Спринг Потрошитель

Евгений Борисов

bsevgeny@gmail.com

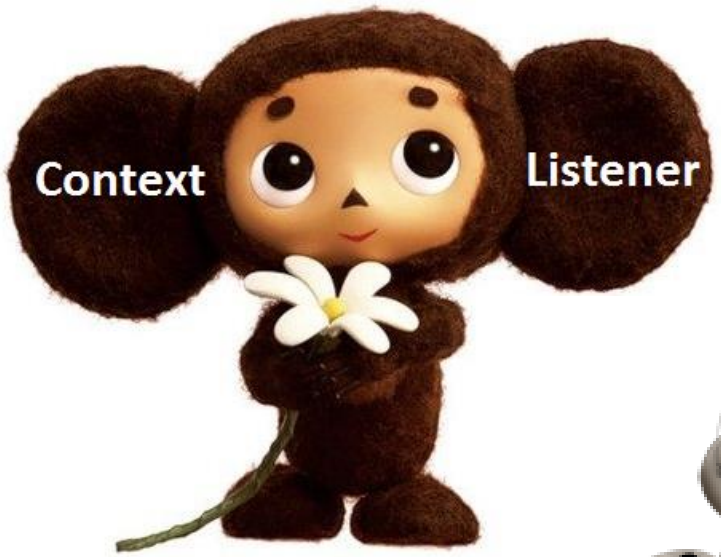
Сегодня в программе

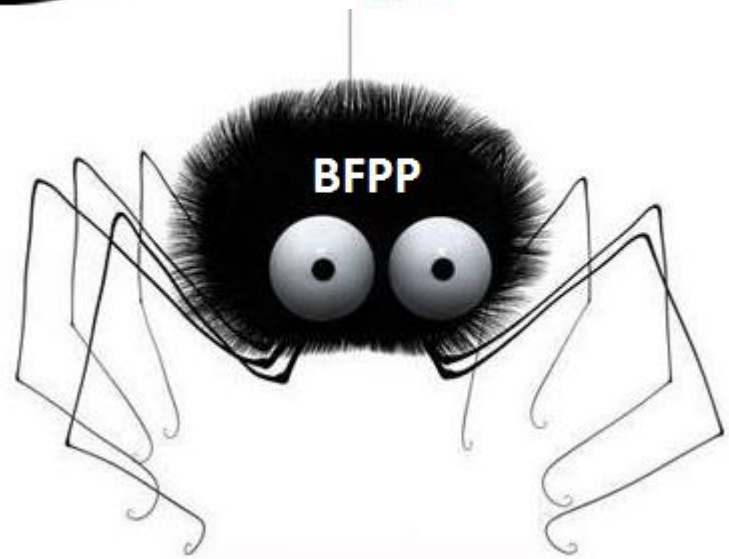
- Как работает Спринг
  - 4 вида контекста + напишем пятый
  - Сравнение контекстов, обсуждение плюсов и минусов
- Что входит в жизненный цикл Спринга
  - BeanDefinitionReader
  - BeanFactoryPostProcessor
  - FactoryBeans
  - BeanPostProcessors
  - ApplicationListener

Сегодня в программе

- А как Спринг влияет на производительность?
  - Цена создания объекта
    - Prototype – против синглтона
  - Цена создания прокси
  - Цена вызова метода через прокси
  - Аспекты: как разные поинткаты бьют по производительности
- А как это сделать?..
  - Обновление прототайпов в синглтоне при помощи JavaConfig
  - Протухание бинов
  - Custom Scopes

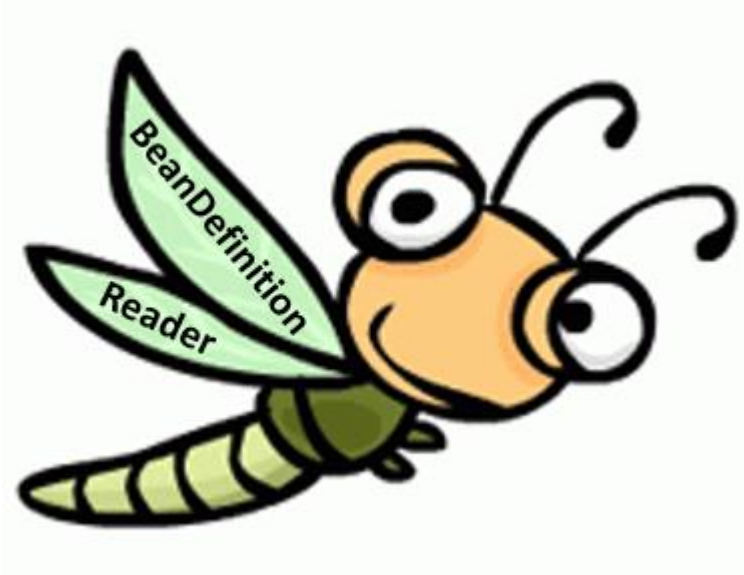
Спринг в картинках...





