Spring(Boot) Configuration

Łukasz Sikora
sikora.lukasz.sl@gmail.com

About...

Spring Configuration history

- XML configuration was available since always. At that time it was considered lighter than Java EE which was industry standard
- Spring took one part of EJB specification, namely container based dependency injection and it became Spring Core

Spring Configuration history

- `<context:annotation-config/>`
  `<context:component-scan base->`
  introduced in spring 2.5

- `@ComponentScan`
  introduced in Spring 3.0

- `@Configuration`
  introduced in Spring 3.0

Component scan

- When Spring 3.0 came out people where already allergic to XML
- Most projects moved to use XML mostly for

```
<context:annotation-config />
<context:component-scan base-package="com.xxx" />
```

Spring Configuration history

- `@Configuration` **classes are with us as long as** `@ComponentScan`
- **Extensive use of Java configuration classes came with time and partially Spring Boot**

Component scan hell

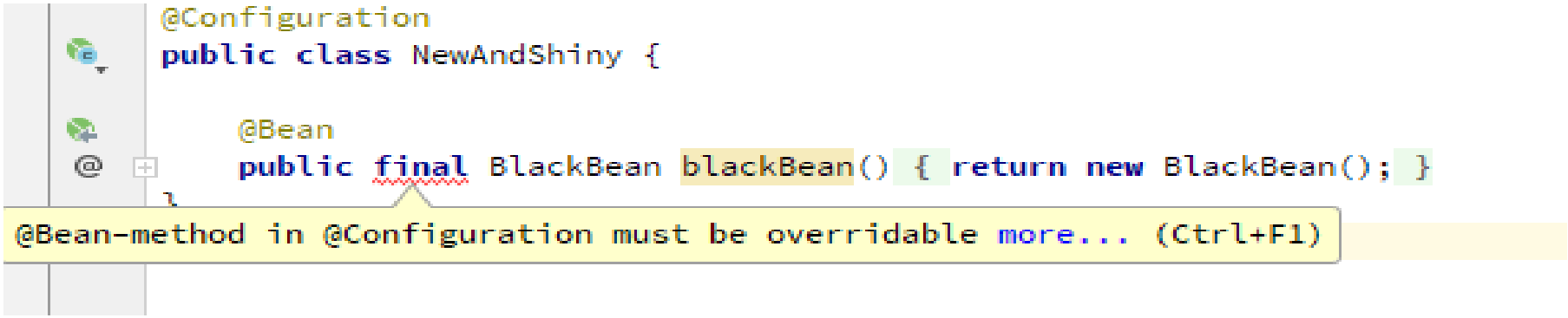
- Package scan has been overused to the extent when core application package has been scanned
- This led to problems with beans implementing same interfaces (Where the heck is second bean instance coming from?)
- Programmers were left with huge contexts of beans floating around just like global variables

Fun fact

- Spring was created as lightweight alternative to Java EE, both based on XML configuration
- With Spring 3.0 released at 2009, Spring bandwagon started to laugh about Java EE XML configuration heaviness when Spring had configuration through annotations
- But Java EE had annotations like `@Remote` for EJB creation since 1.5 which was released 2006

Final method that creates BlackBean

- `@Configuration` will show error if method has incorrect modifier



- and fails miserably

```
org.springframework.beans.factory.parsing.BeanDefinitio
nParsingException: Configuration problem: @Bean
method 'blackBean' must not be private or final;
change the method's modifiers to continue
Offending resource: class path resource
[somebody/somewhere/xmlvsjavaconfig/configuration/NewAn
dShiny.class]
```

XML version

- Just works

```
7
8       </bean>
9       <bean id="someBlackBean" factory-bean="newAndShiny"
10            factory-method="blackBean"/>
11  </beans>
```

```
someBlackBean
xmlvsjavaconfig.commonbeans.BlackBean
@338c99c8
```

Chickpea bean has private constructor

```
public class Chickpea {
    private SimpleBeanFactoryAwareAspectInstanceFactory
springBean;

    public Chickpea(SimpleBeanFactoryAwareAspectInstanceFactory springBean)
{
        this.springBean = springBean;
    }

    private Chickpea() {
    }

    public SimpleBeanFactoryAwareAspectInstanceFactory getSpringBean() {
        return springBean;
    }
}
```

Our application

```
@Bean
public CommandLineRunner commandLineRunner(ApplicationContext ctx){
    return args->{

    System.out.println("Let's inspect the beans provided by Spring Boot:");
    System.out.println("chickpea"+" "+ctx.getBean("chickpea"));
    System.out.println("simpleBeanFactoryAwareAspectInstanceFactory"+" "+

    ((Chickpea)ctx.getBean("chickpea")).getSpringBean());
    };
  }
}
```

Java Config does not compile

- **Compilation simply fails because non parameter constructor is private**

```
@Configuration
public class NewAndShiny {

    @Bean
    public Chickpea chickpea() {
        return new Chickpea();
    }
}
```

XML creates bean without field injected

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

    <bean id="newAndShiny"
class="xmlvsjavaconfig.configuration.NewAndShiny"/>


    <bean id="chickpea"
class="xmlvsjavaconfig.commonbeans.Chickpea"/>
</beans>
```

```
2018-06-22 22:08:04.309  INFO 10400 --- [           main]
s.s.x.app.XmlVsJavaconfigApplication     : Started
XmlVsJavaconfigApplication in 0.953 seconds (JVM running for
1.258)
Let's inspect the beans provided by Spring Boot:
chickpea
xmlvsjavaconfig.commonbeans.Chickpea@30c93896
simpleBeanFactoryAwareAspectInstanceFactory     null
```

Bean without parameterless constructor

```
public class BlackEyedBean {
    private final SimpleBeanFactoryAwareAspectInstanceFactory springBean;

 public BlackEyedBean
(SimpleBeanFactoryAwareAspectInstanceFactory springBean){
        this.springBean = springBean;
    }

    public SimpleBeanFactoryAwareAspectInstanceFactory getSpringBean() {
        return springBean;
    }
}
```

Bean without parameterless constructor

- **Compilation failure**

