Spring the ripper

Evgeny Borisov

bsevgeny@gmail.com

Who are you?

Big Data & Java Technical Leader

Mentoring

Consulting

Lecturing

Writing courses

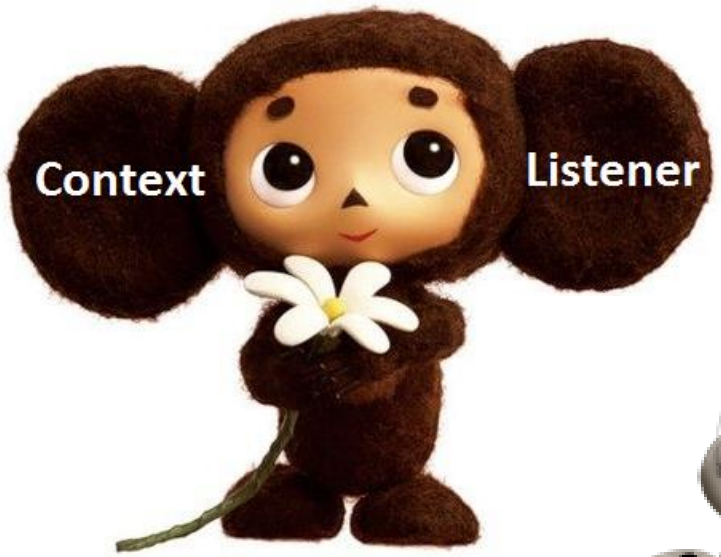
Writing code

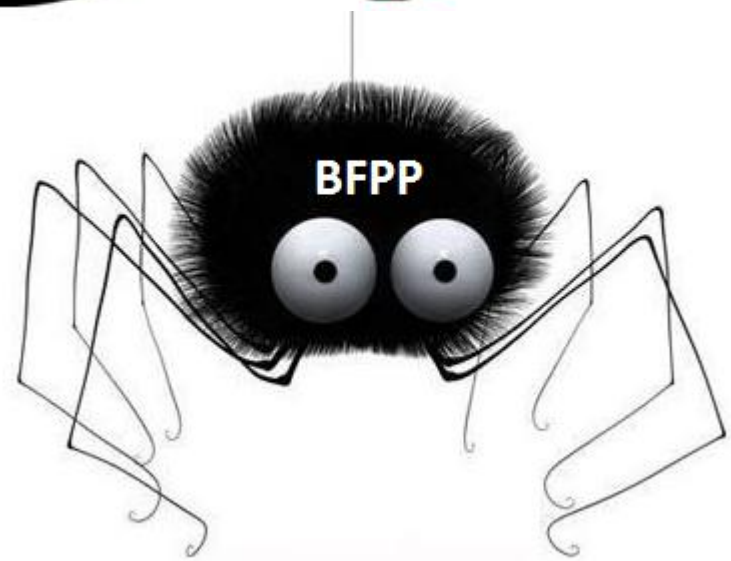
Never writing singletons
I drink them and
I know Spring

Agenda

- Spring internals
- Spring lifecycle
- Spring contexts
- How does spring influence the performance