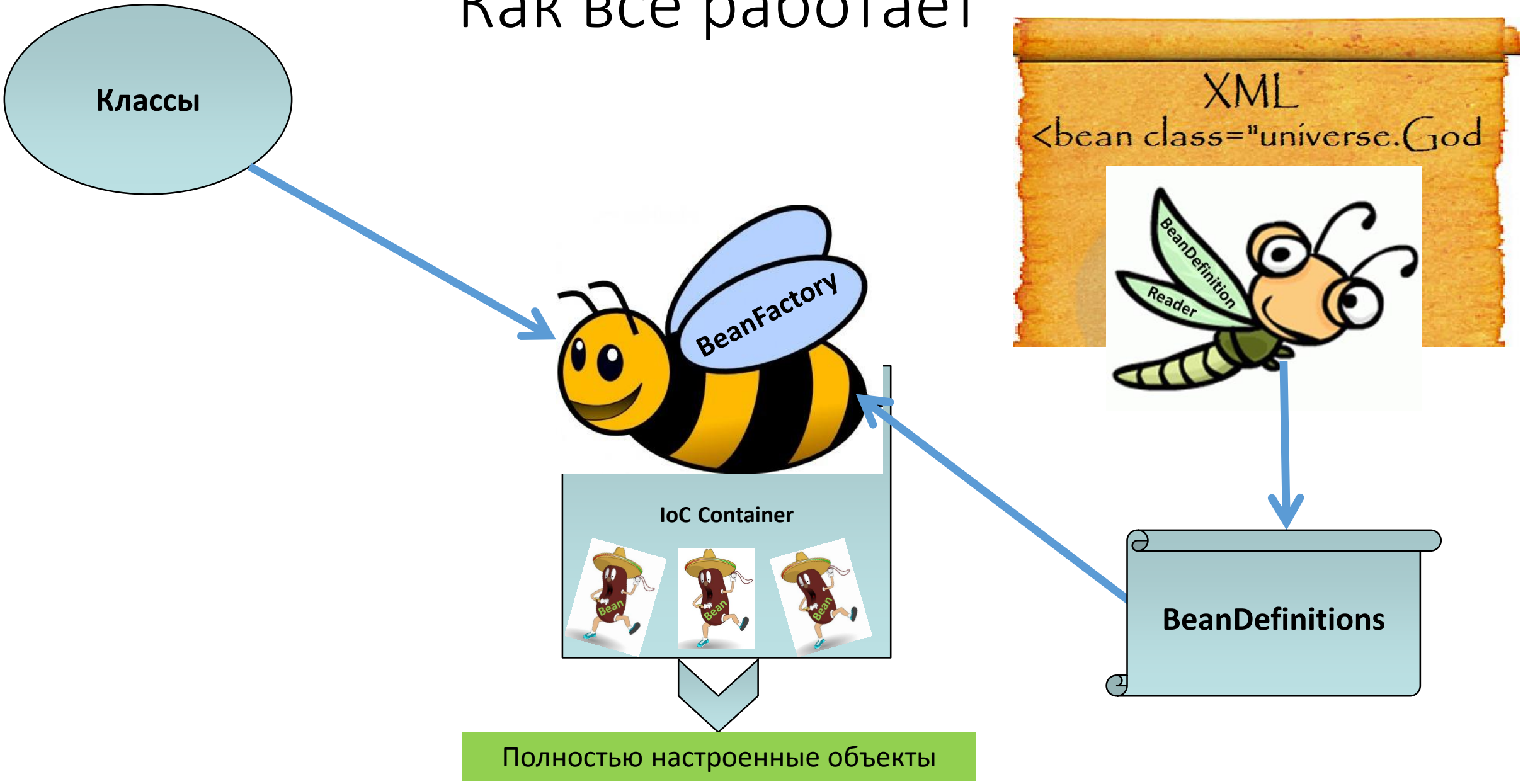
26.11.2003

XmlBeanDefinitionReader



Давайте посмотрим как декларировался БИН

Как всё работает



BeanPostProcessor

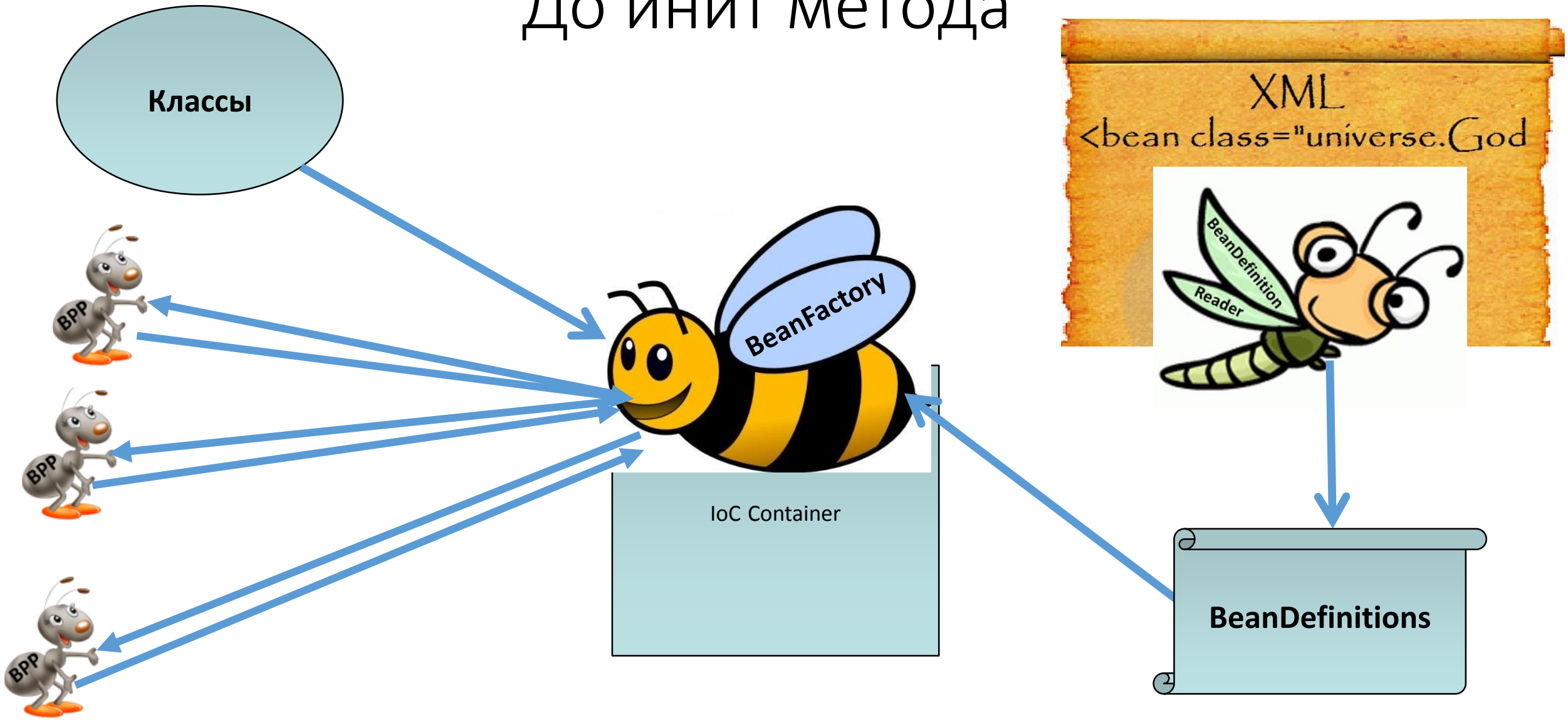
- Позволяет настраивать наши бины до того, как они попадают в контейнер
- У этого интерфейса 2 метода:
  - Object postProcessBeforeInitialization(Object bean, String beanName)
  - Object postProcessAfterInitialization(Object bean, String beanName)
- А между ними вызывается init метод
  - init-method
  - afterPropertiesSet
  - @PostConstruct

