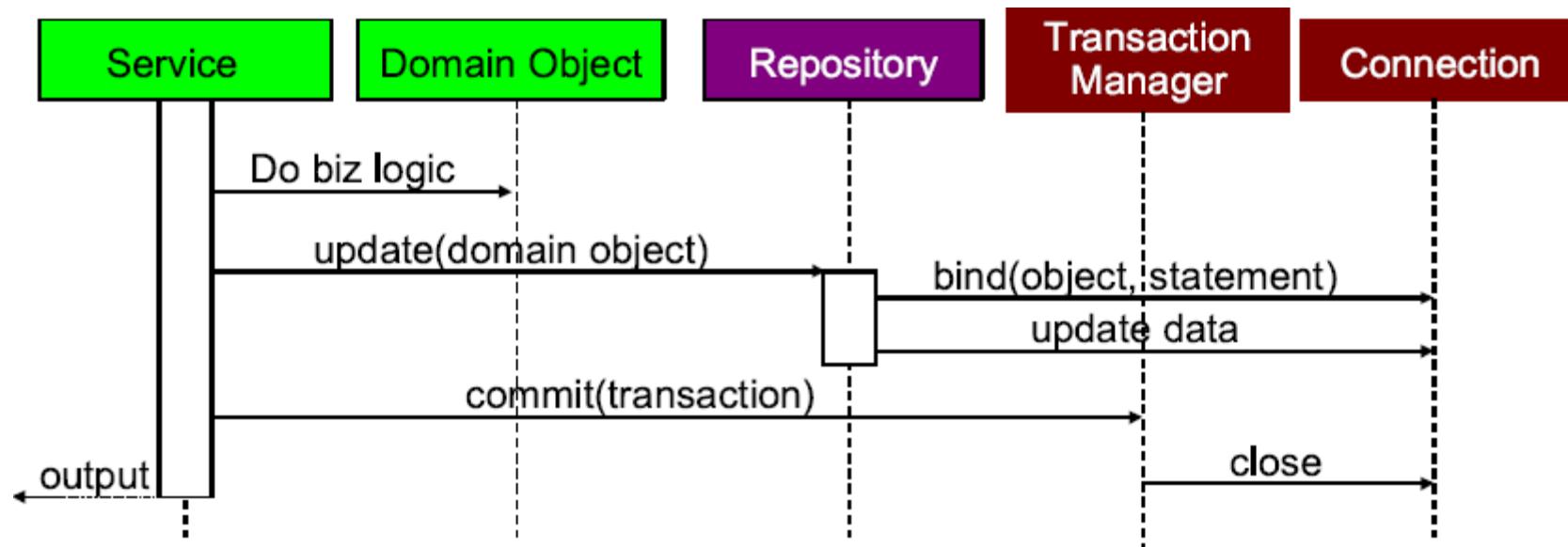
Использование уровней вместе

- Сервис продолжает работу
 - Выполняя бизнес-логику
 - Координируя разные объекты из разных репозиториев
- И заканчивает обработку
 - Обновляя данные и закрывая транзакцию
 - Технологии вроде Hibernate/JPA могут сами обновлять данные

Application Service - Sequence (1)



Application Service - Sequence (2)



Декларативное управление транзакциями

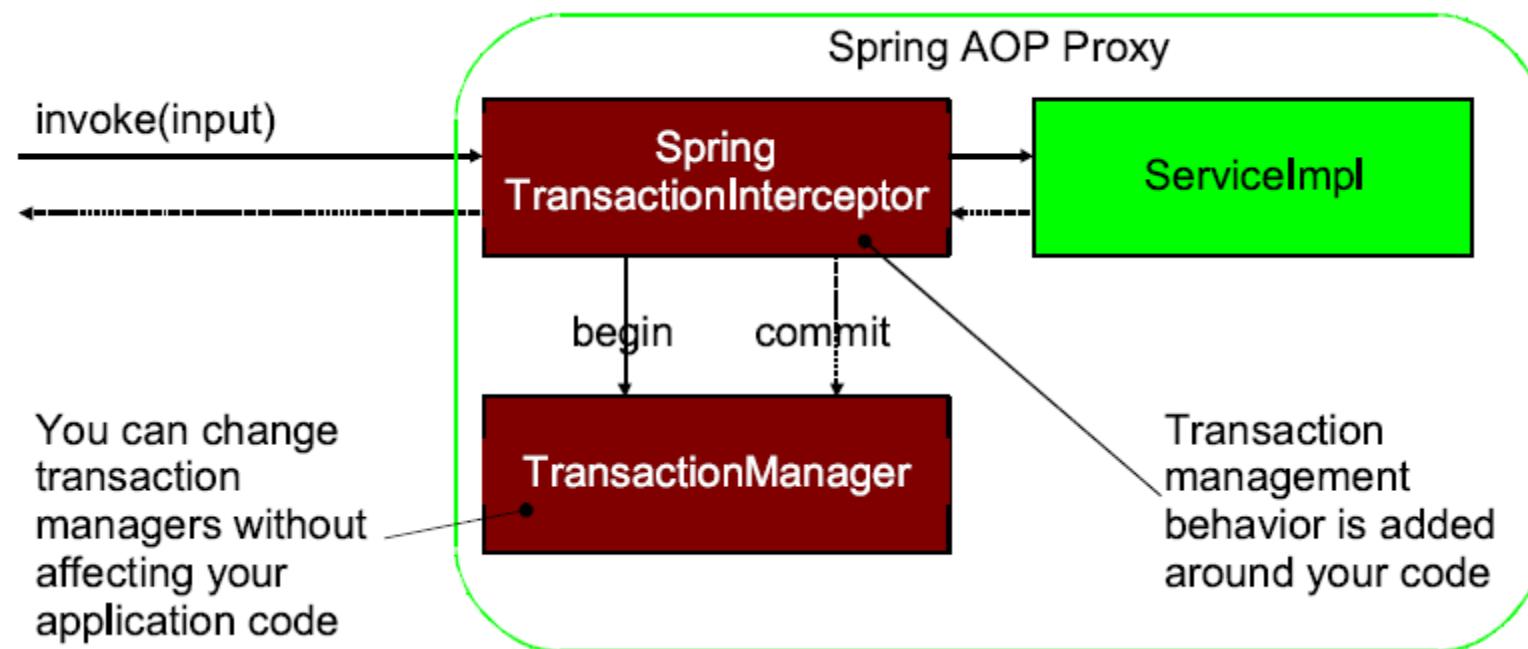
- Спринг добавляет вызовы управления транзакциями вокруг бизнес методов
- Достаточно объявить политику транзакций и Спринг их выполняет

Объявление политики транзакций

```
public class ServiceImpl implements ServiceInterface {  
    @Transactional  
    public void invoke(...) {  
        // your application logic  
    }  
}
```

