Spring Patterns

Evgeny Borisov

Director, Software Engineering

evgeny_borisov@epam.com

Ращу новое поколение

Спойлер:

- Никаких BeanPostProcessor-ов
- Только практические задачи


Home mage Singletons

We know how to make singletons

- **Enum**
- **Intelij**
- **Lazy**

enum

```
public enum Singleton {
    INSTANCE;

    public void doWork(){
        System.out.println("singleton is working...");
    }
}
```

```
public static void main(String[] args) {
    Singleton.INSTANCE.doWork();
}
```

JetBrains

```
public class Singleton {
    private static Singleton ourInstance = new Singleton();

    public static Singleton getInstance() {
        return ourInstance;
    }

    private Singleton() {
    }
}
```

Lazy singleton


```
public class Singleton {
    private static Singleton singleton;

    private Singleton() {
    }

    public static Singleton getInstance() {
        if (singleton==null) {
            singleton = new Singleton();
        }
        return singleton;
    }

    // all business methods
}
```

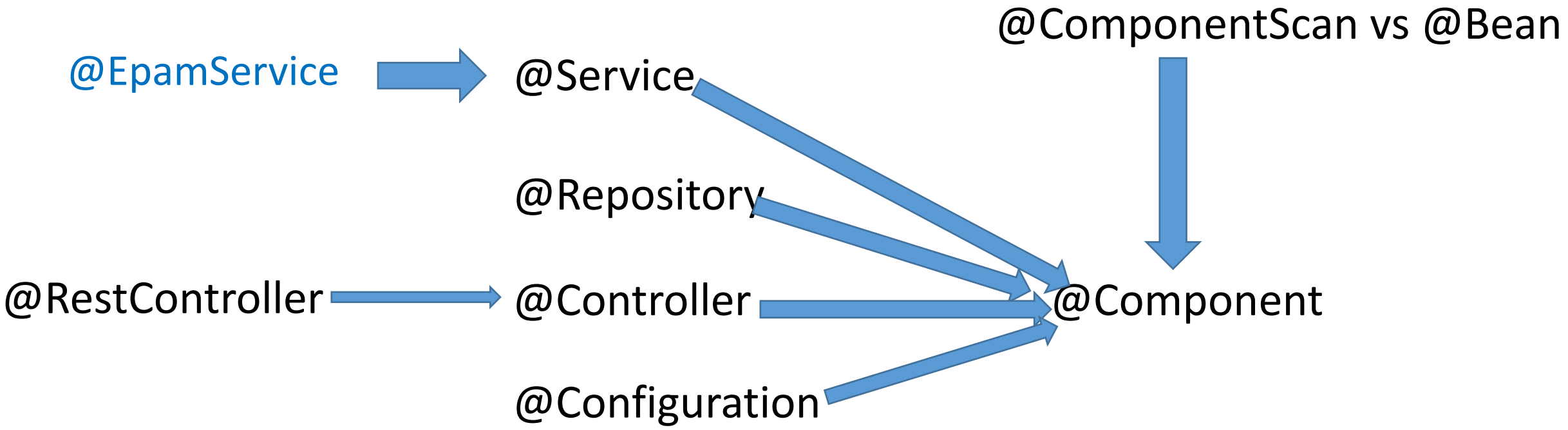
```
public class Singleton {
    private static Singleton singleton;

    private Singleton() {
    }

    public static Singleton getInstance() {
        if (singleton == null) {
            synchronized (Singleton.class) {
                if (singleton == null) {
                    singleton = new Singleton();
                }
            }
        }
        return singleton;
    }

    // all business methods
}
```

```java
public class Singleton {
    private static volatile Singleton singleton;

    private Singleton() {
    }

    public static Singleton getInstance() {
        if (singleton == null) {
            synchronized (Singleton.class) {
                if (singleton == null) {
                    singleton = new Singleton();
                }
            }
        }
        return singleton;
    }

    // all business methods
}
```

Why singleton is anti-patterns

- Makes coupling
- Provoke to write ugly code
- Can’t test without PowerMock
- Against Single Responsibility

Declaring singleton



Вывод – употребление синглтона Вредит Вашему здоровью

Пользуйтесь Spring-ом

Lazy singleton

@Lazy

@Lazy
@Component

@Lazy
@Service

@Lazy
@Bean

Before 4.3 we had a problem…

Why do we need **lazy** singleton at all by the way?