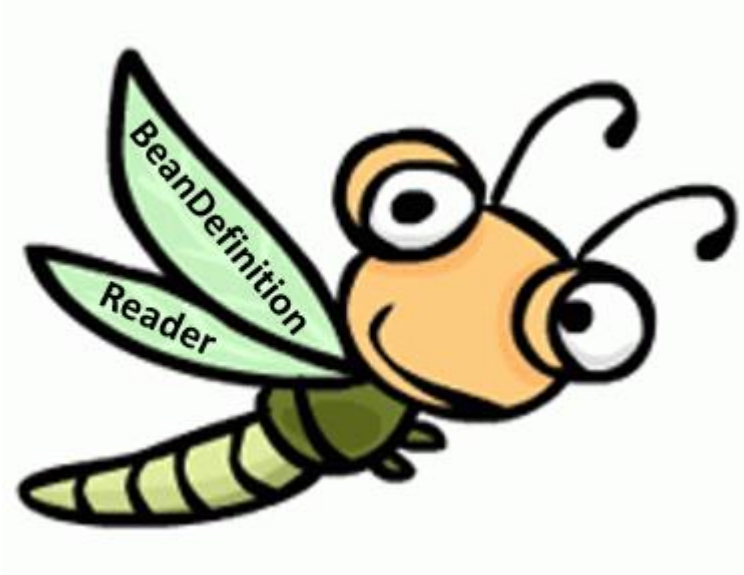
Spring in pictures





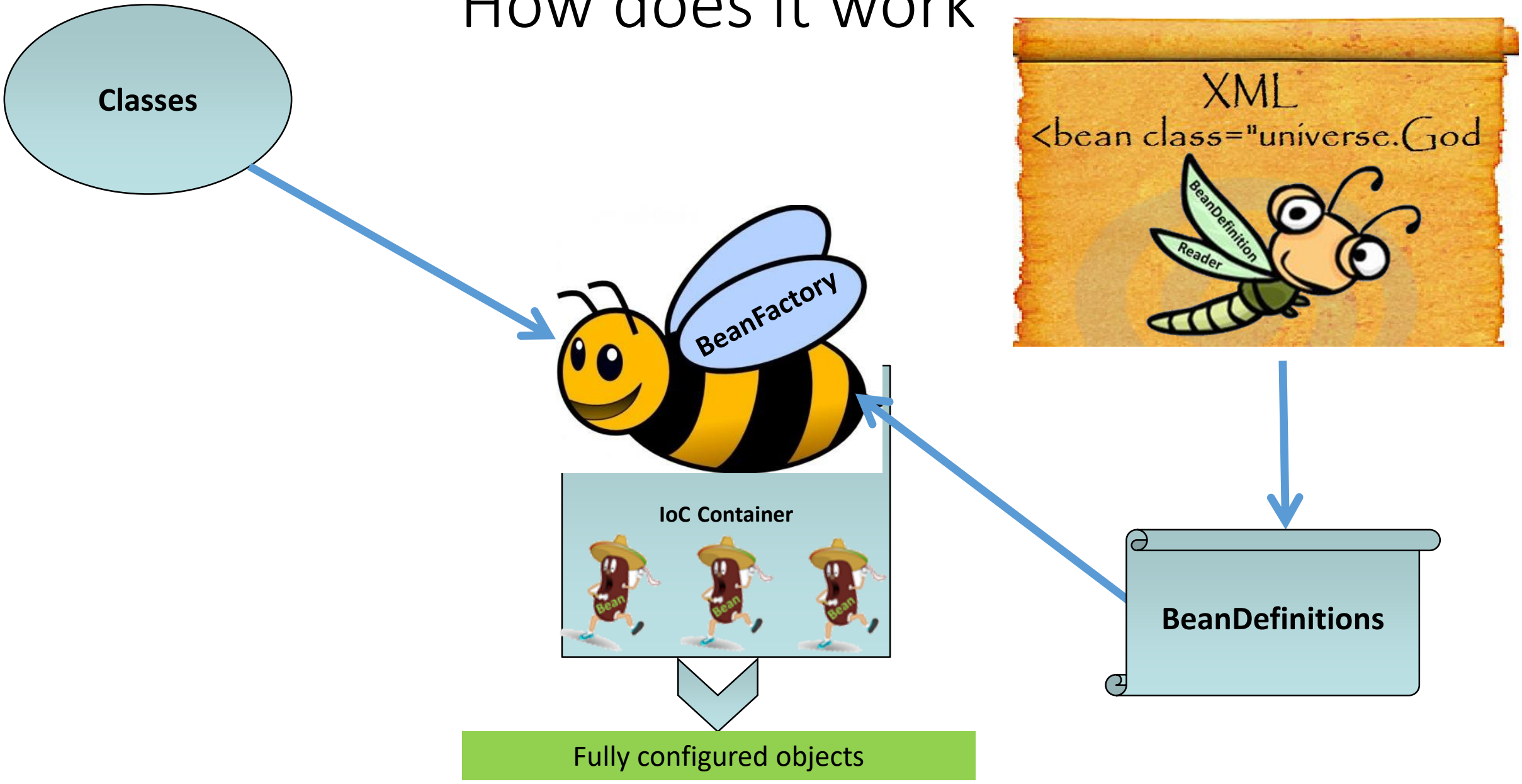
26.11.2003

XmlBeanDefinitionReader



Lets see how was declared bean via xml

How does it work



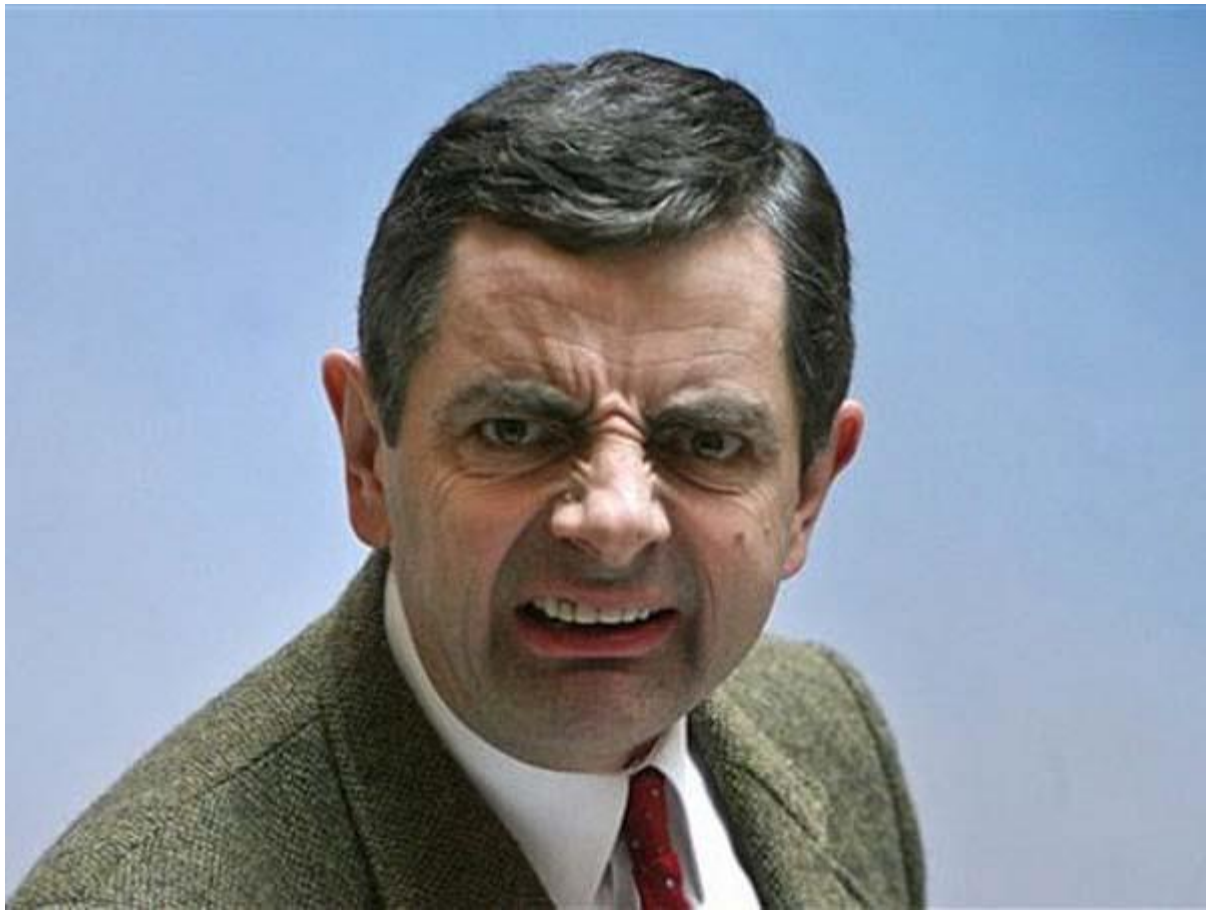
BeanPostProcessor

- Allows to configure our beans before they will be inserted into IOC
- This interface has 2 methods
  - Object postProcessBeforeInitialization(Object bean, String beanName)
  - Object postProcessAfterInitialization(Object bean, String beanName)

- Init method will be invoked in between
  - init-method
  - afterPropertiesSet
  - @PostConstruct

