

Spring глубоко и не очень

# Who are you?

## Big Data & Java Technical Leader

Mentoring

Consulting

Lecturing

Writing courses

Writing code



# ВЫВОДЫ

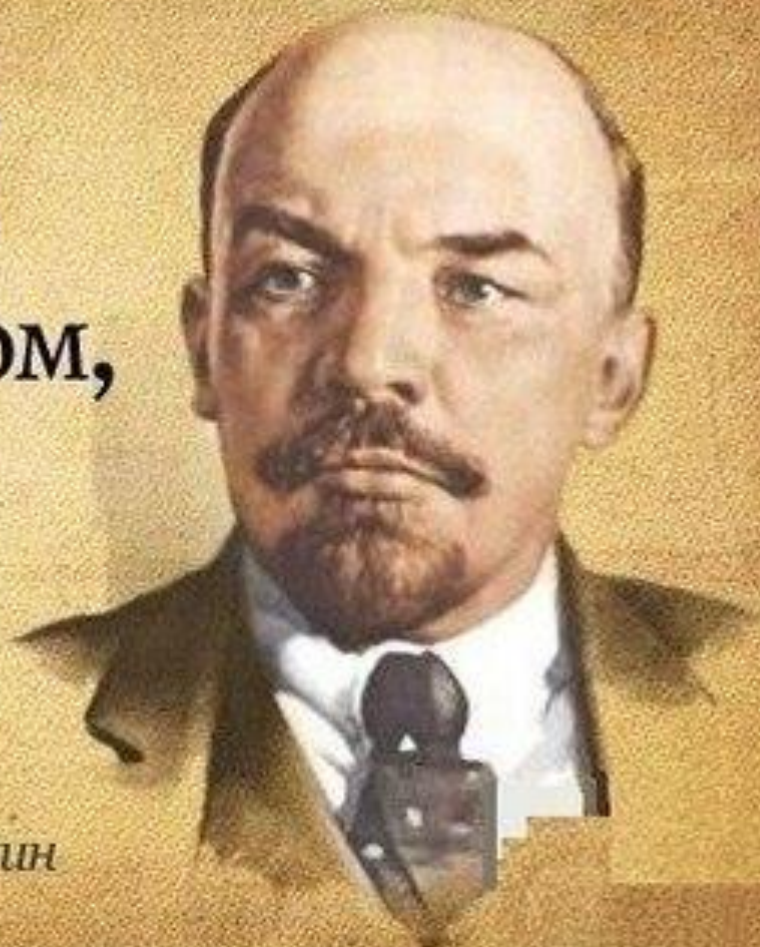
1. СПРИНГ НЕ ИДЕАЛЕН, НО ЛУЧШЕ НИЧЕГО НЕТ
2. СТРОЙТЕ СВОЮ ПЛАТФОРМУ НА СПРИНГЕ, НЕ ОГРАНИЧИВАЙТЕСЬ ТЕМА, ЧТО ОН МОЖЕТ САМА
3. НЕ БОЙТЕСЬ ЕГО ЧИНИТЬ, НО НЕ ЗАБЫВАЙТЕ ЧТО ОН ЧИНИТ СЕБЯ САМА
4. ЧИТАЙТЕ В СПЕКЕ ПРО ОБНОВЛЕНИЯ