У меня вопрос

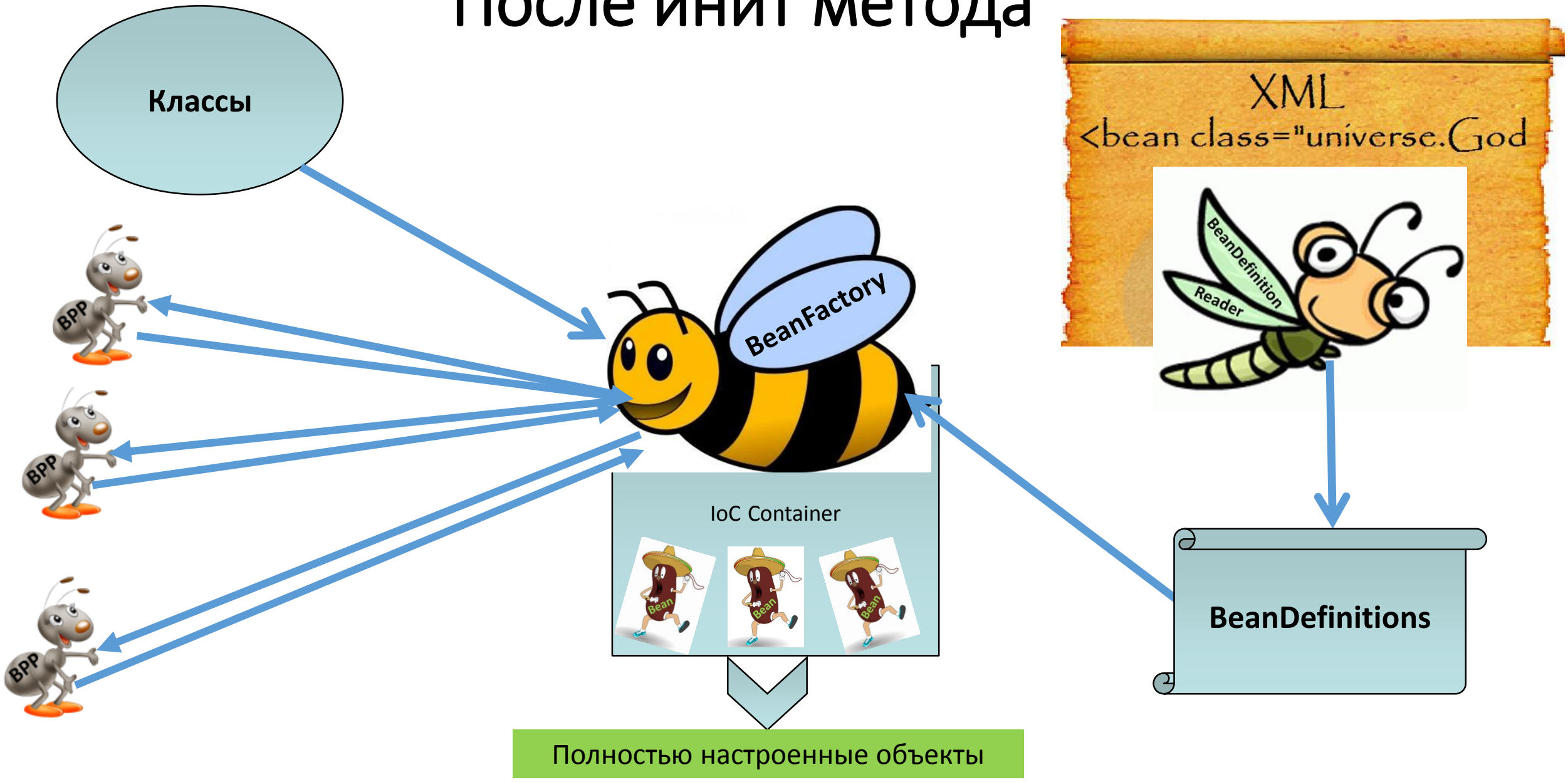
А на хрена нужны инит методы?
Конструктора мало что ли?

А ты про двухфазовый конструктор ничего не слышал???

