

# Spring Boot

## The Ripper





### Big Data & Java Technical Leader

- Mentoring
- Consulting
- Lecturing
- Writing courses
- Writing code



I Never write singletons  
I drink them and  
I know Spring





@tolkv

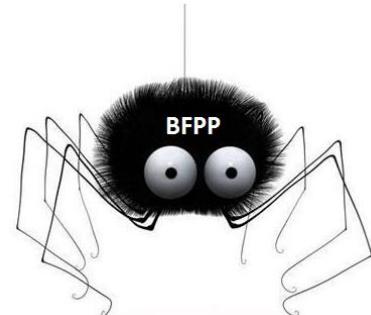
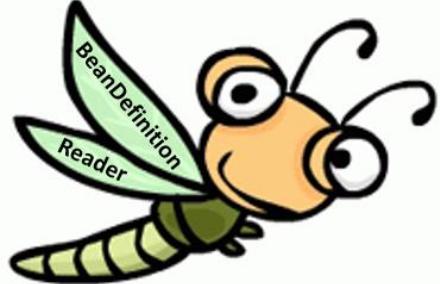


@lavcraft



Thanks to Kirill Tolkachev

# Spring retrospective









# Spring Boot

The Ripper



Demo application



The background features the Game of Thrones logo, which consists of a circular emblem with a three-headed dragon on the left and a two-headed eagle on the right, facing each other. The emblem is set against a dark, textured background that looks like stone or metal. The title "GAME OF THRONES" is written in large, gold-colored letters across the center of the emblem.

# GAME OF THRONES

Demo

# Developer doesn't like

Think about

1. Dependencies...
2. Configurations

Do:

1. Deployment (test/local/prod)

# Developer doesn't like

Think about



1. Dependencies...
2. Configurations

Do:



1. Deployment (test/local/prod)

# Dependencies

# pom.xml

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.1.5.RELEASE</version>
</parent>
```

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.1.5.RELEASE</version>
</parent>
```



```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-dependencies</artifactId>
  <version>2.1.5.RELEASE</version>
</parent>
```

```
2414 <groupId>org.webjars</groupId>
2415 <artifactId>webjars-locator</artifactId>
2416 <version>${webjars-locator.version}</version>
2417 </dependency>
2418 <dependency>
2419   <groupId>org.yaml</groupId>
2420   <artifactId>snakeyaml</artifactId>
2421   <version>${snakeyaml.version}</version>
2422 </dependency>
2423 <dependency>
2424   <groupId>redis.clients</groupId>
2425   <artifactId>jedis</artifactId>
2426   <version>${jedis.version}</version>
2427 </dependency>
2428 <dependency>
2429   <groupId>wsdl4j</groupId>
2430   <artifactId>wsdl4j</artifactId>
2431   <version>${wsdl4j.version}</version>
2432 </dependency>
2433 <dependency>
2434   <groupId>xml-apis</groupId>
2435   <artifactId>xml-apis</artifactId>
2436   <version>${xml-apis.version}</version>
2437 </dependency>
2438 </dependencies>
2439 </dependencyManagement>
```





Not my parent!

ВЕРНИТЕ

НАШИХ

РОДИТЕЛЕЙ

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>io.spring.platform</groupId>
      <artifactId>platform-bom</artifactId>
      <version>2.0.8</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

# build.gradle

```
dependencyManagement {  
    imports {  
        mavenBom 'org.springframework.cloud:spring-cloud-dependencies:Dalston.RELEASE'  
    }  
}
```

# No more versions in out build.gradle...

```
'org.springframework.boot:spring-boot-starter-web'
```

```
'org.springframework.boot:spring-boot-starter-data-jpa'
```

```
'com.h2database:h2'
```

# Spring Context configuration

# Where is the context?

```
@SpringBootApplication  
class App {  
    public static void main(String[] args) {  
        SpringApplication.run(App.class, args);  
    }  
}
```



# The context

A dark, atmospheric image of the Night King from Game of Thrones. He is a pale, skeletal figure with blue glowing eyes, wearing a black cloak and a crown of white, fang-like bones. His hands are raised in a commanding gesture. The background is a cold, snowy landscape under a dark sky.

```
@SpringBootApplication  
class App {  
    public static void main(String[] args) {  
        ApplicationContext context =  
            SpringApplication.run(App.class, args);  
    }  
}
```

# Run method arguments

Дано: RipperApplication.class

```
public... main(String[] args) {  
    SpringApplication.run(?, args);  
}
```

1. RipperApplication.class
2. String.class
3. "context.xml"
4. new ClassPathResource("context.xml")
5. Package.getPackage("conference.spring.boot.ripper")



SpringApplication.run **Object[] sources, String[] args**)

# Run method arguments

Lets Vote!

RipperApplication.class:

```
public... main(String[] args) {  
    SpringApplication.run(?, args);  
}
```

1. RipperApplication.class
2. String.class
3. "context.xml"
4. new ClassPathResource("context.xml")
5. Package.getPackage("conference.spring.boot.ripper")

A woman with blonde hair tied back in a bun is laughing heartily, her mouth wide open. She is wearing a blue patterned top. The background shows a coastal landscape with rocky cliffs and green grass on the left, and a calm sea under a cloudy sky on the right.

All correct



```
SpringApplication.run(Object[] sources, String[] args)
```

```
# APPLICATION SETTINGS (SpringApplication)
```

```
spring.main.sources= # class name, package name, xml location
```

```
spring.main.web-environment= # true/false
```

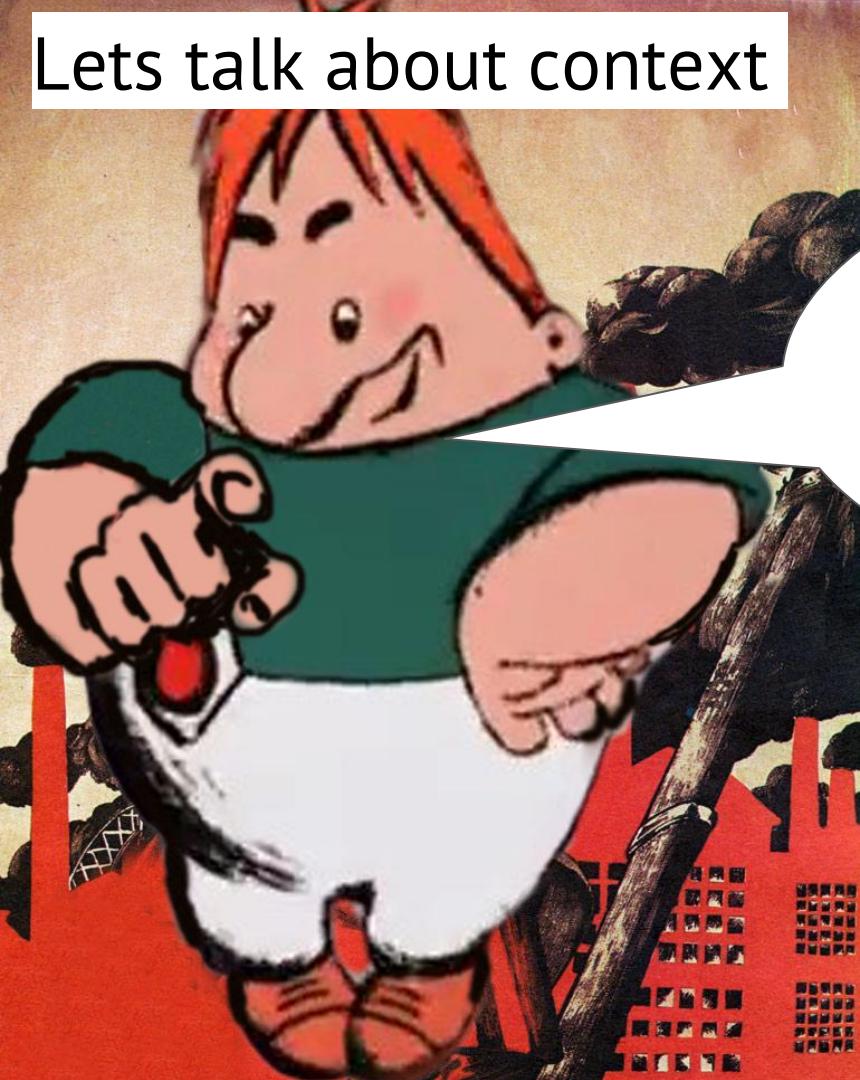
```
spring.main.banner-mode=console # Log/off
```



# SpringApplication



# Lets talk about context



Every child knows about  
`ClassPathXmlApplicationContext`  
And you?