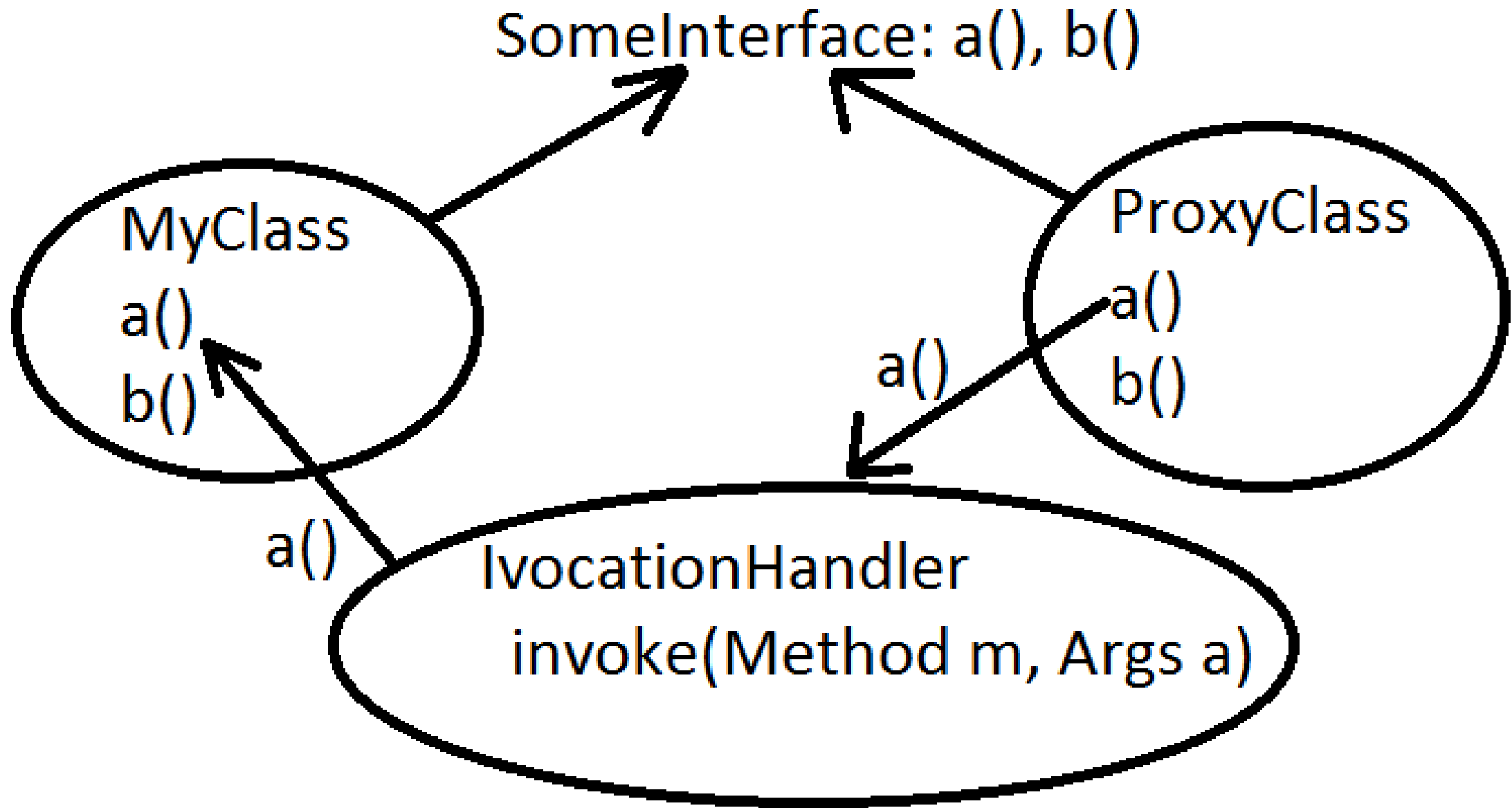


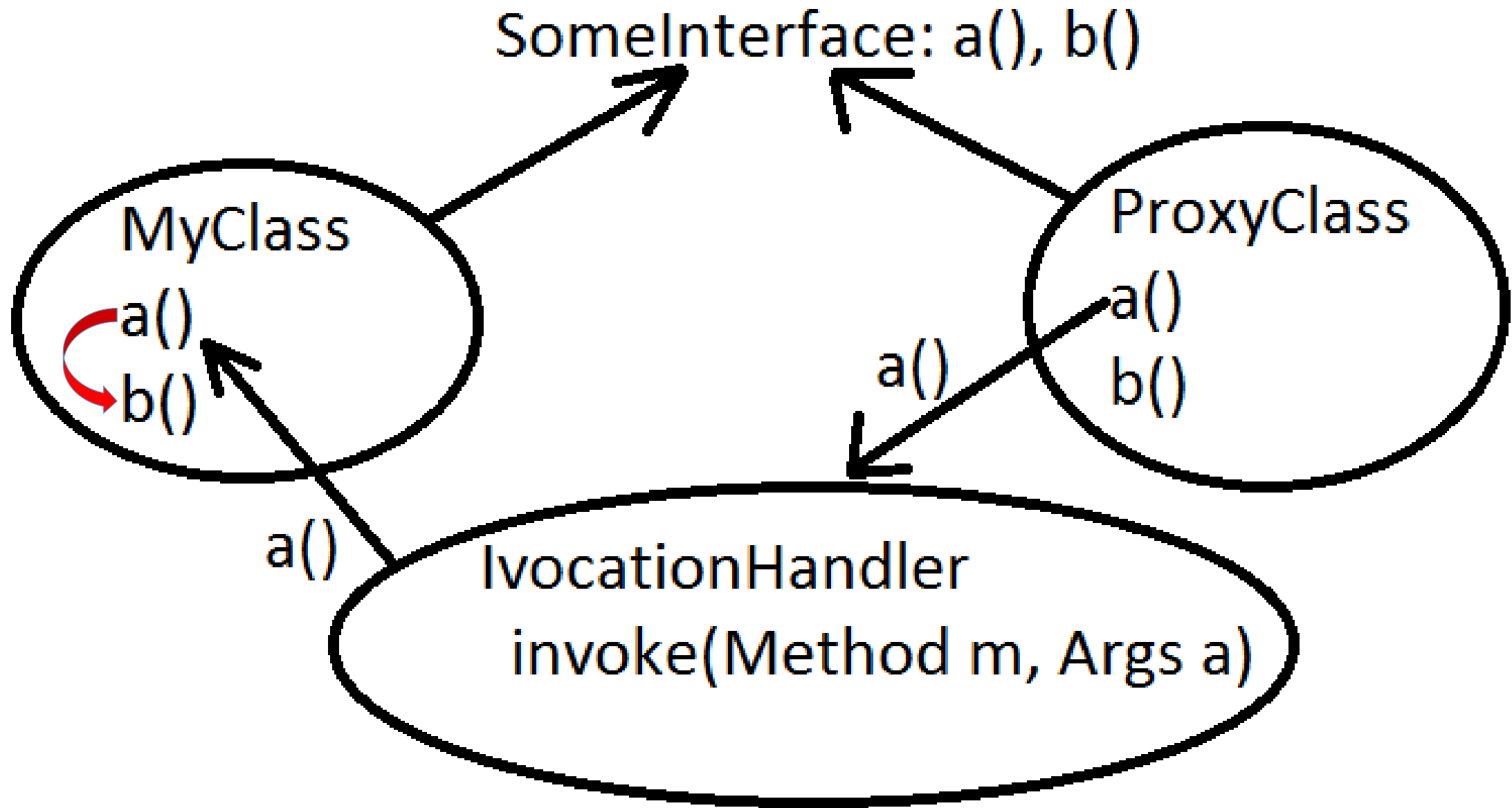
Давайте обновим старое

Проблема цитат  
в сети Интернет  
заключается в том,  
что люди  
**безоговорочно**  
верят в их  
подлинность.