До инит метода

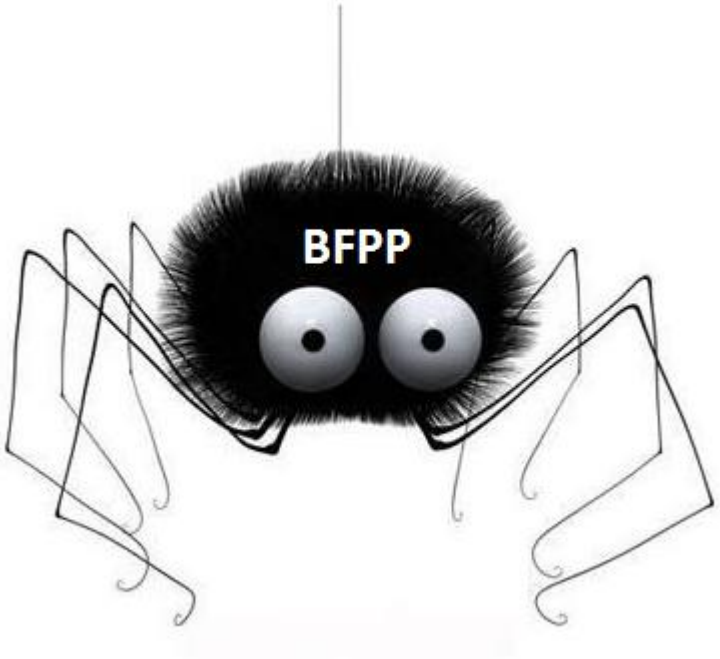


После инит метода

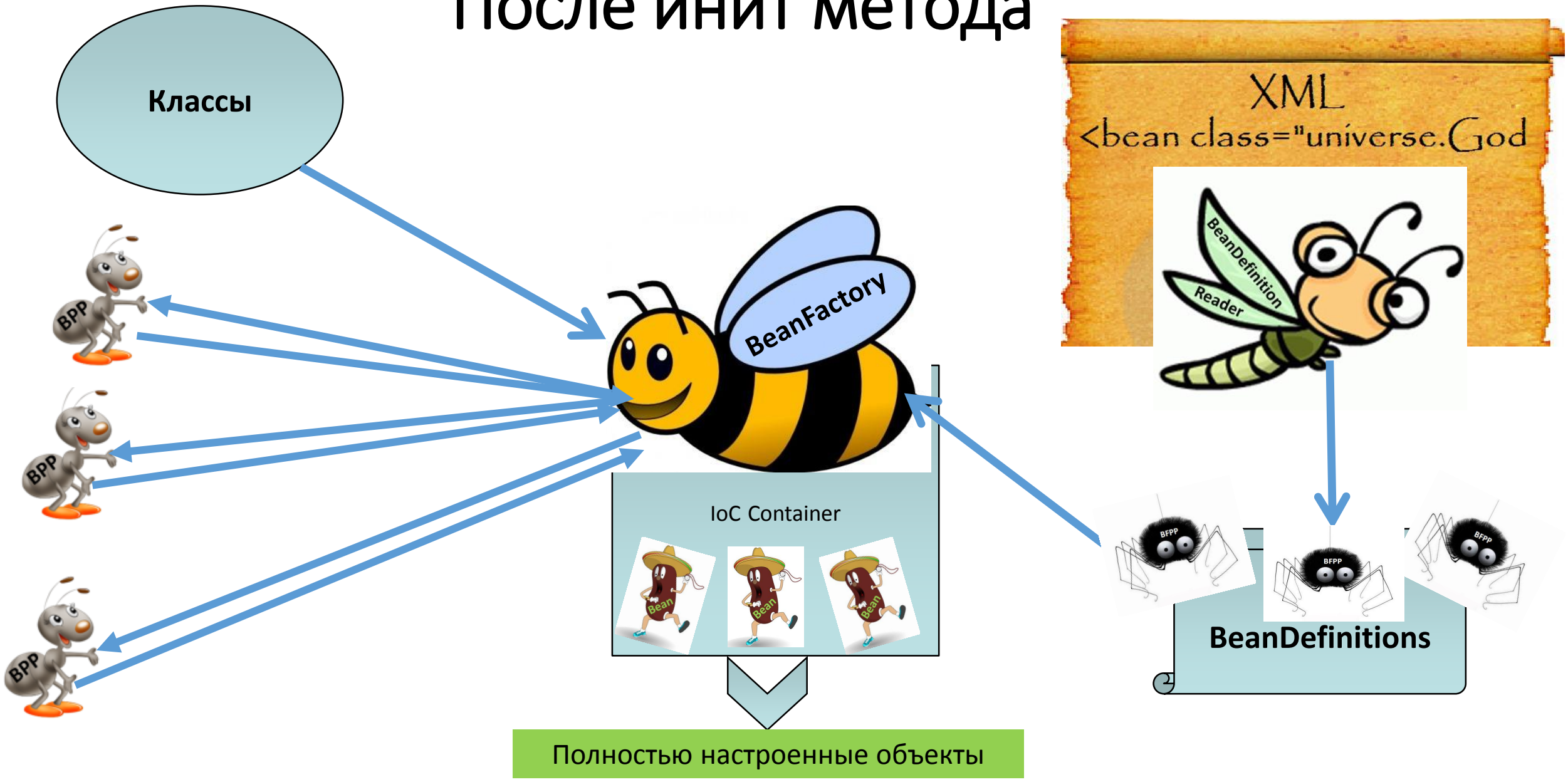


BeanFactoryPostProcessor

- Позволяет настравить бин дифиншоны, до того, как создаются бины
- Этот интерфэйс имеет один единственный метод:
- postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory)
- Этот метод запуститься на этапе, когда други
  и есть только BeanDefinitions



После инит метода



@Component

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

ClassPathBeanDefinitionScanner

- Не является ни BeanPostProcessor-ом, ни BeanFactoryPostProcessor-ом
- Он ResourceLoaderAware
- Создаёт BeanDefinitions из всех классов,
  над которыми стоит @Component,
  или другая аннотация включающая @Component

Java Config

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

- Казалось бы, его должен парсировать, какой-нибудь BeanDefinitionReader, как это было с XML
- И даже его класс его называется схоже: AnnotatedBeanDefinitionReader.
- Но нет AnnotatedBeanDefinitionReader, вообще ничего не имплементирует
- Он просто является часть ApplicationContext-а
- Он только регестрирует все JavaConfig-и

Кто обрабатывает JavaConfig?

- ConfigurationClassPostProcessor (особый BeanFactoryPostProcessor)
- Его регистрирует AnnotationConfigApplicationContext
- Он создаёт бин-дифигишоны по @Bean
- А так же относится к:
  - @Import
  - @ImportResource
  - @ComponentScan (да да там опять будет задействован крот)

Groovy Config

```
beans {
    myDao(DaoImpl)

    jPointService(JPointServiceImpl){bean->
        bean.scope = 'prototype'
        dao = myDao
    }
}
```

- Создаётся вот так:
  `new GenericGroovyApplicationContext("context.groovy");`

- Парсируется `GroovyBeanDefinitionReader`

А может мы свой контекст напишем?

Еще один компонент ApplicationListener

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

- Из любого ивента можно вытащить контекст

А теперь давайте решим что-нибудь **не** «Элементарное»…


Почём Spring для народа?