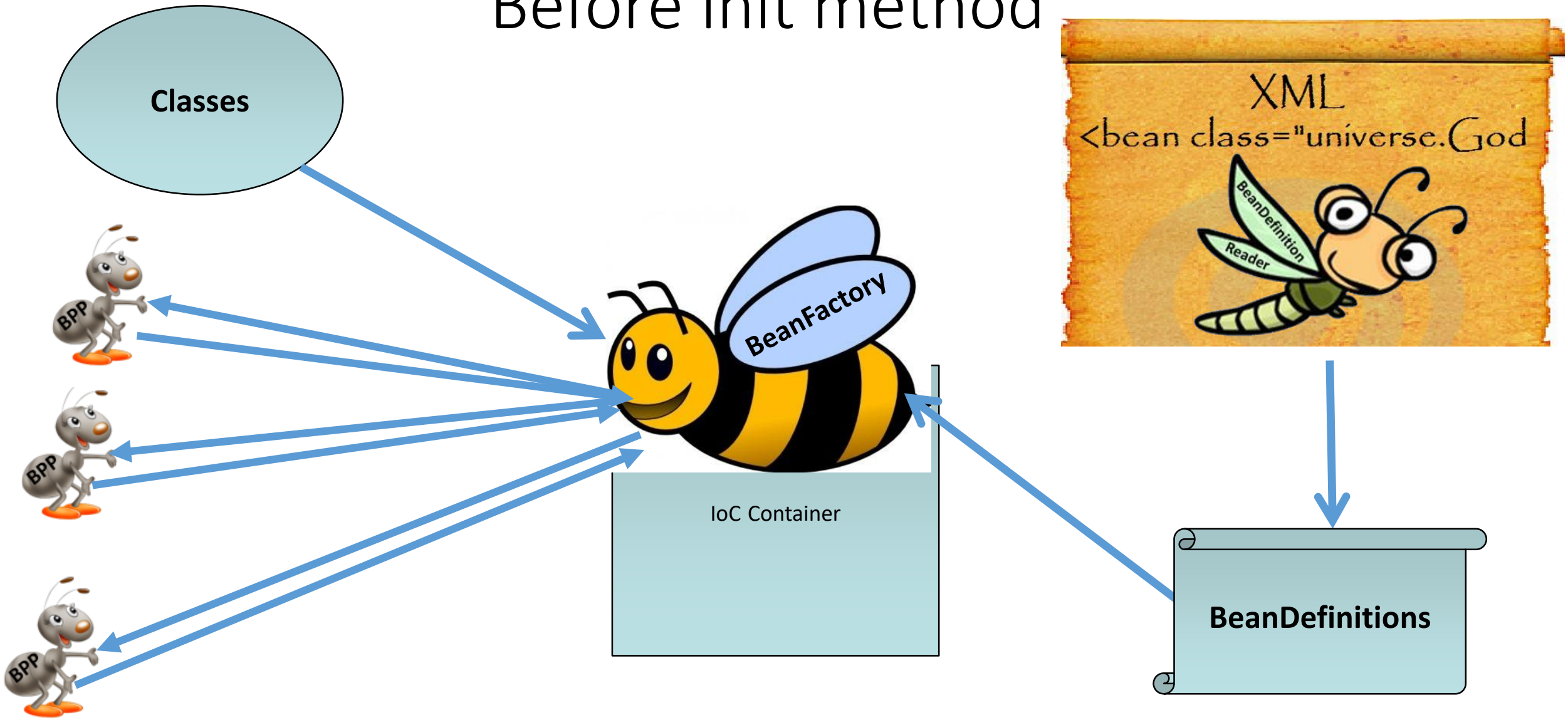
I have a question

Why do we need init methods if we have the constructor?

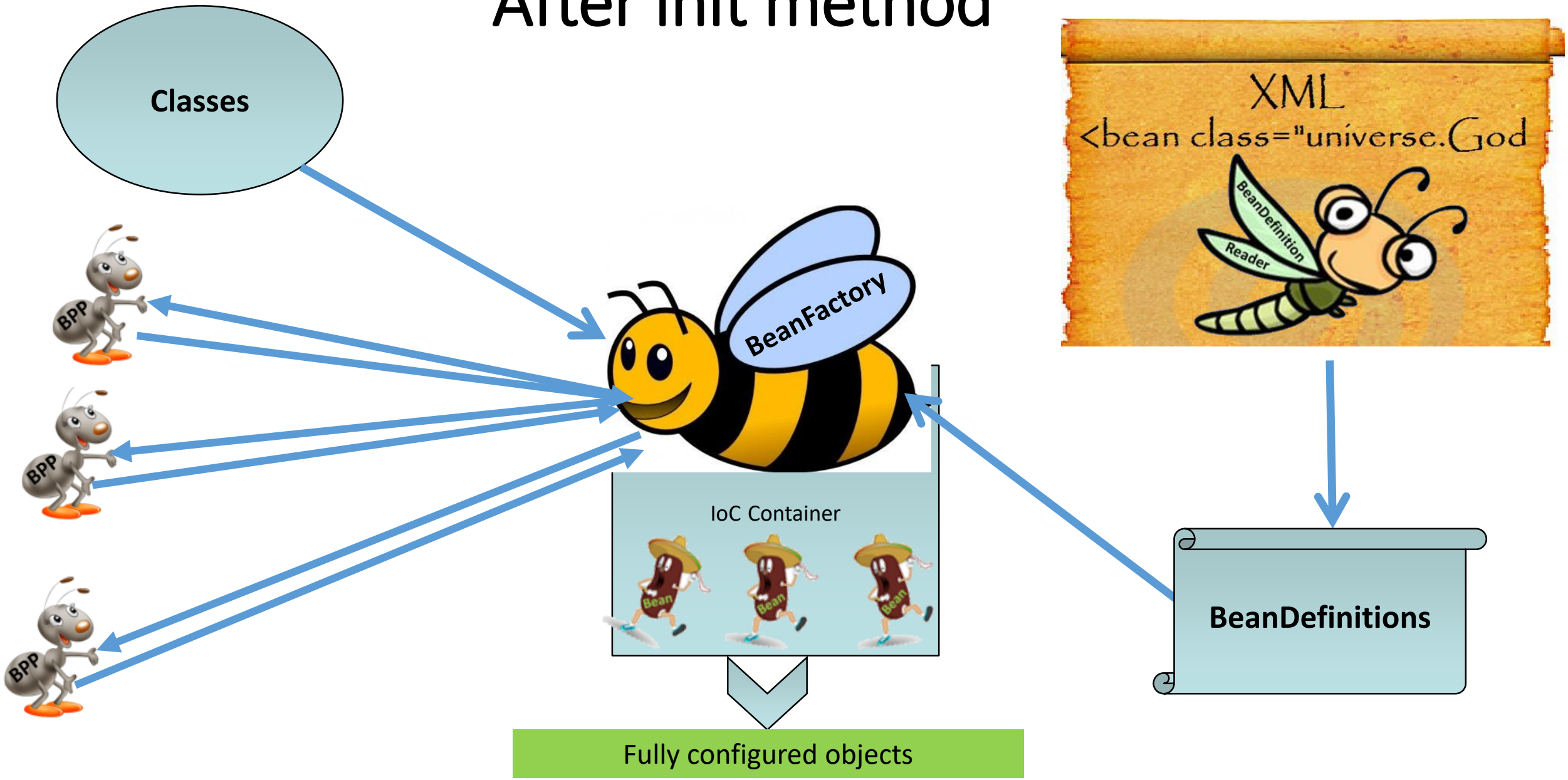


Don’t you heard about 2 phase constructor?