# ConfigurableApplicationContext types

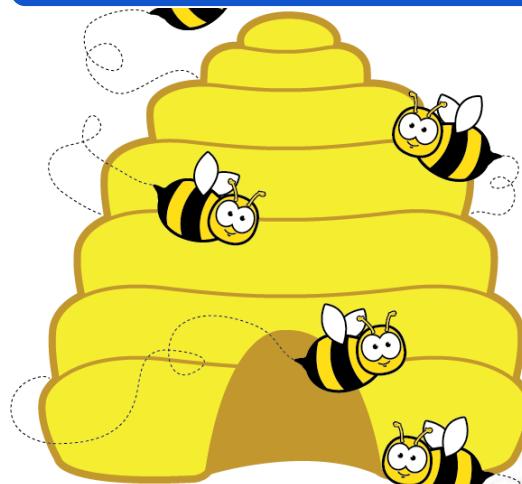
- c AnnotationConfigApplicationContext (org.springframework.context.annotation)
- c AnnotationConfigEmbeddedWebApplicationContext (org.springframework.boot.context.embedded)
- c AnnotationConfigWebApplicationContext (org.springframework.web.context.support)
- c ClassPathXmlApplicationContext (org.springframework.context.support)
- I ConfigurableWebApplicationContext (org.springframework.web.context)
- c EmbeddedWebApplicationContext (org.springframework.boot.context.embedded)
- c FileSystemXmlApplicationContext (org.springframework.context.support)
- c GenericApplicationContext (org.springframework.context.support)
- c GenericGroovyApplicationContext (org.springframework.context.support)
- c GenericWebApplicationContext (org.springframework.web.context.support)
- c GenericXmlApplicationContext (org.springframework.context.support)
- c GroovyWebApplicationContext (org.springframework.web.context.support)
- c StaticApplicationContext (org.springframework.context.support)
- c StaticWebApplicationContext (org.springframework.web.context.support)
- c XmlEmbeddedWebApplicationContext (org.springframework.boot.context.embedded)

# I decide what context should be

I do a lot of things  
before context  
creation...



## Web Context



## Generic Context



If there Servlet in  
classpath...



If

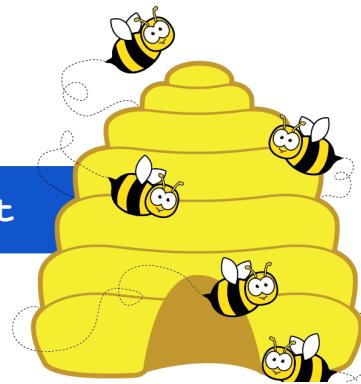
`javax.servlet.Servlet`



`ConfigurableWebApplicationContext`



`AnnotationConfigEmbeddedWebApplicationContext`



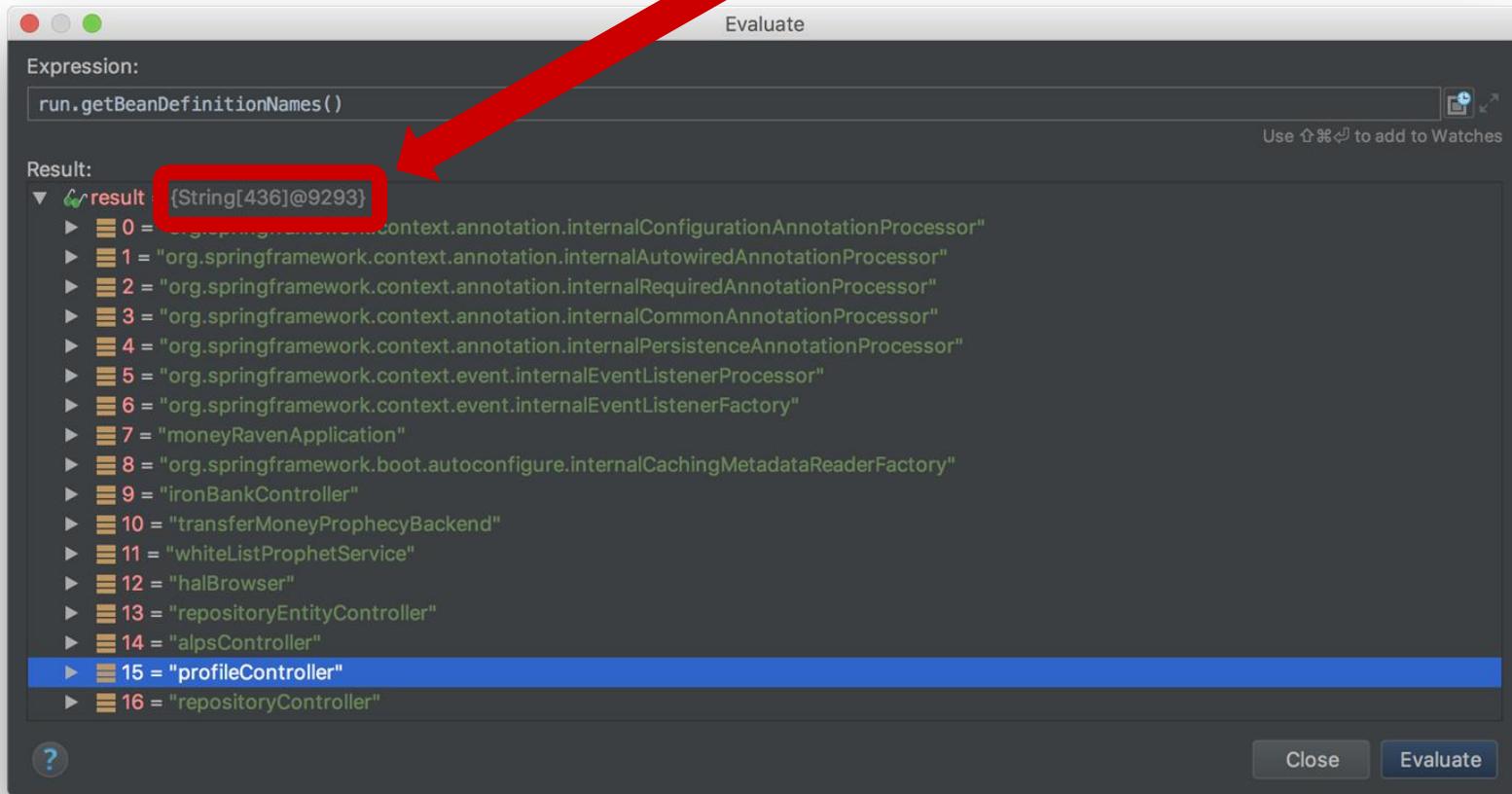
Else



`AnnotationConfigApplicationContext`

How many beans already exists in context?

# 436 spring beans





Do you know where all  
this beans came from?