Tells Spring to always run this method
in a database transaction

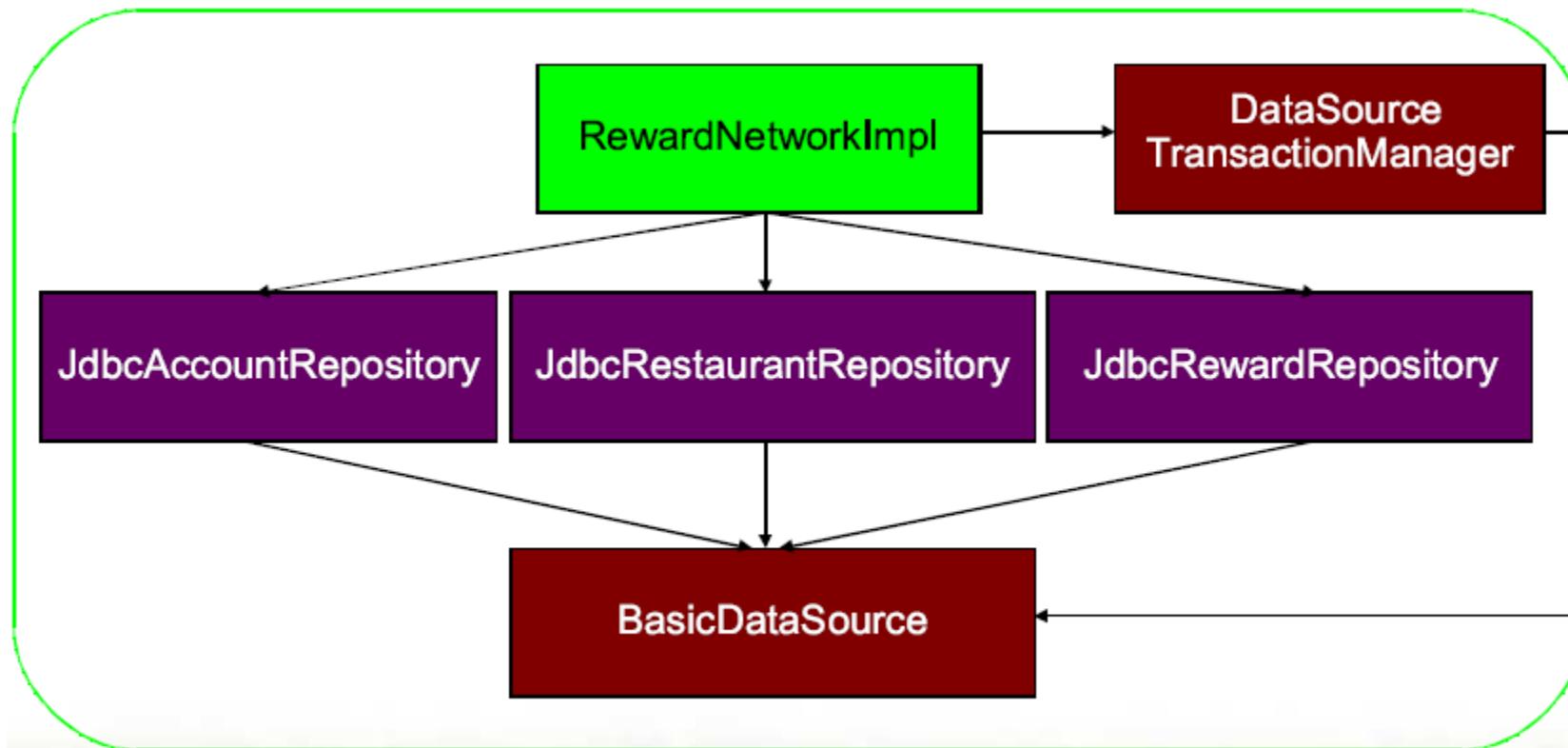
Управление транзакциями в Спринге



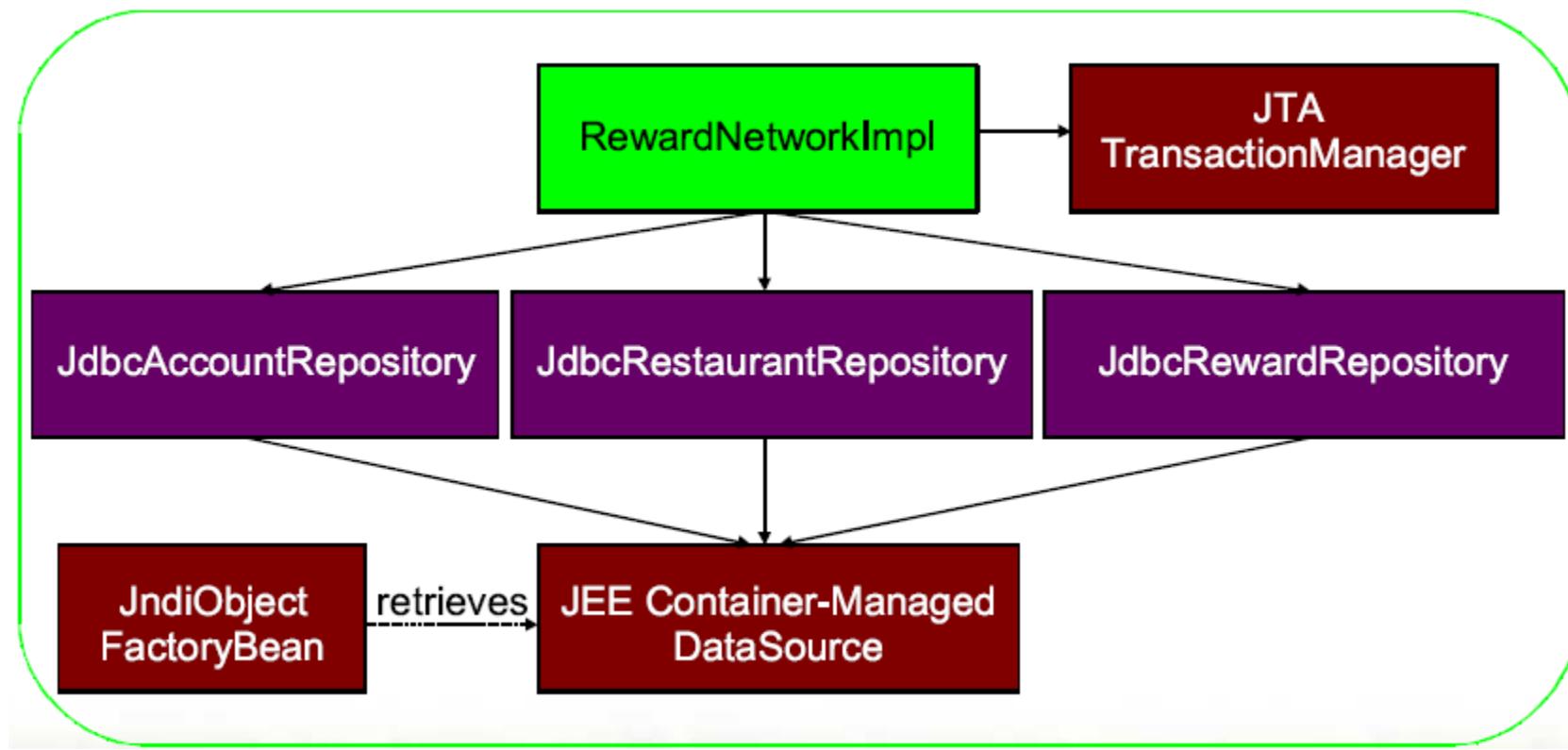
Управление соединениями

- Соединения управляются за нас
 - В соответствии с границами транзакций
 - Не наша головная боль
 - Нет утечек
- Репозитории всегда работают в транзакциях
 - У Спринга есть несколько способов этого достичь

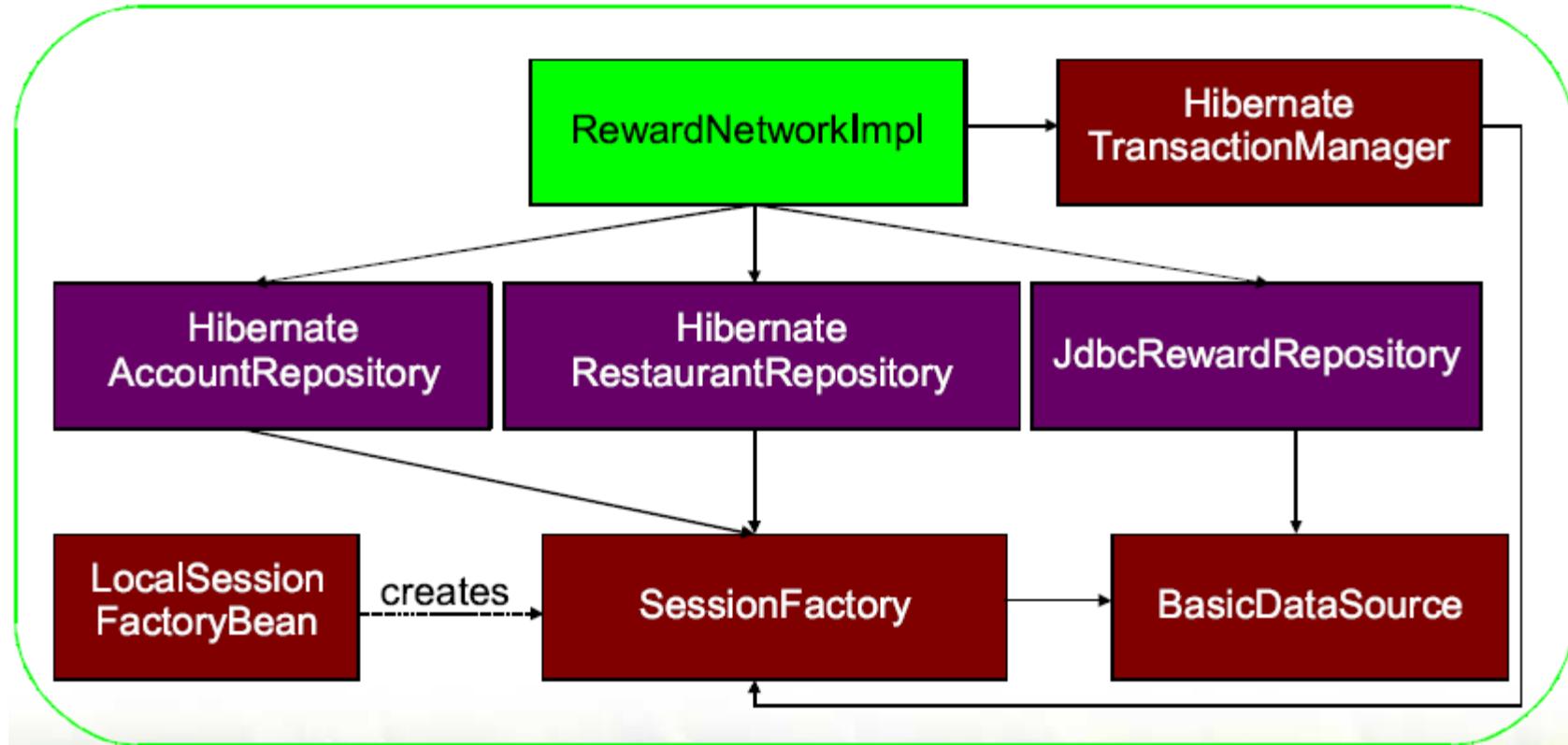
Локальная конфигурация JDBC



Конфигурация JDBC в Java EE Контейнере



Локальная Конфигурация Hibernate



Поддержка Data Access

Purpose	JDBC	Hibernate	TopLink
Technology configuration support	Several local DataSource implementations; JNDI lookup support	LocalSessionFactoryBean; JNDI lookup support	SessionFactoryBean; JNDI lookup support
DAO implementation support	JdbcDaoSupport	HibernateDaoSupport	TopLinkDaoSupport
API Helper	JdbcTemplate	HibernateTemplate	TopLinkTemplate
Transaction management	DataSource Transaction Manager or JTA	HibernateTransaction Manager or JTA	TopLinkTransaction Manager or JTA

Введение в Spring JDBC

Ад JDBC

```
public List findByLastName(String lastName) {
    List personList = new ArrayList();
    Connection conn = null;
    String sql = "select first_name, age from PERSON where last_name=?";
    try {
        DataSource dataSource = DataSourceUtils.getDataSource();
        conn = dataSource.getConnection();
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, lastName);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            String firstName = rs.getString("first_name");
            int age = rs.getInt("age");
            personList.add(new Person(firstName, lastName, age));
        }
    } catch (Exception e) { /* ??? */ }
    finally {
        try {
            conn.close();
        } catch (SQLException e) { /* ??? */ }
    }
    return personList;
}
```

Ад JDBC

```
public List findByLastName(String lastName) {
    List personList = new ArrayList();
    Connection conn = null;
    String sql = "select first_name, age from PERSON where last_name=?";
    try {
        DataSource dataSource = DataSourceUtils.getDataSource();
        conn = dataSource.getConnection();
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, lastName);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            String firstName = rs.getString("first_name");
            int age = rs.getInt("age");
            personList.add(new Person(firstName, lastName, age));
        }
    } catch (Exception e) { /* ??? */ }
    finally {
        try {
            conn.close();
        } catch (SQLException e) { /* ??? */ }
    }
    return personList;
}
```

The bold matters - the
rest is boilerplate

Исключения – тоже не подарок

```
public List findByLastName(String lastName) {
    List personList = new ArrayList();
    Connection conn = null;
    String sql = "select first_name, age from PERSON where last_name=?";
    try {
        DataSource dataSource = DataSourceUtils.getDataSource();
        conn = dataSource.getConnection();
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, lastName);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            String firstName = rs.getString("first_name");
            int age = rs.getInt("age");
            personList.add(new Person(firstName, lastName, age));
        }
    } catch (Exception e) { /* ??? */ }
    finally {
        try {
            conn.close();
        } catch (SQLException e) { /* ??? */ }
    }
    return personList;
}
```

What can
you do?