Before init method

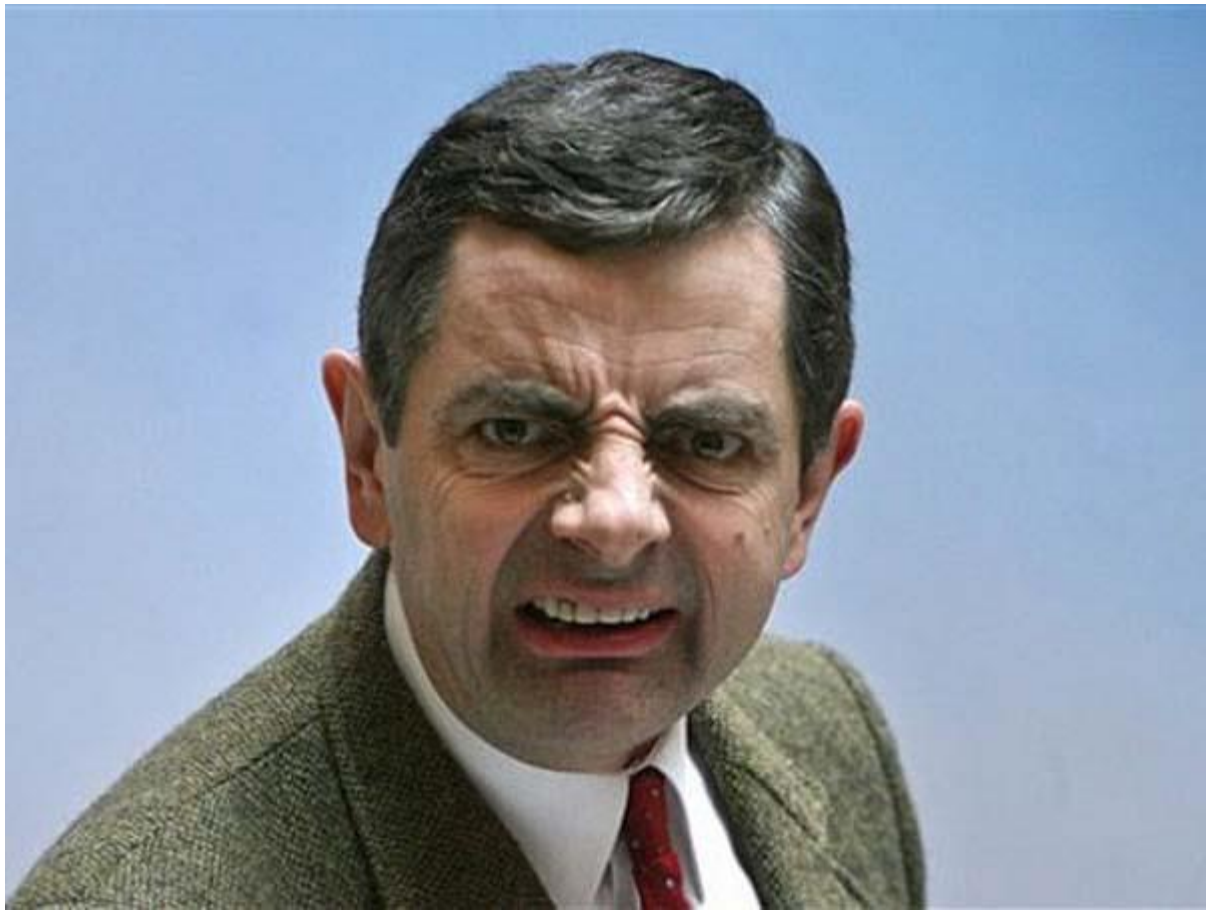


After init method



I have a question

Why do we need 2 methods?

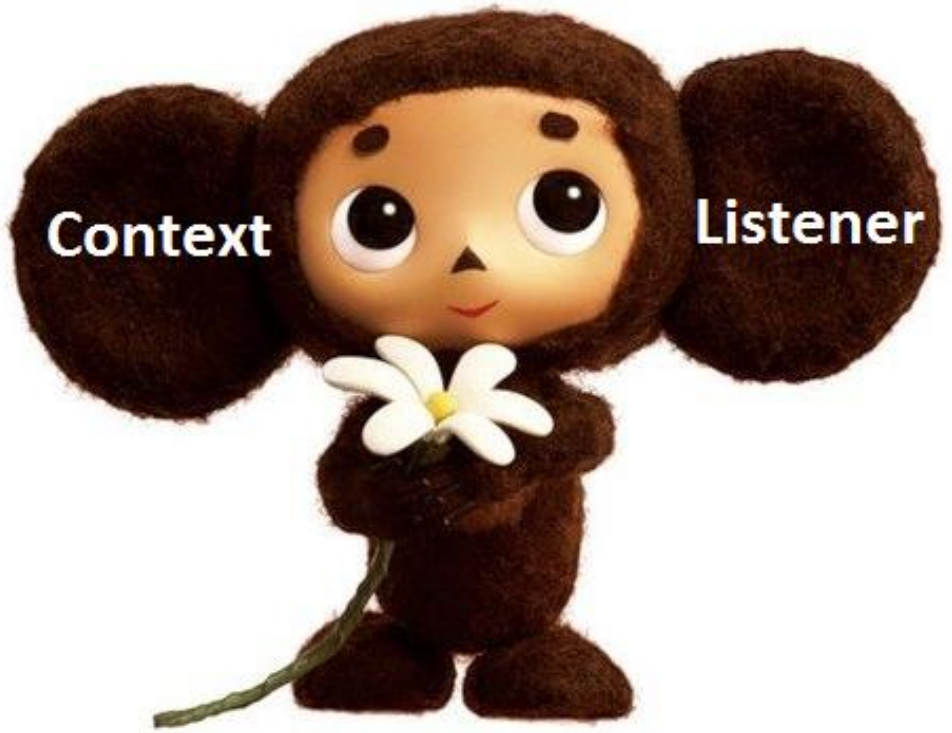


Don’t you heard about proxy?

Another component ApplicationListener

- ContextStartedEvent
- ContextStoppedEvent
- **ContextRefreshedEvent**
- ContextClosedEvent

