The Spring Puzzlers:

Начало.

Ты кто такой?

github.com/jbaruch

@jbaruch

linkd.in/jbaruch

stackoverflow.com/users/402053/jbaruch

Ты кто такой?

jeka@inwhite.pro

@jekaborisov

linkedin.com/in/evborisov

Вместе мы:

Свистнули идейку:

Эпичные Груви Паззлеры

Второй Сезон:
Месть Скобок!

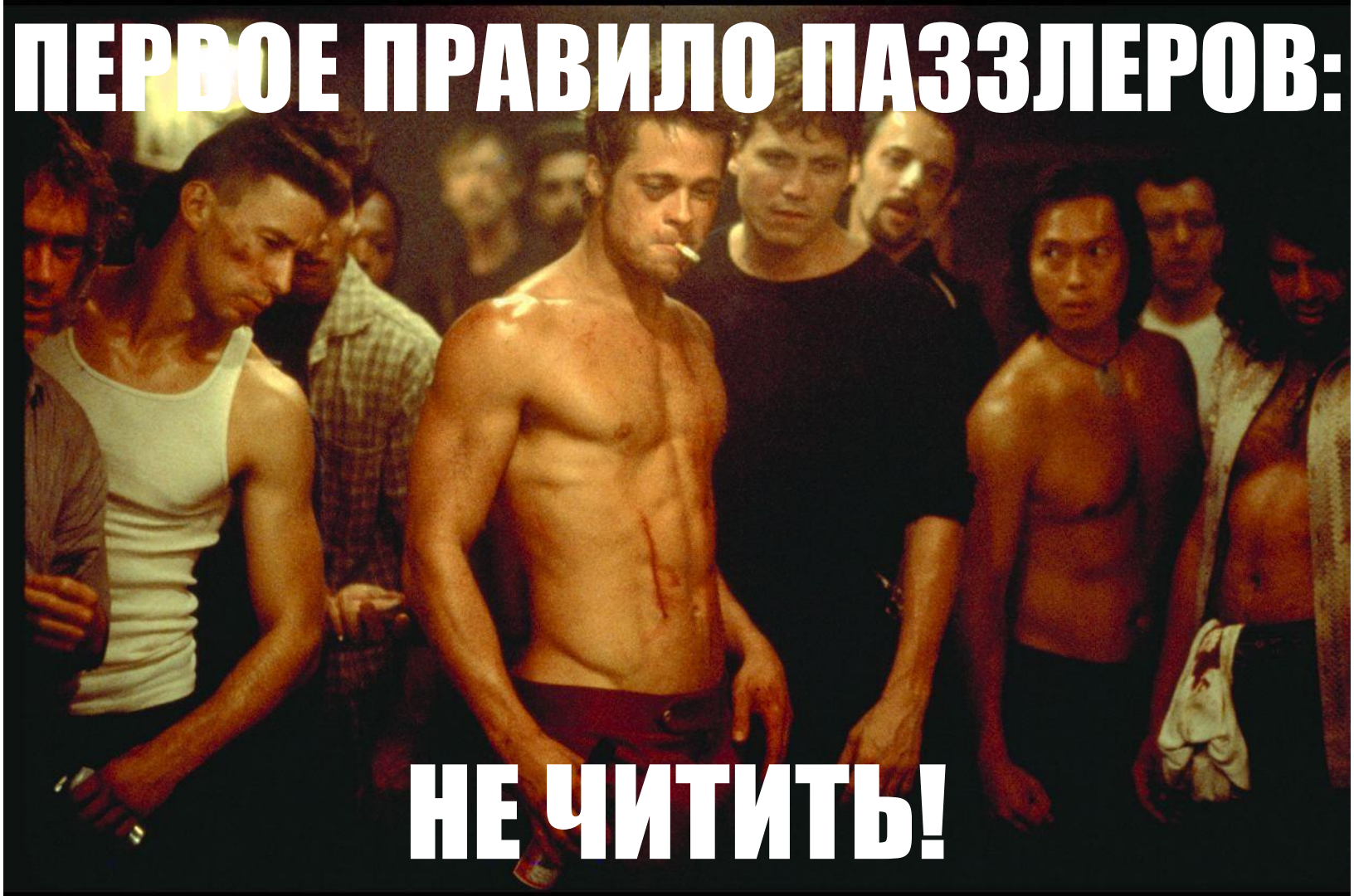
Свистнули идейку:

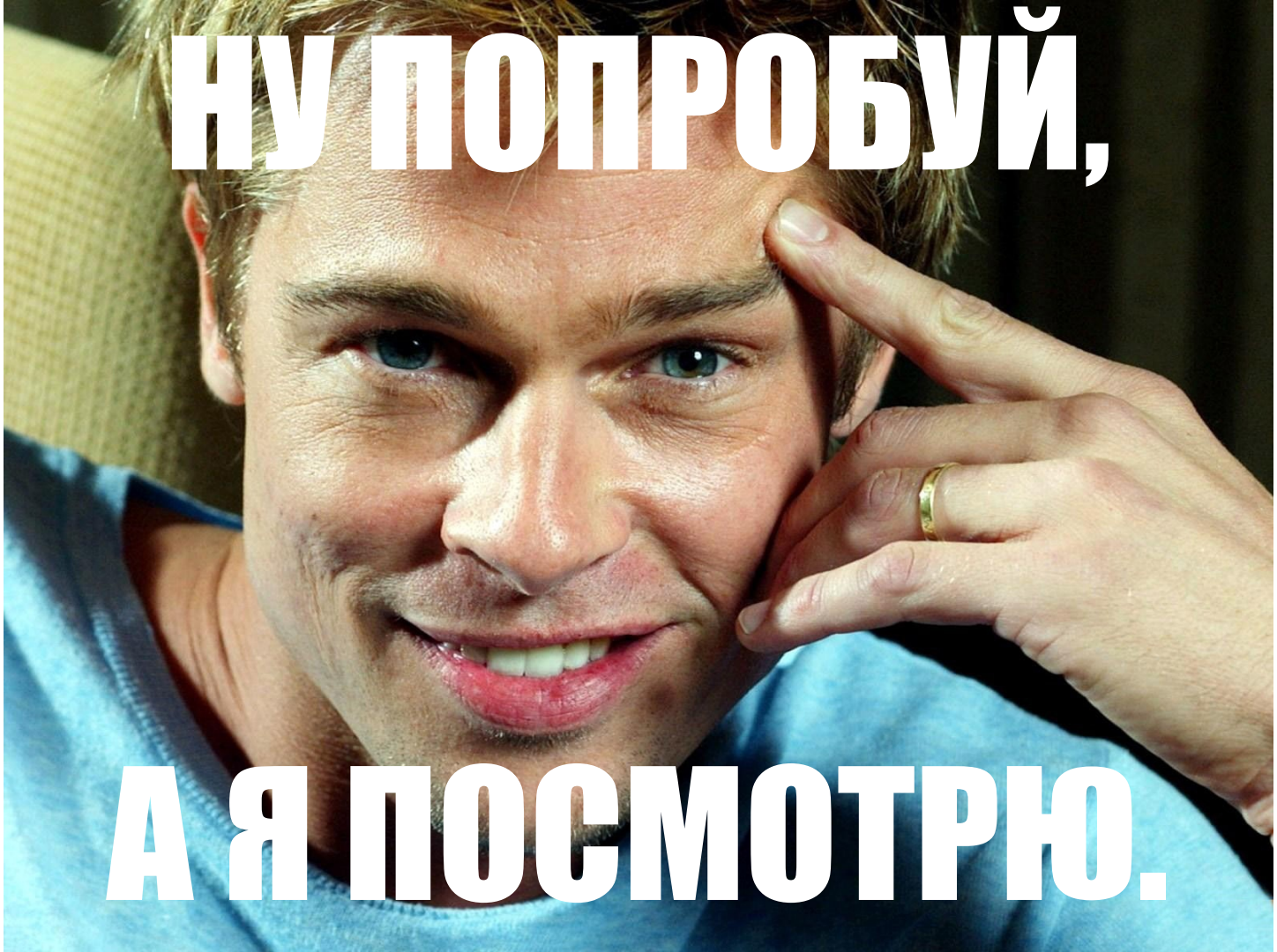
Свистнули идейку:

Свистнули идейку:

Похоже, но не совсем

1. Все еще Два клевых пацана на сцене
2. Меньше хи-хи, больше хардкора!
3. Вы все равно голосуете!
4. Спасибо luxoft и Jetbrains за призы!


ПЕРВОЕ ПРАВИЛО ПАЗЗЛЕРОВ:
НЕ ЧИТИТЬ!


НУ ПОПРОБУЙ,
А Я ПОСМОТРЮ.

```java
@Service
public class ConferenceService {
    @Autowired(required = false)
    private Greeter jokerGreeter;

    @Autowired(required = false)
    private Greeter jPointGreeter;

    @PostConstruct
    public void init(){
        jokerGreeter.greet();
        jPointGreeter.greet();
    }
}
```

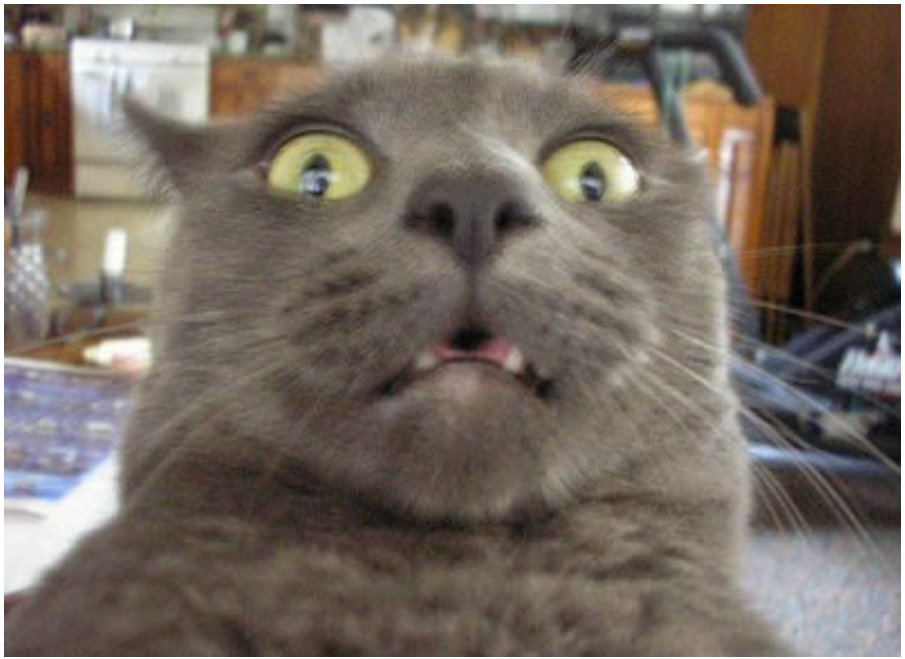
A. Injection JokerGreeter
B. Injection JPointGreeter
C. Injection обоих
D. Нет проблемы, напечатается:
   Привет Джигурда,
   Привет Галкин
E. NPE

В чём проблема?