```
@Bean
public BlackEyedBean chickpea() {
    return new BlackEyedBean();
}
```

Bean without parameterless constructor

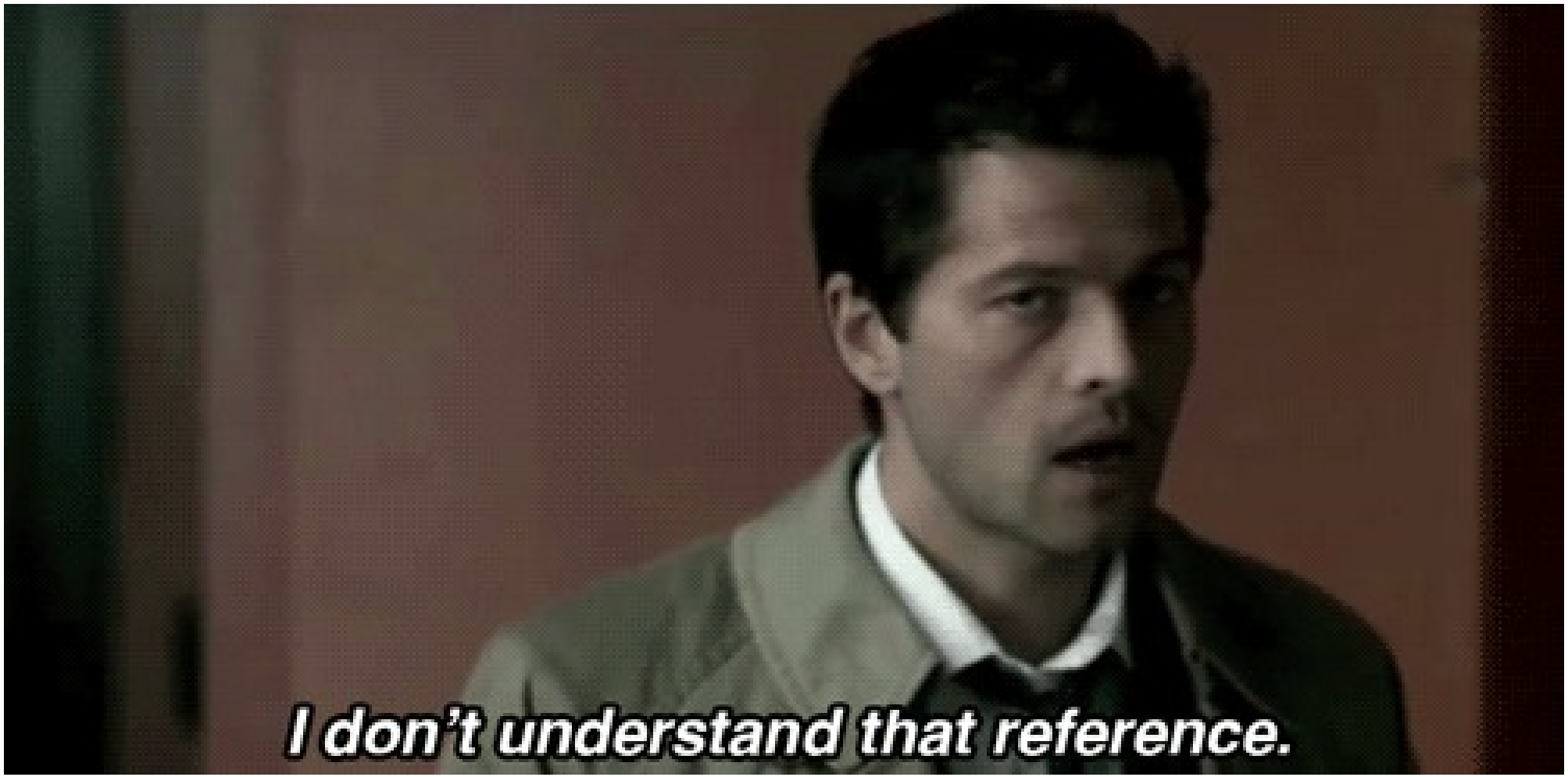
- **IDE error**

```
    <bean id="blackEyedBean"
          class="somebody.somewhere.xmlvsjavaconfig.commonbeans.BlackEyedBean"/>
</beans>
```

No matching constructor found in class 'BlackEyedBean'

BlackEyedBean(...):	Bean:
SimpleBeanFactoryAwareAspectInstanceFactory simpleBeanFactoryAwareAspectInstanceFactory	???

more... (Ctrl+F1)


I don't understand that reference.

Package scan hell (* as feature)

- We introduce new bean which will serve as container for our common beans and use `@Autowire`

```
@Configuration
public class NewAndShiny {
    @Bean
    public BagOfBeans bagOfBeans(@Autowired List<CommonBean> commonBeans) {
        return new BagOfBeans(commonBeans);
    }

}
```

Package scan hell (* as feature)

- We add component scanning inside configuration
- Feature: we scan only external package but all beans will be picked up

```
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
xmlns="http://www.springframework.org/schema/beans"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://
www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <bean id="newAndShiny" class="xmlvsjavaconfig.configuration.NewAndShiny"/>

    <context:annotation-config/>
    <context:component-scan base-package=

"xmlvsjavaconfig.commonbeans"/>
</beans>
```

Package scan hell (* as feature)

- New configuration will return a list with BlackEyedBean and instance of LimeBean

```
@Configuration
public class BeansThatAreNotPackageScanned {

    @Bean
    public List<CommonBean> commonBeans() {
        return asList(new BlackEyedBean());
    }

    @Bean
    public LimeBean limeBean() {
        return new LimeBean();
    }
}
```

Package scan hell (* as feature)

We create subpackage with bean in it

```
package xmlvsjavaconfig.commonbeans.notsocommon;

import org.springframework.stereotype.Component;
import xmlvsjavaconfig.commonbeans.CommonBean;