— В.И. Ленин







# Самовпрыскивание







# Самовпрыскивание. До Спринга 4.3

- A. ☐ @Autowired
- B. ☐ @Inject
- C.  @Resource
- D. ☐ JavaConfig
- E.  XML



# Самовпрыскивание. После Спринга 4.3

- A.  @Autowired
- B.  @Inject
- C.  @Resource
- D. ☐ JavaConfig
- E.  XML



# TTR: 4.5 года!



Spring Framework / SPR-8450

## Support @Autowired-like self injection

Agile Board

### Details

Type:	New Feature	Status:	<b>CLOSED</b>
Priority:	Major	Resolution:	Complete
Affects Version/s:	3.0.5	Fix Version/s:	4.3 RC1
Component/s:	Core		
Labels:	None		
Last commented by a User:	false		

### Description

#### Background

Autowiring a bean with an instance of itself is not something one would normally do, but there are cases where it might be useful – for example, to route method calls *through* the proxy that wraps the bean. There are of course alternatives to this, such as using load-time weaving with AspectJ proxies instead of JDK dynamic proxies.

Note that self-autowiring *by name* via `@Resource` is permitted by the framework; whereas, self-autowiring *by type* is **not** permitted by the framework as can be seen in the following code snippet from `DefaultListableBeanFactory`'s `findAutowireCandidates(String, Class, DependencyDescriptor)` method.

### People

Assignee:	Juergen Hoeller
Reporter:	Sam Brannen
Last updater:	Sam Brannen
Votes:	18 Vote for this issue
Watchers:	22 Start watching this issue

### Dates

Created:	13/Jun/11 9:03 AM
Updated:	26/Jun/16 12:20 PM
Resolved:	25/Jan/16 8:18 PM
Days since last comment:	32 weeks, 6 days ago

### Agile

[View on Board](#)

# Статические методы для декларации бинов

```
@Service
public class SomeService {
    @Value("${best.conference}")
    private void init(String javaHome) {
        System.out.println(javaHome);
    }
}
```

```
@Configuration
@PropertySource("${application.properties}")
public class Config {

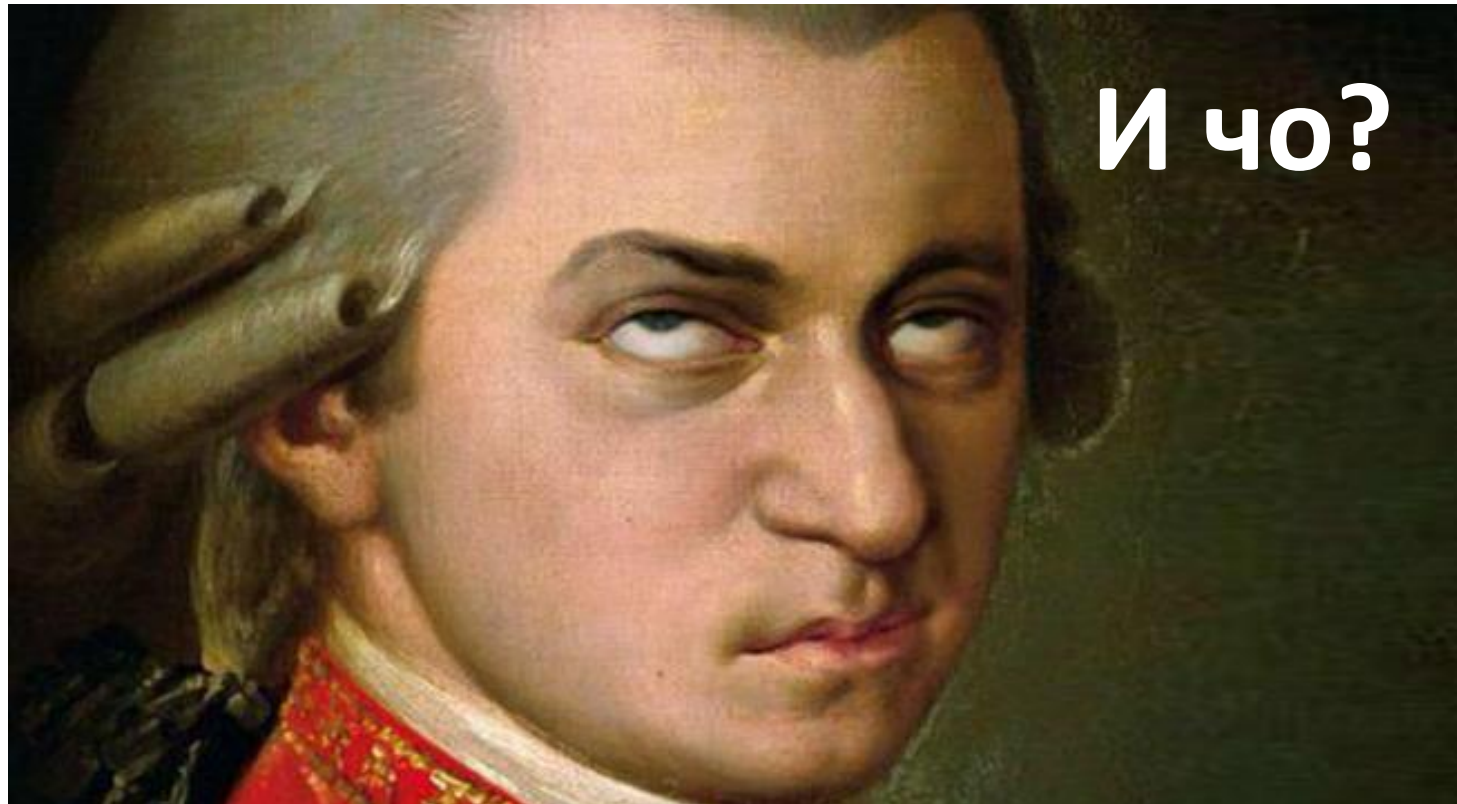
    @Bean
    public PropertySourcesPlaceholderConfigurer configurer(){
        return new PropertySourcesPlaceholderConfigurer();
    }
}
```

Что будет?

OK

# Вот такой warning:

- WARNING: @Bean method Conf.configurer is non-static and returns an object assignable to Spring's BeanFactoryPostProcessor interface





# Как мы это чиним?

- Добавляем static
- Не прописываем этот бин вообще – переходим на нормальный Spring

# Циркулярная зависимость



Последний раз спрашиваю:  
- Где зарплата?!