А ГДЕ
ПРИВЕТ ДЖИГУРДА?

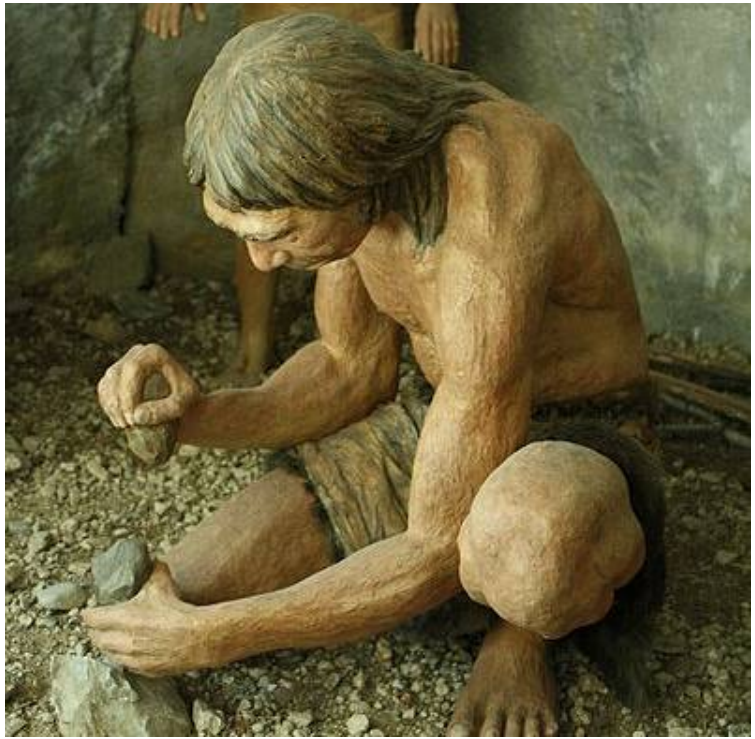
Как мы это чиним?

- Переименуйте класс



Будни Кровавого Энтерпрайза

Когда-то не было аннотаций



Pre-annotation days

- InitMethodInvokerBeanPostProcessor

```
 public Object postProcessBeforeInitialization(Object bean, String beanName)
{
     Method[] methods = bean.getClass().getMethods();
     for (Method method : methods) {
         if (method.getName().startsWith( "init")) {
             ReflectionUtils. invokeMethod(method, bean);
         }
     }
     return bean;
}
```

Java 5, Spring 2

Dependency Injection

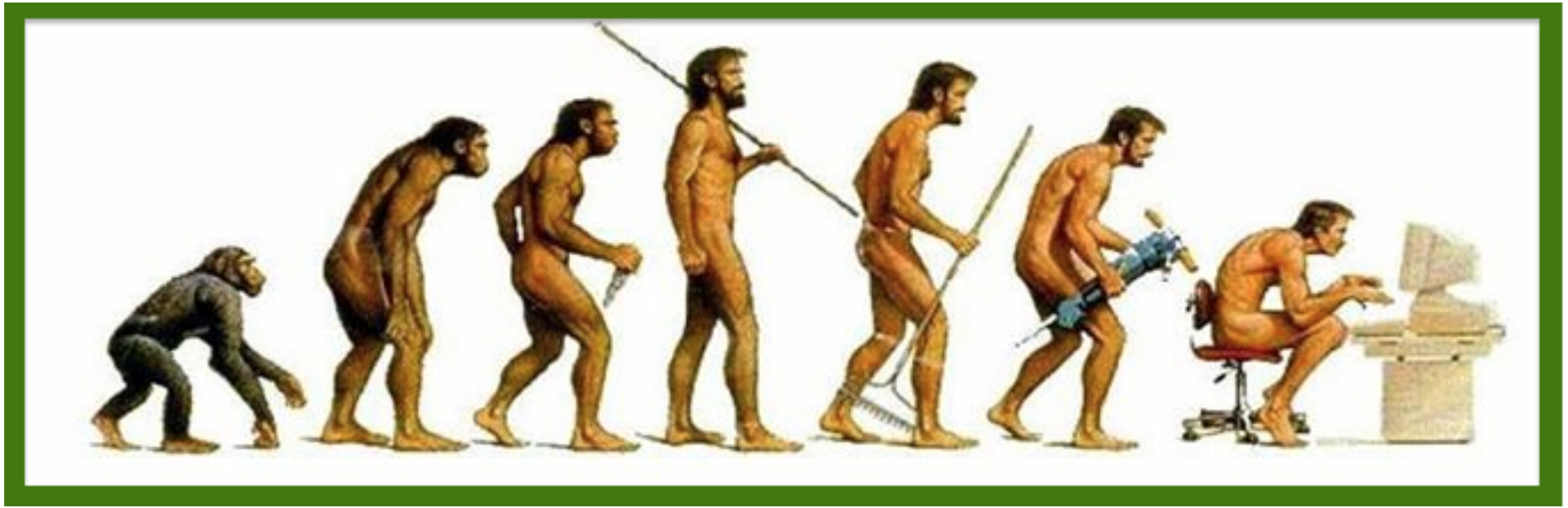
```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.CONSTRUCTOR, ElementType.FIELD,
ElementType.METHOD, ElementType.ANNOTATION_TYPE})
public @interface Dependency {
}
```