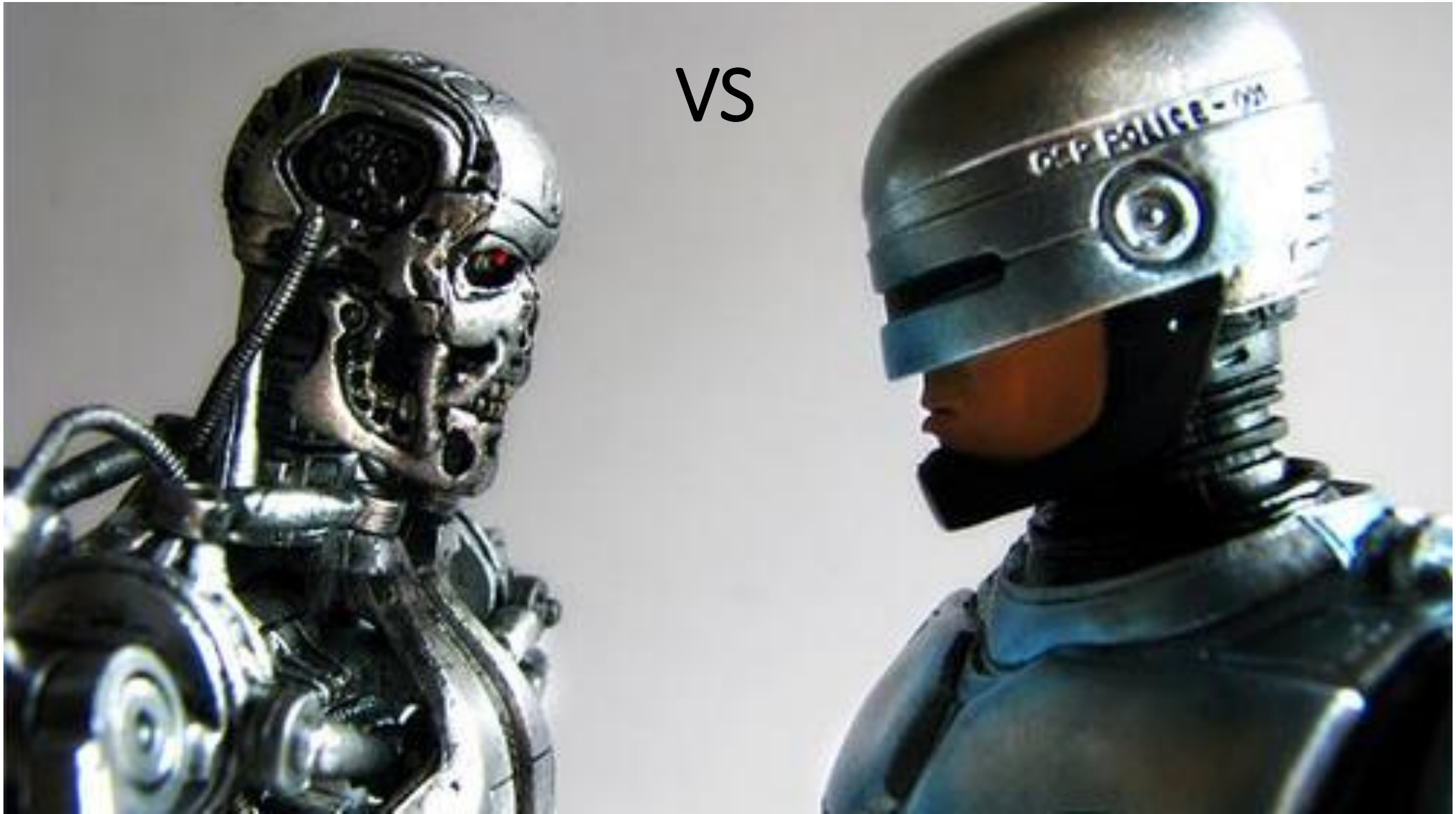
Что будем мерить?

- Время создания объекта (new / reflection / Spring)
- Время на лукап и инжекшн
- Время создания прокси
- Время вызова метода через прокси
- Аспекты

CGLIB

VS

Dynamic Proxy



Как будем мерить?

- Есть разные уровни понимания как делать MicroBenchmark

Уровень первый - Студент

Уровень второй - 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);
}
```

Уровень второй - Middle Software Engineer

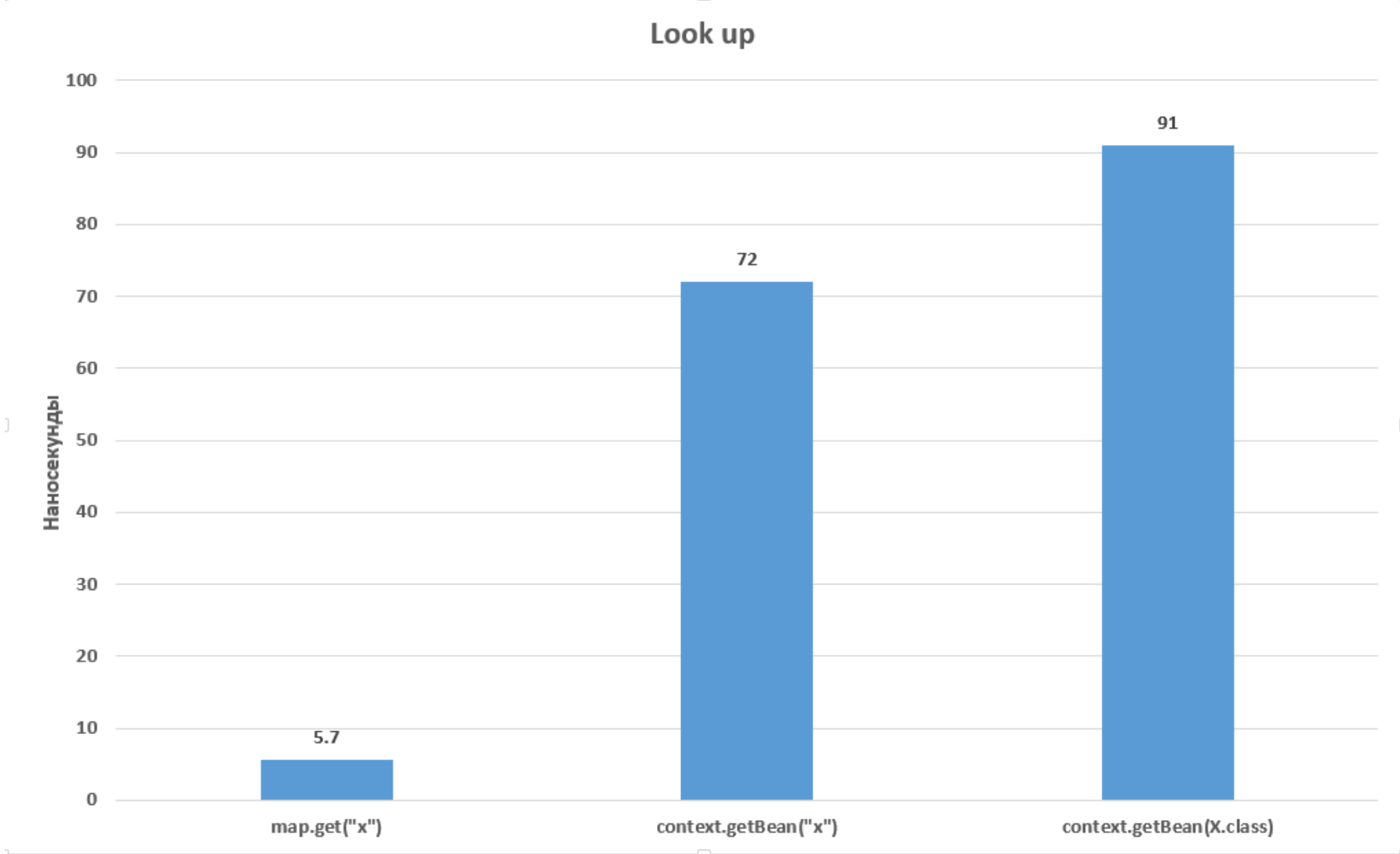
```
public static void main(String[] args) throws Exception {
    ApplicationContext context = new AnnotationConfigApplicationContext("com");
    long before = System.nanoTime();
    for (int i=0;i<1000000;i++) {
        Dao dao = context.getBean(Dao.class);
    }
    long after = System.nanoTime();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