Выводы

**ЛЮБОВЬ важнее, чем 5000\$**  
**5000\$ фигня, главное любовь**

# Выводы

- Старайтесь избегать циркулярных зависимостей -  
Разбивайте класс на части
- Нельзя использовать @PostConstruct для вызова бизнес логики
- Используйте @Main



Сделаем для ленивых поддержку из коробки

**Я КОТЭ И Я НИЧЕГО  
НЕ ХОЧУ ДЕЛАТЬ**



**Я ХОЧУ МУР МУР МУР**

# Мальчик, который доверял интерфейсам



```
public interface УТЮГ {  
    @PostConstruct  
    void разогреваться();  
}
```

```
@Component  
public class Вода {  
    @Override  
    public String toString() {  
        return "вода";  
    }  
}
```

```
@Component  
public class СуперДуперУтюг implements УТЮГ {  
    @Override  
    public void разогреваться() {  
        System.out.println("разогреваюсь");  
    }  
  
    @PostConstruct  
    @Autowired  
    public void гретьВоду(Вода вода) {  
        System.out.println(вода+" грейся...");  
    }  
}
```

Что будет?

# IllegalStateException

- method annotation requires a no-arg method: `public void iLoveInterfaces.СуперДуперУтюг.гретьВоду`





# А зачем @PostConstruct, если стоит @Autowired

```
@PostConstruct
```

```
@Autowired
```

```
public void гретьВоду (Вода вода) {  
    System.out.println (вода+" грейся...") ;  
}
```

Winter is coming – утюги не работают



A still from the movie 'The Iron Giant' showing a young boy with blonde hair and a red knitted sweater looking at a large, injured man with a white bandage on his forehead. The background is a blurred indoor setting.

**АННОТАЦИИ НЕ РАБОТАЮТ**

**В ИНТЕРФЕЙСАХ**



Придётся написать BeanPostProcessor





Достал уже со своими BeanPostProcessors



Будем сегодня писать VRR?



# Будем сегодня писать VPP?

- Напишем BeanFactoryPostProcessor



# Не будем повторять ошибки Спринга

- Это не правильно, что за `PostConstruct` отвечает `VRP`



# BeanDefinition – это интерфейс

- AbstractBeanDefinition
- AnnotatedBeanDefinition
- ConfigurationClassBeanDefinition
- GenericBeanDefinition
- ScannedGenericBeanDefinition



# BeanDefinition – это интерфейс

- AnnotatedBeanDefinition
- AbstractBeanDefinition
- **ConfigurationClassBeanDefinition**
- GenericBeanDefinition
- ScannedGenericBeanDefinition

Хрен тебе

- Annotation
- AbstractBeanDefinition
- **ConfigurationClassBeanDefinition**
- GenericBeanDefinition
- ScannedGenericBeanDefinition





```
class ConfigurationClassBeanDefinitionReader {
```

Package  
friendly

```
    private static class ConfigurationClassBeanDefinition  
        extends RootBeanDefinition implements AnnotatedBeanDefinition
```

```
}
```

Когда не получается закастить по хорошему...



Пошли ковырять то, что спрятано



А если нам надо 2 init метода в интерфейсе?

Хочу дефолтный  
дефолтный запускать!!!





В 4.2 что что писали про default methods

## Framework 4.2

### 5.1 Core Container Improvements

- Annotations such as `@Bean` get detected and processed on Java 8 default methods as well, allowing for composing a configuration class from interfaces with default `@Bean` methods.
-

# Призрачные отголоски XML



Ошибки архитектуры  
14 лет назад ещё  
аукнутся вам...

# Выводы

- Init method есть один, @PostConstruct сколько угодно
- @PostConstruct Defaults methods работают с 4.2
- Задумайтесь, надо ли оно
- Если во время не сделан refactor, потом всю жизнь ходить на костылях

# Сага о двух программистах



```
@Configuration
public class BaruchConfig {

    @Bean
    public String str1() {return "Groovy";}

    @Bean
    public String str2() {return "Spring";}

}
```

```
@Service
public class BaruchService {
    @Autowired
    private List<String> list;
}
```

```
@Configuration
public class JekaConfig {
    @Bean
    public List<String> messages() {
        ArrayList<String> strings = new ArrayList<>();
        strings.add("Groovy");
        strings.add("Spring");
        return strings;
    }
}
```

```
@Service
public class JekaService {
    @Autowired
    private List<String> list;
}
```

## Что заинжектится в лист?

# Как-то на корпоративе





Пока Жека дров не наломал я подстрахуюсь

Добавлю ка я Qualifier  
на свои бины



А теперь можно добавить Артифактори



Какого хрена тут Артифактори делает??




# Как мы это чиним?

 ArrayList<String> вместо List<String>?

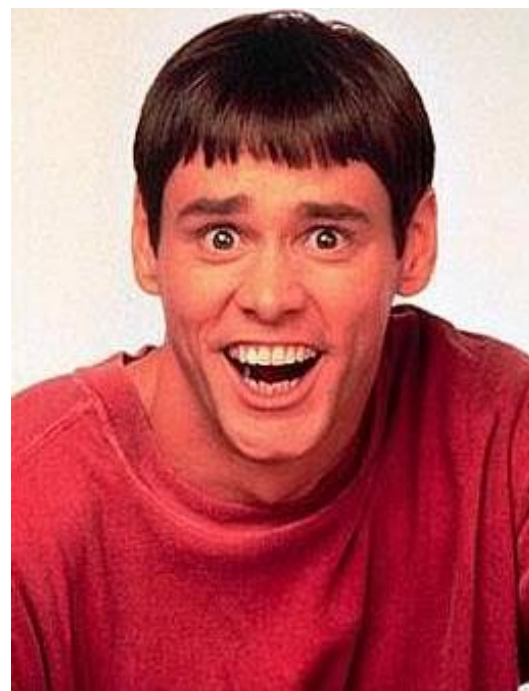
 `@Autowired`  
`@JekaQualifier`  
`private List<List<String>> list;`

