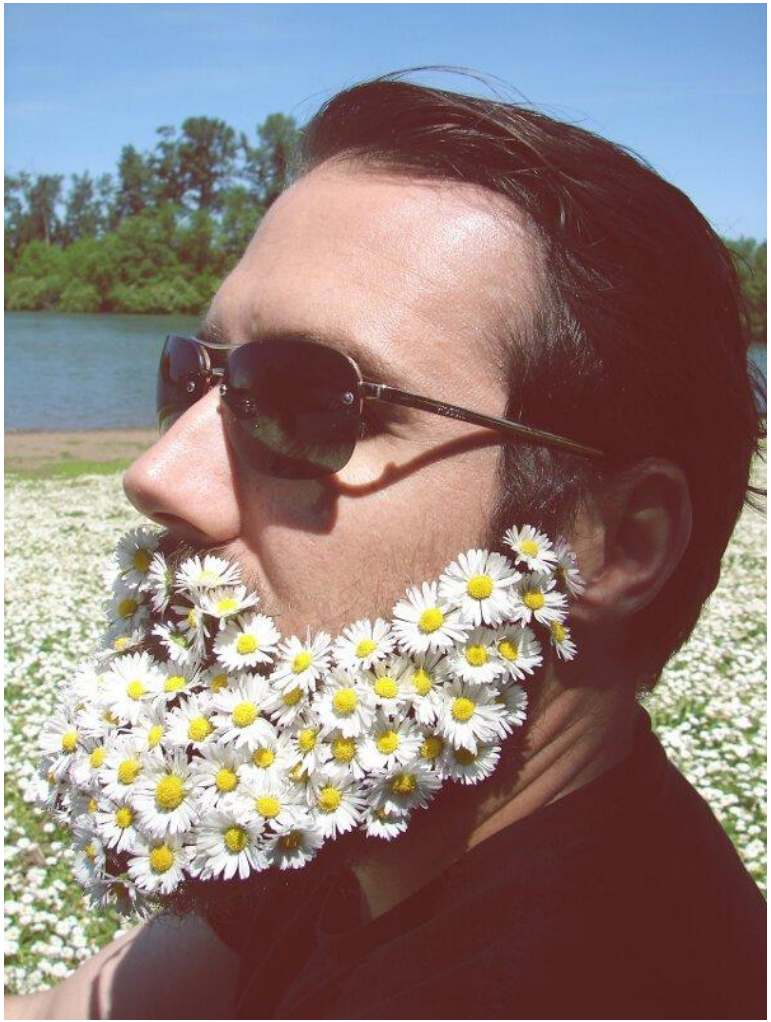
Университет ИТМО

Введение в Spring Framework

Кафедра ВТ, ауд. 373.
Письмак Алексей

Санкт-Петербург, 2017 год



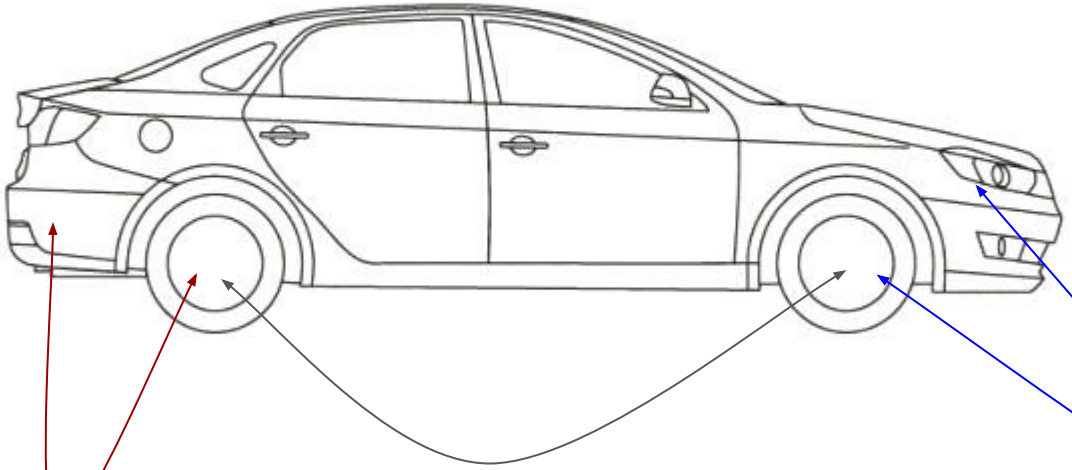
* example of Spring developer

Spring - это… ~~Между нами тает лед~~

- это универсальный фреймворк с открытым исходным кодом для Java-платформы. (Wikipedia)
- это явление на Олимпе индустрии разработки программного обеспечения, достойное внимания (интернеты)
- это легковесный opensource J2EE Framework, разработка которого началась в феврале 2003 года (javaportal.ru)
- это облегченная платформа для построения Java приложений (Кларенс Хо и Роб Харроп, “Spring 3 для профессионалов”, Apress.

Реализация паттерна **IoC** (Inversion of control)
и механизмов ***DI** (Dependency Injection)

DI в картинках



колесо знает о машине, которую оно перемещает

колеса не знаю друг о друге

Машина знает о колесах, которые она будет вращать

DI without Spring

```
public class Wheel {

   private Car owner;

   public Wheel(Car car) {
       setOwner(car);
   }

   public Car getOwner() {...}
   public void setOwner(Car owner) {...}

   public void rotate() {
       // change position of car
   }
}
```

```
public class Car {

   private Wheel  firstWheel;
   private Wheel secondWheel;
   private Vector3  position;

   public Car() {
       firstWheel = new Wheel(this);
       secondWheel = new Wheel(this);
   }

   public Wheel getFirstWheel() {...}
   public void setFirstWheel(Wheel firstWheel) {...}
   public Wheel getSecondWheel() {...}
   public void setSecondWheel(Wheel secondWheel) {...}
   public Vector3 getPosition() {...}
   public void setPosition(Vector3 position) {...}

   public void go() {
       firstWheel.rotate();
       secondWheel.rotate();
   }
}
```

```