No, but I want to do the same



# Why do we need microservices architecture?

- Dependency hell
- Bootstrap time
- Testability
- Sharding
- Work share

# Why do we need starters

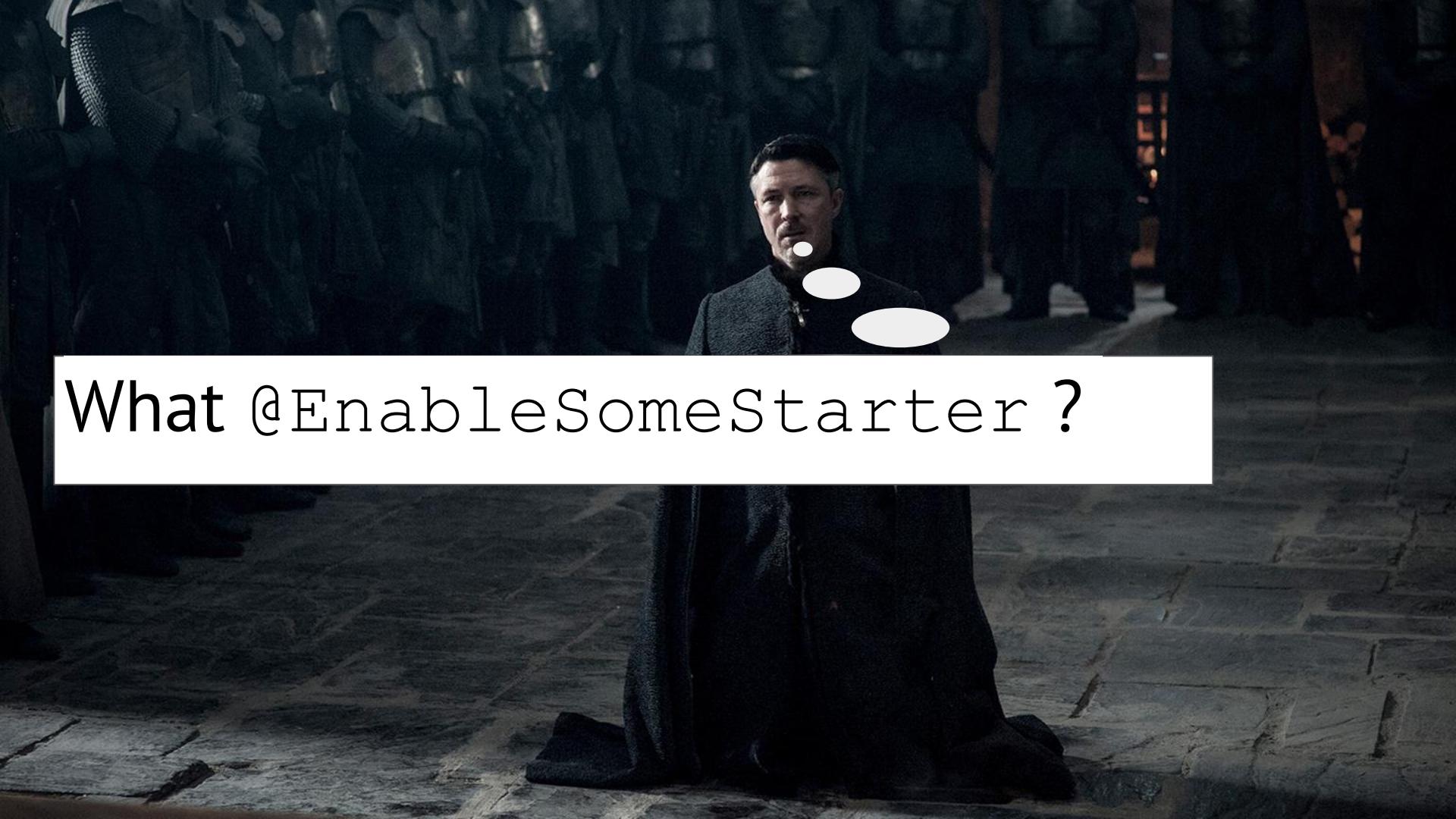
- All infrastructure configurations and beans comes from starters



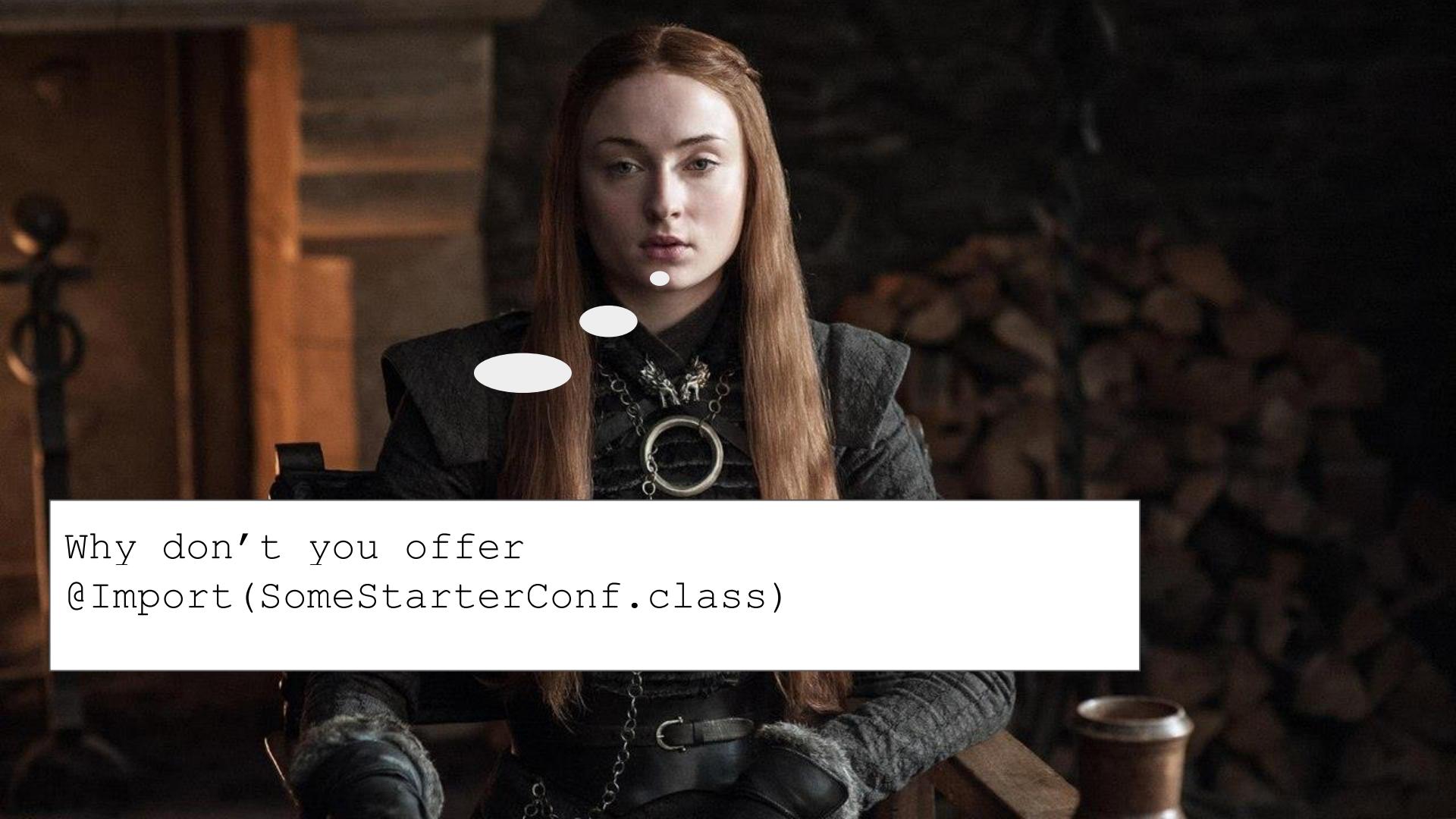
# Iron bank law №1

---

Every time when NotEnoughMoneyException –send the raven



What @EnableSomeStarter ?



Why don't you offer  
@Import(SomeStarterConf.class)



We will use **spring.factories**!

# What is `spring.factories` ?



## § 44.1 Understanding auto-configured beans

Under the hood, auto-configuration is implemented with standard `@Configuration` classes. Additional `@Conditional` annotations are used to constrain when the auto-configuration should apply. Usually auto-configuration classes use `@ConditionalOnClass` and `@ConditionalOnMissingBean` annotations. This ensures that auto-configuration only applies when relevant classes are found and when you have not declared your own `@Configuration`.

You can browse the source code of [spring-boot-autoconfigure](#) to see the `@Configuration` classes that we provide (see the [META-INF/spring.factories](#) file).

A dramatic close-up of a young man with curly brown hair, wearing dark armor with shoulder guards. He has a wide-eyed, shocked expression with his mouth wide open, looking upwards and to the right. The background is dark and out of focus.

Inversion of control!!!