 Не надо делать бин, который лист. Работаем с квалифаером

 Используйте нормальную версию Спринга (4.3+)







```
@Retention(RUNTIME)
@Qualifier
public @interface Comedy{}
```

```
@Retention(RUNTIME)
@Qualifier
public @interface Action{}
```

```
@Retention(RUNTIME)
@Qualifier
public @interface Melodrama{}
```

```
@Retention(RUNTIME)
@Qualifier
@Comedy @Action @Melodrama
public @interface AnyGenre{}
```

```
public interface Actor {
    void play();
}
```

```
@Component @Comedy
public class Jim implements Actor {
```



```
@Component @Melodrama
public class Julia implements Actor {
```



```
@Component @Action
public class Tobey implements Actor {
```



```
@Component @AnyGenre
public class Chuck implements Actor {
```



```
@Component @Action @Comedy
public class Stallone implements Actor {
```



```
@Component
public class Katy implements Actor {
```



## Что заинжектится в ЭТОТ ЛИСТ?

```
@Service
public class Film {
    @Autowired
    @Comedy
    @Action
    private List<Actor> actors;
}
```

А как же я...



Что делает Spring, если есть больше чем 1 qualifier?

&

# Как мы это чиним?

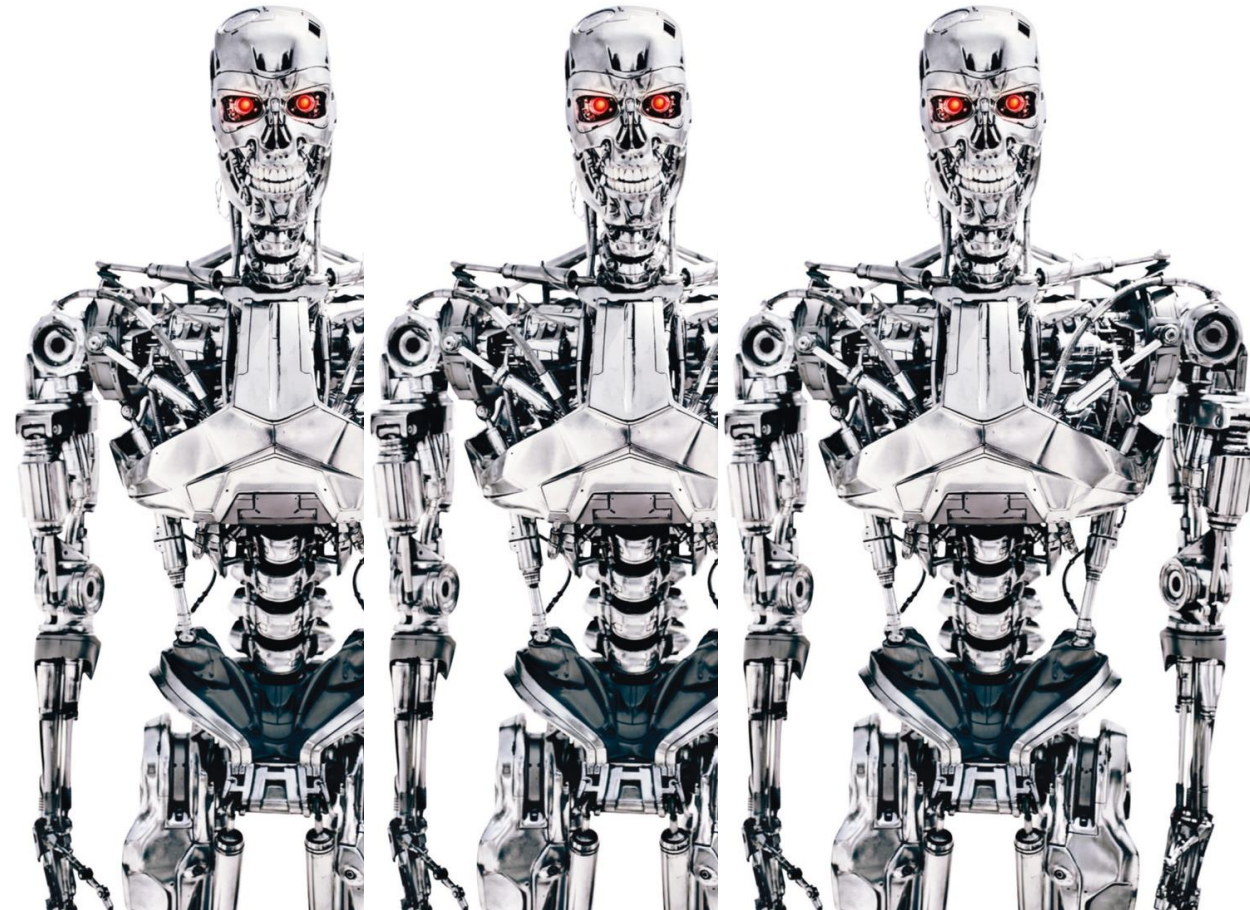
- Пишем свою обработку квалифаера



Так всё таки напишем VRR?



# Singleton vs Prototypes



```

@Component
@Scope("prototype")
public class T800 {
    @PostConstruct
    public void init() {
        System.out.println("Give me your clothes");
    }

    @PreDestroy
    public void destroy() {
        System.out.println("Astalavista baby");
    }
}

```

`context.close();`

```

@Component
@Scope("singleton")
public class T1000 {

    @Autowired
    private T800 t800;

    @PostConstruct
    public void init() {
        System.out.println("Where is John Connor");
    }

    @PreDestroy
    public void destroy() {
        System.out.println("Страшные звуки");
    }
}

```

# Чего не будет?





Кто знает почему не сработал  
destroy method у T800?



# А если прописать destroy method в xml-е?

- Что изменится?
- Появится xml!