@Component
public class GreatNorthernBean implements CommonBean {
}
```

Package scan hell (* as feature)

At that point we have 5 beans in total
in our application

```
package xmlvsjavaconfig.commonbeans;

@Component
public class Chickpea implements CommonBean {
}
```

Package scanned

```
public class LimeBean implements CommonBean {
}
```

Created with @Bean

```
public class BlackEyedBean implements CommonBean {
}
```

List containing this instance
created with @Bean

Package scan hell (* as feature)

```
@Component
public class BlackBean implements CommonBean {
    public BlackBean() {
    }
}
```

Package scanned

```
package xmlvsjavaconfig.commonbeans.notsocommon;

import org.springframework.stereotype.Component;
import xmlvsjavaconfig.commonbeans.CommonBean;

@Component
public class GreatNorthernBean implements
CommonBean {
}
```

Package scanned ?

Q: How many CommonBeans do we get?

```
@Bean
public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
        return args -> {
            System.out.println("Let's inspect the beans provided by
            Spring Boot:");
System.out.println("bagOfBeans");
System.out.println(ctx.getBean("bagOfBeans"));
        };
    }
}
```

A: Lots but not all

- **We do not get bean instance created in `@Configuration` and wrapped in list**

```
Let's inspect the beans provided by Spring Boot:
bagOfBeans
BagOfBeans{commonBeans=
xmlvsjavaconfig.commonbeans.LimeBean@275bf9b3,
From java config
xmlvsjavaconfig.commonbeans.BlackBean@1b8a29df,
xmlvsjavaconfig.commonbeans.Chickpea@4fbe37eb,
xmlvsjavaconfig.commonbeans.notsocommon.GreatNort
hernBean@12a94400
]}Package scanned
```

Answer in a moment

Package scan hell (* as feature)

Lets remove package scan

```
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
xmlns="http://www.springframework.org/schema/beans"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <bean id="newAndShiny" class="xmlvsjavaconfig.configuration.NewAndShiny"/>
</beans>
```

and LimeBean

```
@Configuration
public class BeansThatAreNotPackageScanned {

    @Bean
    public List<CommonBean> commonBeans() {
        return asList(new BlackEyedBean());
    }
}
```

Package scan hell (* as feature)

And we get our list of beans
from Java configuration

```
Let's inspect the beans provided by Spring
Boot:
bagOfBeans
BagOfBeans{commonBeans=[xmlvsjavaconfig.co
mmonbeans.BlackEyedBean@181e731e]}
```

Package scan hell (* as feature)

Lets add back package scan

```
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
xmlns="http://www.springframework.org/schema/beans"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/
beans/spring-beans.xsd http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <bean id="newAndShiny" class="xmlvsjavaconfig.configuration.NewAndShiny"/>

    <context:annotation-config/>
    <context:component-scan base-package="xmlvsjavaconfig.commonbeans"/>
</beans>
```

and LimeBean

```
@Configuration
public class BeansThatAreNotPackageScanned {

    @Bean
    public List<CommonBean> commonBeans() {
        return asList(new BlackEyedBean());
    }

    @Bean
    public LimeBean limeBean() {
        return new LimeBean();
    }
}
```

What about @Resource injection?

```
@Configuration
public class NewAndShiny {
     @Resource
      private List<CommonBean> commonBeans;

     @Bean
     public BagOfBeans bagOfBeans() {
          return new BagOfBeans(commonBeans);
     }

}
```

@Resource collection injection

```
Let's inspect the beans provided by Spring Boot:

bagOfBeans

BagOfBeans{commonBeans=[xmlvsjavaconfig.commonbeans.BlackEyedBean@21337f7b]}
```

CDI @Inject

```
@Configuration
public class NewAndShiny {
    @Inject
     private List<CommonBean> commonBeans;

    @Bean
    public BagOfBeans bagOfBeans() {
        return new BagOfBeans(commonBeans);
    }

}
```

build.gradle :

```
compile('org.jboss.weld.se:weld-se-shaded:3.0.5.Final')
```

More about CDI & Java EE later

We get same beans as with @Autowired