Java 5 Spring 2

- DependencyInjectorBeanPostProcessor

```
public class DependencyInjectorBeanPostProcessor  implements BeanPostProcessor, ApplicationContextAware
{
    private ApplicationContext  context;

    public void setApplicationContext(ApplicationContext context)  throws BeansException {
        this.context = context;
    }

    public Object postProcessBeforeInitialization(Object bean, String beanName)  throws BeansException {
        ReflectionUtils.doWithFields(bean.getClass(), field -> {
            field.setAccessible( true);
            field.set(bean, context.getBean(field.getType()));
        }, field -> field.isAnnotationPresent( Dependency.class));

        return bean;
    }

    public Object postProcessAfterInitialization(Object bean, String beanName)  throws BeansException {
        return bean;
    }
}
```

Спринг начал поддерживать аннотации



Annotation Mix

Аннотации Спринга в наследовании

Аннотации спринга в наследовании

```
public class Parent {

    @PostConstruct
    private void init(){
        System.out.println("Папа");
    }
}
```

```
@Component
public class Son extends Parent {

    @PostConstruct
    public void init(){
        System.out.println("Сын");
    }
}
```

Создаём сына, что с папой?

A. @PostConstruct не inherited, Поэтому Папа не придёт

B. init у Папы private, поэтому Папа не придёт

C. Не может быть больше, чем один init method – Папа не придёт

D. init method переопределён у сына, поэтому папа не придёт

E. Папа придёт

Папа точно придёт

Аннотации спринга в наследовании

```
public class Parent {

    @PostConstruct
    private void init(){
        System.out.println("Папа");
    }
}
```

```
@Component
public class Son extends Parent {

    @PostConstruct
    public void init(){
        System.out.println("Сын");
    }
}
```

Создаём сына, что с папой?

A. @PostConstruct не inherited, Поэтому Папа не придёт

B. init у Папы private, поэтому Папа не придёт

C. Не может быть больше, чем один init method – Папа не придёт

D. init method переопределён у сына, поэтому папа не придёт

Папа придёт

Annotation Mix

Микс наших и не наших

```java
public class Son extends Parent {
    @Autowired
    private Integer year;

    @PostConstruct
    public void postConstruct(){
        System.out.println(year);
    }
}
```

```java
public class Parent {

    @Dependency
    private String confName;

    public void init(){
        System.out.println(confName);
    }
}
```

```
<bean class="..InitMethodInvokerBeanPostProcessor"/>
<bean class="..DependencyInjectorBeanPostProcessor"/>

<bean class="..CommonAnnotationBeanPostProcessor"/>
<bean class="..AutowiredAnnotationBeanPostProcessor"/>
```

A. null, null

null, 2015

C. JPoint, 2015

D. JPoint, null

Микс наших и не наших

```java
public class Son extends Parent {
    @Autowired
    private Integer year;

    @PostConstruct
    public void postConstruct(){
        System.out.println(year);
    }
}
```

```java
public class Parent {

    @Dependency
    private String confName;

    public void init(){
        System.out.println(confName);
    }
}
```

```
<bean class="..InitMethodInvokerBeanPostProcessor"/>
<bean class="..DependencyInjectorBeanPostProcessor"/>

<bean class="..CommonAnnotationBeanPostProcessor"/>
<bean class="..AutowiredAnnotationBeanPostProcessor"/>
```

A. null, null

null, 2015

C. JPoint, 2015

D. JPoint, null

Как чиним?

A. Назвать в нужном алфавитном порядке
B. Прописать в нужном порядке в xmle
C. Воспользоваться Аннотацией @Order

Имплементировать интерфейс Ordered

И вот мы сделали так

- Для DependencyInjectorBeanPostProcessor
  Мы ставим Ordered.HIGHEST_PRECEDENCE;
  Чтобы он точно был первым
- Для InitMethodInvokerBeanPostProcessor
  ставим Ordered.LOWEST_PRECEDENCE
  Чтобы точно был последним
- А теперь внимание вопрос:

Микс наших и не наших

```java
public class Son extends Parent {
    @Autowired
    private Integer year;

    @PostConstruct
    public void postConstruct(){
        System.out.println(year);
        System.out.println(confName);

    }
}
```

```java
public class Parent {

    @Dependency
    private String confName;

    public void init(){
        System.out.println(confName);
    }
}
```

```xml
<bean class="..InitMethodInvokerBeanPostProcessor"/>
<bean class="..DependencyInjectorBeanPostProcessor"/>

<bean class="..CommonAnnotationBeanPostProcessor"/>
<bean class="..AutowiredAnnotationBeanPostProcessor"/>
```

A. 2015, JPoint, JPoint

JPoint, 2015, JPoint

C. null, 2015, JPoint

D. JPoint, null, JPoint

Хочешь подсказку?

- CommonAnnotationBeanPostProcessor : Ordered.LOWEST_PRECEDENCE – 3

Хочешь подсказку?