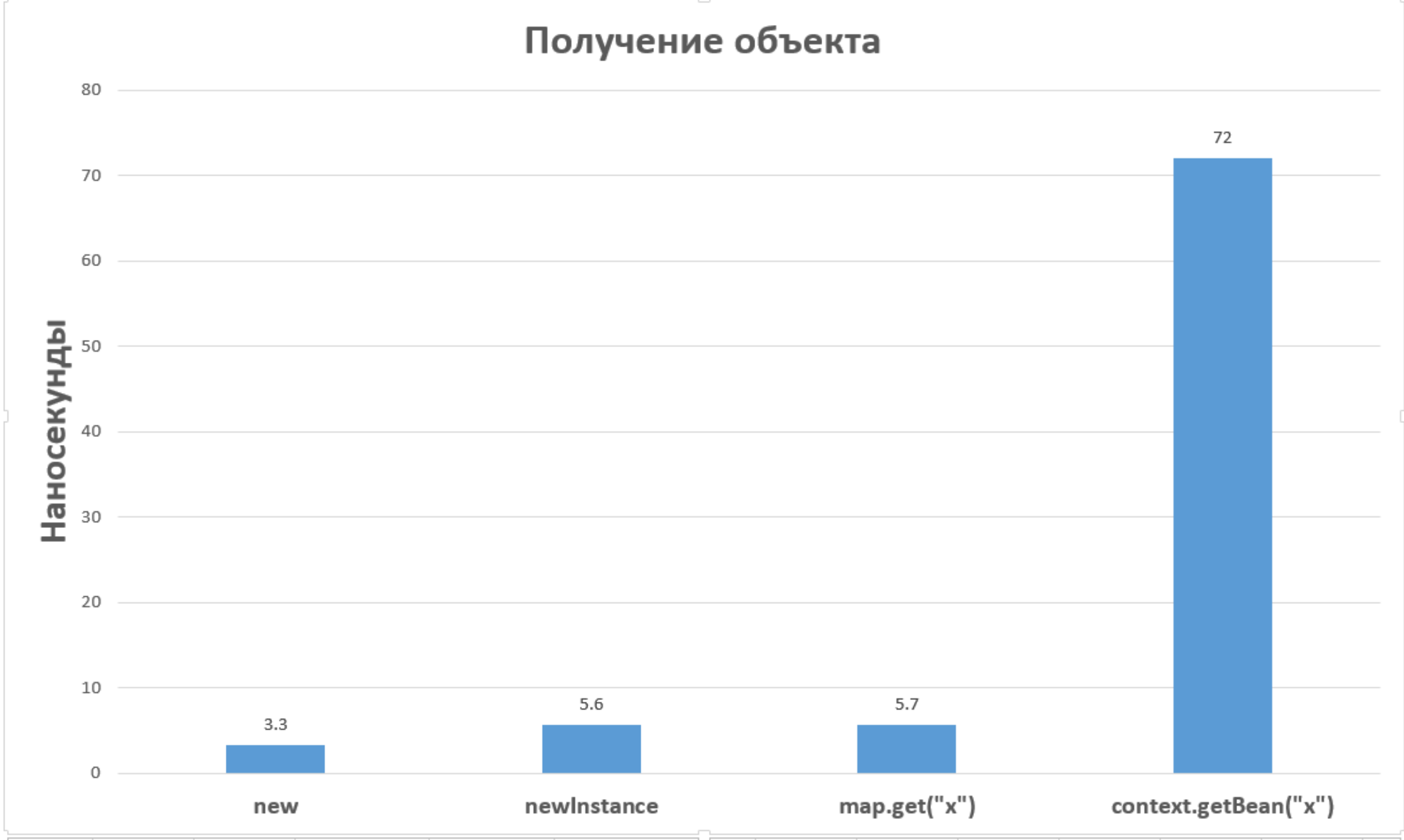
Уровень третий - Senior Software Engineer