```
Let's inspect the beans provided by Spring Boot:
bagOfBeans
BagOfBeans{commonBeans=[
xmlvsjavaconfig.commonbeans.LimeBean@7ef27d7f,
xmlvsjavaconfig.commonbeans.BlackBean@490caf5f,
xmlvsjavaconfig.commonbeans.Chickpea@6337c201,
xmlvsjavaconfig.commonbeans.notsocommon
.GreatNorthernBean@5c669da8
]}
```

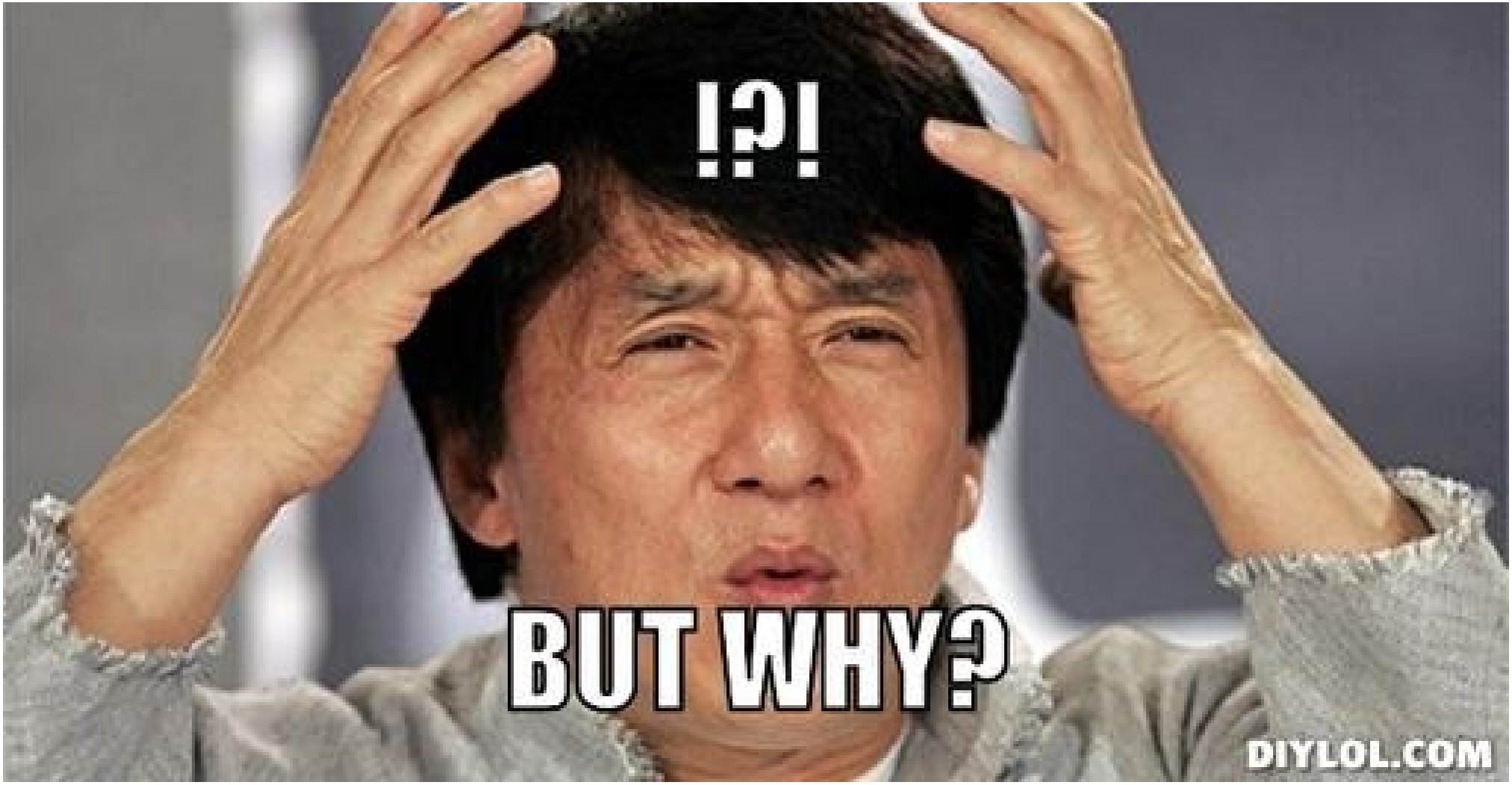
Black magic f*ckery (SpEL & @Value)

```
@Configuration
public class NewAndShiny {
      @Value("#{commonBeans}")
     //@Qualifier("commonBeans")
     //@Named("commonBeans")

     @Autowired
     //@Inject
     private List<CommonBean> commonBeans;

     @Bean
     public BagOfBeans bagOfBeans() {
          return new BagOfBeans(commonBeans);
     }

}
```

What now?

```
Let's inspect the beans provided by Spring
Boot:
bagOfBeans
BagOfBeans{commonBeans=[xmlvsjavaconfig.comm
onbeans.BlackEyedBean@27eedb64]}
```

Finally ...



@Autowired is not @Resource

- **https://jira.spring.io/browse/SPR-8519**
- **`@Autowire/@Inject` not supported up until 3.0 at least, available as single candidate in Spring Boot**
- **`@Resource` to be used for injection of collections explicitely (it autowires by name as seen later)**
- **Spring >4.3 is able to inject collections by `@Autowire` with restrictions as shown by examples**
- **Qualifying bean with `@Value` &SpEL or `@Qualifier` as option**
- **Also works with @Map**

Package scan hell rehersal

- Tread carefully with mixing XML and Java Configuration and wiring collections
- Multiple options for unambiguous collection injection by name:

  - Use `@Resource` if you allow field or setter injection (Preferred option in Spring)

  - Use `@Qualifier @Autowired` in other cases

- Spring will create collection out of all beans flying around in context but will not merge it with explicit collection declarations

Spring how to annotation scan

- When having a set of common interface implementations, strategies, commands, put them inside single package an scan it.
- Watch out for default filters
- Define this in single encapsulated `@Configuration` class, treat this package just as a bag full of closely related beans where it is very easy to put them in future or retrieve them all at once.
- Always use `@ComponentScan(includeFilters="")`

Q

Q

- **Which of programming principles do the last two points from previous slide represent?**

&A

Open-Closed Principle – we open for adding particular interface implementations but through white list inclusion we close for adding classes unrelated to the interface.

Spring configuration approach evolution

- Spring favoured configuration over convention
- This approach was created against Java EE convention over configuration approach in terms of functionalities available out of the box and available to application
- Some people where overwhelmed by number of features that were coming from Java EE servers when just simple servlet would suffice...

Spring configuration approach evolution

- Spring Boot uses auto configuration
- Spring Boot scans class path for resources and libraries connected to supported frameworks
- Spring Boot automatically creates all needed beans e.g. if we have data source configured and hibernate properties it will think that we want to use hibernate in our project
- This behaviour can be disabled

Pivotal stance

15. Configuration Classes

Spring Boot favors Java-based configuration. Although it is possible to use `SpringApplication` with XML sources, we generally recommend that your primary source be a single `@Configuration` class. Usually the class that defines the `main` method is a good candidate as the primary `@Configuration`.

Many Spring configuration examples have been published on the Internet that use XML configuration. If possible, always try to use the equivalent Java-based configuration. Searching for `Enable*` annotations can be a good starting point.

15.1 Importing Additional Configuration Classes

You need not put all your `@Configuration` into a single class. The `@Import` annotation can be used to import additional configuration classes. Alternatively, you can use `@ComponentScan` to automatically pick up all Spring components, including `@Configuration` classes.

15.2 Importing XML Configuration

If you absolutely must use XML based configuration, we recommend that you still start with a `@Configuration` class. You can then use an `@ImportResource` annotation to load XML configuration files.

Spring configuration approach evolution

- An example why old Spring configuration approach was making people want to it all the chicken in the room …

Property injection Spring vs Spring Boot