Regular mode

Cheap

Speacial mode

Expensive

I need lazy Rocket

@Lazy

It didn’t worked

Lazy injection – since 4.3

```
@Service
@Lazy
public class LazySingleton {
```

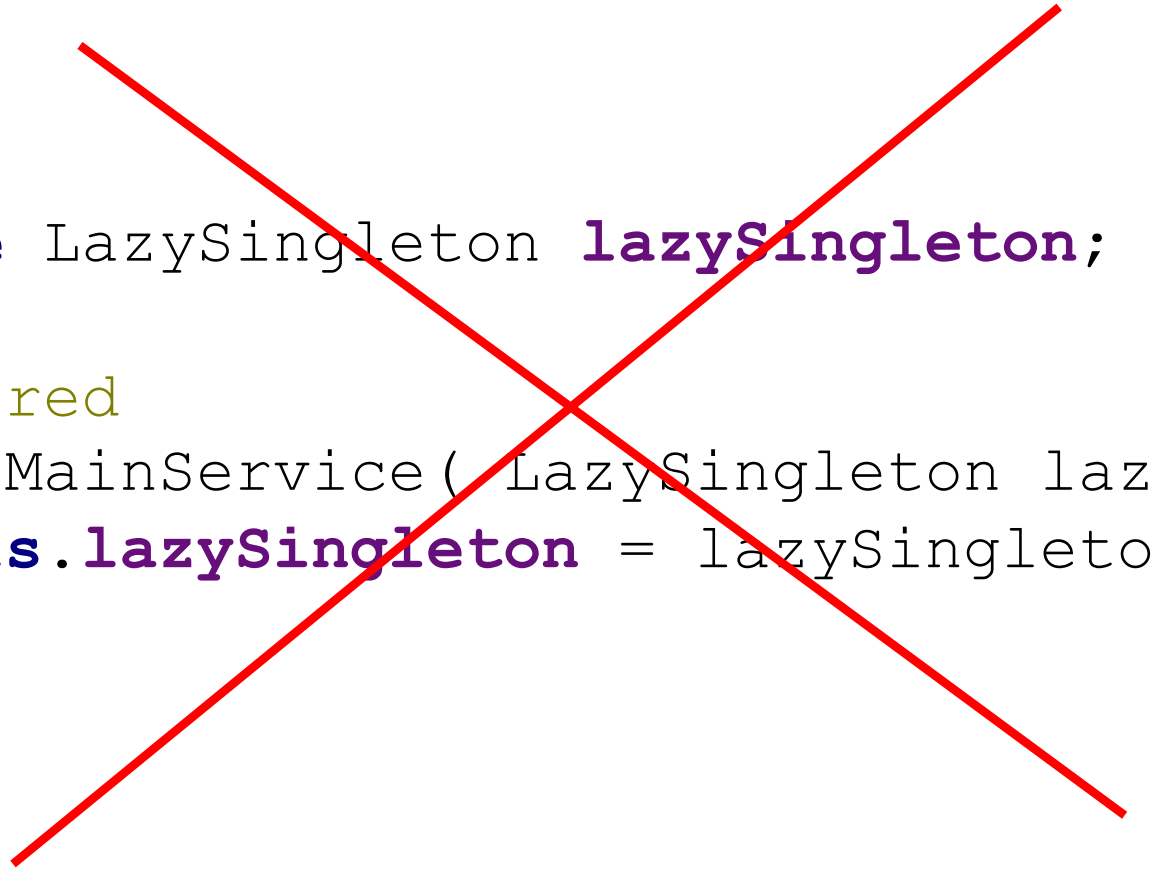
```
@Autowired
@Lazy
private LazySingleton lazySingleton;
```

```
@Autowired
public MainService(@Lazy LazySingleton lazySingleton) {
    this.lazySingleton = lazySingleton;
}
```

```
@Lazy
public MainService(LazySingleton lazySingleton) {
    this.lazySingleton = lazySingleton;
}
```

```
@RequiredArgsConstructor(onConstructor_={@Lazy})
```