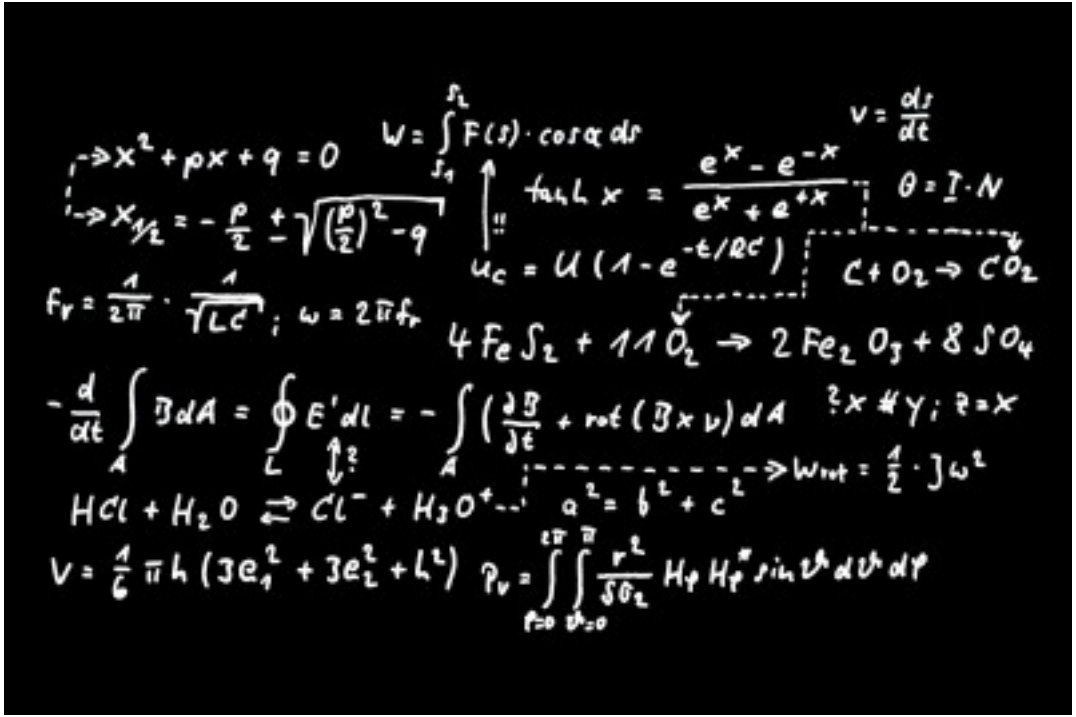
- CommonAnnotationBeanPostProcessor : Ordered.LOWEST_PRECEDENCE – 3
- А у AutowiredAnnotationBeanPostProcessor: Ordered.LOWEST_PRECEDENCE – 2

Хочешь подсказку?

- CommonAnnotationBeanPostProcessor : Ordered.LOWEST_PRECEDENCE – 3
- А у AutowiredAnnotationBeanPostProcessor: Ordered.LOWEST_PRECEDENCE – 2

Объясни мне!

Непонятно? Зато работает.