```
    @Bean
    public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
        return args -> {

            System.out.println("Let's inspect the beans provided by Spring Boot:");
            System.out.println("cannelliniBean");
            System.out.println(((CannelliniBean)
ctx.getBean("cannelliniBean")).getCannelliniBeanName());

        };
    }
}
```

Property injection Spring vs Spring Boot

```
package xmlvsjavaconfig.commonbeans;

import org.springframework.lang.NonNull;

public class CannelliniBean {
    private final String cannelliniBeanName;

    public CannelliniBean(@NonNull String cannelliniBeanName) {
        this.cannelliniBeanName = cannelliniBeanName;
    }

    public String getCannelliniBeanName() {
        return cannelliniBeanName;
    }
}
```

Property injection Spring vs Spring Boot

```
package xmlvsjavaconfig.configuration;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
import org.springframework.context.annotation.PropertySource;
import xmlvsjavaconfig.commonbeans.CannelliniBean;

@Configuration
@PropertySource("classpath:common.properties")
public class NewAndShiny {


    @Bean
    public CannelliniBean cannelliniBean(

@Value("${cannellini.bean.name}") String beanName) {
        return new CannelliniBean(beanName);
    }

}
```

Property injection Spring vs Spring Boot

- common.properties

```
cannellini.bean.name="italianKidneyBean"
```

Property injection Spring vs Spring Boot

- **Spring Boot:**

```
2018-06-23 11:55:11.599  INFO 2812 ---
[           main] s.s.x.app.XmlVsJavaconfigApplication
   : Started XmlVsJavaconfigApplication in 0.89 seconds
(JVM running for 1.196)
Let's inspect the beans provided by Spring Boot:
cannelliniBean
"italianKidneyBean"
```

- **In spring nothing gets injected**

Property injection Spring vs Spring Boot

- **What is missing in bare Spring?**
- **Java Config:**

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

- **XML:**

```
<bean
class="org.springframework.beans.factory.config.P
ropertyPlaceholderConfigurer"
      name="propertiesBean"/>
```

Property injection Spring vs Spring Boot

- When using Spring ~~Boot~~ and using only XML context configuration all that would be needed is

```
<context:property-placeholder
location="classpath:application.properties"/>
```

Property injection Spring vs Spring Boot

- In Spring boot as shown there is not need to add `PropertyPlaceholderConfigurer` by hand
- File named `application.properties` is by default read in order to retrieve properties
- If environment is defined in spring file `application-environment.properties` is read where environment is name of Spring profile

Property injection Spring

- **Properties defined in environment file will override properties in default properties file in case they clash**
- `@TestPropertySource` or `@SpringBootTest(properties="some.properties")`

  can be used to configure test specific properties


PLEASE STAND BY

Property injection Spring Boot

- Spring Boot is able to map properties from files to Java object graph with root in class annotated with

```
@ConfigurationProperties
(prefix = "characteristics")
public class Characteristics {
    String color;
    String size;
    String taste;
    int length;
```

Property injection Spring Boot

```
@Configuration
@EnableConfigurationProperties(Characteristics.class)
public class NewAndShiny {

    @Bean
    public BlackBean blackBean() {
        return new BlackBean();
    }

    //OR

    @Bean
    public Characteristics characteristics() {
        return new Characteristics();
    }
}
```

Property injection Spring Boot

- application.properties

```
characteristics.length:42
```

- application.yaml

```
characteristics:
   color: black
    size: small
  taste: chicken
```

Property injection Spring Boot

```
BlackBean{characteristics=Characterist
ics{color='black', size='small',
taste='chicken', length='42'}}
```

Property injection Spring Boot

- `@PropertySource` **does not work with .yaml**
- **For more:**

  http://www.baeldung.com/properties-with-spring
- `@Value` **vs** `@ConfigurationProperties`
- **https://tuhrig.de/why-using-springs-value-annotation-is-bad/**

Nobody expects the Spanish Inquisition

Spring proxying problems

```
    @Bean
    public CommandLineRunner
commandLineRunner(ApplicationContext ctx) {
        return args -> {
            System.out.println("Let's inspect the beans
provided by Spring Boot:");

Stream.of(ctx.getBeanNamesForType(CommonBean.class))
                    .forEach(
                            beanName ->
                    System.out.println(beanName
                                    + "      " +
                    ctx.getBean(beanName).toString()));
        };
    }
```

Spring proxying problems

```
@Configuration
public class NewAndShiny {

    @Bean
    public static PintoBean publicStaticPintoBean() {
        return new PintoBean();
    }

    public BlackBean noAnotationBlackBean() {
        return new BlackBean();
    }

    @Bean
    private KidneyBean privateKidneyBean() {
        return new KidneyBean();
    }