@Predestroy не работает для прототипов





И даже warning-а не будет!  
Сами разбирайтесь

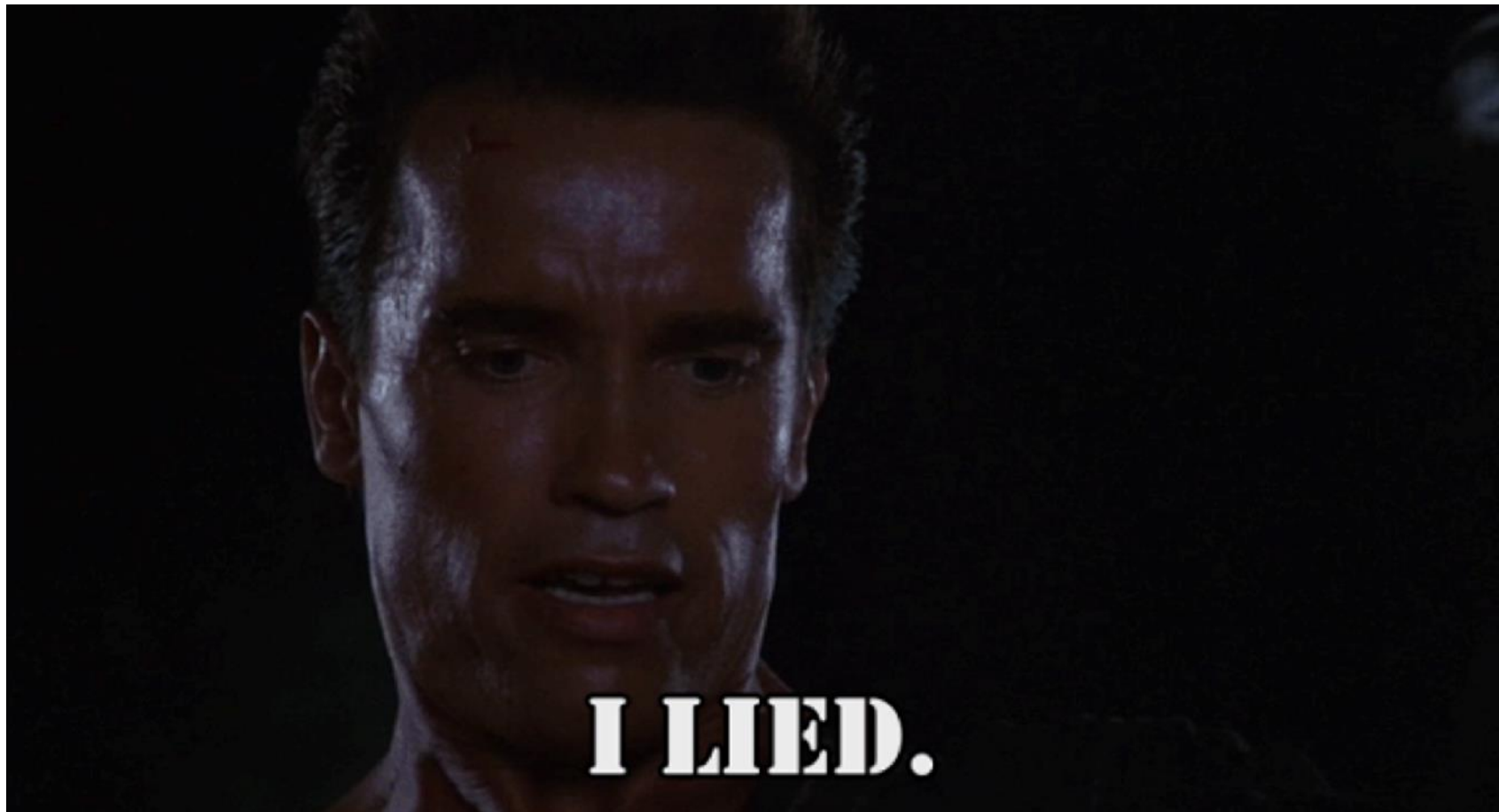
# Что надо сделать чтобы был Warning?

- Написать BeanFactoryPostProcessor?

A close-up, low-key photograph of Arnold Schwarzenegger. He is looking directly at the camera with a serious, intense expression. He is holding a handgun in his right hand, which is raised towards his face. The lighting is dramatic, with strong highlights on his face and the gun, and deep shadows in the background.

**REMEMBER WHEN I SAID I'D KILL YOU LAST?**

Мы напишем ВРР



# Где destroy method будет бежать асинхронно?

1. 

```
@Service
public class Тормоз {
    @PreDestroy
    public void всёЗаккрыть () {
        ... медленный код
    }
}
```
2. 

```
@Service
public class Тормоз {
    @PreDestroy
    @Async
    public void всёЗаккрыть () {
        ... медленный код
    }
}
```
3. 

```
@Bean(destroyMethod = "всёЗаккрыть")
```
4. 

```
<bean class="Тормоз" destroy-method="всёЗаккрыть"/>
```

A. Во всех случаях



Ни в каком

C. Только 2, если не забыли @EnableAsync

D. 3 & 4





А мы этой фигнёй не  
занимаемся

# Как мы это чиним?

- Ну например...

```
@PreDestroy
public void killAll() {
    Executors.newSingleThreadExecutor().submit(() ->
        System.out.println("close all"));
}
```

Спринг и Груви. Друзья или нет?