JdbcTemplate

- Упрощает работу с JDBC API
 - Убивает весь лишний код
 - Не дает писать баги
 - Обрабатывает SQLExceptions как надо
- Без ограничений
 - Полный доступ ко всем конструкциям JDBC

JdbcTemplate в одном слайде

```
int count = jdbcTemplate.queryForInt(  
    "SELECT COUNT(*) FROM CUSTOMER");
```

- Получение подключения
- Работа в транзакции
- Выполнение вызова
- Обработка результата
- Обработка исключений
- Отключение

All handled
by Spring

Подход JdbcTemplate

```
List results = jdbcTemplate.query(sql,
    new RowMapper() {
        public Object mapRow(ResultSet rs, int row) throws SQLException {
            // map the current row to an object
        }
    });
}

class JdbcTemplate {
    public List query(sql, rowCallback) {
        try {
            // acquire connection
            // prepare statement
            // execute statement
            // for each row in the result set
            results.add(rowCallback.mapRow(rs, rowNum));
        }
        return results;
    } catch (SQLException e) {
        // convert to root cause exception
    } finally {
        // release connection
    }
}
```

Создаем JdbcTemplate

- Нужен DataSource

```
JdbcTemplate template = new JdbcTemplate(dataSource);
```

- Можно использовать повторно
 - Не нужно каждый раз новый
 - Thread safe после создания

Когда использовать JdbcTemplate

- Сам по себе
 - Когда нужен JDBC
 - Во всяких утилитах и в тестах
 - Разгребая старый код
- Для реализации **репозитория** в многоуровневой системе
 - A.K.A. data access object (DAO)

Реализация репозитория на JDBC

- Создаем новый класс
 - Реализуя интерфейс доступа к данным
- Реализуем методы с помощью JdbcTemplate или SimpleJdbcTemplate
- Тестируем
- Интегрируем в приложение
 - Где он будет участвовать в бизнес-транзакциях

Создаем класс репозитория на JDBC

- Можно использовать супер-класс как основу
 - SimpleJdbcDaoSupport
 - Получаем JdbcTemplate по наследству, но нужно впрыснуть DataSource

Реализуем репозиторий на JDBCRepository

```
public class JdbcCustomerRepository implements CustomerRepository {  
  
    private SimpleJdbcTemplate jdbcTemplate;  
  
    public JdbcCustomerRepository(DataSource dataSource) {  
        this.jdbcTemplate = new SimpleJdbcTemplate(dataSource);  
    }  
  
    public int getCustomerCountOlderThan(int age) {  
        String sql = "select count(*) from customer where age > ?";  
        return jdbcTemplate.queryForInt(sql, age);  
    }  
}
```

Интегрируем репозиторий в приложение

```
<bean id="creditReportingService" class="example.CreditReportingService">
    <property name="orderRepository" ref="orderRepository" />
    <property name="customerRepository" ref="customerRepository" />
</bean>
<jee:jndi-lookup id="d" > Configure the repository's DataSource <!-- bc/credit -->
<bean id="orderRepository" class="example.order.JdbcOrderRepository">
    <constructor-arg ref="dataSource"/>
</bean>

<bean id="customerRepository"
      class="example.customer.JdbcCustomerRepository">
    <constructor-arg ref="dataSource" />
</bean>
```

↑
Inject the repository into application services

↑
Configure the repository's DataSource

Запросы с JdbcTemplate

- JdbcTemplate умеет запрашивать:
- Простые типы(int, long, String)
 - Generic Maps
 - Domain Objects

Простые типы

```
public int getPersonCount() {  
    String sql = "select count(*) from PERSON";  
    return jdbcTemplate.queryForInt(sql);  
}
```

Простые типы

```
private SimpleJdbcTemplate jdbcTemplate;  
  
public int getCountOfPersonsOlderThan(int age) {  
    return jdbcTemplate.queryForInt(  
        "select count(*) from PERSON where age > ?", age);  
}
```

No need for `new Object[] { new Integer(age) }`



Общие запросы

- JdbcTemplate может возвращать каждую строчку ResultSet как Map
- Если строчка ожидается одна:
 - Use queryForMap(..)
- Если больше чем одна:
 - Use queryForList(..)
- Полезно, когда не нужен объект

Общие запросы

- Запрос на одну строку

```
public Map getPersonInfo(int id) {  
    String sql = "select * from PERSON where id=?";  
    return jdbcTemplate.queryForMap(sql, id);  
}
```

- Ответ:
Map { ID=1, FIRST_NAME="John", LAST_NAME="Doe" }

Общие запросы

- Запрос на много строчек

```
public List getAllPersonInfo() {  
    String sql = "select * from PERSON";  
    return jdbcTemplate.queryForList(sql);  
}
```

- Ответ:

```
List {  
    0 - Map { ID=1, FIRST_NAME="John", LAST_NAME="Doe" }  
    1 - Map { ID=2, FIRST_NAME="Jane", LAST_NAME="Doe" }  
    2 - Map { ID=3, FIRST_NAME="Junior", LAST_NAME="Doe" }  
}
```

Строительство Domain Object

- Мапирование из базы данных в объекты
 - Например ResultSet в Account
- JdbcTemplate поддерживает это через коллбэки

Строим Domain Objects

- Одна строчка

```
public Person getPerson(int id) {  
    return (Person) jdbcTemplate.queryForObject(  
        "select first_name, last_name from PERSON where id=?",  
        new Object[] { new Long(id) },  
        new PersonMapper()); ← Maps rows to Person objects  
}
```

```
class PersonMapper implements RowMapper {  
    public Object mapRow(ResultSet rs, int i) throws SQLException {  
        return new Person(rs.getString(1), rs.getString(2));  
    }  
}
```

Строим Domain Objects

- Много строк

```
public List getAllPersons() {  
    return jdbcTemplate.query(  
        "select first_name, last_name from PERSON",  
        new PersonMapper());  
}
```

Same row mapper can be used

```
class PersonMapper implements RowMapper {  
    public Object mapRow(ResultSet rs, int i) throws SQLException {  
        return new Person(rs.getString(1), rs.getString(2));  
    }  
}
```

Один объект с SimpleJdbcTemplate

```
public Person getPerson(int id) {  
    return jdbcTemplate.queryForObject(  
        "select first_name, last_name from PERSON where id=?",  
        new PersonMapper(), id);  
}
```

No need to cast

```
class PersonMapper implements ParameterizedRowMapper<Person> {  
    public Person mapRow(ResultSet rs, int i) throws SQLException {  
        return new Person(rs.getString(1), rs.getString(2));  
    }  
}
```

Parameterizes return type

Много объектов с SimpleJdbcTemplate

```
public List<Person> getAllPersons() {  
    return jdbcTemplate.query(  
        "select first_name, last_name from PERSON",  
        new PersonMapper());
```

No need to cast

```
class PersonMapper implements ParameterizedRowMapper<Person> {  
    public Person mapRow(ResultSet rs, int i) throws SQLException {  
        return new Person(rs.getString(1), rs.getString(2));  
    }  
}
```

RowCallbackHandler

- Удобен, если не нужно возвращать значение
 - Например, записать строчки в файл

```
public interface RowCallbackHandler {  
    void processRow(ResultSet rs) throws SQLException;  
}
```

RowCallbackHandler

```
public class JdbcOrderRepository {  
    public void generateReport() {  
        // select all orders for a full report  
        jdbcTemplate.query("select...from order o, item i",  
            new OrderReportWriter());  
    }  
}
```

```
class OrderReportWriter implements RowCallbackHandler {  
    public void processRow(ResultSet rs) throws SQLException {  
        // stream row to a file  
    }  
}
```

ResultSetExtractor

- ResultSetExtractor удобен, если нужно смапировать весь ResultSet в объект

```
public interface ResultSetExtractor {  
    Object extractData(ResultSet rs) throws SQLException,  
        DataAccessException;  
}
```

ResultSetExtractor

```
public class JdbcOrderRepository {  
    public Order findByConfirmationNumber(String number) {  
        // execute an outer join between order and item tables  
        return (Order) jdbcTemplate.query(  
            "select...from order o, item i...conf_id = ?",
            new Object[] { number },
            new OrderExtractor());  
    }  
}
```

```
class OrderExtractor implements ResultSetExtractor {  
    public Object extractData(ResultSet rs) throws SQLException {  
        // create an Order object from multiple rows  
    }  
}
```

Интерфейсы коллбэков

- RowMapper
 - Если каждая строчка ResultSet мапируется в объект
- RowCallbackHandler
 - Если не нужно возвращать значение
- ResultSetExtractor
 - Если нужна больше чем одна строчка ResultSet чтобы создать объект