    @Bean
    LimaBean defaultLimaBean() {
        return new LimaBean();
    }
```

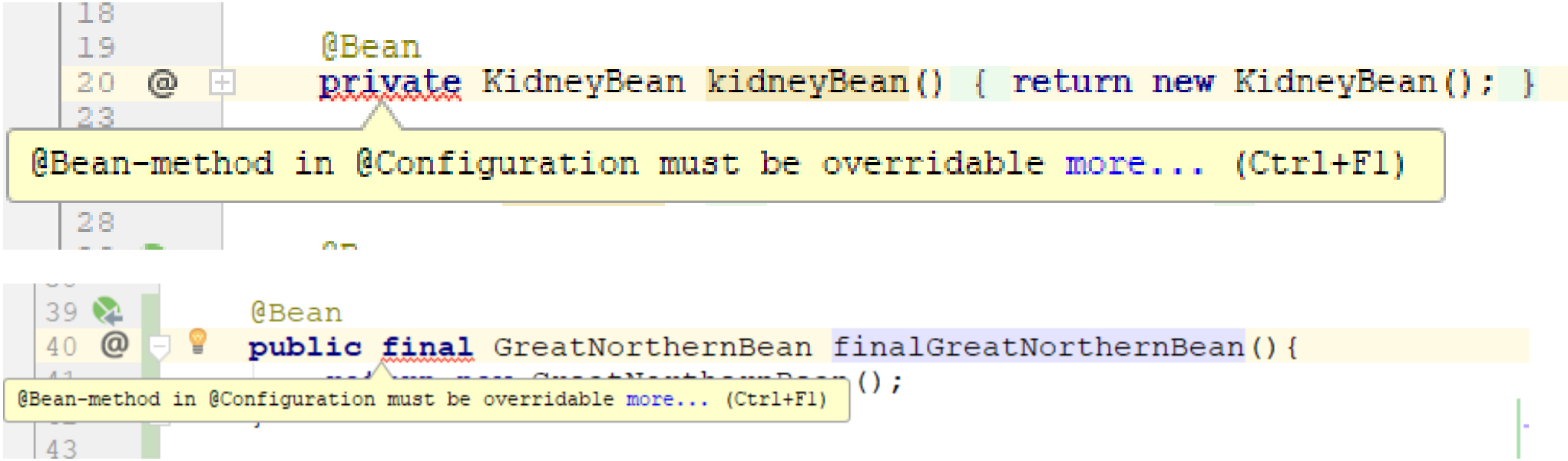
Spring proxying problems

```
    @Bean
    protected BlackedEyePea protectedBlackedEyePea() {
        return new BlackedEyePea();
    }

    @Bean
    public MrBean mrBean() {
        return new MrBean();
    }

    @Bean
    public final
    GreatNorthernBean finalGreatNorthernBean() {
        return new GreatNorthernBean();
    }
}
```

Spring proxying problems

```
2018-07-30 21:09:55.176 ERROR 7788 --- [ main]
o.s.boot.SpringApplication  : Application run failed
org.springframework.beans.factory.parsing.BeanDefinitionParsingEx
ception: Configuration problem: @Bean method
'kidneyBean' must not be private or final;
change the method's modifiers to continue
Offending resource: xmlvsjavaconfig.configuration.NewAndShiny
```

IDE to the rescue



Spring proxying problems

```
@Configuration
public class NewAndShiny {

    @Bean
    public static PintoBean publicStaticPintoBean() {
        return new PintoBean();
    }

    @Bean
    LimaBean defaultLimaBean() {
        return new LimaBean();
    }

    @Bean
    protected BlackedEyePea protectedBlackedEyePea() {
        return new BlackedEyePea();
    }

    @Bean
    public MrBean mrBean() {
        return new MrBean();
    }
}
```

Spring proxying problems

```
Let's inspect the beans provided by Spring Boot:
publicStaticPintoBean
xmlvsjavaconfig.commonbeans.PintoBean@60b4beb4

defaultLimaBean      xmlvsjavaconfig.commonbeans.LimaBean@7fcf2fc1

protectedBlackedEyePea
xmlvsjavaconfig.commonbeans.BlackedEyePea@2141a12

mrBean      xmlvsjavaconfig.commonbeans.MrBean@4196c360
```

Spring proxy approaches

- When you create a bean in Spring proxy object may be created (transaction proxies, db exception translation proxies, aop proxies).
- When the bean is instance of a class that implements interface proxy is created as `JdkDynamicProxy`.
- When the bean does not implement any interface and CGLib is available on classpath **CGLib proxy** is created.
- From Spring 3.2 CGLib comes bundled with Spring
- This is why casting bean instance will fail with `ClassCastException`
- Actual implementation class/object can be retrieved

Proxying sideeffects

- Private & final methods cannot be proxied
- Final classes cannot be proxied
- Proxies are not seriallizable
- There's little performance difference between CGLIB proxying and dynamic proxies.
- As of Spring 1.0, dynamic proxies are slightly faster. However, this may change in the future.
- Performance should not be a decisive consideration in this case.

When does Spring creates proxy

- **Bean is annotated `@Repository`.**
- **`@Transational`, `@Transaction` is defined at any level.**
- **Bean is `@RestController` or mapping annotations are used.**
- **Bean is `@Entity`.**
- **Inside Java configuration if bean is referenced through method but only if configuration is picked up through Spring and not created by new!**

Spring proxying problems

```
@Configuration
public class NewAndShiny {

    @Bean
    public static PintoBean pintoBean() {
        return new PintoBean();
    }

    @Bean
    LimaBean limaBean() {
        return new LimaBean(blackedEyePea());
    }

    @Bean
    protected BlackedEyePea blackedEyePea() {
        return new BlackedEyePea();
    }