public static void main(String[] args)
{
   Car car = new Car();
   car.go();
}
```

DI with Spring

```
public class Wheel {

    @Autowired private Car owner;

    public Wheel() {}

    public Car getOwner() {...}
    public void setOwner(Car owner) {...}

    public void rotate() {
        // change position of car
    }
}
```

```
public class Car {

    @Autowired private Wheel  firstWheel;
    @Autowired private Wheel secondWheel;
    private Vector3  position;

    public Car() {}

    public Wheel getFirstWheel() {...}
    public void setFirstWheel(Wheel firstWheel) {...}
    public Wheel getSecondWheel() {...}
    public void setSecondWheel(Wheel secondWheel) {...}
    public Vector3 getPosition() {...}
    public void setPosition(Vector3 position) {...}

    public void go() {
        firstWheel.rotate();
        secondWheel.rotate();
    }
}
```

А что если...

колеса связаны с машиной

колеса связаны с валом, который их крутит

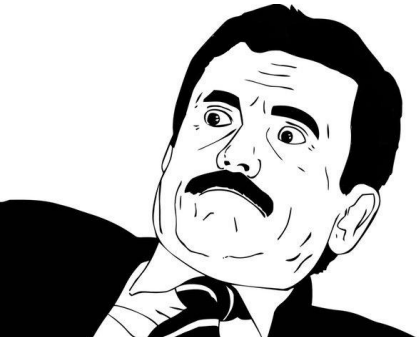
вал связан с машиной

А если много объектов? Но я же…

```
public class Wheel {

    private Car owner;

    public Wheel(Car car) {
        setOwner(car);
    }

    public void rotate() {
        // change position of car
    }
}
```

```
public class Val {

    private Wheel wheel;
    private Vector3 position;

    public Val (Wheel wheel) {
        setWheel(wheel);
    }

    public void work() {
        wheel.rotate();
    }
}
```

```
public class Car {

    private Val firstVal;
    private Val secondVal;
    private Vector3  position;

    public Car() {
        Wheel w1 = new Wheel(this);
        Wheel w2 = new Wheel(this);
        firstVal = new Val(w1);
        secondVal = new Val(w2);
    }

    public void go() {
        firstVal.work();
        secondVal.work();

    }
}
```