Will not affect



```
@Lazy
private LazySingleton lazySingleton;

@Autowired
public MainService( LazySingleton lazySingleton) {
    this.lazySingleton = lazySingleton;
}
```

Професионаллы ни пишут тесты

Професионалы ни ошибаються

Standard situation

```
@TestConfiguration
public class MyTestConf {
    @Bean
    public ServiceIWannaTest serviceIWannaTest() {
        return new ServiceIWannaTest();
    }
}
```

Standard situation

```
@TestConfiguration
public class MyTestConf {
    @Bean
    public ServiceIWannaTest serviceIWannaTest() {
        return new ServiceIWannaTest();
    }
    @Bean
    public ServiceINeedForMyService serviceINeedForMyService() {
        return new ServiceINeedForMyService();
    }
}
```

Standard situation

```
@TestConfiguration
public class MyTestConf {
    @Bean
    public ServiceIWannaTest serviceIWannaTest() {
        return new ServiceIWannaTest();
    }
    @Bean
    public ServiceINeedForMyService serviceINeedForMyService() {
        return new ServiceINeedForMyService();
    }
    @Bean
    public Service2INeedForMyService service2INeedForMyService() {
        return new Service2INeedForMyService();
    }
}
```

```java
@TestConfiguration
public class MyTestConf {
    @Bean
    public ServiceIWannaTest serviceIWannaTest() {
        return new ServiceIWannaTest();
    }
    @Bean
    public ServiceINeedForMyService serviceINeedForMyService() {
        return new ServiceINeedForMyService();
    }
    @Bean
    public Service2INeedForMyService service2INeedForMyService() {
        return new Service2INeedForMyService();
    }
    @Bean
    public Service123НеЗнаюКомуОнНуженНоБезНегоПадает service123INeedForMyService() {
        return new Service123INeedForMyService();
    }
}
```

```
public ServiceIWannaTest serviceIWannaTest() {
    return new ServiceIWannaTest();
}
@Bean
public ServiceINeedForMyService serviceINeedForMyService() {
    return new ServiceINeedForMyService();
}
@Bean
public Service2INeedForMyService service2INeedForMyService() {
    return new Service2INeedForMyService();
}
@Bean
public Service123НеЗнаюКомуОнНуженНоБезНегоПадает service123INeedForMyService() {
    return new Service123INeedForMyService();
}
@Bean
public Service124НеЗнаюКомуОнНуженНоБезНегоПадает service123INeedForMyService() {
    return new Service123INeedForMyService();
}
@Bean
public Service125НеЗнаюКомуОнНуженНоБезНегоПадает service123INeedForMyService() {
    return new Service123INeedForMyService();
}
@Bean
public Service126НеЗнаюКомуОнНуженНоБезНегоПадает service123INeedForMyService() {
    return new Service123INeedForMyService();
}
@Bean
public Service126НеЗнаюКомуОнНуженНоБезНегоПадает service123INeedForMyService() {
    return new Service123INeedForMyService();
}
@Bean
```

Standard situation

Testing part of the system

Finally using main conf

All beans will be created, not only related to test scenario

We need to test the Predator

How can scan for all packages, and configurations without creating beans, which are not in use

To many mocks… Use lazy scanning

Since 4.1

```
@ComponentScan(lazyInit = true)
```

Spring Boot in application.properties:

```
spring.main.lazy-initialization=true
```


АРНОЛЬД
ШВАРЦЕНЕГГЕР
РЕКОМЕНДУЕТ
Constructor
injection


field injection


setter
injection


АРНОЛЬД
ШВАРЦЕНЕГГЕР
РЕКОМЕНДУЕТ
Constructor injection


field injection


METHOD INJECTION

@Primary usages…

```
@Bean
public RestTemplate restTemplate(){
    return new RestTemplate();
}

@Bean
public RestTemplate restTemplateWithZul(RestTemplateBuilder builder) {
    return builder.build();
}
```