    @Bean
    public MrBean mrBean() {
        return new MrBean();
    }
}
```

Spring config debugging

- If bean definition is not found:
- Check if you scan all required packages
- Create explicit configuration based on java config to see at which point compilation fails
- `@Bean` method is missing, `@Component` is missing, package containing bean implementation is not scanned

Spring config debugging

- If multiple definitions of bean are found:

  - Check how many contexts create instances of given bean class either through package scan (check each package level) or explicitly

  - Check which of those contexts are used by your context manually if there is more than one (IDE may help).
- Double check that you should really have two contexts imported which create same bean instance
- If injected bean is defined in multiple contexts that are part of same context dependency tree, refactor contexts configuration to remove duplication e.g define bean in first common node between two branches of dependency tree (move bean definition up on dependency tree)
- Extract bean to separate configuration in order to get rid of entire branches of dependencies if multiple bean definitions were result of overgrown dependency tree where entire context is imported in order to inject one bean(corner case is bean which has no dependencies or fixed dependencies and is injected through dependency tree just because “we already are creating this bean instance”)

Spring config debugging

- **Resolving bean ambiguity**
- **Add `@Qualifier` to `@Autowired` in order to choose correct bean implementation**
- **There is also `@Resource` which is part of JSR-250 which is meant to be injected by name but works only on field and setter injection and is not recommended by Spring since it does now work with constructor injection**


EVIL
memegenerator.net

Spring config debugging evil

- All non-Java EE JSR 250 annotations were added to the Java SE with version 6 (`@Generated`, `@PostConstruct`, `@PreDestroy`, `@Resource`, `@Resources`). They are located in the package `javax.annotation`

  - This is a little warning since it shows that `javax.` Packages are not only related to java ee but also cross cutting concepts

  - Actually `@Qualifier` is part of `javax.` package

  - Make sure you have Java classes for Java ee packages on classpath (Spring boot adds them, Spring framework on tomcat doesn't) and

  -"JLS 13.5.7: " ... removing annotations has no effect on the correct linkage of the binary representations of programs in the Java programming language."

  - This means that Java ignores annotation if they are not present in runtime

  - https://4programmers.net/Profile/78878/Microblog?page=4 - search for `javax.transaction.Transactional`, zamiast `org.springframework.transaction.annotation.Transactional` "Ale dziś magię rozwaliłem"

Spring config debugging PURE EVIL

- How not to have issues with Java EE packages missing in runtime – deploy on Java EE server]:->