# @SpringBootApplication



# @SpringBootApplication

- @ComponentScan
- @Configuration
- @EnableAutoConfiguration

# @SpringBootApplication

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
    @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
    @Filter(type = FilterType.CUSTOM, classes =
AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {
    ...
}
```

# @SpringBootApplication

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
    @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
    @Filter(type = FilterType.CUSTOM, classes =
AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {

    ...
}
```

# @EnableAutoConfiguration

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@AutoConfigurationPackage
@Import({EnableAutoConfigurationImportSelector.class})
public @interface EnableAutoConfiguration {
    String ENABLED_OVERRIDE_PROPERTY = "spring.boot.enableautoconfiguration";
    Class<?>[] exclude() default {};
    String[] excludeName() default {};
}
```

# @EnableAutoConfiguration

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@AutoConfigurationPackage
@Import({EnableAutoConfigurationImportSelector.class})
public @interface EnableAutoConfiguration {
    String ENABLED_OVERRIDE_PROPERTY = "spring.boot.enableautoconfiguration";
    Class<?>[] exclude() default {};
    String[] excludeName() default {};
}
```

# @EnableAutoConfiguration

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@AutoConfigurationPackage
@Import({EnableAutoConfigurationImportSelector.class})
public @interface EnableAutoConfiguration {
    String ENABLED_OVERRIDE_PROPERTY = "spring.boot.enableautoconfiguration";
    Class<?>[] exclude() default {};
    String[] excludeName() default {};
}
```



# @EnableAutoConfiguration



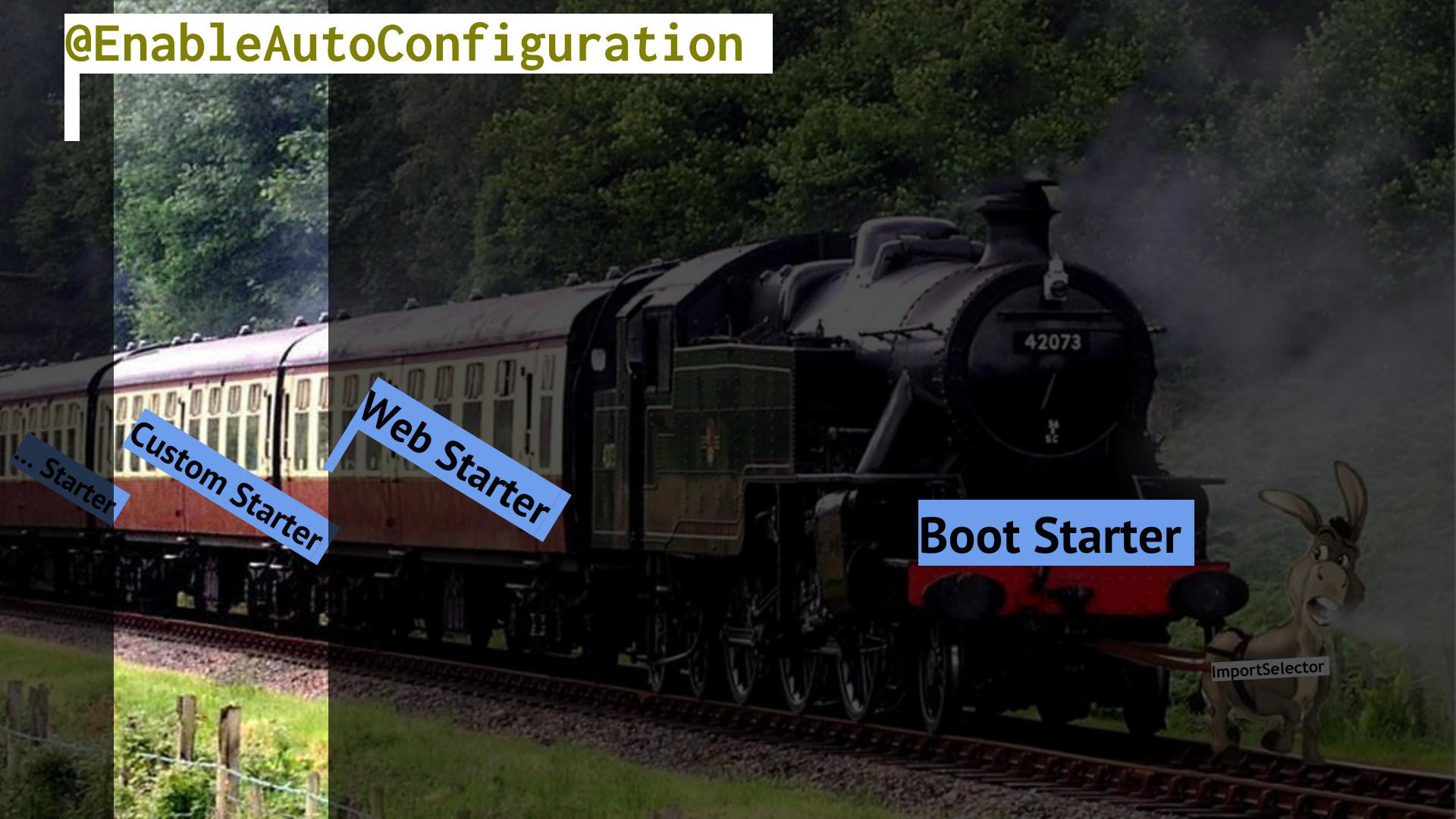
Iron Starter  
... more jars

Web Starter

Boot Starter

ImportSelector

# @EnableAutoConfiguration



# Spring Factories Loader



# SpringFactoriesLoader

```
static <T> List<T> loadFactories(  
    Class<T> factoryClass,  
    ClassLoader cl  
)
```

```
static List<String> loadFactoryNames(  
    Class<?> factoryClass,  
    ClassLoader cl  
)
```



A dramatic, close-up shot of a man with dark, curly hair. He has a shocked or intense expression, with his mouth wide open and eyes wide. He appears to be wearing a dark, textured collar or armor. The background is dark and out of focus.

Inversion of control!!!



Not totally

# spring-boot-autoconfigure.jar/spring.factories

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\ 
org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,\ 
org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,\ 
org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration.\
```

...

...

# spring-boot-autoconfigure.jar/spring.factories

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\ 
org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,\ 
org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,\ 
org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration.\
```

...

...

# spring-boot-autoconfigure.jar

```
@Import(CacheConfigurationImportSelector.class)  
public class CacheAutoConfiguration {
```

# Somewhere deep in CacheAutoConfiguration

```
for (int i = 0; i < types.length; i++) {  
    Imports[i] = CacheConfigurations.getConfigurationClass(types[i]);  
}  
return imports;
```

# CacheConfigurations

```
private static final Map<CacheType, Class<?>> MAPPINGS;
static {
    Map<CacheType, Class<?>> mappings = new HashMap<CacheType, Class<?>>();
    mappings.put(CacheType.GENERIC, GenericCacheConfiguration.class);
    mappings.put(CacheType.EHCACHE, EhCacheCacheConfiguration.class);
    mappings.put(CacheType.HAZELCAST, HazelcastCacheConfiguration.class);
    mappings.put(CacheType.INFINISPAN, InfinispanCacheConfiguration.class);
    mappings.put(CacheType.JCACHE, JCacheCacheConfiguration.class);
    mappings.put(CacheType COUCHBASE, CouchbaseCacheConfiguration.class);
    mappings.put(CacheType.REDIS, RedisCacheConfiguration.class);
    mappings.put(CacheType.CAFFEINE, CaffeineCacheConfiguration.class);
    addGuavaMapping(mappings);
    mappings.put(CacheType.SIMPLE, SimpleCacheConfiguration.class);
    mappings.put(CacheType.NONE, NoOpCacheConfiguration.class);
    MAPPINGS = Collections.unmodifiableMap(mappings);
}
```

JL.

Hardcoded



Will all this configuration always be loaded??



Calm down @Conditional  
Will filter most of them



What conditions?





# Iron bank law №1.2

Raven should be sent only in production

**@ConditionalOnProduction – Demo**

# Puzzler

```
@Configuration
@ConditionalOnWinter
public class UndeadArmyConfiguration {
    ...
}