@Primary usages…

```
@Bean
public RestTemplate restTemplate(){
    return new RestTemplate();
}

@Bean(name = "restTemplateWithZul")
public RestTemplate restTemplate(RestTemplateBuilder builder) {
    return builder.build();
}
```

```
@Autowired
public ExamComposerController(RestTemplate restTemplate) {
    this.restTemplate = restTemplate;
}
```

Not Unique Exception

@Primary usages…

```
@Bean
@Primary
public RestTemplate restTemplate(){
    return new RestTemplate();
}

@Bean
@LoadBalanced
public RestTemplate restTemplateWithZul(RestTemplateBuilder builder) {
    return builder.build();
}
```

```
@Autowired
private RestTemplate restTemplate;

@LoadBalanced
@Autowired
private RestTemplate restTemplate;
```

Primary bean injection

restTemplateWithZuul injection

```
@Qualifier
public @interface LoadBalanced {
}
```

Disclaimer

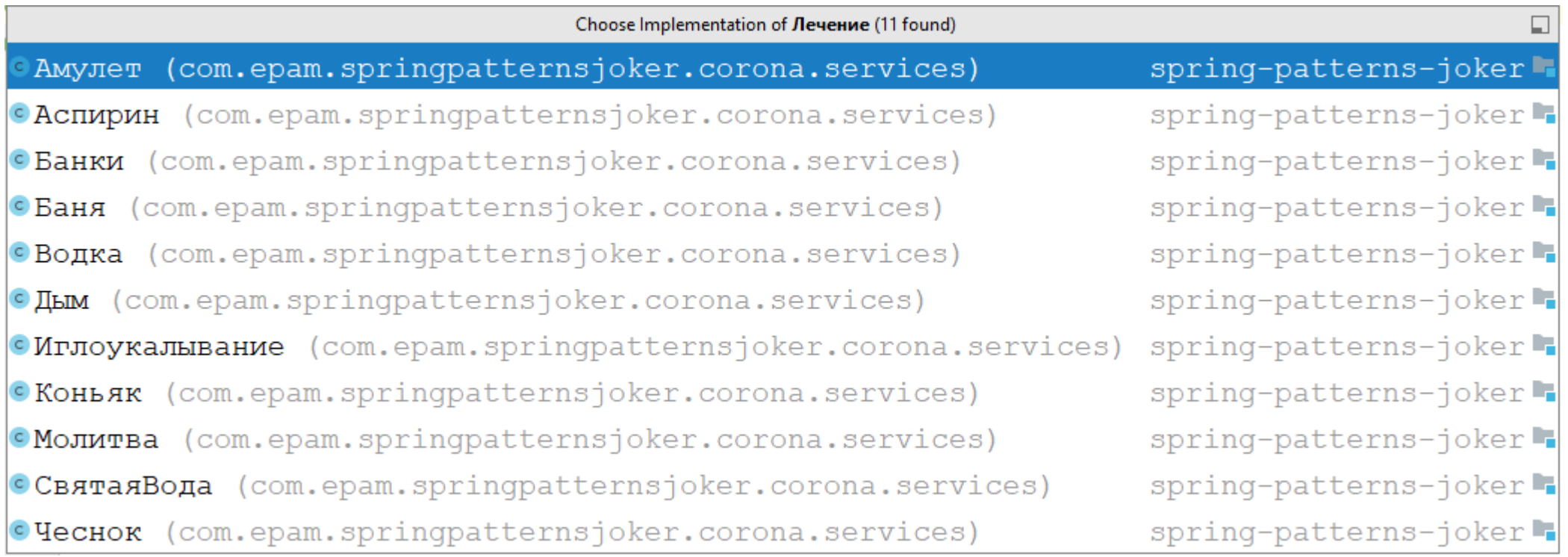
Все ситуации/персонажи выдуманы и любая схожесть с реальностью это совпадение