- The context can be fetched from each event

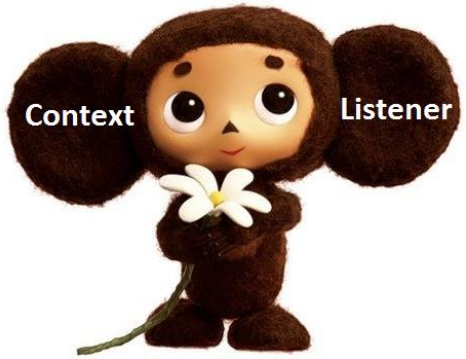


3 phase constructor

- Constructor

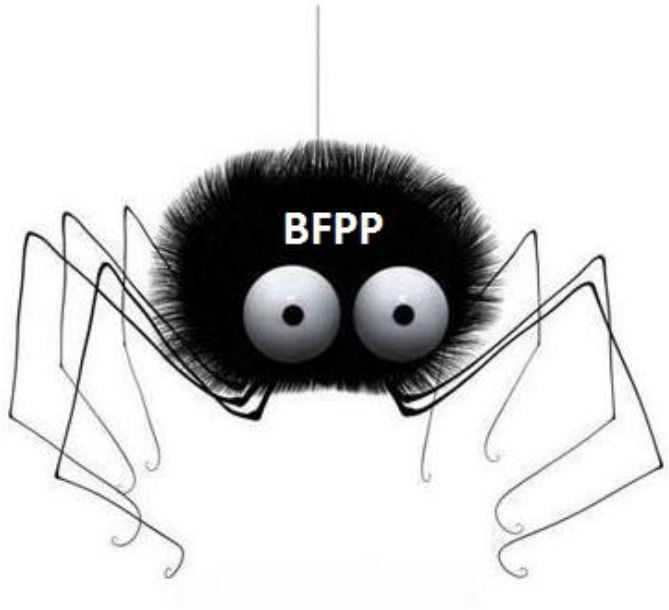
- @PostConstruct

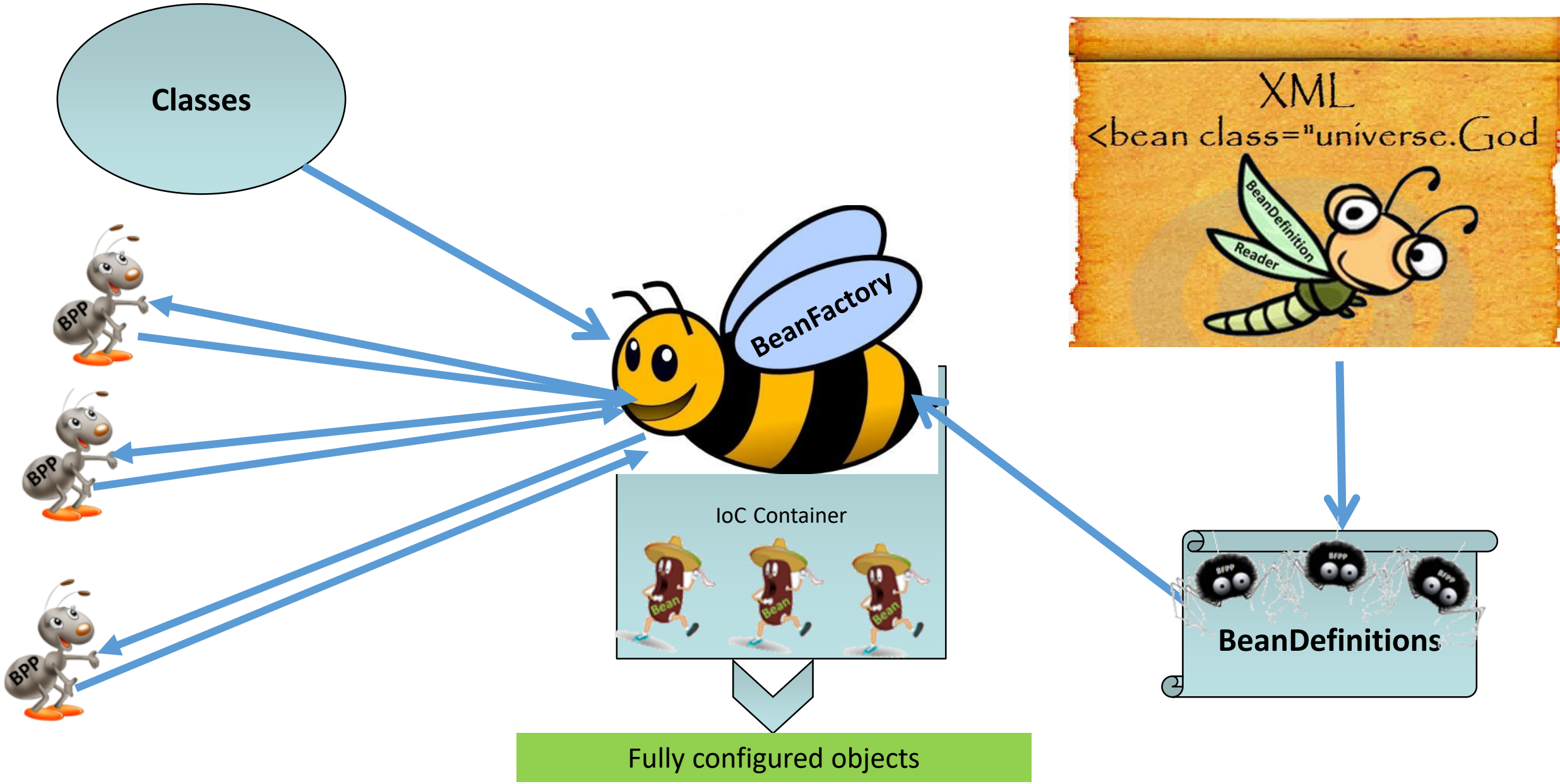
- @AfterProxy



BeanFactoryPostProcessor

- Allows to configure bean definitions before beans creation
- This interface has only one method
- postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory)
- This method will invoked when there are only BeanDefinitions and BeanFactory exists




Classes
BPP
BPP
BPP
BeanFactory
IoC Container
Bean
Bean
Bean
Fully configured objects
XML
<bean class="universe.God
BeanDefinition
Reader
BeanDefinitions

@Component

- `<context:component-scan base-package="com…"/>`
- `new AnnotationConfigApplicationContext("com");`

ClassPathBeanDefinitionScanner

- It is not BeanPostProcessor, niether BeanFactoryPostProcessor
- It is ResourceLoaderAware
- It creates BeanDefinitions from all classes annotated,
  with @Component,
  or any other annotation annotated @Component

Java Config

- `new` AnnotationConfigApplicationContext(JavaConfig.`class`);

- It seems that it must be parsed with some BeanDefinitionReader, like it was with xml
- Even the name of the class is AnnotatedBeanDefinitionReader
- But no, AnnotatedBeanDefinitionReader doesn’t implement anything
- It is just a part of ApplicationContext
- It just registers all JavaConfigs

```java
@Configuration
@ComponentScan("root")
public class JavaConfig {
    @Bean
    public CoolDao dao() {
        return new CoolDaoImpl();
    }

    @Bean(initMethod = "init")
    @Scope(BeanDefinition.SCOPE_PROTOTYPE)
    public CoolService coolService() {
        CoolServiceImpl service = new CoolServiceImpl();
        service.setDao(dao());
        return service;
    }
}
```

Who handle JavaConfig?

- ConfigurationClassPostProcessor (special BeanFactoryPostProcessor)
- AnnotationConfigApplicationContext will register it
- It creates bean definitions according @Bean
- And also handles
  - @Import
  - @ImportResource
  - @ComponentScan (Scanner will be used again)

Groovy Config

```
beans {
    myDao(DaoImpl)



}
```

- To create context:
- `new GenericGroovyApplicationContext("context.groovy");`

- Will be parsed by `GroovyBeanDefinitionReader`

May be we can write our own context?


What performance price should I pay for spring?