- Josh Long https://spring.io/blog/2014/03/07/deploying-spring-boot-applications
- There is a section What about the Java EE Application Server? Which shows how to make Spring Boot `.war` running on Java EE servers
- By Josh Long the "make `.jar` not `.war`" guy
- May require more work with Spring Boot 2.x, War packaging is still available.

Spring config debugging

- `@Resource javax.annotation` **Java**
- `@Inject javax.inject` **Java**
- `@Qualifier javax.inject` **Java**
  Watch out!
- `@Qualifier org.springframework.bean.factory.annotation` **Spring**
- `@Autowired org.springframework.bean.factory` **Spring**

Spring config debugging

“ `@Autowired` and `@Inject`

Matches by Type

Restricts by Qualifiers

Matches by Name

`@Resource`

Matches by Name

Matches by Type

Restricts by Qualifiers (ignored if match is found by name)
”

https://stackoverflow.com/questions/4093504/resource-vs-autowired

Spring config debugging

- **Spring =**

  1) `@Bean(name = "")` **or** `@Component + @Qualifier`

  2) `@Autowired + @Qualifier`

- **CDI =**

  1)Create spring bean with explicitly given name

  2) Inject with `@Resource`

Spring config debugging

- **Use CDI's `@Alternative` in order to manage bean ambiguity**
- **Since CDI 2.0 it has been moved from Java EE to Java SE but requires at least Weld SE implementation on `Classpath`**
- **`@Alternative` shows that there are multiple bean implementations and requires programmer to choose explicitly which one will be used**

Spring config debugging

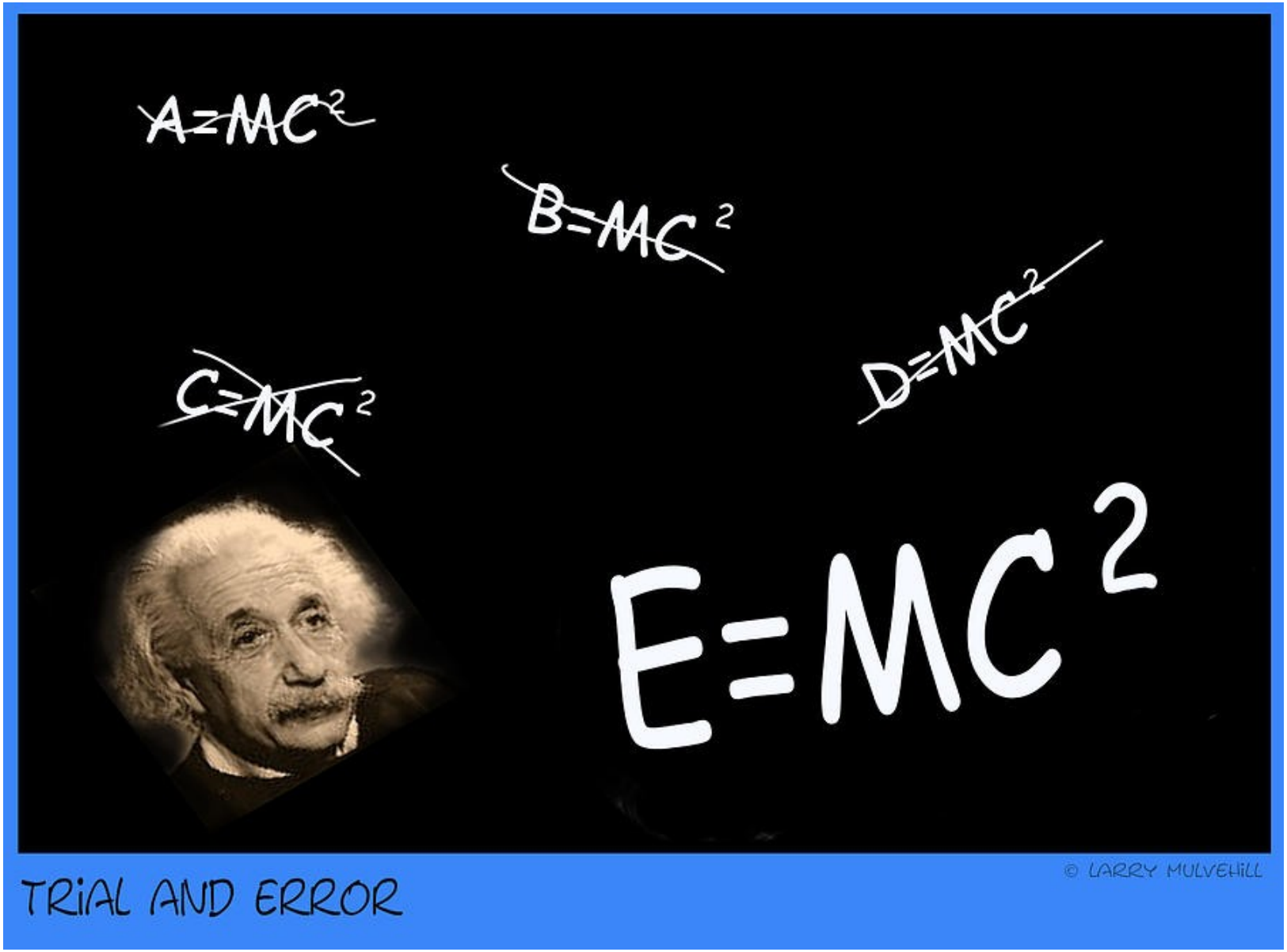
- `@Primary` forces Spring to use annotated bean when multiple bean definitions are found
- Every time you use `@Primary` small kitten dies, especially if class created by package scan is set to `@Primary`. This means that it will be primary for all contexts which instantiate this beans which may not be wanted behaviour
- Unless you do it for debugging

Spring config debugging

Information about Spring

- Spring documentation (Even minor version matters!)
- Baeldung
- Safari Books
- Coursera
- GitHub

Information about Spring



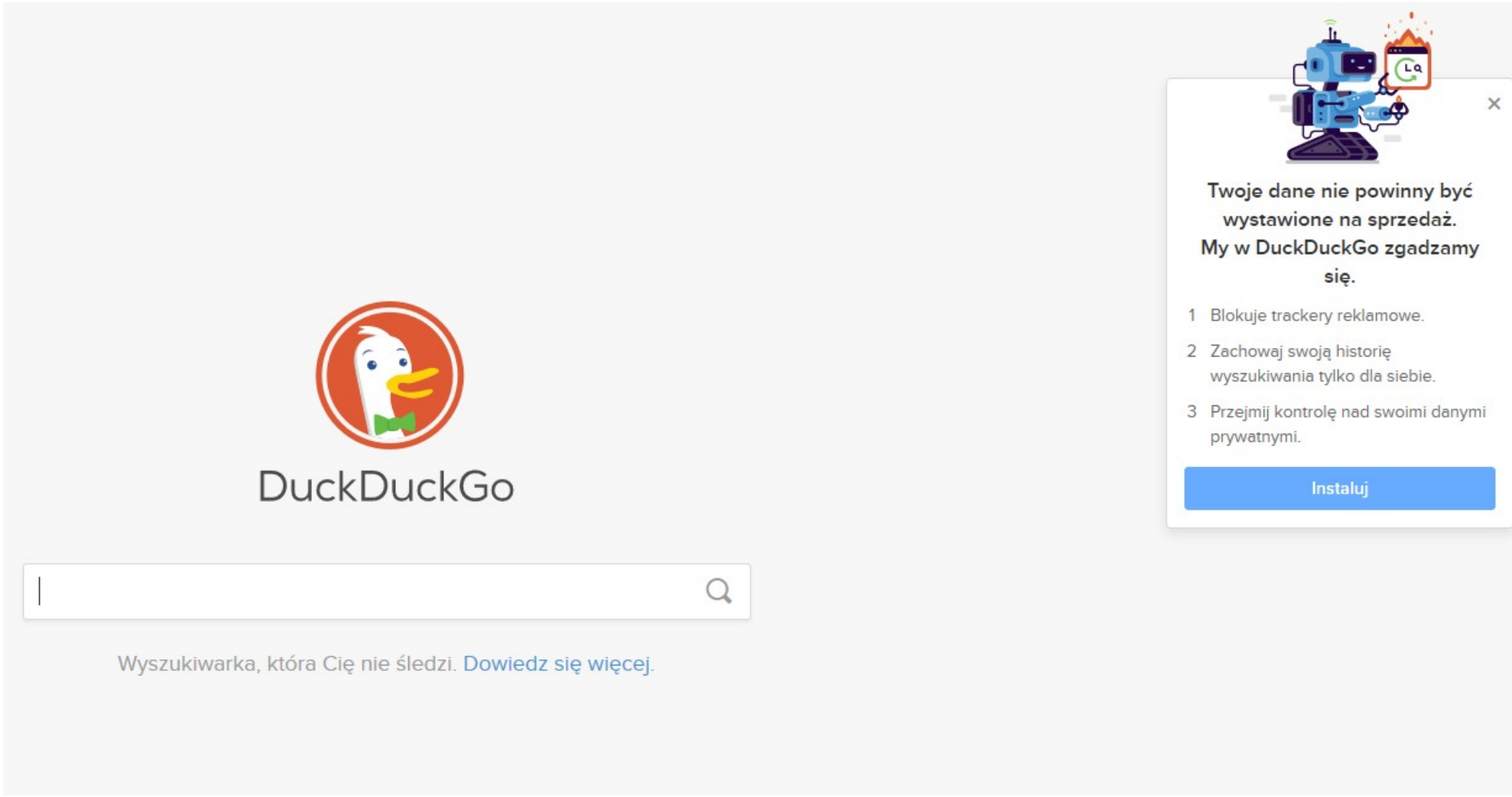
TBC?

- Spring Profiles vs Environment vs build time configurations vs configuration hell
- Deep dive in Spring/Java properties
- Problem investigation in enterprise environment

Q&A

42

Q&A


DuckDuckGo
Wyszukiwarka, która Cię nie śledzi. Dowiedz się więcej.
Twoje dane nie powinny być wystawione na sprzedaż. My w DuckDuckGo zgadzamy się.
1 Blokuje trackery reklamowe.
2 Zachowaj swoją historię wyszukiwania tylko dla siebie.
3 Przejmij kontrolę nad swoimi danymi prywatnymi.
Instaluj

DuckDuckGo

- Blocks Ad Trackers
- Full control about search data - even before GDPR
- Supports !bangs, !w will search Wikipedia directly
- Favours StackOverflow answers
- Drops out pure searches instead of tailored ones like G

Q&A

www.facebook.com/dtkociegniazdko

Common beans

- **https://www.realsimple.com/food-recipes/shopping-storing/food/common-types-beans**