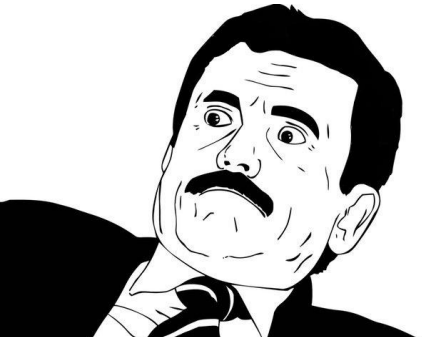
Как бы Вы разрешили такие зависимости?

А если много объектов? Но я же…

```
public class Wheel {

   private Car owner;

   public Wheel(Car car) {
       setOwner(car);
   }

   public void rotate() {
       // change position of car
   }
}
```

```
public class Val {

   private Wheel wheel;
   private Vector3 position;

   public Val (Wheel wheel) {
       setWheel(wheel);
   }

   public void work() {
       wheel.rotate();
   }
}
```

```
public class Car {

    private Val firstVal;
    private Val secondVal;
   private Vector3  position;

   public Car() {
       Wheel w1 = new Wheel(this);
       Wheel w2 = new Wheel(this);
       firstVal = new Val(w1);
       secondVal = new Val(w2);
   }

   public void go() {
       firstVal.work();
       secondVal.work();

   }
}
```



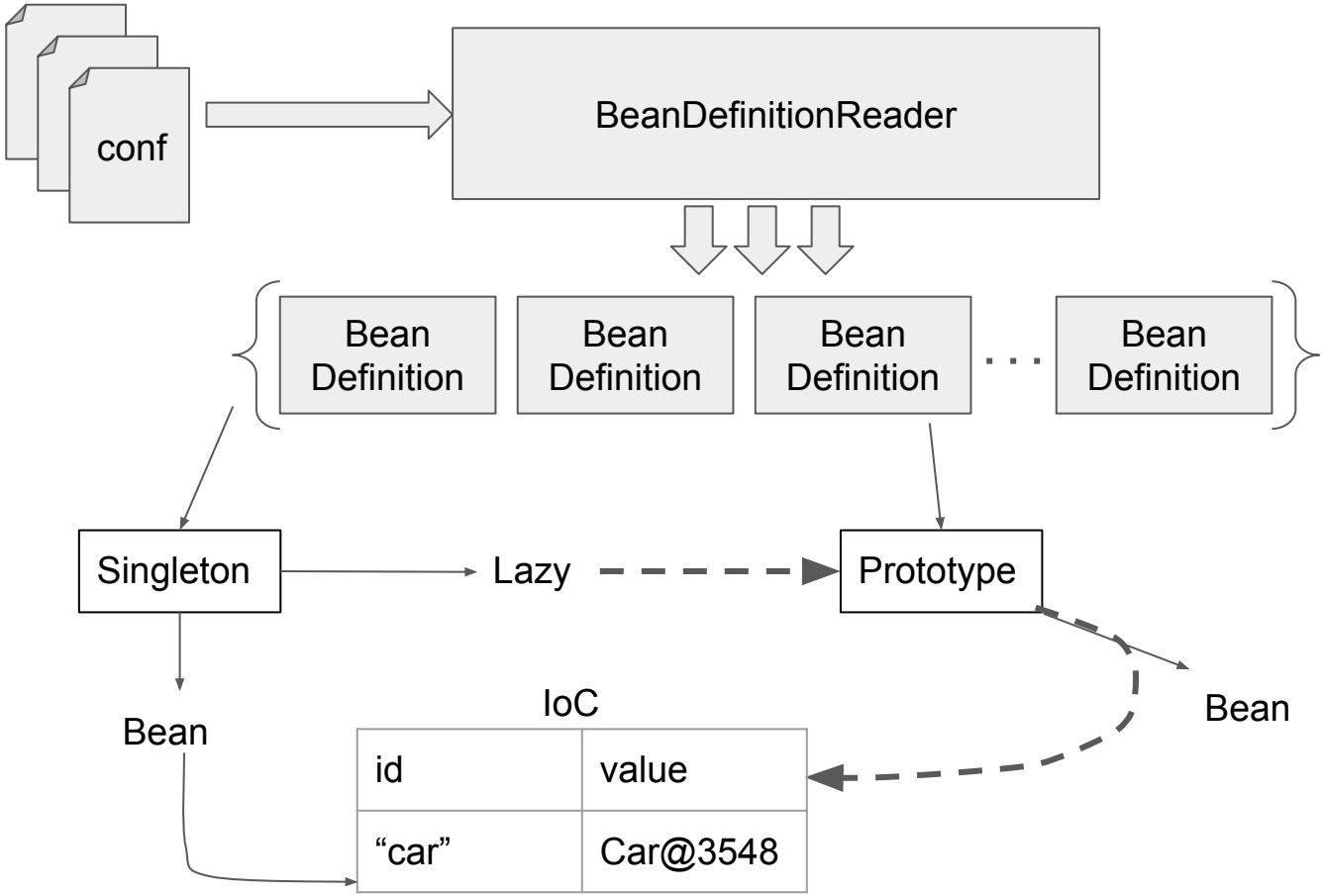
Если Вы подумали сделать поля ***static***, то Вы не пройдете собеседование на Java Junior.