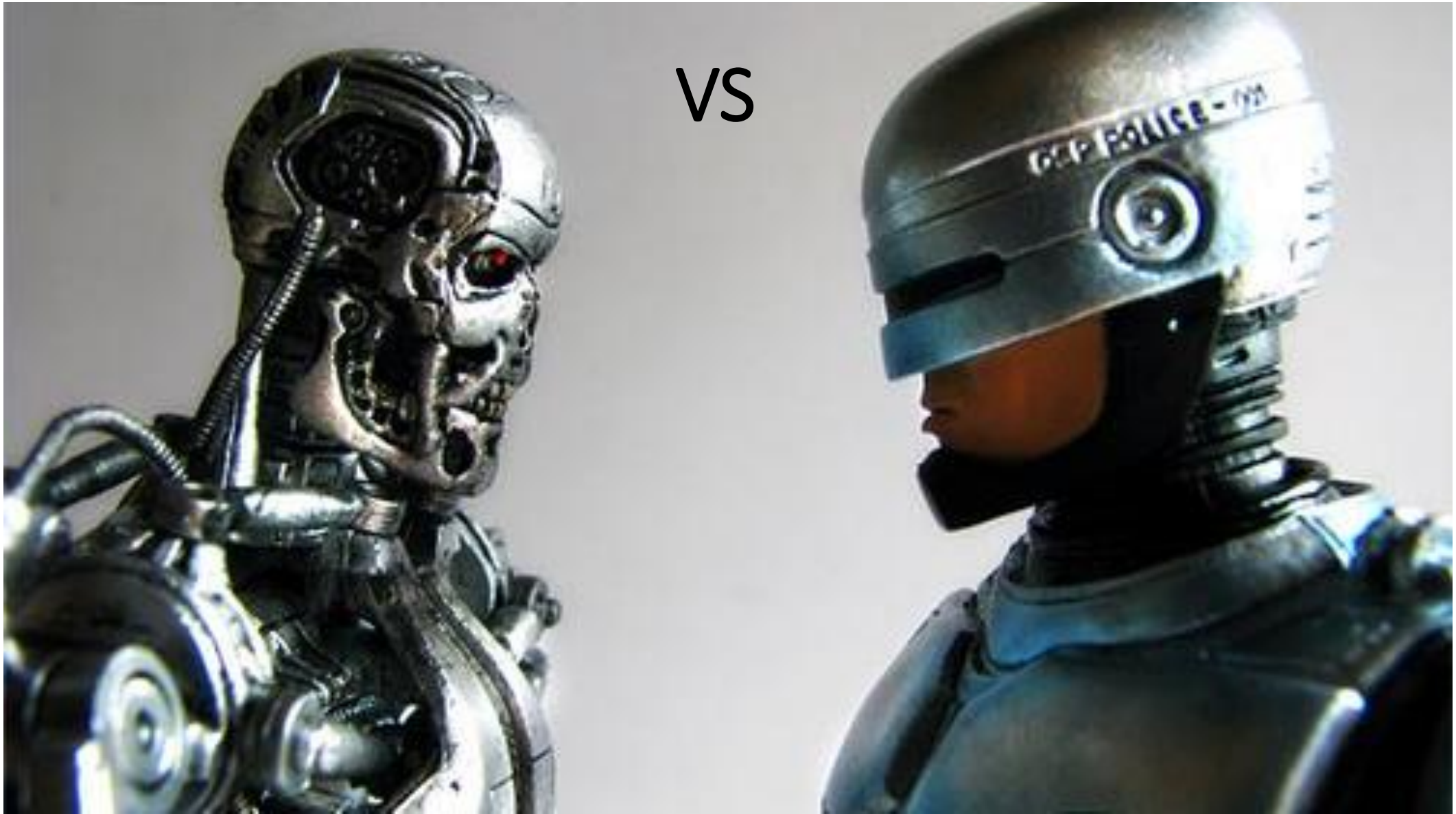
What will we benchmark?

- Object creation (new / reflection / Spring)
- Lookup & Injection
- Creating proxy (generating new classes)
- Method invocation via proxy
- Aspects

CGLIB

VS

Dynamic Proxy



How to do benchmark?

Students think that benchmark is:

Junior Software Engineer

```
public static void main(String[] args) throws Exception {
    ApplicationContext context = new AnnotationConfigApplicationContext("com");
    long before = System.currentTimeMillis();
    Dao dao = context.getBean(Dao.class);
    long after = System.currentTimeMillis();
    System.out.println(after-before);
}
```

Middle Software Engineer

```
public static void main(String[] args) throws Exception {
    ApplicationContext context = new AnnotationConfigApplicationContext("com");
    long before = System.nanoTime();
    for (int i=0;i<1000000;i++) {
        Dao dao = context.getBean(Dao.class);
    }
    long after = System.nanoTime();
    System.out.println((after-before)/1000000);
}
```