@Configuration
public class DragonIslandConfiguration {
    @Bean
    @ConditionalOnWinter
    public DragonGlassFactory dragonGlassFactory() {
        return new DragonGlassFactory();
    }
    ...
}
```

# ConditionalOnWinter

1. 100 \$
2. 200 \$
3. 300 or 400 \$
4. between 500 and 1000 \$
5. More than 1000 \$

# Correct answer

- 
1. 100 \$
  2. 200 \$
  - 300 or 400 \$
  4. between 500 and 1000 \$
  5. more than 1000 \$

A close-up photograph of a man with light brown hair and a beard, looking upwards with a surprised or questioning expression. He is wearing a dark, textured jacket. The background is blurred.

How does  
it work?

Как то так, через хитро  
закрученную жопу  
оно и работает



# How to cache condition answer?



# How to cache condition answer? It's not working



# Use static





Where will the Raven fly?

Лететь то куда?

# Iron bank law №1.3

Raven should be sent only if destination is configured

*Autocomplete for our properties will be nice...*

# Autocomplete for properties

Generate file: additional-spring-configuration-metadata.json

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

# Conditional and friends

@ConditionalOnBean

@ConditionalOnClass

@ConditionalOnCloudPlatform

@ConditionalOnExpression

@ConditionalOnJava

@ConditionalOnJndi

@ConditionalOnMissingBean

@ConditionalOnMissingClass

@ConditionalOnNotWebApplication

@ConditionalOnProperty

@ConditionalOnResource

@ConditionalOnSingleCandidate

@ConditionalOnWebApplication

...

# Iron law №1.4

ExceptionHandler from starter should work only in there is no custom bean: ExceptionHandler



# ConditionalOnPuzzler

```
@Configuration
public class ExecutionConfig {

    @Bean
    @ConditionalOnClass({Soap.class, Roap.class})
    @ConditionalOnMissingBean({ExecutionFactory.class})
    public ExecutionFactory gallows() { return new GallowsFactory("..."); }

}
```

# ConditionalOnPuzzler

```
@Configuration
public class ExecutionConfig {

    @Bean
    @ConditionalOnClass({Soap.class, Roap.class})
    @ConditionalOnMissingBean({ExecutionFactory.class})
    public ExecutionFactory gallows() { return new GallowsFactory("..."); }

    @Bean
    @ConditionalOnClass({Chair.class, Electricity.class})
    @ConditionalOnMissingBean({ExecutionFactory.class})
    public ExecutionFactory chairs() { return new ChairFactory("zzz zzz"); }

}
```

# ConditionalOnPuzzler

```
@Configuration
public class ExecutionConfig {

    @Bean
    @ConditionalOnClass({Soap.class, Roap.class})
    @ConditionalOnMissingBean({ExecutionFactory.class})
    public ExecutionFactory gallows() { return new GallowsFactory("..."); }

    @Bean
    @ConditionalOnClass({Chair.class, Electricity.class})
    @ConditionalOnMissingBean({ExecutionFactory.class})
    public ExecutionFactory chairs() { return new ChairFactory("zzz zzz"); }

    @Bean
    @ConditionalOnClass({Guillotine.class, GoodMood.class})
    @ConditionalOnMissingBean({ExecutionFactory.class})
    public ExecutionFactory guillotine() { return new GuillotineFactory ("vjik"); }
}
```

# ConditionalOnPuzzler

```
@Configuration
```

```
public class ExecutionConfig {
```

1. ClassDefNotFound ?
2. Will not compile
3. Will work
4. Will work perfectly

```
@Bean
```

```
@ConditionalOnClass({Soap.class, Roap.class})
```

```
@ConditionalOnMissingBean({ExecutionFactory.class})
```

```
public ExecutionFactory gallows() { return new GallowsFactory("..."); }
```

```
@Bean
```

```
@ConditionalOnClass({Chair.class, Electricity.class})
```

```
@ConditionalOnMissingBean({ExecutionFactory.class})
```

```
public ExecutionFactory chairs() { return new ChairFactory("zzz zzz"); }
```

```
@Bean
```

```
@ConditionalOnClass({Guillotine.class, GoodMood.class})
```

```
@ConditionalOnMissingBean({ExecutionFactory.class})
```

```
public ExecutionFactory guillotine() { return new GuillotineFactory(); }
```

```
}
```

```
}
```

# ConditionalOnPuzzler

```
@Configuration
```

```
public class ExecutionConfig {
```

- 1. ClassDefNotFound ?
- 2. Will not compile
-  3. Will work
- 4. Will work perfectly

```
@Bean
```

```
@ConditionalOnClass({Soap.class, Roap.class})
```

```
@ConditionalOnMissingBean({ExecutionFactory.class})
```

```
public ExecutionFactory gallows() { return new GallowsFactory("..."); }
```

```
@Bean
```

```
@ConditionalOnClass({Chair.class, Electricity.class})
```

```
@ConditionalOnMissingBean({ExecutionFactory.class})
```

```
public ExecutionFactory chairs() { return new ChairFactory("zzz zzz"); }
```

```
@Bean
```

```
@ConditionalOnClass({Guillotine.class, GoodMood.class})
```

```
@ConditionalOnMissingBean({ExecutionFactory.class})
```

```
public ExecutionFactory guillotine() { return new GuillotineFactory(); }
```

```
}
```

```
}
```

Because of ASM

# Iron bank law №1.5

enable/disable raven

@ConditionalOnProperty



# OnPropertyCondition

```
@Conditional(OnPropertyCondition.class)
public @interface ConditionalOnProperty {

    String[] value() default {};
    String prefix() default "";
    String[] name() default {};
    String havingValue() default "";

    boolean matchIfMissing() default false;
    boolean relaxedNames() default true;

}
```

# OnPropertyCondition

```
@ConditionalOnProduction
@ConditionalOnProperty(
    name = "raven.enabled",
    havingValue = "true"
)
public IronBankApplicationListener applicationListener() {
    ...
}
```

# OnPropertyCondition

```
@ConditionalOnProduction
@ConditionalOnProperty(
    name = {
        "raven.enabled",
        "winter.enabled",
        "stark.enabled"
    },
    havingValue = "true"
)
public IronBankApplicationListener applicationListener() {
    ...
}
```

# OnPropertyCondition

```
@ConditionalOnProduction
@ConditionalOnProperty(
    name = {
        "raven.enabled",
        "winter.enabled",
        "stark.enabled"
    },
    havingValue = "true" ← Not a list
)
public IronBankApplicationListener applicationListener() {
    ...
}
```

# OnPropertyCondition

```
@ConditionalOnProduction
@ConditionalOnProperty(name = "raven.enabled", havingValue="true")
@ConditionalOnProperty(name = "winter.enabled", havingValue="true")
@ConditionalOnProperty(name = "stark.enabled", havingValue="false")
public IronBankApplicationListener applicationListener() {
    ...
}
```

# OnPropertyCondition

```
@ConditionalOnProduction  
@ConditionalOnProperty(name = "raven.enabled", havingValue="true")  
@ConditionalOnProperty(name = "winter.enabled", havingValue="true")  
@ConditionalOnProperty(name = "stark.enabled", havingValue="false")  
public IronBankApplicationListener applicationListener() {  
    ...  
}  
  
@ConditionalOnProperty ← Not @Repeatable
```

A photograph from a movie scene showing six men in medieval or fantasy-style armor standing in a misty, wooded area. Some are on horseback. The men are wearing tunics, breeches, and various types of armor, including chainmail and leather. They have beards and long hair. The background is a dense forest with fallen leaves on the ground.

AllNestedConditions

& &

AnyNestedCondition

# OnPropertyCondition

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ ElementType.TYPE, ElementType.METHOD })
@Conditional(CompositeCondition.class)
public @interface ConditionalOnRaven {
}
```

# OnPropertyCondition

```
public class OnRavenCondition extends AllNestedConditions {  
  
    @ConditionalOnProperty(  
        name = "raven.destination",  
        havingValue = "false")  
    public static class OnRavenProperty { }  
  
    @ConditionalOnProperty(  
        name = "raven.enabled",  
        havingValue = "true",  
        matchIfMissing = true)  
    public static class OnRavenEnabled { }  
  
    ...  
}
```



# OnPropertyCondition

```
@ConditionalOnRaven  
public IronBankApplicationListener applicationListener() {  
    ...  
}
```

A Night King stands in a desolate, snow-covered landscape, his pale, featureless face and blue glowing eyes looking directly at the viewer. He wears his signature white, scale-like armor and has his arms outstretched wide. The background is a vast, cold wasteland under a dark, overcast sky.