Inserts иUpdates

- Insert

```
public int insertPerson(Person person) {  
    return jdbcTemplate.update(  
        "insert into PERSON (first_name, last_name, age)" +  
        "values (?, ?, ?)",  
        person.getFirstName(),  
        person.getLastName(),  
        person.getAge());  
}
```

Inserts и Updates

- Update

```
public int updateAge(Person person) {  
    return jdbcTemplate.update(  
        "update PERSON set age=? where id=?",  
        person.getAge(),  
        person.getId());  
}
```

Ад SQLExceptions

- SQLExceptions чаще всего не ловятся
- Но если ловятся, то они leaky abstraction
 - Завязывают на конкретную технологию
 - И с тестами все плохо
- Использование JdbcTemplate гарантирует правильную обработку

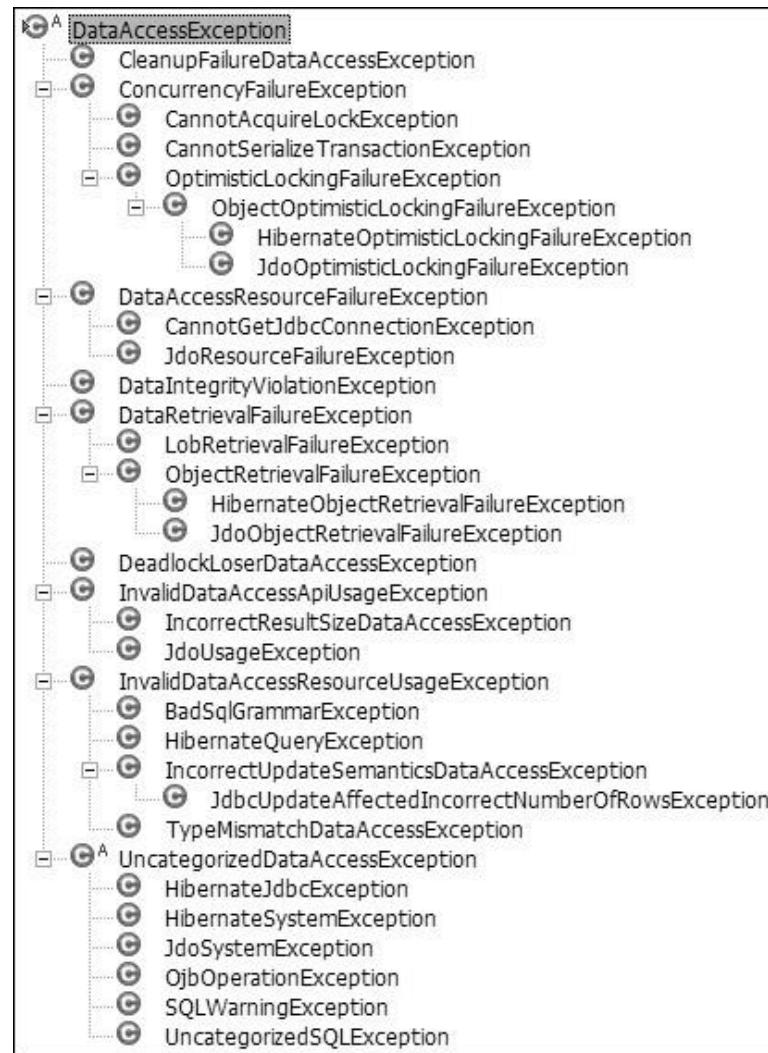
Ад SQLExceptions

- Часто непонятно в чем проблема
 - Всякие коды ошибок из разных баз данных
- SQLException is a leaky abstraction
 - Checked exception: приходится объявлять в сигнатуре методов!
- Чаще всего делается одно из двух:
 - “Catch and wrap”
 - Вообще проглатываются

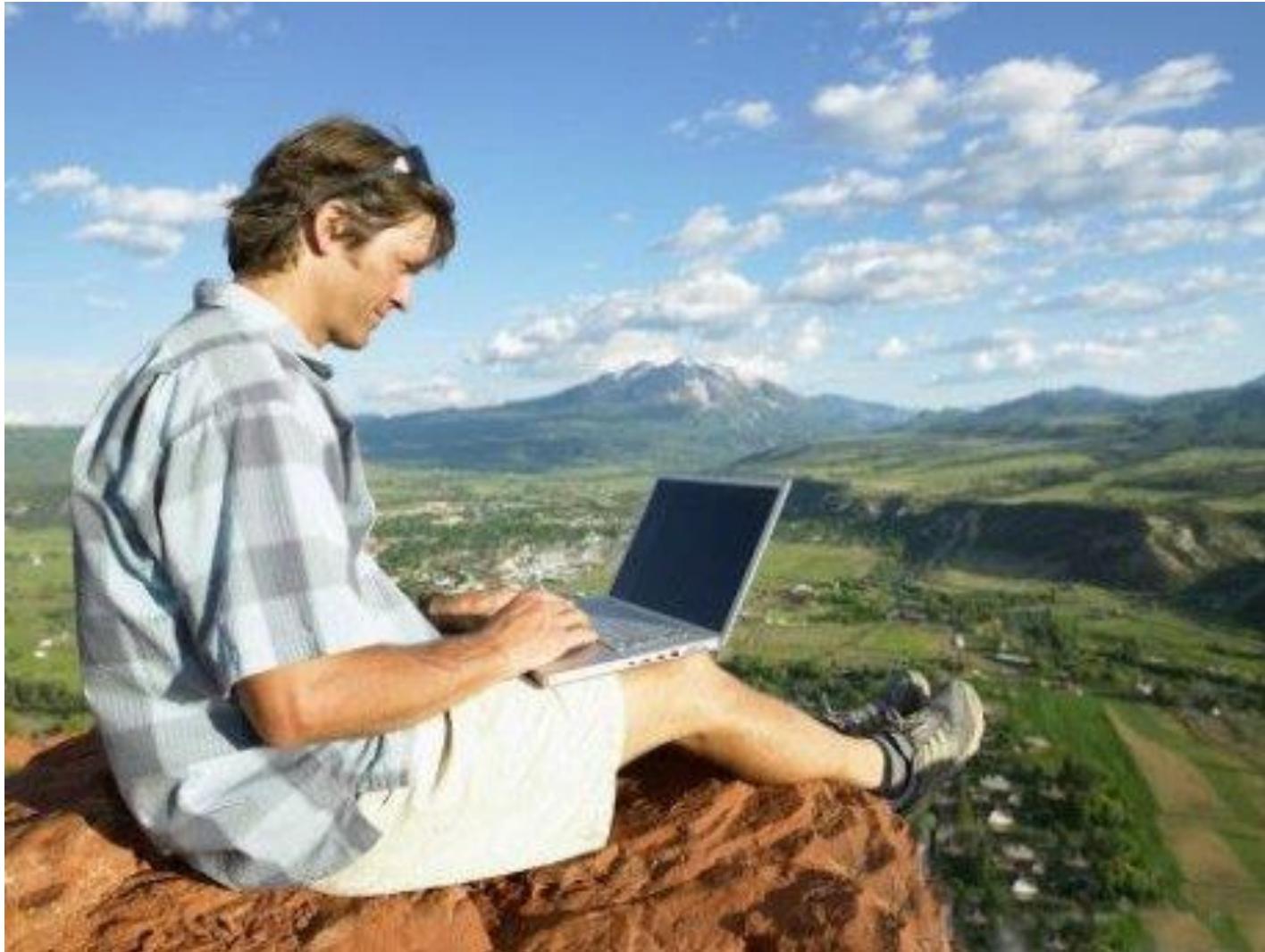
Обработка SQLEception

- SQLEception обрабатываются консистентно
 - Ресурсы высвобождаются как следует
- Общие SQLEception переводятся в конкретные, с корневой ошибкой
- DataAccessExceptions
 - Консистентные вне зависимости от конкретной технологии
 - Не надо постоянно catch-and-wrap
 - Можно обрабатывать в зависимости от типа
 - Всегда всплывают, если не пойманы

Иерархия DataAccessException

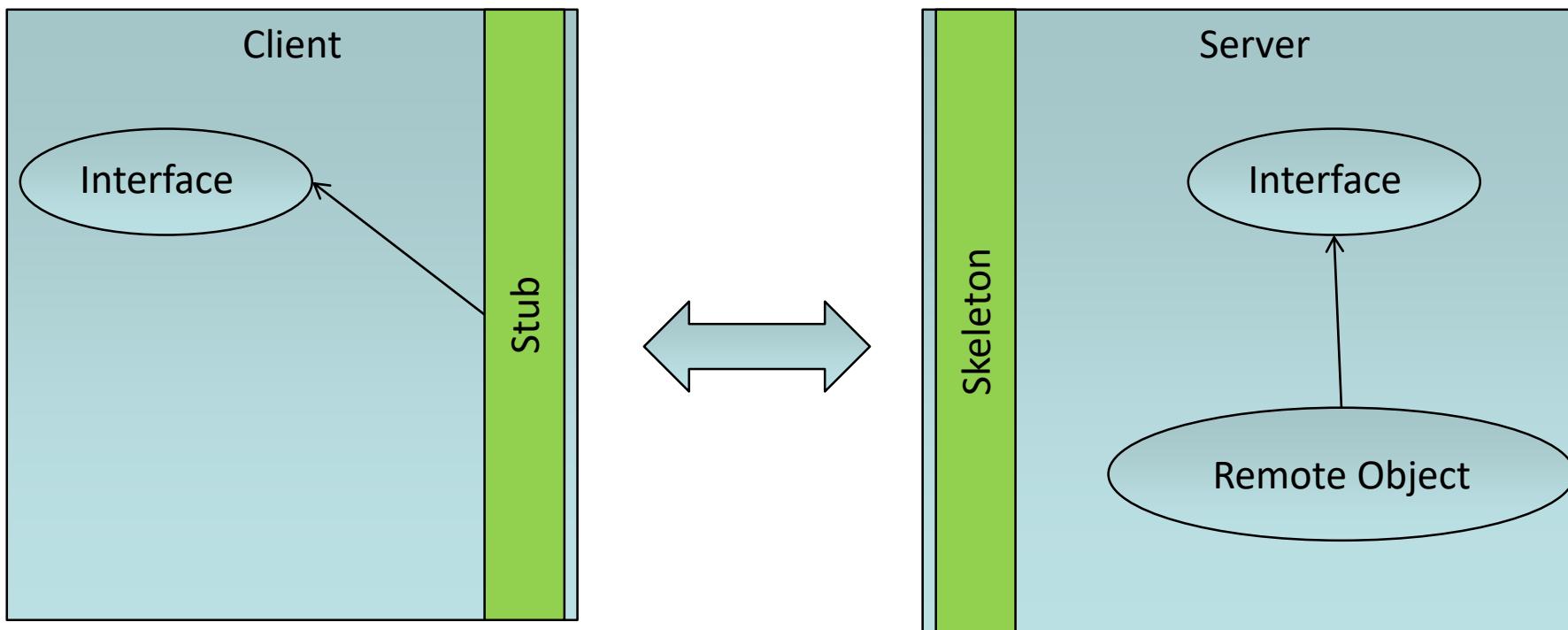


Spring Remoting



Как оно работает

- RMI архитектура:



Сервер

- We are going to expose the following POJO:

```
public interface Greeter{  
    String greet();  
}
```

```
public class GreeterImpl implements Greeter{  
    private String message;  
  
    public void setMessage(String message) {  
        this.message = message;  
    }  
    @Override  
    public String greet() {  
        return message;  
    }  
}
```

Конфигурация на сервере

```
<beans ...>

    <bean name="greeter"
          class="com.inwhite.examples.spring.remoting.GreeterImpl">
        <property name="greet" value= "Привет Доичь Банк!" />
    </bean>

    <bean class="org.springframework.remoting.rmi.RmiServiceExporter">
        <property name="serviceName" value="GreeterService" />
        <property name="service" ref="greeter" />
        <property name="serviceInterface"
                  value="com.inwhite.examples.spring.remoting.Greeter" />
        <property name="registryPort" value="1199" />
    </bean>

</beans>
```

Конфигурация у клиента

```
<beans>

    <bean id="greeter"
          class="org.springframework.remoting.rmi.RmiProxyFactoryBean">
        <property name="serviceUrl"
                  value="rmi://localhost:1199/GreeterService" />
        <property name="serviceInterface"
                  value="com.trainologic.examples.spring.remoting.Greeter" />
    </bean>

</beans>
```

Код клиента

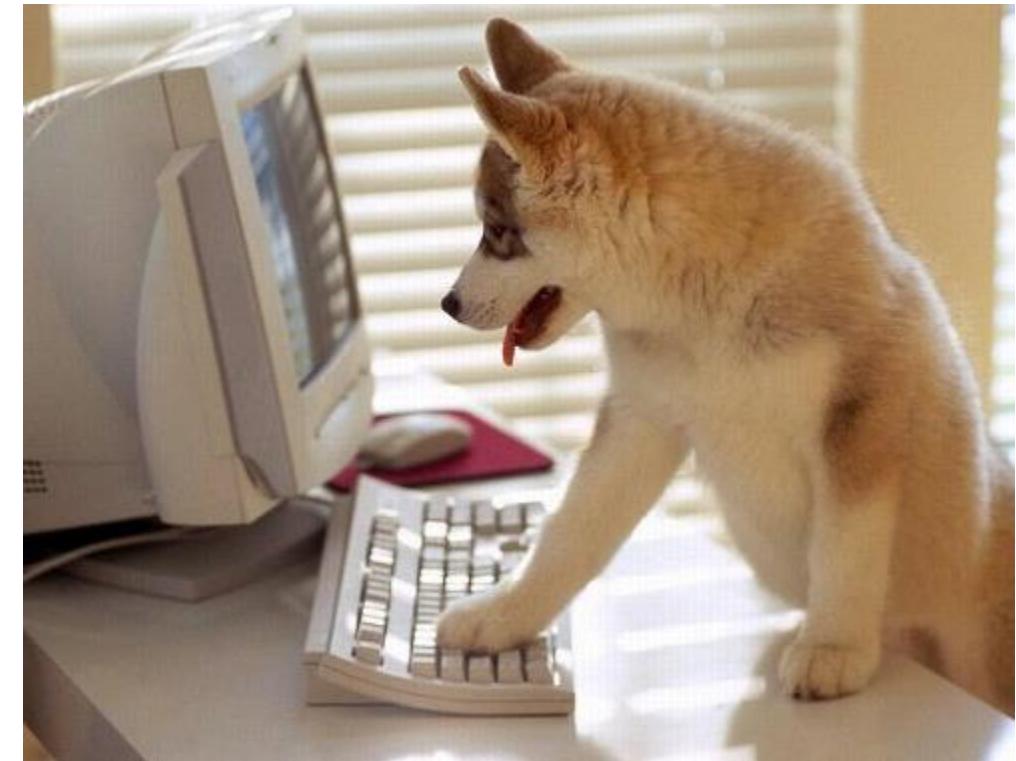
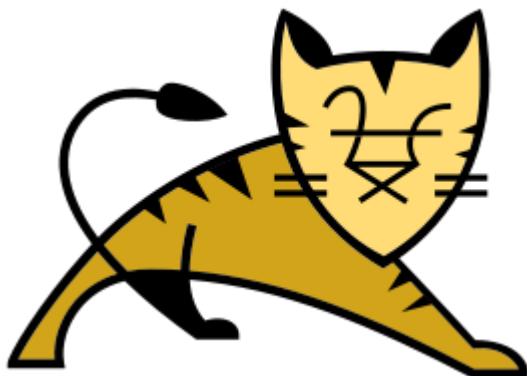
```
public static void main(String[] args) {  
    context= new FileSystemXmlApplicationContext("applicationContext.xml");  
  
    Greeter greeter = context.getBean(Greeter.class);  
  
    System.out.println(greeter.greet());  
}
```

Задание

- Сделайте тоже самое, но используя Java Config

Hessian & Burlap

- Если мы хотим запускать удалённые методы объектов, которые находятся в сервере приложений или в веб контейнере



Server Side

```
<web-app>
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
  <servlet>
    <servlet-name>hessianGreeter</servlet-name>
    <servlet-class>org.springframework.web.context.support.HttpRequestHandlerServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>hessianGreeter</servlet-name>
    <url-pattern>/remoting/samplehessian.do</url-pattern>
  </servlet-mapping>

</web-app>
```

Server Side – application-context.xml

```
<bean name="hessianGreeter" class="org.springframework.remoting.caucho.HessianServiceExporter">
    <property name="service" ref="greeter"/>
    <property name="serviceInterface" value="common.Greeter"/>
</bean>
```

Client Side – application-context.xml

```
<bean id="hessianGreeter"
      class="org.springframework.remoting.cauchy.HessianProxyFactoryBean">
    <property name="serviceUrl"
              value="http://localhost:8080/remoting/samplehessian.do"/>
    <property name="serviceInterface" value="common.Greeter"/>
</bean>

<context:component-scan base-package="client"/>
```

Validations

- Валидации являются часть спека JSR-310
- Основная идея заключается в том, чтобы всю повторяющуюся логику проверок держать в одном месте
- Но как водится у большинства JSR-ов, своей имплементации у них нет
- Поэтому будем пользоваться Хайбернетовскими

Пример валидного работника

```
public class Employee {  
    @NotNull  
    private String name;  
  
    @AssertFalse  
    private boolean всехДостал;  
  
    @Min(18)  
    @Max(60)  
    private int age;  
  
    @Past  
    private Calendar birthday;  
}
```



А кто будет делать валидации?

Спринг конечно

```
@Bean public LocalValidatorFactoryBean validatorFactoryBean() {  
    return new LocalValidatorFactoryBean();  
}  
  
@Autowired private Validator validator;  
public void printErrors(Employee employee) {  
    Set<ConstraintViolation<Employee>> violationSet = validator.validate(employee);  
    for (ConstraintViolation<Employee> violation : violationSet) {  
        System.out.println(violation.getMessage());  
    }  
}
```

Это потому что спринг не пишет
имплементаций



Добавте джар Хайберната

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>5.0.1.Final</version>
</dependency>
```

Встроенные Constraints

- @AssertFalse: checks that the field/property is *false*.
- @AssertTrue: checks that the field/property is *true*.
- @DecimalMax: checks that the field/property (must be one of: *BigDecimal*, *BigInteger*, *String*, *byte*, *short*, *int*, *long*, or wrappers) is lower or equal to the annotation's parameter.
- @DecimalMin: the same as @DecimalMax only for minimum value.

Встроенные Constraints

- @Digits: Works on the same types as `@DecimalMax` and checks for a defined maximum number of digits and a number of fractional digits.
- @Future: checks whether the field/property (*Date* or *Calendar*) is in the future.
- @Past: same as `@Future`, but for the past.
- @Max: like `@DecimalMax`, only that the parameter is a number and not a *String*.
- @Min: well, the opposite of `@Max`.

Встроенные Constraints

- @NotNull.
- @Null.
- @Pattern: checks for match against the supplied regular expression.
- @Size: checks whether the element's size (*String, Collection, Map* or array) is in a specified range.
- @Valid: recursively performs validations on the annotated member/property.

Задание

- Напишите класс Водитель
 - Имя не может быть null
 - Возраст должен быть больше чем 18
 - Не пьющий
-
- Проверьте валидатором объект водителя и распечатайте все ошибки

Добавь message в аннотацию



Следующее задание

- А теперь напишите Taxi, в который будет впрыскиваться валидный Driver
- Проверьте при помощи валидатора, что Taxi валидный.
- В данном случае это значит, что водитель, который в него впрыснут прошел все проверки

Создание кастомных constraints

- Create a constraint annotation
- Implement a validator
- Define a default error message
- Сейчас покажу вживую

Задание

- Написать кастомную аннотацию `@ValidCredentials`
- Которая будет проверять что у объекта логин пароль не длиннее, чем имя