# Что будет?

```
@RestController
class MyController {
    @RequestMapping(value = '/rent', method = RequestMethod.GET)
    def rent(@RequestParam Integer count = 1) {

        ... красивый код на груви

    }
}
```



IllegalStateException

- B. Всё будет работать
- C. ArrayBoundException
- D. Compilation Failure



# Кто знает какой message у exception-a?

```
@RestController
class MyController {
    @RequestMapping(value = '/rent', method = RequestMethod.GET)
    def rent(@RequestParam Integer count = 1) {

        ... красивый код на груви

    }
}
```

- A. IllegalStateException
- B. Всё будет работать
- C. ArrayBoundException
- D. Compilation Failure

**java.lang.IllegalStateException: Ambiguous mapping**

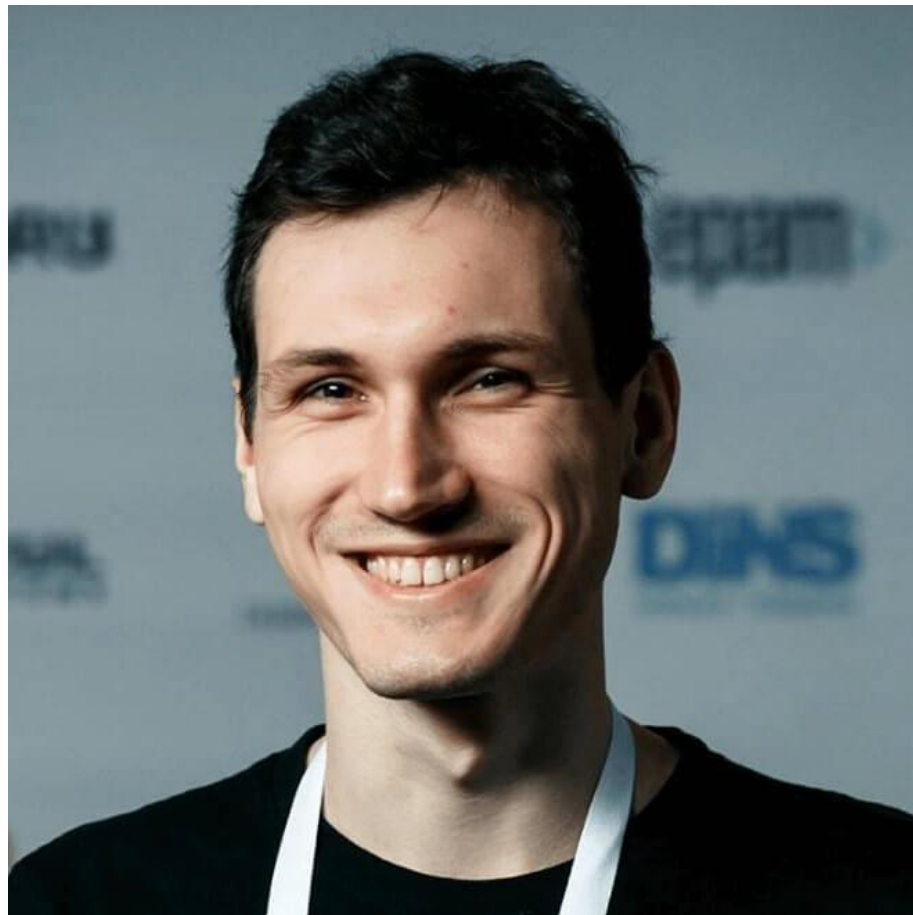


Как мы это чиним?

Не используем Груви



Это был паззлер от Кирилла Толкачёва



# Смерть XML-y







ДЕВОЧКА, КОТОРАЯ ЛЮБИЛА XML

# Битва фабрик





# Где Integer будет prototype?

1)

```
<bean id="integer"
      class="factoryBeans.IntegerFactory"
      scope="prototype"/>
```

```
public class IntegerFactory implements FactoryBean<Integer> {
    private Random random = new Random();

    @Override
    public Integer getObject() throws Exception {
        return random.nextInt(10);
    }

    @Override
    public Class<?> getObjectType() {
        return Integer.class;
    }

    @Override
    public boolean isSingleton() {
        return true;
    }
}
```

2)

```
<bean id="integerFactory"
      class="factoryBeans.IntegerFactory"
      scope="prototype"/>
```

```
<bean id="integer"
      class="java.lang.Integer"
      scope="singleton"
      factory-bean="integerFactory"
      factory-method="getInt"/>
```

A. Только 1

B. Только 2



Оба

D. Ни один



# Где integer будет prototype?

1)

```
<bean id="integer"
      class="factoryBeans.IntegerFactory"
      scope="prototype"/>
```

```
public class IntegerFactory implements FactoryBean<Integer> {
    private Random random = new Random();

    @Override
    public Integer getObject() throws Exception {
        return random.nextInt(10);
    }

    @Override
    public Class<?> getObjectType() {
        return Integer.class;
    }

    @Override
    public boolean isSingleton() {
        return true;
    }
}
```

2)

```
<bean id="integerFactory"
      class="factoryBeans.IntegerFactory"
      scope="prototype"/>
```

```
<bean id="integer"
      class="java.lang.Integer"
      scope="singleton"
      factory-bean="integerFactory"
      factory-method="getInt"/>
```

A. Только 1

B. Только 2



Оба

D. Ни один

И чо?





# Попробуйте сделать scope session

1)

```
<bean id="integer"  
      class="factoryBeans.IntegerFactory"  
      scope="???" />
```

```
public class IntegerFactory implements FactoryBean<Integer> {  
    private Random random = new Random();  
  
    @Override  
    public Integer getObject() throws Exception {  
        return random.nextInt(10);  
    }  
  
    @Override  
    public Class<?> getObjectType() {  
        return Integer.class;  
    }  
  
    @Override  
    public boolean isSingleton() {  
        return ???;  
    }  
}
```

# Попробуйте сделать scope session

1)

```
<bean id="integer"
      class="factoryBeans.IntegerFactory"
      scope="session"/> //не забыть proxy-mode
```

```
public class IntegerFactory implements FactoryBean<Integer> {
    private Random random = new Random();

    @Override
    public Integer getObject() throws Exception {
        return random.nextInt(10);
    }

    @Override
    public Class<?> getObjectType() {
        return Integer.class;
    }

    @Override
    public boolean isSingleton() {
        return true;
    }
}
```