WINTER IS COMING  
Here

A dark, atmospheric scene from Game of Thrones. A dragon, likely Drogon, is shown flying through a sky filled with swirling, dark clouds. The dragon's body is mostly obscured by the clouds, but its long tail and a portion of its wing are visible, glowing with a faint blue light. Below the dragon, a vast, snow-covered landscape stretches across the horizon, with distant mountains visible under a heavy, overcast sky.

Season 8 – Winter is here

A close-up photograph of a man with a shocked or surprised expression. He has short brown hair, a beard, and some dark marks on his forehead and around his eyes. He is wearing a dark leather collar. The background is dark and appears to be a wooden interior.

The iron bank credit rules have been changed

– Demo

# Iron bank law №2

Should fast fail without profile



# Demo



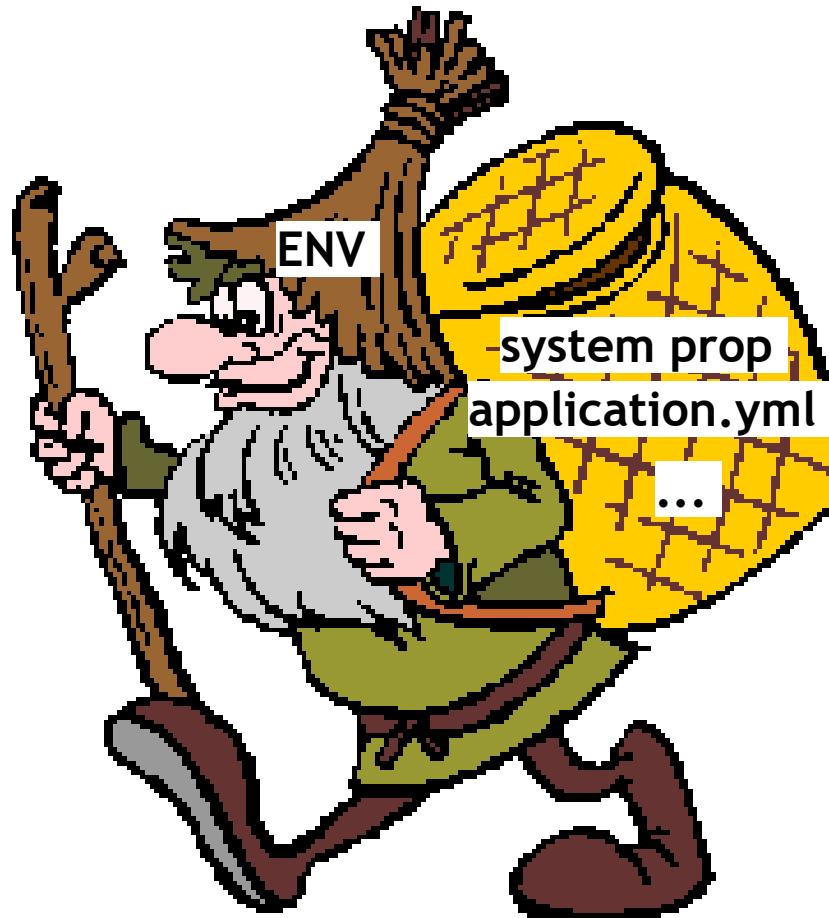
# Environment

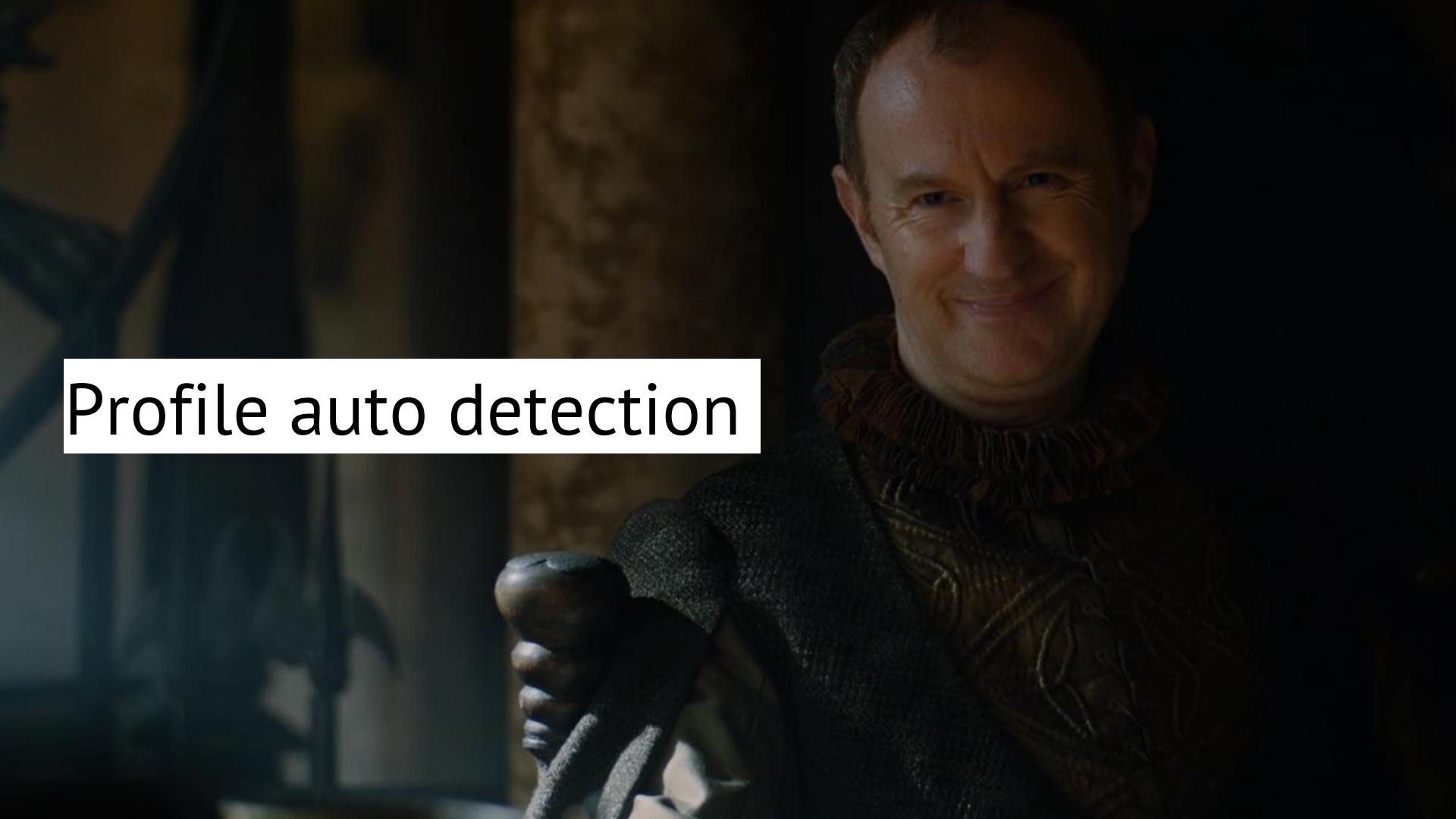
Property sources:

- systemProperties
- system environment
- random property
- application.properties
- ...

Active Profiles

Default Profiles



A close-up photograph of a man with short, light-colored hair, smiling warmly at the camera. He is wearing a dark, ribbed, high-collared sweater or jacket. His right hand is resting on a dark, textured steering wheel, suggesting he is in a vehicle. The background is dark and out of focus.