```
public class User {  
    @ValidCredentials  
    private Login login;
```

Hibernate – я задолбался без него



Persistence

- Persistence is often a required application feature.
- Persistence means that the data in the objects is retained between program executions.
- The most popular persistence storage is a relational database.

ORM Solution

- Mapping the object domain to a relational domain is a complicated task.
- Many projects have failed because they tried to develop an in-house persistence framework.
- The common solution today is to use a library, an **ORM** library.
- ORM = Object/Relational Mapping.

JPA History

- In the previous version (J2EE 1.4/EJB 2.1) the persistence solution was: Entity Beans.
- Entity Beans provided a poor ORM solution.
- Some major problems:
 - No support for inheritance/ polymorphism.
 - Weak query language (e.g., no support for group by).
- Most projects didn't use it!

Говорим Ленин – подразумеваем Партия,
Говорим Партия - подразумеваем Ленин



Говорим JPA – подразумеваем HIBERNATE,
Говорим HIBERNATE- подразумеваем JPA



Hibernate

- Many projects opted to use some non-standard ORM framework.
- The most popular ORM framework was Hibernate.
- The persistence solution of JEE 5 (JPA) is largely based on Hibernate.



Какие базы данных поддерживает hibernate

- Oracle
- DB2
- Microsoft SQL Server
- Sybase
- MySQL
- PostgreSQL
- TimesTen
- HypersonicSQL
- SAP
- InterSystems Cache
- Apache Derby
- HP NonStop SQL/MX
- Firebird
- FrontBase
- Informix
- Ingres
- Interbase (6.0.1 tested)
- Mckoi SQL
- Pointbase Embedded
- Progress 9
- Microsoft Access
- Corel Paradox
- flat text , CSV file, TSV file, fixed-length, and variable-length binary file
- Xbase database
- Microsoft Excel

В чём суть?

- The general mapping idea is:
 - Map a class to a table.
 - Map a member/property to a column.
- Of course, there are many different variations and exceptions to these guidelines.
- We will go into it later...

Что такое Entity

- Класс, который является отражением таблицы
- Рекомендуется, чтобы такой класс:
 - Соблюдал конвенции JavaBeans
 - Имплементировал **Serializable**
- Обязательно:
 - Пустой конструктор
 - Если есть primary key нужен **@id** property
 - Нельзя final

Persistent Fields

- Mapping can be done either by fields or by properties.
- Mapping by properties means that the JPA framework will use the getters/setters convention for the mapping.
- By default all the fields/properties are mapped.
- We will see how to control this in the next slides...

Mapping

- Mapping can be using annotations, XML files or both.
- Note, if both are used, the XML configuration overrides the annotations.
- All configuration/mapping options can be done in both ways.
- In this course, we will use the annotation-based configuration.

The Entity Annotation

- To state that a class is mapped in JPA, it must be annotated with **@Entity**.
- By default, the class will be mapped to a table that has the same name as the class.
- Also, the properties will be mapped to columns that have the same names.

Primary Key

- Each entity must have a primary key field/property.
- The annotation used to map it is: **@Id**.
- The primary key will be mapped to the primary key column in the DB.
- By default, the column name is the same as the mapped field.
- For now, we will concentrate on simple primary keys.

```
@Entity           // marking the class as being MAPPED
public class Company {
    private String name;
    private String symbol;
    private int id;

    public Company() {
        // required constructor for JPA
    }
    public Company(String name, String symbol) {
        this.name = name;
        this.symbol = symbol;
    }

    @Id      // marking the primary key
    public int getId() {return id;}
    public void setId(int id) {this.id = id; }
    public String getName() {return name; }
    public void setName(String name) {this.name = name; }
    public String getSymbol() {return symbol; }
    public void setSymbol(String symbol) {this.symbol = symbol; }
}
```

Controlling The Table Name

- By default, the table name will be the same as the class name.
- In order to change this, use the **@Table** annotation.
- Important members of @Table:
 - name – the name of the table.
 - catalog.
 - schema.

Controlling The Column Name

- By default, the column name will be the same as the field/property name.
- In order to change this, use the **@Column** annotation.
- Important members of @Column:
 - name – the name of the column.
 - insertable – controls whether this column is included in the generated INSERT statement.
 - updatable – controls whether this column is included in the generated UPDATE statement.

Persistent Fields

- By default, all the fields/properties are mapped.
- In order to specify an **unmapped** field/property, use the **@Transient** annotation.
- If you are using field-access, the field will not be persisted if it is declared as *transient*.

Field Types

- According to the field type, some annotations may be present:
 - All persistent fields which are not a part of a relationship can be annotated with the **@Basic** annotation.
 - `java.util.Date` and `java.util.Calendar` types **must** be annotated with the **@Temporal** annotation.
 - Enums can be annotated with the **@Enumerated** annotation.

@Basic Annotation

- The @Basic annotation is optional.
- With the @Basic annotation you can provide the following members:
 - fetch – fetching mode, defaults to: Eager.
 - optional – controls whether the field can be null. Defaults to *true* (used for schema generation).

@Temporal Annotation

- The @Temporal annotation must be used for persistent fields of the types: Date & Calendar.
- Its value is either DATE, TIME or TIMESTAMP.
- It is used to map between a Java object and an appropriate column representation.

Суррогаты

или

натуральные?



Primary Keys

- A primary key can be either simple (a single column) or composite (several columns).
- By default, the value for the primary key is assigned by the program (like any other field).
- However, the primary key can also be **generated**.
- This is done by using the **@GeneratedValue** annotation.

@GeneratedValue

- The @GeneratedValue annotation specifies that the primary key is generated.
- It has the following members:
 - strategy: can be one of *TABLE*, *SEQUENCE*, *IDENTITY* or *AUTO* (default).
 - generator: the name of the generator (used for *SEQUENCE* or *TABLE*).

Generation Type

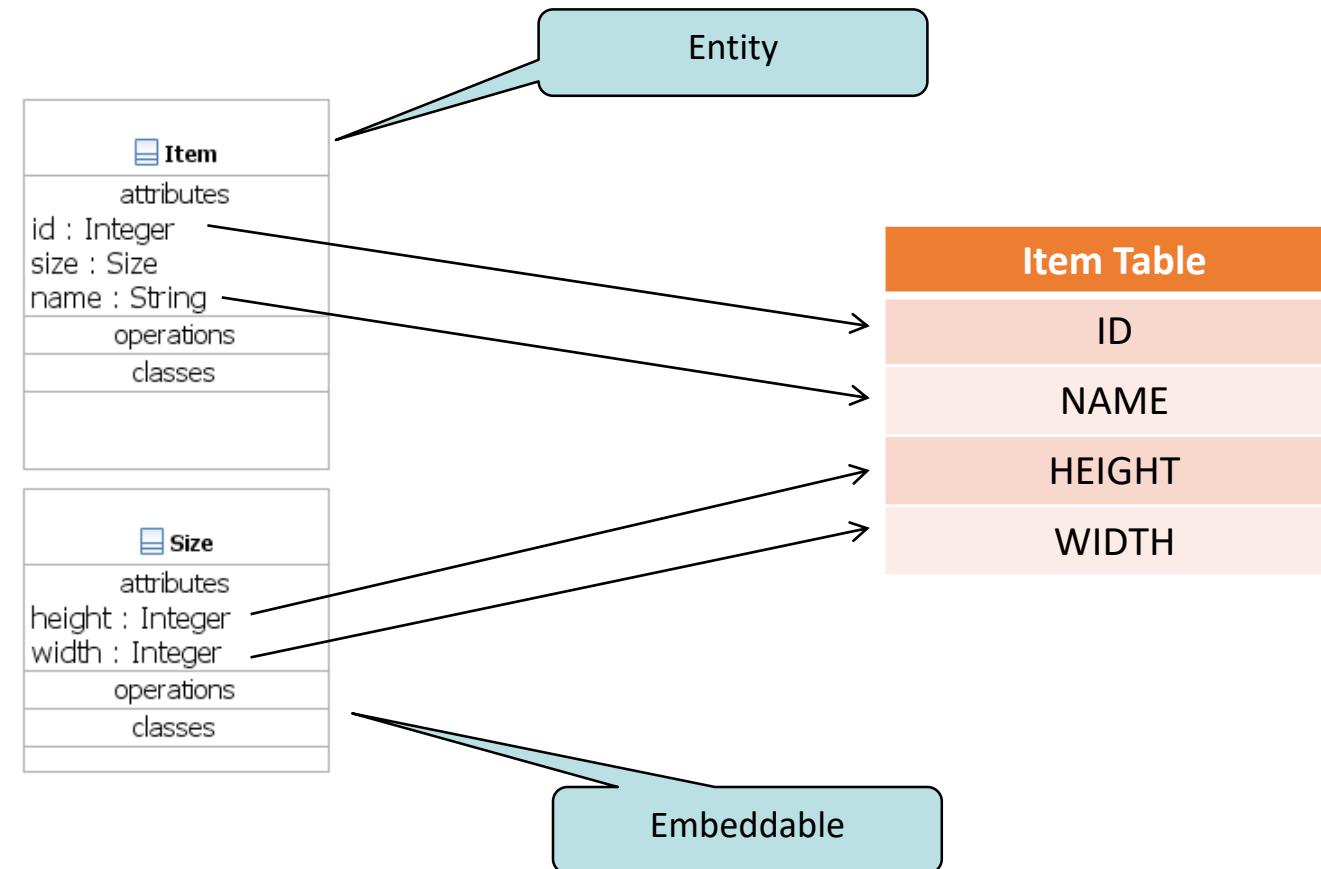
- TABLE: the framework will use a special table for generating unique keys (used with `@TableGenerator`).
- SEQUENCE: the framework will use a sequence for generating unique keys (used with `@SequenceGenerator`).
- IDENTITY: the framework will use an auto-increment column.
- AUTO: the framework will choose the appropriate mechanism, depending on the underlying database.