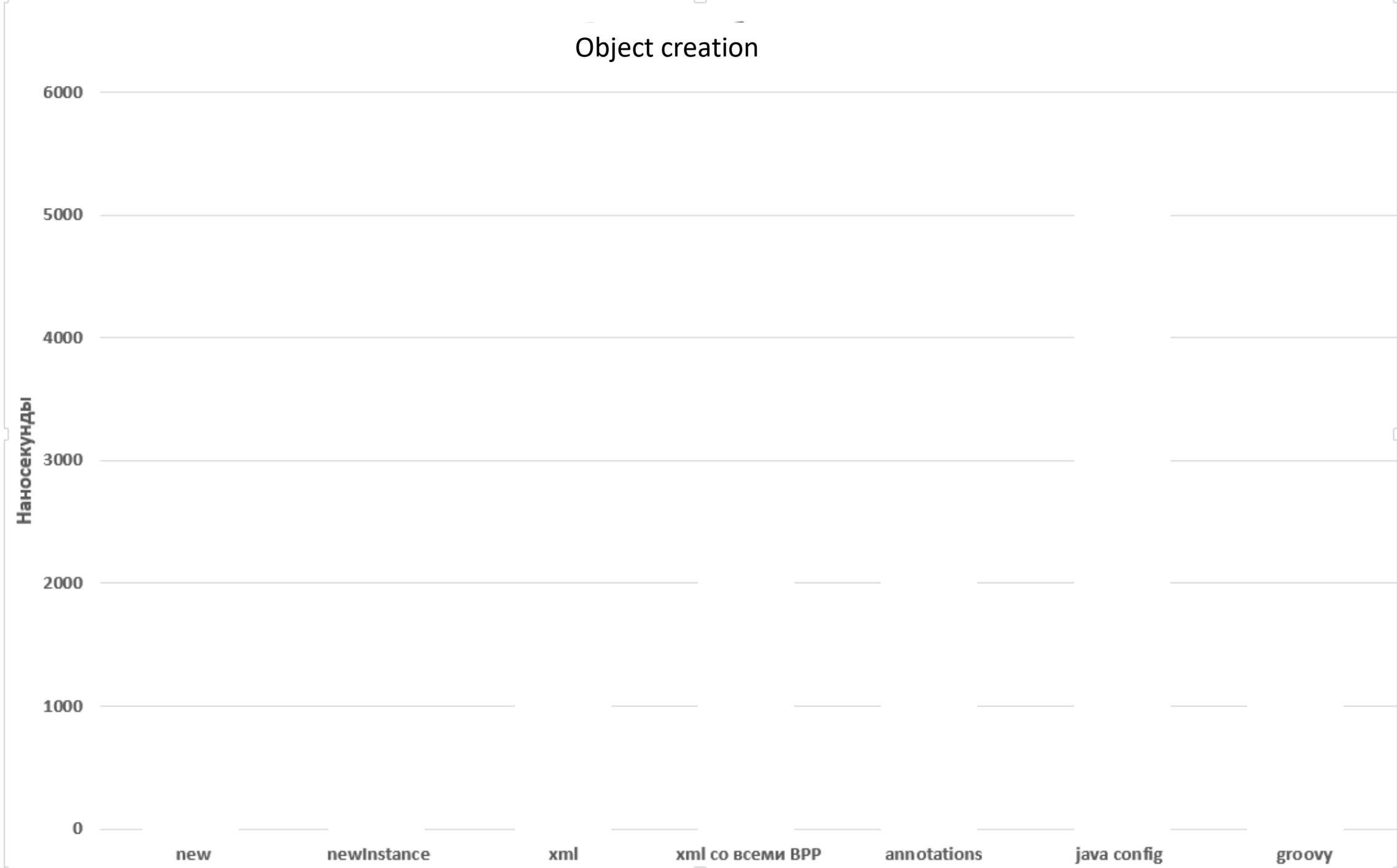
Senior Software Engineer

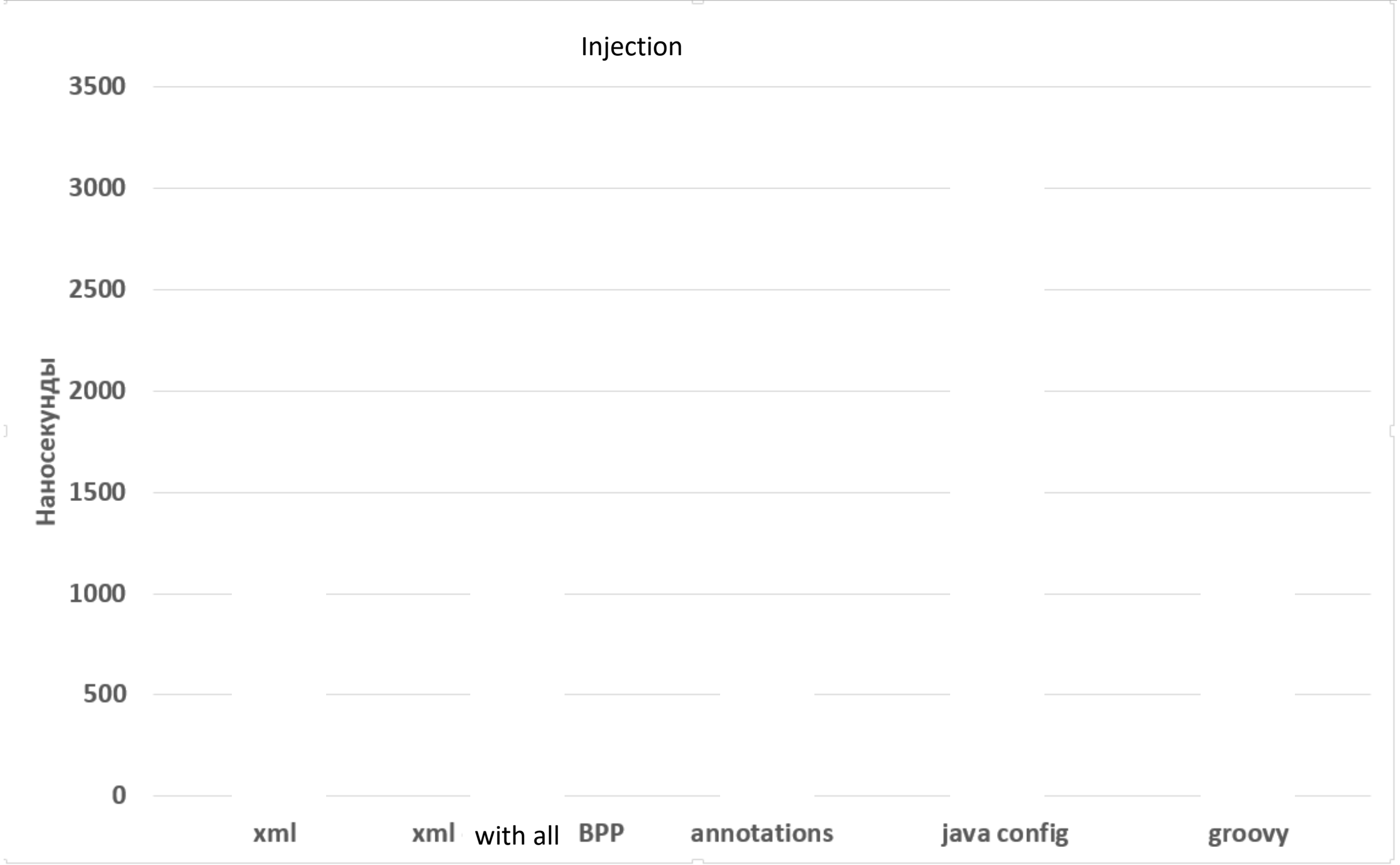
```
public static void main(String[] args) throws Exception {
    ApplicationContext context = new AnnotationConfigApplicationContext("com");
    Dao dao=null;
    long before = System.nanoTime();
    for (int i=0;i<1000000;i++) {
        dao = context.getBean(Dao.class);
    }
    long after = System.nanoTime();
    System.out.println((after-before)/1000000);
    System.out.println(dao);
}
```

The architect

You drink,
I'll benchmark


Look up
100
90
80
70
60
50
40
30
20
10
0
Наносекунды
map.get("x")
context.getBean("x")
context.getBean(X.class)


Object creation
6000
5000
4000
3000
2000
1000
0
Наносекунды
new
newInstance
xml
xml со всеми BPP
annotations
java config
groovy