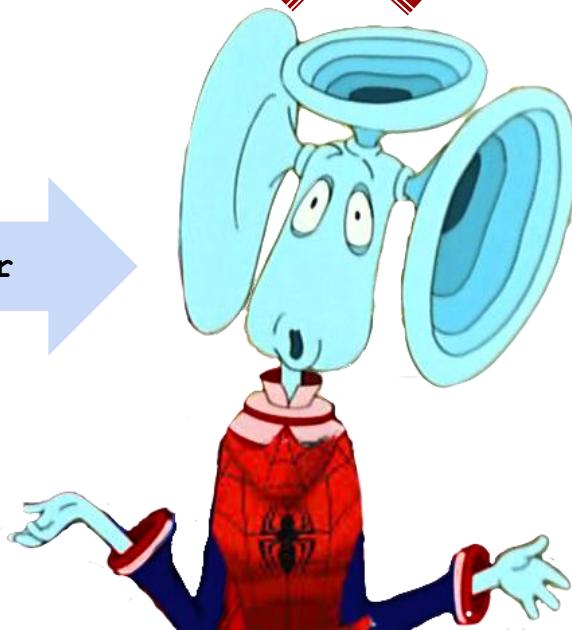
Profile auto detection

# EnvironmentPostProcessor



demo





ConfigFileApplicationListener

# First step: Spring application is building environment



Second step: listener receives environment and  
SpringApplication



# ConfigFileApplicationListener

As listener, listens for

- **ApplicationPreparedEvent**
- **ApplicationEnvironmentPreparedEvent**

As EnvironmentPostProcessor loads:

- application.yml
- application.properties
- env vars
- cmd args



# onApplicationEnvironmentPreparedEvent

A composite image featuring two characters from Disney Pixar's Cars. On the left is Tow Mater, a rusty tow truck with a smiling face and the text "SpringFactoriesLoader" written diagonally across his front grille. On the right is Mr. Incredible, a superhero with a large blue head and a red superhero suit. He is holding a large, blue, cylindrical device with a speaker-like end, which is emitting a beam of light towards a speech bubble. A speech bubble is positioned between them, containing the text "Bring me EnvironmentPostProcessors".

Bring me  
EnvironmentPostProcessors

# onApplicationEnvironmentPreparedEvent

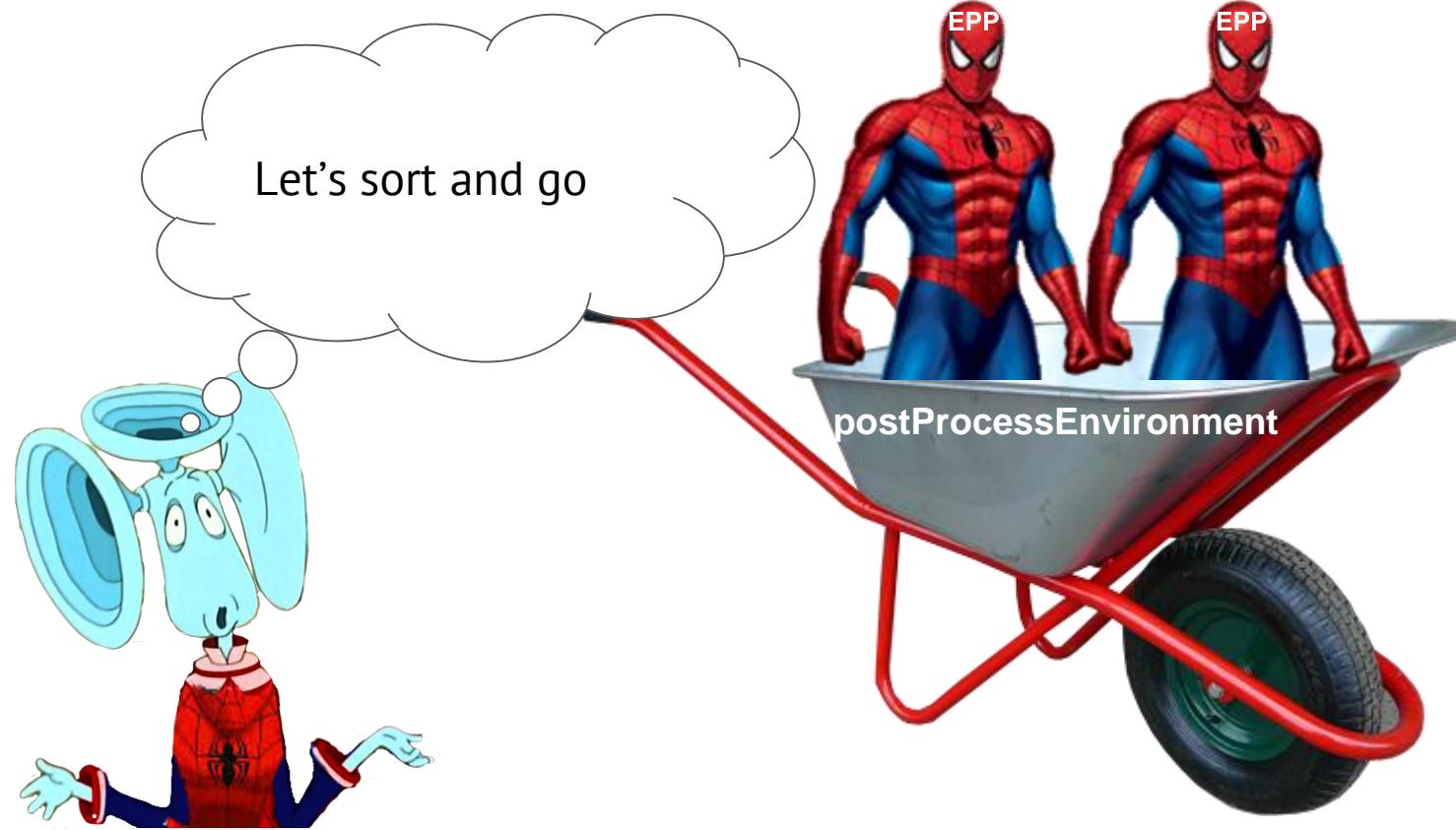
```
SpringFactoriesLoader.  
loadFactories(  
    EnvironmentPostProcessor.class  
    ...)
```



# onApplicationEnvironmentPreparedEvent



# onApplicationEnvironmentPreparedEvent



# onApplicationEnvironmentPreparedEvent



# onApplicationEnvironmentPreparedEvent



# onApplicationEnvironmentPreparedEvent



# ConfigFileApplicationListener

Listens

- ApplicationPreparedEvent
- ApplicationEnvironmentPreparedEvent

Loads

- application.yml
- application.properties
- env vars
- cmd args



# ConfigFileApplicationListener

Слушает

- ApplicationPreparedEvent
- **ApplicationEnvironmentPreparedEvent**

Загружает

- application.yml
- application.properties
- env vars
- cmd args



# I heard about

## Different events

**ContextStartedEvent**

**ContextStoppedEvent**

**ContextClosedEvent**

**ContextRefreshedEvent**





It's Spring Boot, baby  
Much more events

# Application Events

ApplicationStartingEvent

ApplicationEnvironmentPreparedEvent

ApplicationPreparedEvent

ContextRefreshedEvent

EmbeddedServletContainerInitializedEvent

ApplicationReadyEvent

ApplicationFailedEvent

...

# Application Events

ApplicationStartingEvent



ApplicationEnvironmentPreparedEvent



ApplicationPreparedEvent



ContextRefreshedEvent



EmbeddedServletContainerInitializedEvent



ApplicationReadyEvent



ApplicationFailedEvent



ContextClosedEvent



# Application Events

ApplicationStartingEvent

ApplicationEnvironmentPreparedEvent

ApplicationPreparedEvent

ContextRefreshedEvent

EmbeddedServletContainerInitializedEvent

ApplicationReadyEvent

ApplicationFailedEvent

ContextClosedEvent



# Application Events

ApplicationStartingEvent

ApplicationEnvironmentPreparedEvent

ApplicationPreparedEvent

ContextRefreshedEvent

EmbeddedServletContainerInitializedEvent

ApplicationReadyEvent

ApplicationFailedEvent

ContextClosedEvent



# Application Events

ApplicationStartingEvent

ApplicationEnvironmentPreparedEvent

ApplicationPreparedEvent

ContextRefreshedEvent

EmbeddedServletContainerInitializedEvent

ApplicationReadyEvent

ApplicationFailedEvent

ContextClosedEvent



# What about:

`ContextStartedEvent`

`ContextStoppedEvent`



# Where they should be placed?

ApplicationStartingEvent



ApplicationEnvironmentPreparedEvent



ApplicationPreparedEvent



ContextRefreshedEvent



EmbeddedServletContainerInitializedEvent



ApplicationReadyEvent



ApplicationFailedEvent

ContextStartedEvent

ContextStoppedEvent

A close-up photograph of a man with dark hair and a mustache. He is wearing a dark suit jacket over a white shirt and a dark tie. His head is tilted downwards, and he has a serious, perhaps melancholic, expression on his face. The background is blurred, suggesting an indoor setting.