2)

```
<bean id="integerFactory"
      class="factoryBeans.IntegerFactory"
      scope="singleton"/>
```

```
<bean id="integer"
      class="java.lang.Integer"
      factory-bean="integerFactory"
      factory-method="getInt"
      scope="session"/> //не забыть proxy-mode
```



# ВЫВОДЫ

1. СПРИНГ НЕ ИДЕАЛЕН, НО ЛУЧШЕ НИЧЕГО НЕТ
2. НЕ БОЙТЕСЬ ЕГО ЧИНИТЬ, НО НЕ ЗАБЫВАЙТЕ ЧТО ОН ЧИНИТ СЕБЯ САМ
3. ЧИТАЙТЕ В СПЕКЕ ПРО ОБНОВЛЕНИЯ
4. СТРОЙТЕ СВОЮ ПЛАТФОРМУ НА СПРИНГЕ, НЕ ОГРАНИЧИВАЙТЕСЬ ТЕМА, ЧТО ОН МОЖЕТ САМ
5. И ГЛАВНОЕ :



ВЫВОДЫ

АННОТАЦИИ — ЭТО  
ДОБРО!!!

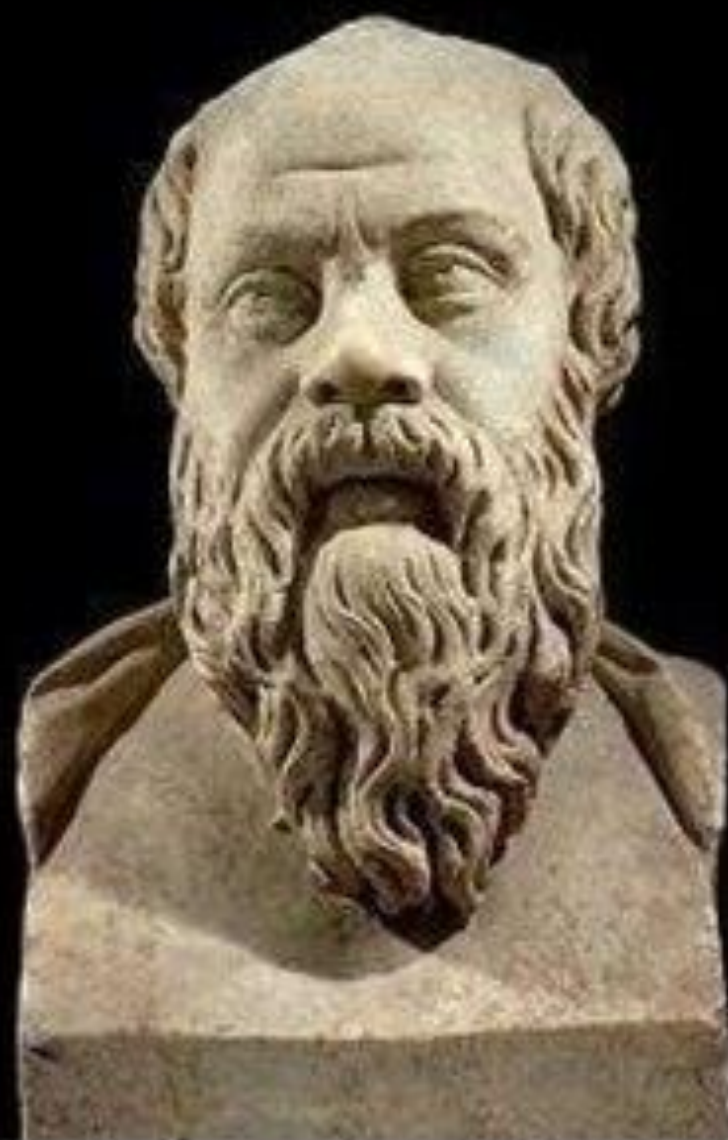


СПАСИБО ЗА ПРИЗЫ

JUG.RU

**"Расставашки -  
всегда пичалька".**

**(с) Сократик**



- Есть несколько конфигураций, у которых что делается в постконтракте и есть обычные бины которые тоже что то делают в постконтракте, вопрос чей постконтракт раньше отработает.
- На самом деле кейс такой: у меня есть одна конфигурация, которая инжектит в себя бин, и что то ему подкручивает в постконтракте, и я должен быть уверен, что когда этот бин заинжектится в другие бины, он уже будет подкручен. Например его какой-то бин инжектит через setter, и вытаскивает у этого бина тот проперти, который уже должен был быть подкручен. Или какой-то бин в постконтракте хочет воспользоваться этим подкрученным проперти.
- Кому это всё нужно? Ну вот например, приходит какой то бин прописанный не в нашей конфигурации, мы её в нашу заимпортировали, и мне ему надо что-то подкрутить перед тем, как он будет всем инжектиться. Причем как не правильный ответ на вопрос как это сделать можно сказать, что мы его инжектим в нашу конфигурацию, и в ней прописываем такой-же бин используя этот. Но это не сработает (надо точно проверить) потому что теперь будет 2 бина этого из одного класса или интерфейса. Правда если у них будут совпадать имена, тогда вообще будет веселье, проверить что будет в этом случае. Короче тут будет на 2 пазлера.

# Наследование бинов в джаваконфиге



@Repetable

# Йорген против Ланистера

- Два бина – один мок и один настоящий
- Два проперти файла с одим и тем же проперти
- Что будет в тестах и что будет в продакшне?
- Проперти для тестов победит в тестах, а в продакшне победит продакшн, А вот бин в тестах создастся оба два.
- Надо разобраться почему когда я в одном тесте добавляю `@ContextConfiguration(class=MockBeansConfig.class)` и в этот конфиге задавливаю не нужные бины это ломает другие тесты, хотя если каждый запускать по отдельности всё работает.

SelfInject От 4.3.RELEASE