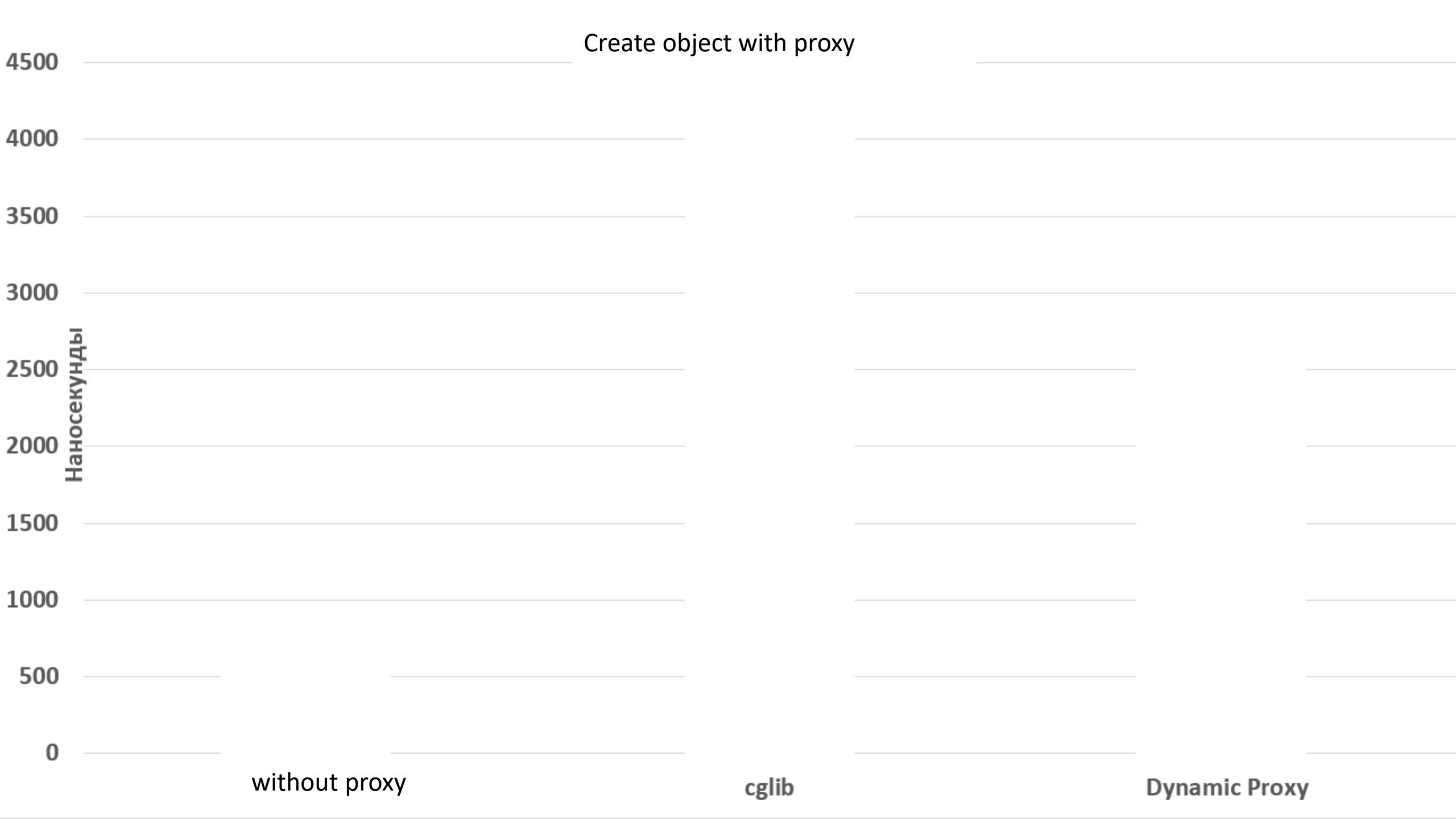

Injection
3500
3000
2500
2000
1500
1000
500
0
Наносекунды
xml
xml with all
BPP
annotations
java config
groovy

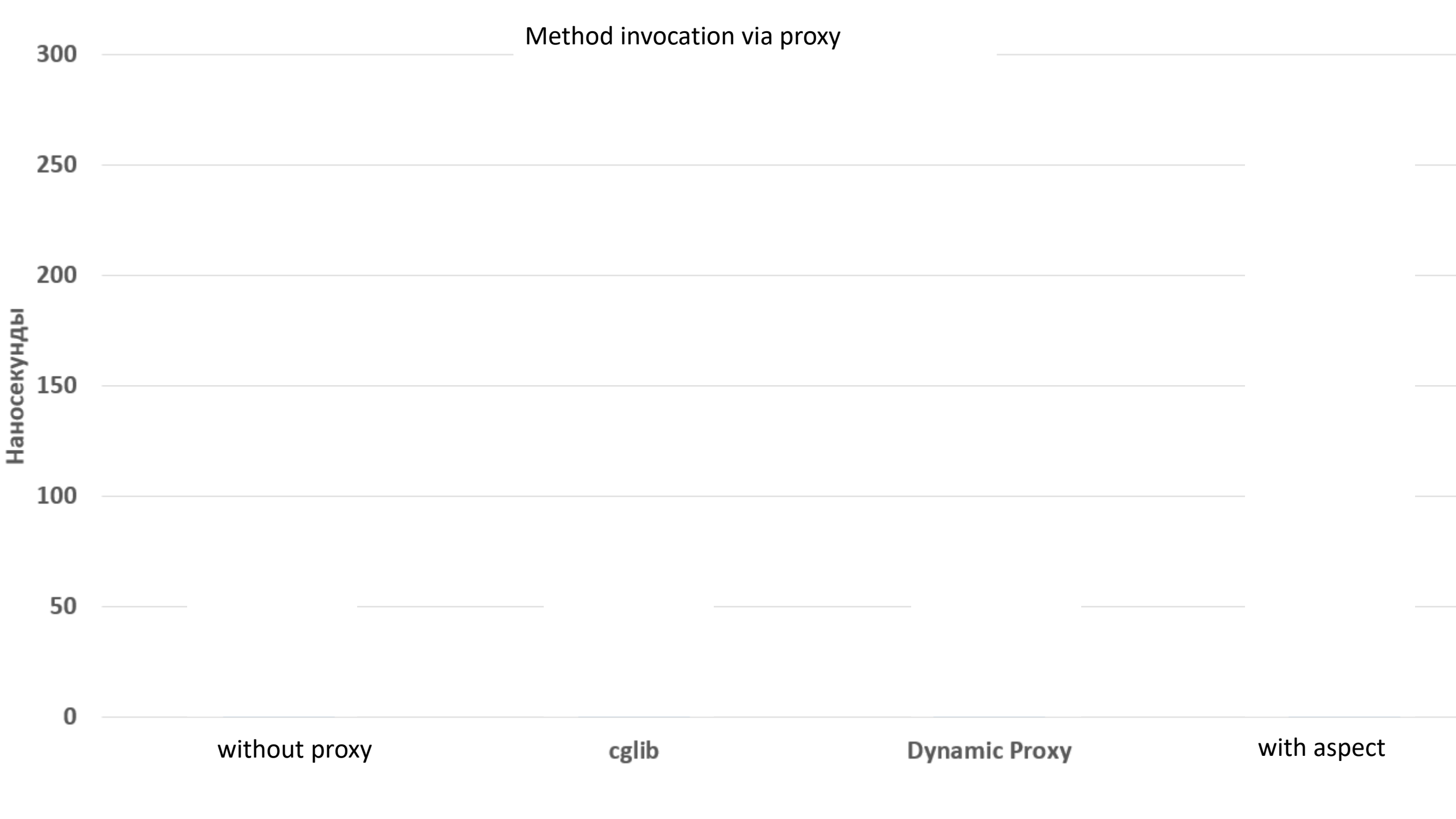
Panic…

You can breathe

- What time need for creation of 1 million prototypes?
- 4.5 sec
- What time need for lookup of 1 million singletons 0.1 sec

You can breathe


Create object with proxy
4500
4000
3500
3000
2500
2000
1500
1000
500
0
Наносекунды
without proxy
cglib
Dynamic Proxy


Method invocation via proxy
300
250
200
150
100
50
0
Наносекунды
without proxy
cglib
Dynamic Proxy
with aspect

Conclusions

- If you want to work well – use Spring
- If you want it to work well – be familiar with Spring internals