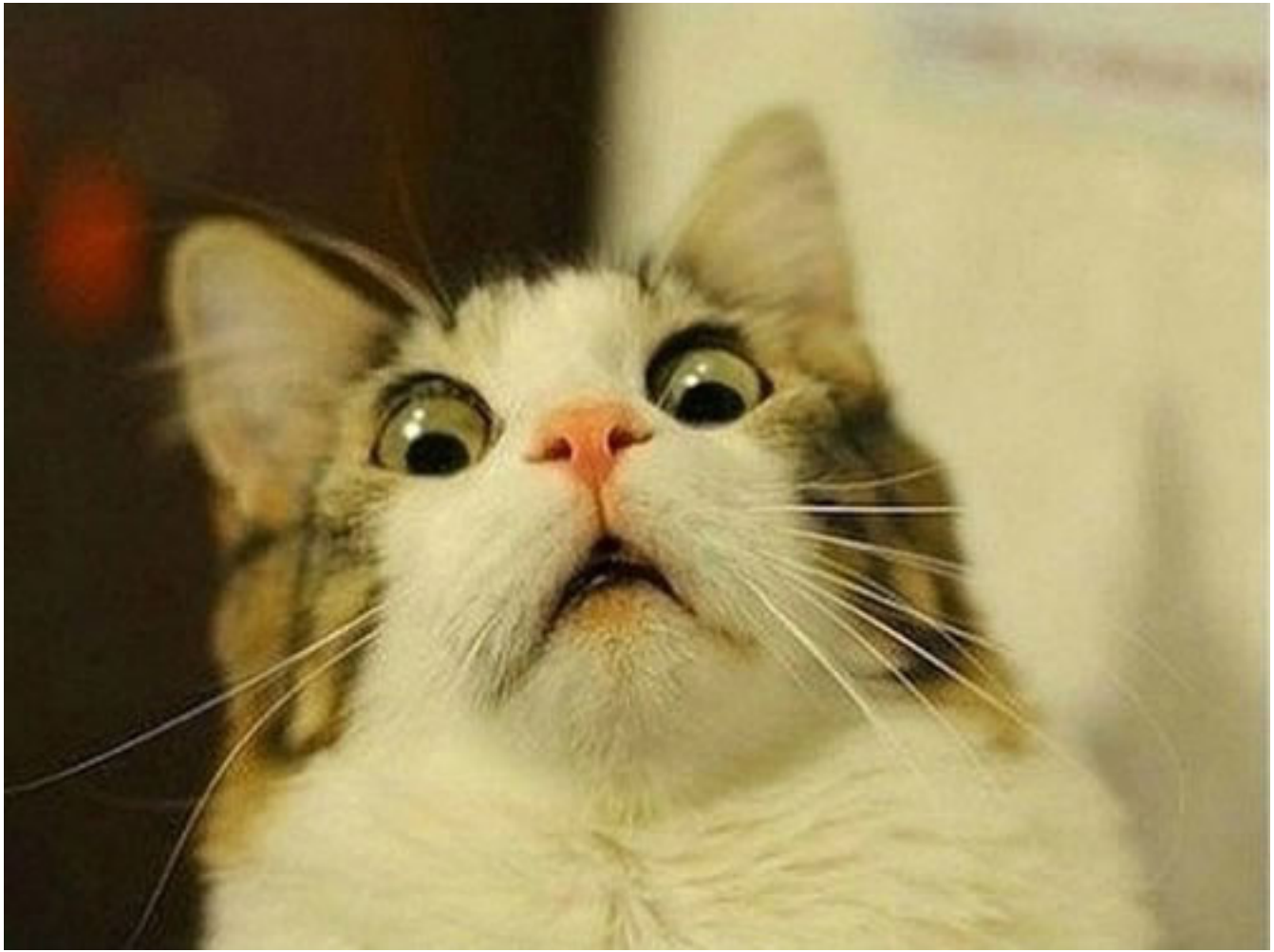


Все ещё впереди

Добавляем наш @Transactional

```
@Service
public class JPointServiceImpl implements
JPointService {
    @Autowired
    private String conference;

    @Transactional
    public void pay() {
     System.out.println("Вчера билеты были дешевле");
    }

    @PostConstruct
    public void postConstruct(){
        System.out.println(conference);
    }
}

 context.getBean(JPointService.class).
 pay();
```

A. Не сработает init method
B. NoSuchBeanDefinitionException
C. Упадёт Autowired из за то, что у прокси нет нужного филда
D. Всё отлично заработает с транзакциями



Добавляем наш @Transactional

```
@Service
public class JPointServiceImpl implements
JPointService {
    @Autowired
    private String conference;

    @Transactional
    public void pay() {
     System.out.println("Вчера билеты были дешевле");
    }

    @PostConstruct
    public void postConstruct(){
        System.out.println(conference);
    }
}

 context.getBean(JPointService.class).
 pay();
```

A. Не сработает init method
B. NoSuchBeanDefinitionException
C. Упадёт Autowired из за то, что у прокси нет нужного филда
D. Всё отлично заработает с транзакциями

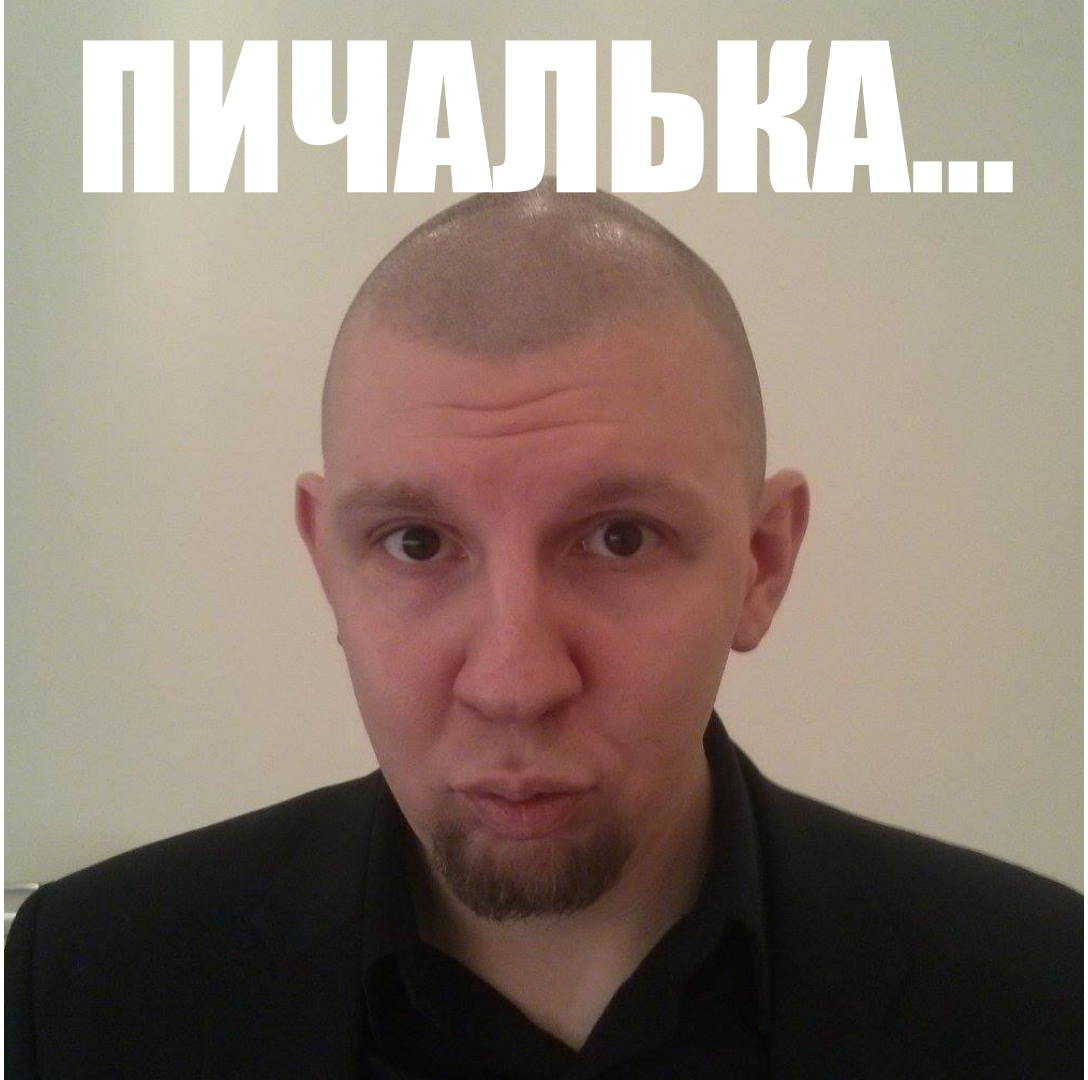
Как мы это чиним?