Если Вы подумали сделать поля ***public static***…
“Небо поможет Вам…”

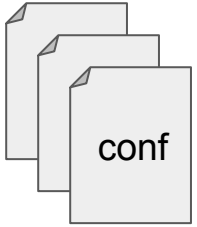
Кто виноват и что… такое Application context?

```
@Configuration
public class Main {

    public static void main(String[] args) {
        ApplicationContext ctx = new AnnotationConfigApplicationContext(Main.class);
        ((Wheel)ctx.getBean("firstWheel")).rotate();
    }

    @Bean
    public Car getCar() {
        return new Car();
    }

    @Bean("firstWheel")
    public Wheel firstWheel() {
        return new Wheel();
    }

    @Bean("firstVal")
    public Val firstVal() {
        return new Val();
    }
}
```

Жизненный цикл Spring-приложения (упрощенный)



Типы конфигов



- property files
- xml files
- groovy config
- java config

```
<?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-3.0.xsd">

   <bean id = "car" class = "ru.ifmo.cse.Car">
      <property name = "firstWheel" ref = "firstWheelBean"/>
   </bean>

   <bean id = "firstWheelBean" class = "ru.ifmo.cse.Wheel">
      <property name = "owner" ref = "car"/>
   </bean>


</beans>
```

Двухфазовый конструктор

```
public class Car {

    @Autowired
    private Val firstVal;

    public Car() {
        // using firstVal
    }

    @PostConstruct
    public void init() {

    }

    @PreDestroy
    public void destroy() {

    }

}
```

```
// В xml конфигурации
// <bean id = …..     init-method = “init” />
```

Типы DI. Constructor injection.

```
// bean code

@Autowired
public Car(Wheel wheel) {
   setFirstVal(null);
}


// java config code

@Bean
public Car getCar() {
   return new Car(firstWheel());
}


// xml config code

<bean id = "car" class = "ru.ifmo.cse.Car">
   <constructor-arg name = "wheel" ref = "firstWheelBean" />
</bean>
```

Типы DI. Property injection.

```
// bean code

@Autowired private Wheel  firstWheel;


// java config code
```

нет, т.к. внедрение происходит автоматически, используя рефлексию

```
// xml config code

<bean id = "car" class = "ru.ifmo.cse.Car" autowired = ”firstWheel” />
```