```
@Entity  
@SequenceGenerator(name="CUSTOMER_SEQ", sequenceName="SEQ_T4471")  
class Employee {  
  
    @Id  
    @GeneratedValue(strategy=GenerationType.SEQUENCE, generator="CUSTOMER_SEQ")  
    @Column(name="CUSTOMER_ID")  
    public Long getId() { return id; }
```

Mapping Granularity

- Sometimes, the “class-per-table” approach is not sufficient.
- For example, we have the class *Size* that has width and height properties.
- The class *Item* contains a *Size* property.
- However, in the DB there is only one table: *Item*.
- See diagram on next page...

Mapping Granularity



Mapping Granularity

- As you can see, not every class is mapped to a separate table.
- Some classes should be **embedded** into others in the DB.
- In the previous example, the *Size* class is embedded into the *Item* class (from persistence point-of-view).

@Embeddable

- A class can be annotated with the **@Embeddable** annotation.
- This annotation specifies that the class is **not** mapped to its own table, but into its container's table.
- Of course, the embeddable object can be used in different classes and thus be mapped into different tables.

Example

```
@Embeddable          // marking the class as Embeddable
public class Size {
    private int width;
    private int height;

    public Size() {           // required no-args constructor
    }

    public Size( int width, int height) {
        this.height = height;
        this.width = width;
    }

    public int getWidth() {return width;}
    public void setWidth(int width) {this.width = width;}
    public int getHeight() {return height;}
    public void setHeight(int height) {this.height = height;}
}
```

@Embedded

```
@Entity
public class Item {
    private String name;
    private int id;
    private Size size;

    public Item() {}
    public Item(String name) {this.name = name;}

    @Id
    public int getId() {return id;}
    public void setId(int id) {this.id = id;}
    @Embedded
    public Size getSize() {return size;}
    public void setSize(Size size) {this.size = size;}
    public String getName() {return name;}
    public void setName(String name) {this.name = name;}
}
```

@AttributeOverride

- When you embed an *Embeddable* class, you can control the mapping of its properties.
- You do so by using the **@AttributeOverrides** annotation.
- Lets see an example...

Example

```
@Embeddable // We changed this class property names
public class Size {
    private int w;
    private int h;

    public Size() { }

    public Size(int w, int h) {
        this.h = h;
        this.w = w;
    }

    public int getW() { return w; }

    public void setW(int w) {this.w = w; }

    public int getH() {return h; }

    public void setH(int h) {this.h = h; }

}
```

Example

```
@Embedded  
@AttributeOverrides( {  
    @AttributeOverride(name = "w", column = @Column(name = "WIDTH")),  
    @AttributeOverride(name = "h", column = @Column(name = "HEIGHT"))  
})  
public Size getSize() {  
    return size;  
}
```

Вот таблица

Event	
PK	<u>EVENT_ID</u>
	EVENT_DATE title

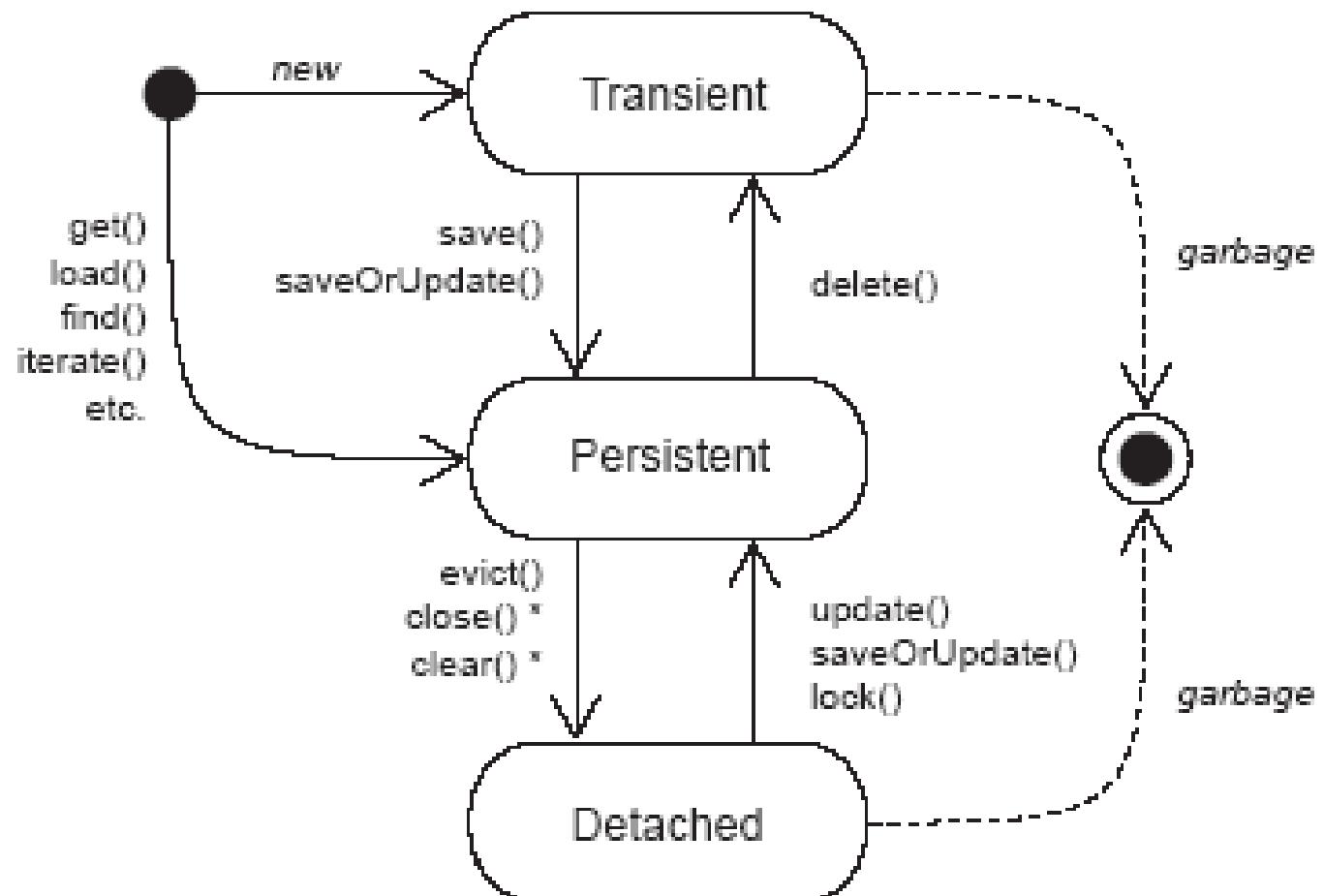
A BOT Entity