- Прокси надо делать после инит методов!
- Для это есть postProcessAfterInit

Добавим Аспект

```
@Component
@Aspect
public class PayInformatorAspect {
    @Pointcut("execution(* jpoint.story.phase4..*.pay*(..))")
    public void allPayMethods(){}

    @After("allPayMethods()")
    public void sendMailToLesha(){
        System.out.println("Ещё билет продали");
    }
}
```

B. Федоров получит извещение о проданном билете, но не будет транзакции
B. Транзакция будет, но Федоров не узнает о проданном билете
C. Будет транзакция и Федоров получит извещение
D. Будет exception из-за двухуровневого прокси


ПИЧАЛЬКА...

Как мы это чиним?

- Пишем нормальный BPP, который не делает bean.getClass()

Начинаем юзать транзакции спринга

```
@Retention(RetentionPolicy.RUNTIME)
@Transactional
public @interface JPointTransaction  {
    Propagation propagation()  default Propagation.REQUIRES_NEW;
}
```

Вложенные транзакции

```
@JPointTransaction
public class JPointService {
    public void transferMoney(Account from, Account to, BigDecimal amount) {
        withdraw(from, amount);
        deposit(to, amount);
    }
    @Transactional(propagation = Propagation.MANDATORY)
    private void withdraw(Account from, BigDecimal amount) {
        //some code here
    }
    @Transactional(propagation = Propagation.REQUIRED)
    private void deposit(Account to, BigDecimal amount) {
        //some code here
        informBankManager(amount);
    }

    private void informBankManager(BigDecimal amount) {
        //some code here
    }
}
```

Случился exception в методе informBankManager. Что будет?

A. Всё откатится до самого начала

B. Откатится только informBankManager

C. Откатится до начала метода deposit

D. Метод withdraw кинет exception

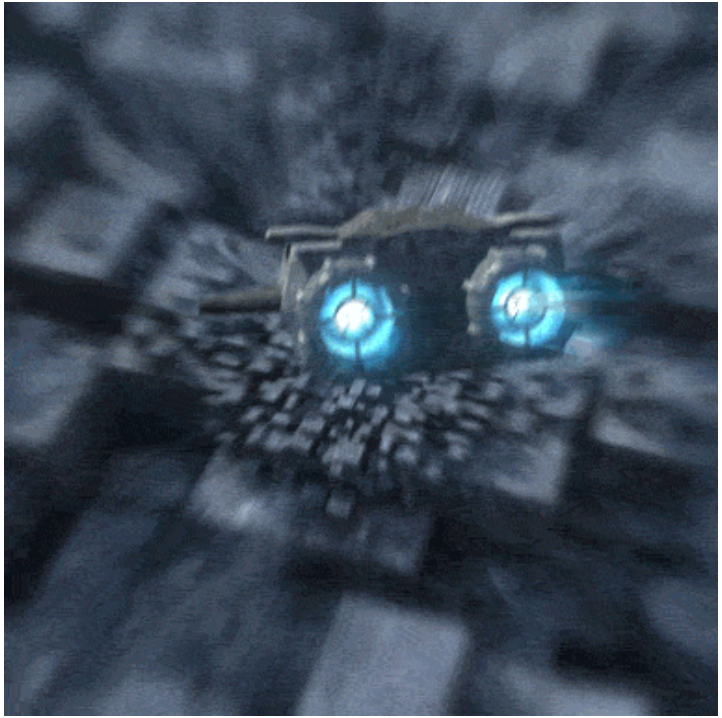
Почему не работает??

```
@Service
public class JPointServiceImpl implements JPointService {

    @Transactional
    public void pay() {
        System.out.println("Yesterday tickets were cheaper");
        informAboutPayment();
    }

    @Override
    @Transactional(requiresNew = true)
    public void informAboutPayment() {
        System.out.println("money were transferred");
    }
}
```

Почему не работает??

```
@Service
public class JPointServiceImpl implements JPointService {

    @Transactional
    public void pay() {
        System.out.println("Yesterday tickets were cheaper");
        this.informAboutPayment();
    }

    @Override
    @Transactional(requiresNew = true)
    public void informAboutPayment() {
        System.out.println("money were transferred");
    }
}
```

Самовпрыскивание



Самовпрыскивание. Как сделать?

A. @Autowired
B. @Inject

@Resource
D. Что за бред?! Это невозможно!

У спринга 4 года, а у нас?