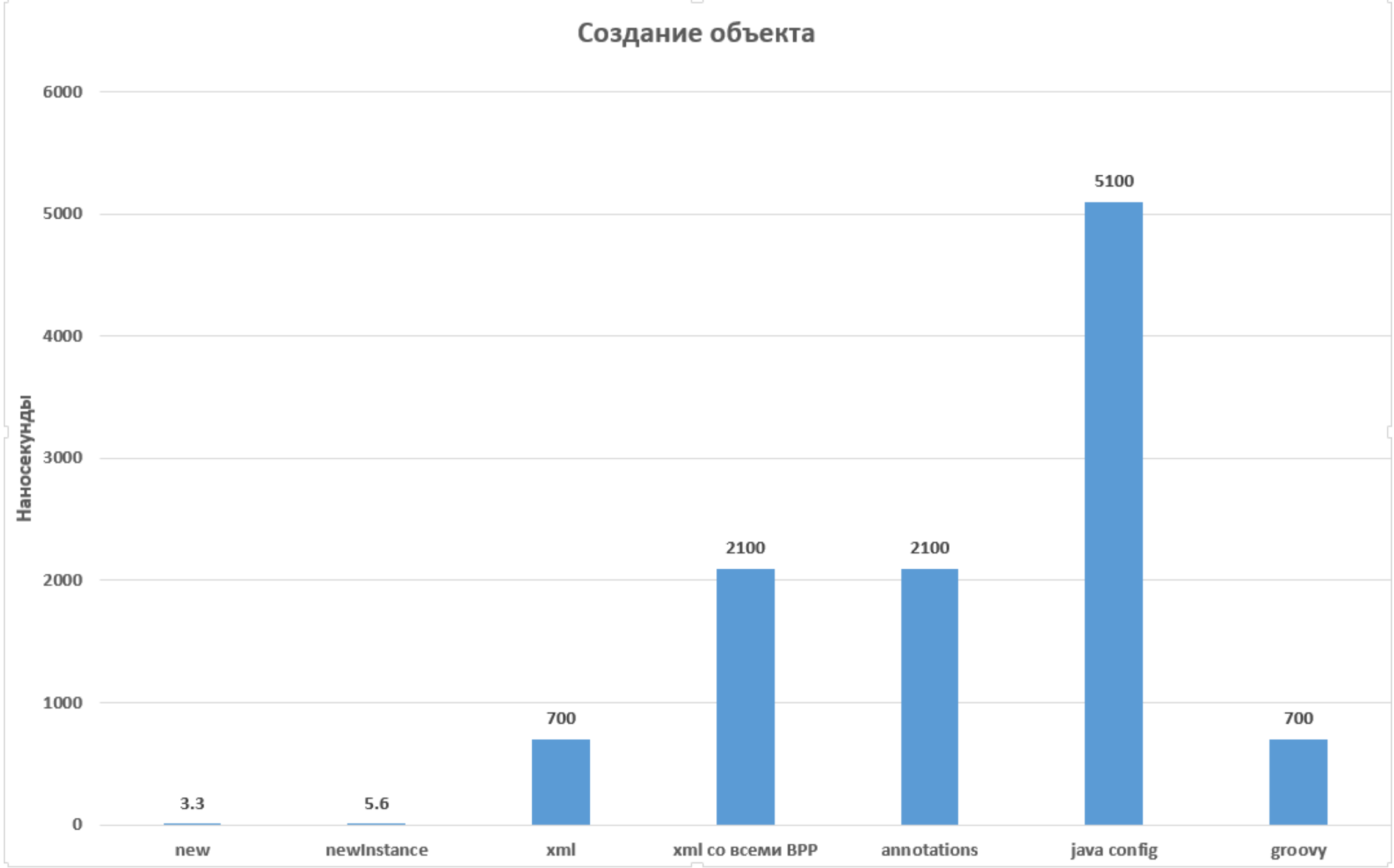
```
public static void main(String[] args) throws Exception {
    ApplicationContext context = new AnnotationConfigApplicationContext("com");
    Dao dao=null;
    long before = System.nanoTime();
    for (int i=0;i<1000000;i++) {
        dao = context.getBean(Dao.class);
    }
    long after = System.nanoTime();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

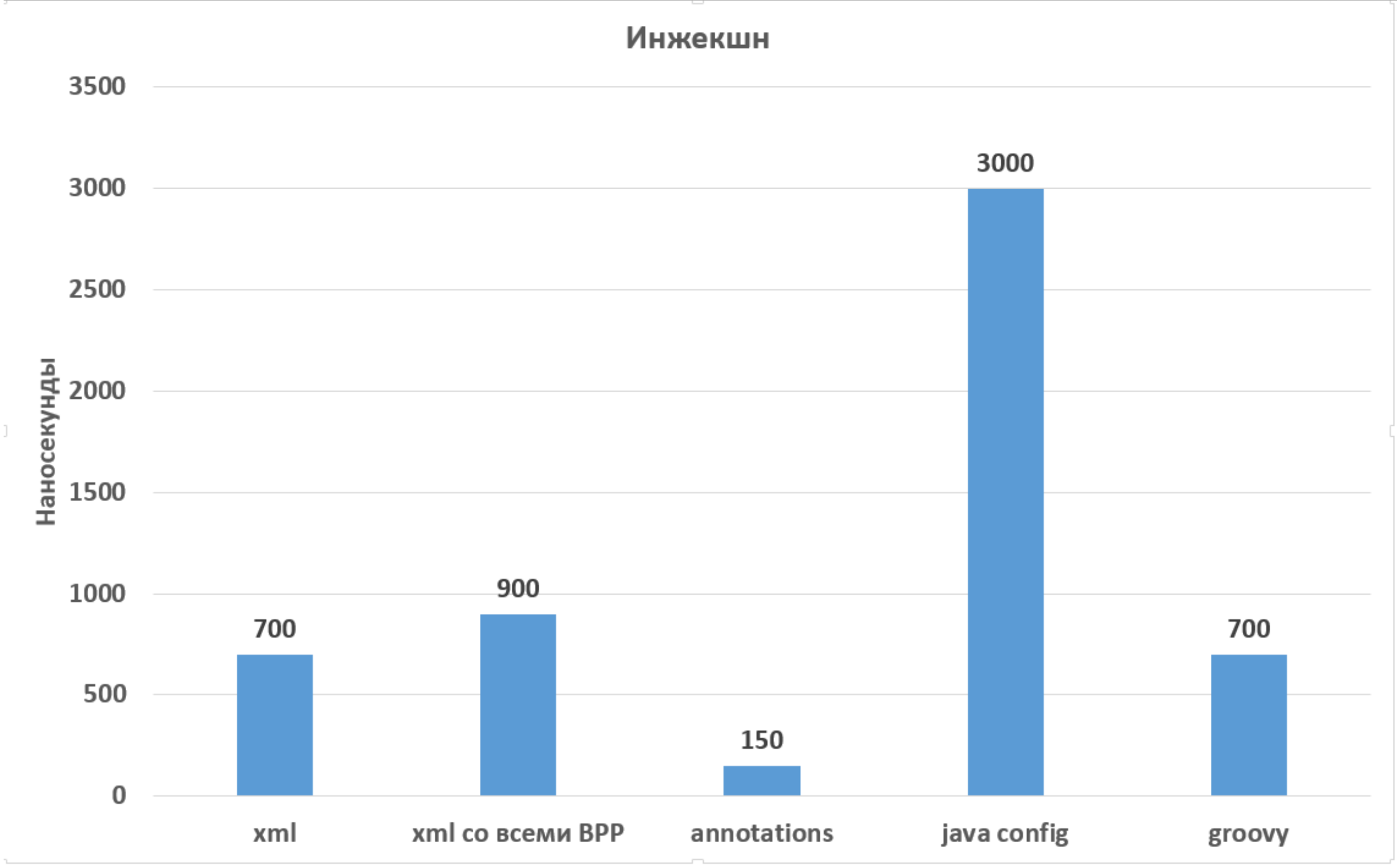
Уровень четвертый - Архитектор

Вы пьёте,
я пишу


Look up
Наносекунды
100
90
80
70
60
50
40
30
20
10
0
5.7
72
91
map.get("x")
context.getBean("x")
context.getBean(X.class)


Получение объекта
Наносекунды
80
70
60
50
40
30
20
10
0
3.3
5.6
5.7
72
new
newInstance
map.get("x")
context.getBean("x")

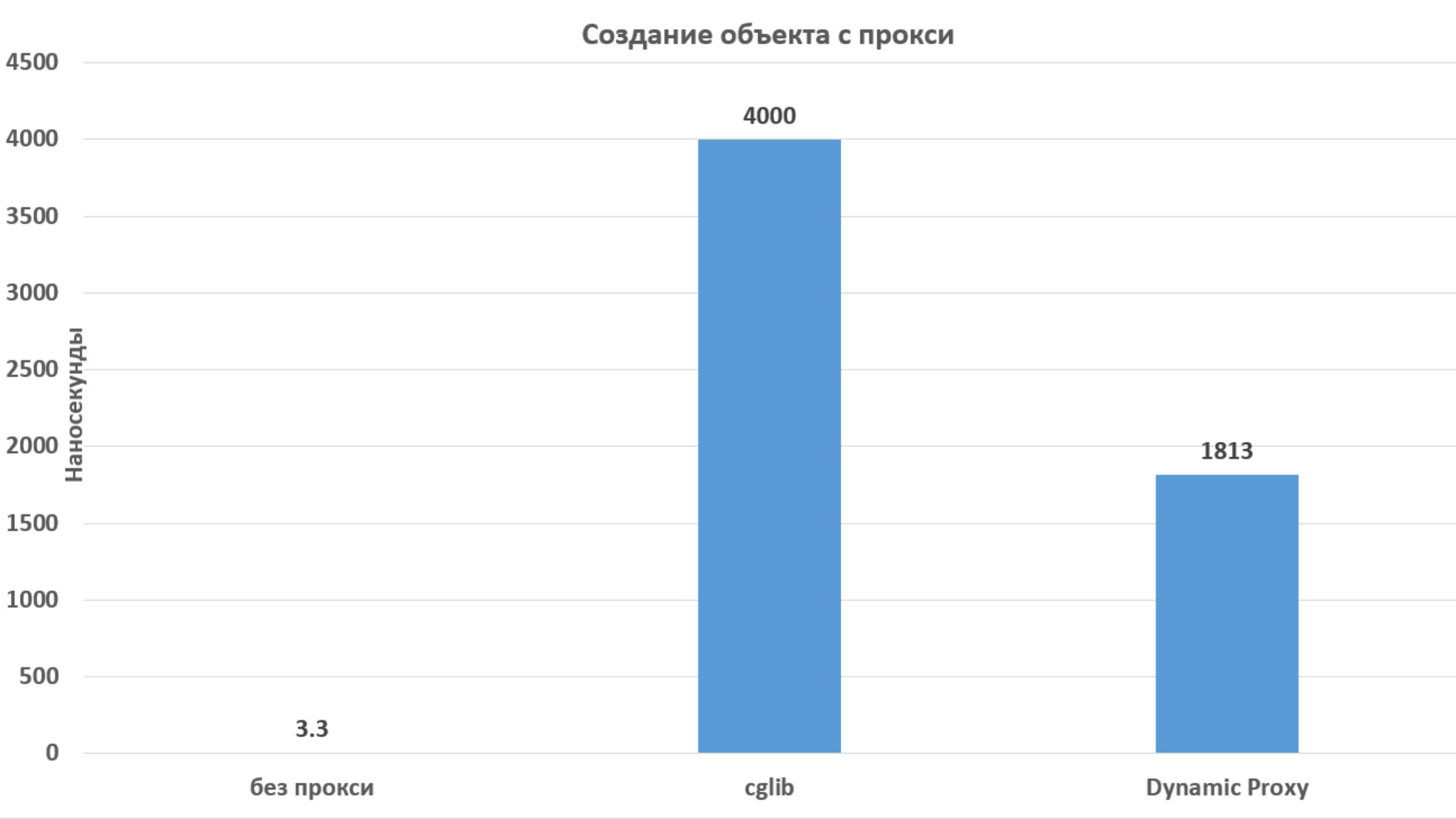

Создание объекта
Наносекунды
6000
5000
4000
3000
2000
1000
0
3.3
5.6
700
2100
2100
5100
700
new
newInstance
xml
xml со всеми BPP
annotations
java config
groovy


Инжекшн
Наносекунды
3500
3000
2500
2000
1500
1000
500
0
700
900
150
3000
700
xml
xml со всеми BPP
annotations
java config
groovy

Паника...

Всё это ещё не страшно

- Сколько времени нужно на создание миллиона прототайпов?
- 4.5 секунды
- Сколько времени нужно чтобы получить миллион сингалтонов?
- 0.1 секунды


Создание объекта с прокси
4500
4000
3500
3000
2500
2000
1500
1000
500
0
Наносекунды
4000
1813
3.3
без прокси
cglib
Dynamic Proxy


Инвокация метода через прокси
Наносекунды
300
250
200
150
100
50
0
3.3
10
10
250
без прокси
cglib
Dynamic Proxy
Через Аспект

Выводы

- Хочешь, хорошо работать – пользуйся Спрингом
- Хочешь, чтобы работало хорошо – знай его кишки