```
1 @Entity(name = "EVENTS")
2 public class Event implements Serializable {
3     @Id
4     @GeneratedValue(strategy = GenerationType.AUTO)
5     @Column(name = "EVENT_ID")
6     public Long getId() {
7         return id;
8     }
9     @Temporal(TemporalType.TIMESTAMP)
10    @Column(name = "EVENT_DATE")
11    public Date getDate() {
12        return date;
13    }
14    ...
15 }
```

Кто главный игрок в hibernate

- Session – интерфейс хайберната
 - save
 - update
 - saveOrUpdate
 - delete
- EntityManager – интерфейс JPA
 - persist
 - merge
 - remove

Persistent Object Lifecycle



* affects all instances in a Session

Как их получают

- Надо настроить SessionFactory если работаем с голым хайбернетом
- Настроить EntityManagerFactory – если работаем через JPA
- Потом его можно впрыскивать в любые бины спринга при помощи:
- @PersistenceContext

НЕЛЬЗЯ ПРОСТО ТАК ВЗЯТЬ

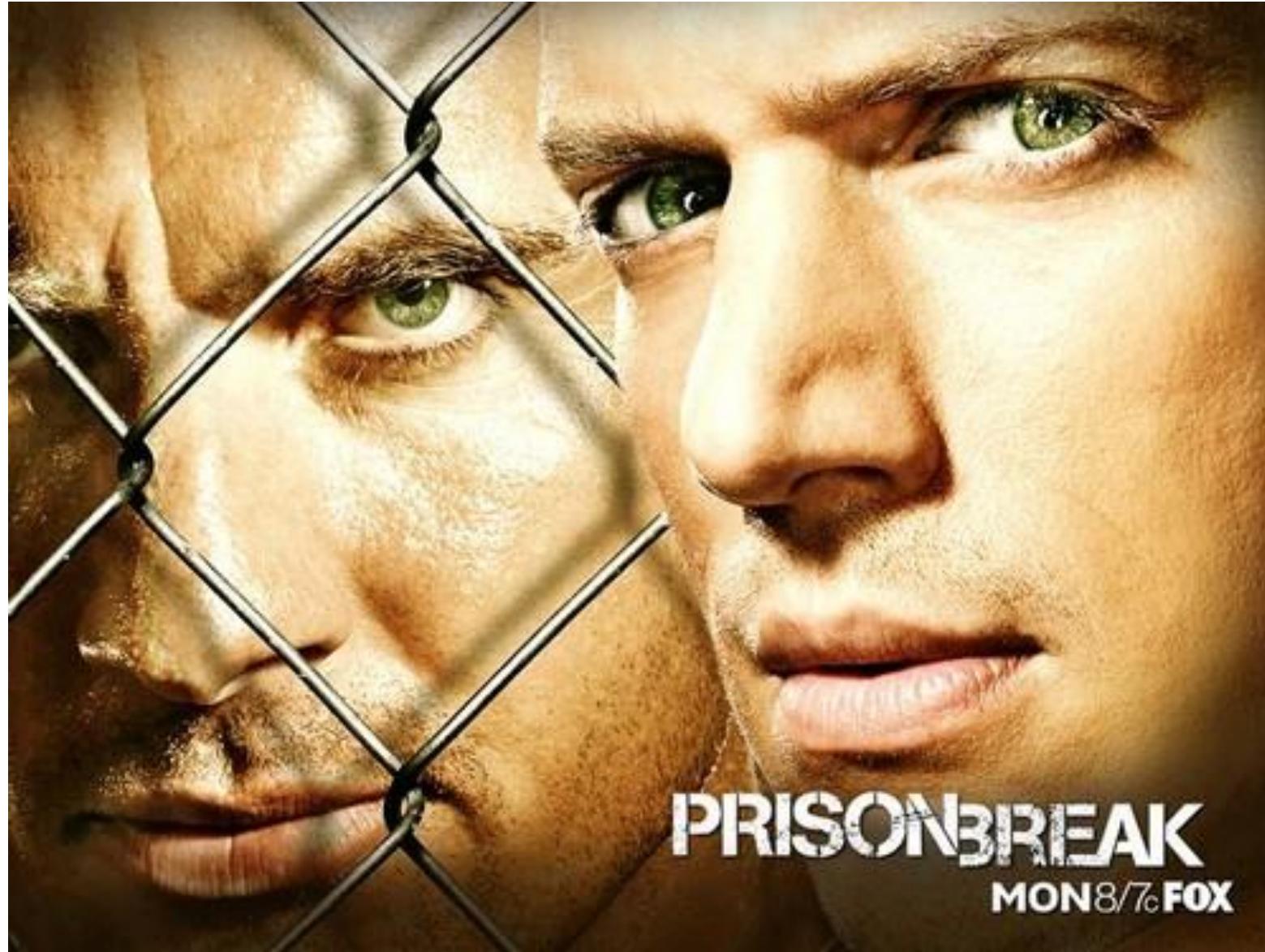


И РАБОТАТЬ БЕЗ ТРАНЗАКЦИЙ

@Transactional – аннотация спринга

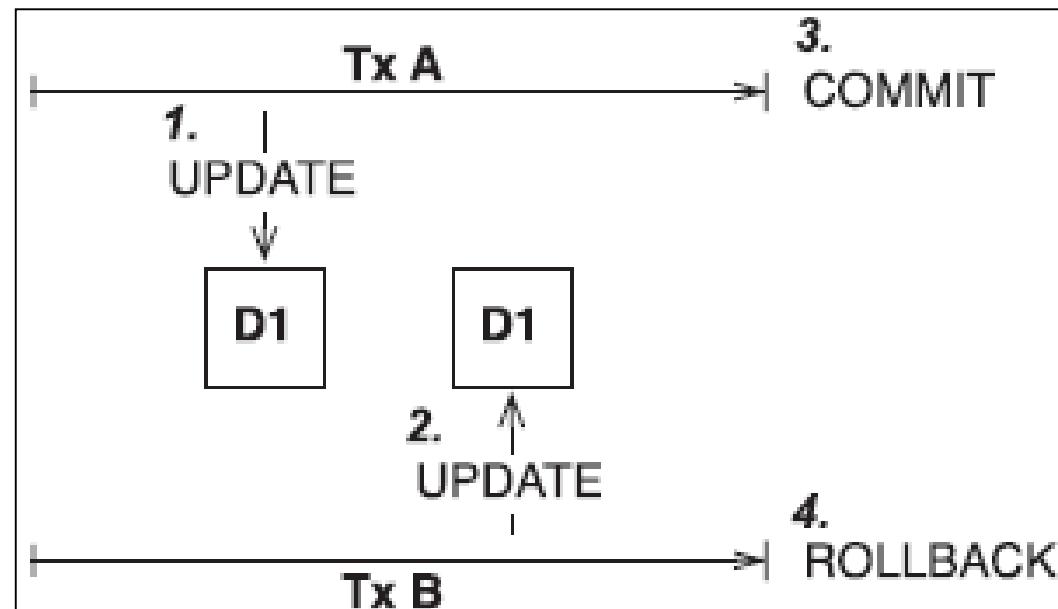
- Можно ставить над классом – тогда все методы будут транзакционные
- Можно переопределять для каждого метода
- Параметры аннотации:
 - `int timeout() default -1;`
 - `boolean readOnly() default false;`
 - `Throwable[] noRollbackFor()`
 - `isolation() : DEFAULT, READ_UNCOMMITTED, READ_COMMITTED, REPEATABLE_READ, SERIALIZABLE`
 - `propagation` : сейчас расскажу

Какие бывают виды изоляции?



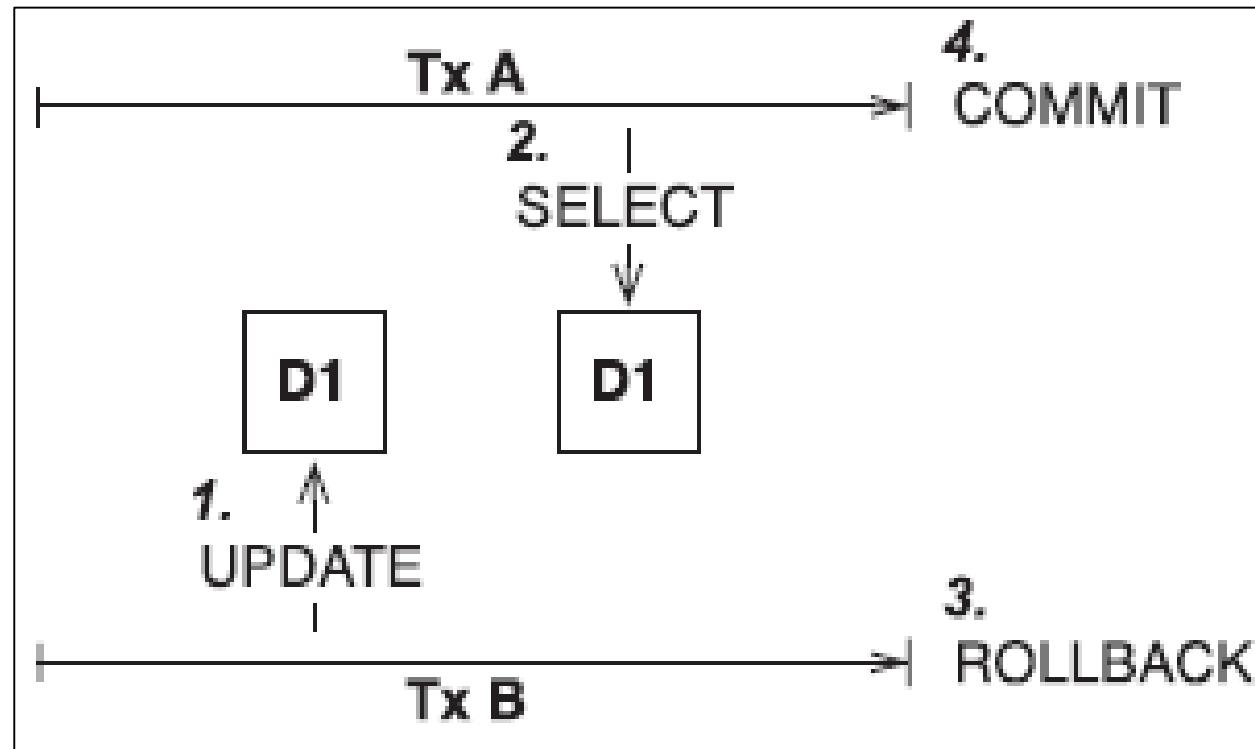
The Isolated Property

- Lost update
 - Two transactions update the same data without locking
 - If one of them aborts, both changes lost



The Isolated Property

- Dirty read
 - Transaction A reads uncommitted data
 - The read data may never be committed



Какие бывают Propagation-ы

- REQUIRED - (default)
- REQUIRES_NEW
- MANDATORY
- SUPPORTS
- NOT_SUPPORTED
- NEVER
- **NESTED**

Propagation Value	Existing Transaction	Transaction Associated with Business Method
REQUIRED	None T1	T T1
REQUIRES_NEW	None T1	T T2
MANDATORY	None T1	Error T1
SUPPORTS	None T1	None T1
NOT_SUPPORTED	None T1	None None
NEVER	None T1	None Error

- <bean name="bookstore" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
 <property name="driverClassName" value="org.apache.derby.jdbc.EmbeddedDriver"/>
 <property name="url" value="jdbc:derby:bookstore;create=true"/>
</bean>
- <bean id="bookstore-persistence-unit"
 class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
 <property name="dataSource" ref="bookstore"/>
 <property name="persistenceProviderClass" value="org.hibernate.ejb.HibernatePersistence"/>
 <property name="jpaVendorAdapter">
 <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"/>
 </property>
 <property name="jpaProperties">
 <props>
 <prop key="hibernate.dialect">org.hibernate.dialect.DerbyTenSevenDialect</prop>
 <prop key="hibernate.hbm2ddl.auto">create</prop>
 <prop key="hibernate.show_sql">true</prop>
 <prop key="hibernate.format_sql">true</prop>
 </props>
 </property>
 <property name="packagesToScan">
 <list>
 <value>bookstore</value>
 </list>
 </property>
</bean>
- <bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">
 <property name="entityManagerFactory" ref="bookstore-persistence-unit"/>
</bean>
- <context:component-scan base-package="bookstore"/>
- <tx:annotation-driven transaction-manager="txManager"/>

The Quartz Scheduler

- Кварц – очень популярный фреймворк, намного лучше чем Java Timers
- Поддерживает транзакции
- Поддерживает кластеризацию
- Может сохранять стэйт в базе данных или на диске
- Очень удобно настраивать тригеры

Что надо настраивать

- Job – что должно выполняться
- JobDetails – указывает на job и позволяет настроить всякие мелочи, например хранить ли стэйт jobа на диске
- Trigger – указывает на JobDetails и определяет когда данный job будет работать
- SchedulerFactoryBean – держит в себе лист всех триггеров, которые должны бежать