A. 4 часа
B. 4 дня
C. 4 минуты
D. 4 секунды

@Transactional @PostConstruct

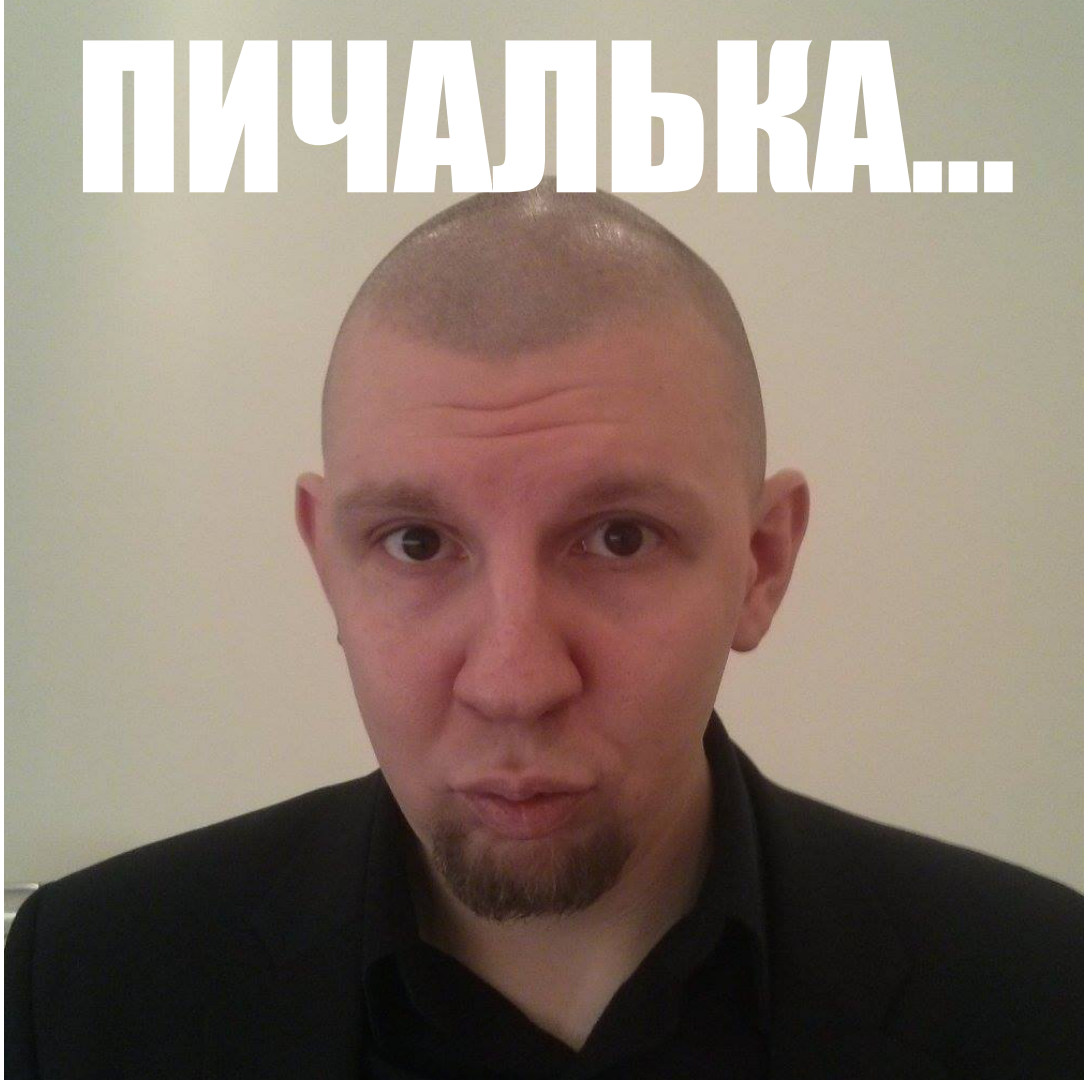
- Что будет, с методом у которого обе аннотации?

A. Сработает с транзакцией

Сработает без транзакции

C. Не сработает вообще

D. MethodMissingException


ПИЧАЛЬКА...

ApplicationListener

- Решаем своей аннотацией @PostInitialize

Чего не будет в BeanDefinition-е?

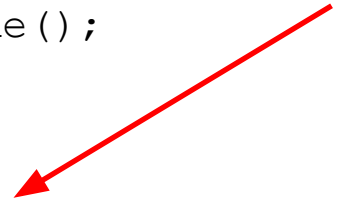
```
@Scope("singleton")
@Bean(destroyMethod = "toString")
public String str() {
    return "chuck norris";
}
```

A. Scope
B. Destroy method name

BeanClass
D. Всё будет

@PostInitialize

```
String[] names = context.getBeanDefinitionNames();
for (String name : names) {
    BeanDefinition beanDefinition = factory.getBeanDefinition
(name);
    String beanClassName = beanDefinition.getBeanClassName();
    try {


        Class<?> originalBeanClass = Class.forName(beanClassName);
```



Но ведь не так же…

```
String[] names = context.getBeanDefinitionNames();
for (String name : names) {
    BeanDefinition beanDefinition =  factory.getBeanDefinition
(name);
    String beanClassName = beanDefinition.getBeanClassName();
    try {
        if (beanClassName == null) {
            continue;
        }
        Class<?> originalBeanClass = Class. forName(beanClassName);
```

Сделаем по взрослому

A. 4 минуты
B. 3 минуты
C. 2 минуты
D. 1 минуты

СТРАШНЫЕ ИСТОРИИ

МАЛЬЧИК, КОТОРЫЙ НЕ ЛЮБИЛ ИНТЕРФЕЙСЫ

Добавляем Comparable…

NoSuchBeanDefinitionException

B. No proxies

C. NoSuchMethodException

D. All OK

Как мы это чиним?

- Любовь к интерфейсам

ДЕВОЧКА, КОТОРАЯ ХОТЕЛА СДЕЛАТЬ НАДЕЖНО

Что будет?

```
<bean class="jpoint.littleGirl.MissionCriticalService"/>

<context:component-scan base-package="jpoint.littleGirl"/>
```

```
@Service
public class MissionCriticalService {
    @PostConstruct
    public void important(){
        System.out.println("Не забудь выключить утюг!");
    }
}
```

```
@Bean
public MissionCriticalService missionCriticalService() {
    return new MissionCriticalService();
}
```

A. не забыть выключить утюг

B. не забыть выключить утюг *2

C. не забыть выключить утюг *3

D. BeanCreationException

Выводы

1. Учите спринг!
2. Читайте документаци
3. Иногда это баги, но и
4. Пользуйте spring inte
   IDEA!
5. Учите спринг, Я сказа

"Расставашки -
всегда пичалька".

(с) Сократик