Типы DI. Setter injection.

```
// bean code

@Autowired
public setWheel(Wheel wheel) {
    …
}


// java config code

тоже самое, что и Property Injection


// xml config code

<bean id = "car" class = "ru.ifmo.cse.Car">
    <property name = "wheel" value-ref = "firstWheelBean" />
</bean>
```

Dependency Lookup - второй тип IoC

Contextualize Lookup

```
public class Val implements ApplicationContextAware {

   public void setApplicationContext(ApplicationContext applicationContext) {
      // using application context
      applicationContext.getBean(...)
   }
}
```

* Вопрос на л/р: Dependency Pull

Инициализация бинов

1. Используя конструктор
2. Метод фабрики (init-method)

```
<bean id ="wheelInstance" factory-bean ="wheelFactory" factory-method ="create">
    <constructor-arg>
        <value>
            ru.ifmo.cse.Wheel
        </value>
    </constructor-arg>
</bean>
```

Singleton имеет особенность в своем жизненном цикле при указании

lazy-init = *"true"* или `@Lazy`

Области видимости бинов

- Singleton
- Prototype (не управляется после инициализации)

Spring web также содержит области

- Request
- Session

По умолчанию все бины - singleton’ы

* Вопрос на л/р: как запилить свой кастомный Scope?

Немного подробностей о жизненном цикле бинов

В какой момент обрабатывается Autowired?

Что делать, если хочется что то сделать с бинами?

```
interface BeanPostProcessor
    postProcessAfterInitialization(Object bean, String beanName)
    postProcessBeforeInitialization(Object bean, String beanName)
```

Оба метода возвращают Object.

Примеры существующих постпроцессоров:
- AutowiredAnnotationBeanPostProcessor
- InitDestroyAnnotationBeanPostProcessor

interface ApplicationListener<ApplicationEvent> - на все случаи жизни

Spring - это

1. BOOT
2. MOBILE
3. FOR ANDROID
4. FRAMEWORK
5. CLOUD
6. SOCIAL
7. DATA
8. INTEGRATION
9. BATCH
10. SECURITY
11. STATE MACHINE
12. SESSION
13. LDAP
14. WEB SERVICES
15. WEB FLOW
16. REST DOCS
17. CLOUD DATA FLOW
18. AMQP
19. HATEOAS
20. TEST

влезло только 20

Framework of frameworks

Откуда все это и что это?

Пара примеров для л/р. Spring Data.

```
public interface WheelRepository extends CrudRepository<Wheel, Long> { }
```

Для использования таблицы в базе данных SpringData предлагает внедрить себе такую зависимость и **использовать ее полностью**!

```
…
@Autowired
private WheelRepository wheelRepository;
…
```

В интерфейсе CrudRepository уже есть некоторые методы, например,

- findAll
- findOne
- save
- delete
- count
- и другие.

Однако, если добавить свой метод, вроде findByVector3(Vector3), то он тоже заработает!

Spring Data умеет понимать названия методов.

Table 4. Supported keywords inside method names

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is,Equals	findByFirstname, findByFirstnameIs, findByFirstnameEquals	… where x.firstname = ?1
Between	findByStartDateBetween	… where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age <= ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull	findByAgeIsNull	… where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	… where x.age not null
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?

Пара примеров для л/р. Spring RESTful.

```
@RestController
public class IncrementController {

    private static final String template = "Hi, %s! Current value = %d";
    private final AtomicLong counter = new AtomicLong();

    @RequestMapping("/increment")
    public ValueWrapper inc(@RequestParam(value="name", defaultValue="Guest") String name) {

        Long value = counter.incrementAndGet();
        return new ValueWrapper(value, String.format(template, name, value));
    }
}
```

Пара примеров НЕ для л/р. Spring Profiles.