This events

Never occurs

# Did you tried to invoke them?



# Did you tried to invoke them?

`ctx.start(); → ContextStartedEvent`

`ctx.stop(); → ContextStoppedEvent`



# In what line exception will occur?

ctx.stop(); (1)

ctx.start(); (2)

ctx.close(); (3)

ctx.start(); (4)

# In what line exception will occur?

ctx.stop(); (1)

ctx.start(); (2)

ctx.close(); (3)

ctx.start()  (4)



Stop before Start

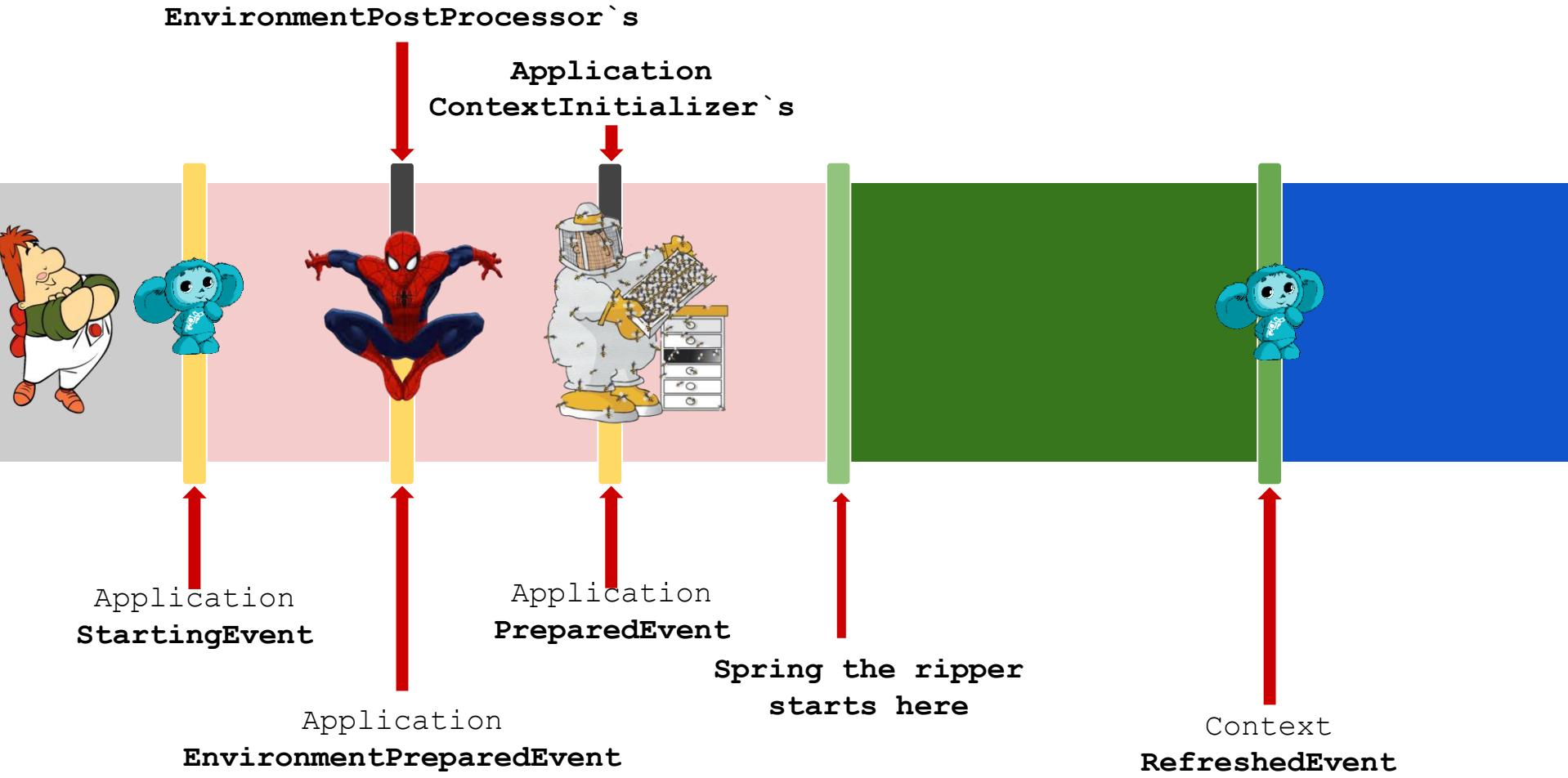
# Где вылетит исключение

ctx.stop(); (1)

ctx.start(); (2)

**ctx.close(); (3)**

ctx.start(); (4)



# Launching an application

1. **tomcat war**
2. idea
3. java -jar/war

# Who is packing the jar?

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

It only seems like regular JAR



# SpringBoot Jar anatomy

jar

- META-INF
- BOOT-INF
  - libs
  - classes
- org
  - springframework
  - boot



# SpringBoot Jar anatomy

jar

- | - META-INF
- | - BOOT-INF
  - | - libs
  - | - classes
- | - org
  - | - springframework
    - | - boot



**MANIFEST.MF**

...

Spring-Boot-Version: 1.5.3.RELEASE  
Implementation-Vendor: Pivotal Software, Inc.  
Main-Class:  
org.springframework.boot.loader.JarLauncher  
Start-Class:ru....ripper.OurMainClass  
Spring-Boot-Classes: BOOT-INF/classes/  
Spring-Boot-Lib: BOOT-INF/lib/  
...

# SpringBoot Jar

jar

- META-INF
- BOOT-INF
- |- libs
- |- classes
- org
- |- springframework
- |- boot



**MANIFEST.MF**

...  
Spring-Boot-Version: 1.5.3.RELEASE  
Implementation-Vendor: Pivotal Software, Inc.

**Main-Class:**

org.springframework.boot.loader.JarLauncher

**Start-Class:** ru....ripper.OurMainClass

Spring-Boot-Classes: BOOT-INF/classes/

Spring-Boot-Lib: BOOT-INF/lib/

...

# SpringBoot Jar

jar

- META-INF
- BOOT-INF
- |- libs
- |- classes
- org
- |- springframework
- |- boot



**MANIFEST.MF**

...  
Spring-Boot-Version: 1.5.3.RELEASE  
Implementation-Vendor: Pivotal Software, Inc.

**Main-Class:**

org.springframework.boot.loader.JarLauncher  
Start-Class:ru....ripper.OurMainClass  
Spring-Boot-Classes: BOOT-INF/classes/  
Spring-Boot-Lib: BOOT-INF/lib/  
...

**JarLauncher**

Who is writing the manifest for this Jar?



```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <mainClass>conference...OurMainClass</mainClass>
    </plugin>
  </plugins>
</build>
```

### MANIFEST.MF

...

Start-Class: conference.spring.boot.ripper.OurMainClass

...

# mainClass resolution strategy

Spring boot plugin scans classpath and looking for main methods

- If there is only one main method, we found it! :)
- In case of more than one main, the one annotated by `@SpringBootApplication` will be chosen
- If there is no `@SpringBootApplication` or >1 – exception



java -jar  
- too complicated for us

Click and go!

# executable jar: Maven

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <executable>true</executable>
      </configuration>
    </plugin>
  </plugins>
</build>
```

# Gradle

```
springBoot {
  executable = true
}
```

# Demo. Executable jar structure

# Jar content

Script

**Oxf4ra -> а это ZIP!**  
jar archive

# Jar content

Script

**0xf4ra -> а это ZIP!**  
jar archive



File beginning

# SpringBoot Jar?

Script

0xf4ra -> а это ZIP!  
jar archive



File beginning

Zip archive beginning

# Conclusions

1. **Spring Boot** is much more than **Spring Context**
2. Starters have a lot of naming conventions
  - 1 It is not always good
3. Spring Boot is not a magic, but it is entire world

Я вижу, это  
Ваш первый  
визит к  
стоматологу ?



@tolkv

@lavcraft



@jekaboris  
ov

@jeka1978



al Falab



NAYA  
TECHNOLOGIES

Вопросы?