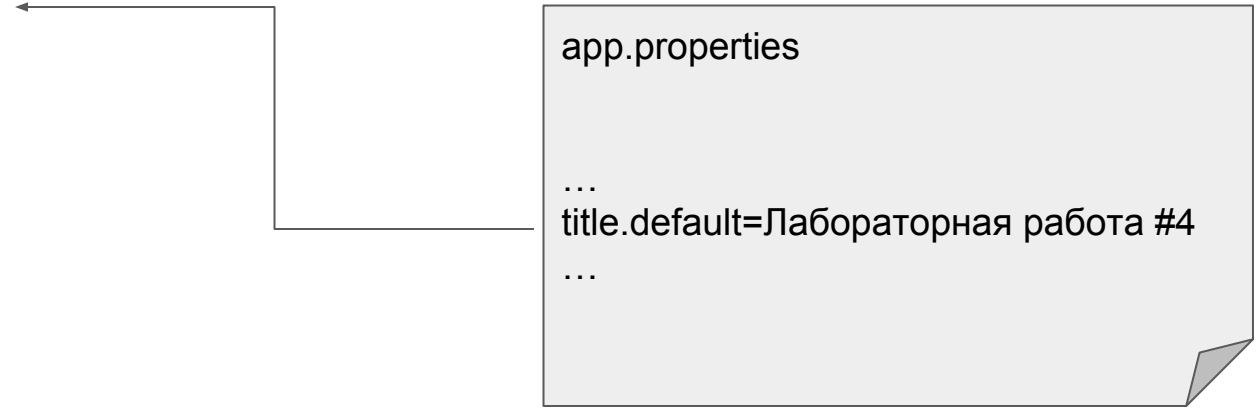
```
@Profile("prod")
@Configuration
public class ProductionConfiguration { }

@Profile("dev")
@Configuration
public class DeveloperConfiguration { }

// main
ApplicationContext ctx = new AnnotationConfigApplicationContext(
                                                    ProductionConfiguration.class,
                                                    DeveloperConfiguration.class);
```

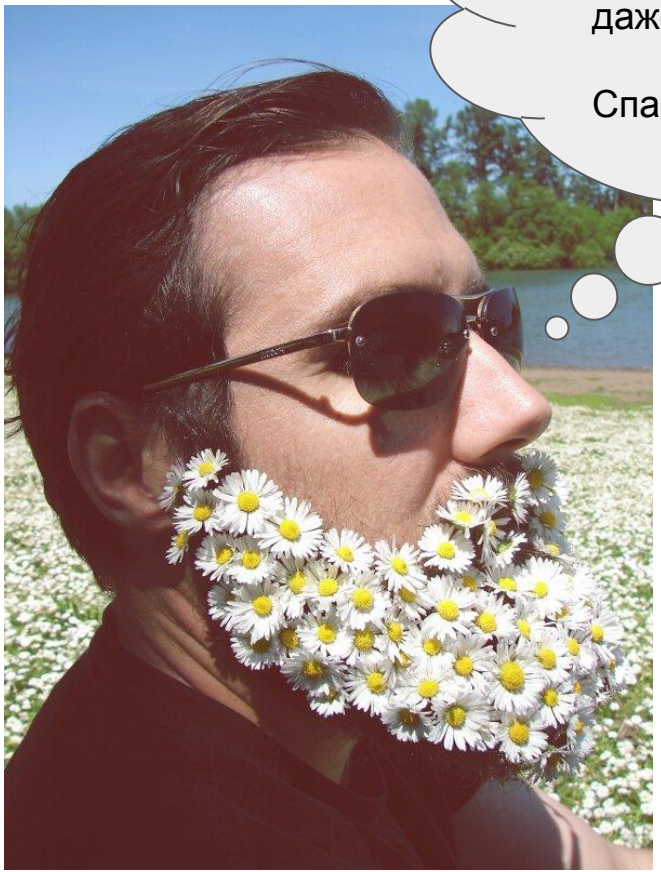
Все случится при старте приложения:

```
java  -Dspring.profiles.active=prod  -jar  App.java
```

Пара примеров НЕ для л/р. Spring Core.

```
@Component
@Scope("prototype")
@PropertySource("classpath:app.properties")
public class MainPage {

    @Value("${title.default}")
    private String title;

}
```





Эй, пссс. Знать Spring придется даже *тестировщику!

Спасибо за внимание.

* Вопрос на л/р: как и зачем использовать spring-test ?