Будем строить центы борьбы с короной

```
public interface Целитель {
    void исцелять(Patient patient);

    String ТРАДИЦИОННАЯ = "traditional";
    String НАРОДНАЯ = "folk";
}
```

```
@Component
public class Знахарь implements Целитель {
    @Autowired
    private Лечение водка;

    @Override
    public void исцелять(Patient patient) {
        System.out.println("определяю лечение...");
        водка.применить(patient);
    }
}
```

```java
public interface Лечение {
    void применить(Patient patient);

}
```



```
public class Амулет implements Лечение {
    @Override
    public void применить(Patient patient) {
        System.out.println("Носить на шее маску. Не снимать ни в душе ни во сне");
    }
}

Чеснок implements Лечение {
e
oid применить(Patient patient) {
em.out.println("Вставить чеснок в ущи, нос и рот, крутить по часовой стрелке во время еды, до конца пандем

ic class Баня implements Лечение {
@Override
public void применить(Patient patient) {
    System.out.println("три захода в баню, по 10 минут при температуре в 3 раза превышающей температуру тел
}
```

```
public class Водка implements Лечение {
    @Override
    public void применить(Patient patient) {
        System.out.println("100 грамм водки внутрь, три раза перед каждой ед
    }
}

public class Молитва implements Лечение {
    @Override
    public void применить(Patient patient) {
        System.out.println("Прочесть три раза КОРОНЫ ИЗЫДИ перед чихнуть");
    }
```

```
    @Override
    public void применить(Patient patient) {
        System.out.println("нюхать дым, стучать в барабан до полного исцеления");
    }
```

```java
@Retention(RUNTIME)
@Component
@Qualifier
@Autowired
public @interface Treatment {
    String value();
}
```

```java
@Treatment(Лечение.АЛКОГОЛЬ)
public class Коньяк implements Лечение {
    @Override
    public void применить(Patient patient) {
        System.out.println("Дышать над коньяч
    }
}
```

```java
@Treatment(Лечение.АЛКОГОЛЬ)
public class Водка implements Лечение {
    @Override
    public void применить(Patient patient) {
        System.out.println("100 грамм водки внутр
    }
}
```

```java
@Component
public class Знахарь implements Целитель {
    @Autowired
    private Лечение водка;

    @Override
    public void исцелять(Patient patient) {
        System.out.println("определяю лечение...");
        водка.применить(patient);
    }
}
```

1) ConflictingBeanDefinitionException
2) Водка will be injected
3) Compilation error
4) NoSuchBeanDefinitionException

```java
@Retention(RUNTIME)
@Component
@Qualifier
@Autowired
public @interface Treatment {
    String value();
}
```

```java
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Indexed
public @interface Component {
    String value() default "";
}
```

```java
@Treatment(Лечение.АЛКОГОЛЬ)
public class Коньяк implements Лечение {
    @Override
    public void применить(Patient patient) {
        System.out.println("Дышать над коньяч
    }
}
```

```java
@Treatment(Лечение.АЛКОГОЛЬ)
public class Водка implements Лечение {
    @Override
    public void применить(Patient patient) {
        System.out.println("100 грамм водки внутр
    }
}
```

```java
@Component
public class Знахарь implements Целитель {
    @Autowired
    private Лечение водка;

    @Override
    public void исцелять(Patient patient) {
        System.out.println("определяю лечение...");
        водка.применить(patient);
    }
}
```

ConflictingBeanDefinitionException

2) Водка will be injected

3) Compilation error

4) NoSuchBeanDefinitionException

```java
@Retention(RUNTIME)
@Component
@Qualifier
@Autowired
public @interface Treatment {
    String value();
}
```

```java
@Retention(RUNTIME)
@Component
@Qualifier
@Autowired
public @interface Treatment {
    String type();
}
```

```java
@Retention(RUNTIME)
@Component
@Qualifier
@Autowired
public @interface Treatment {
    String type();
}
```

```java
@Treatment(type = Лечение.АЛКОГОЛЬ)
public class Водка implements Лечение {
    @Override
    public void применить(Patient patient) {
        System.out.println("100 грамм водки вн
    }
}
```

```java
@Component
public class Знахарь implements Целитель {
    @Treatment(type = Лечение.АЛКОГОЛЬ)
    private Лечение водка;

    @Override
    public void исцелять(Patient patient) {
        System.out.println("определяю лечение...");
        водка.применить(patient);
    }
}
```

```java
@Treatment(type = Лечение.АЛКОГОЛЬ)
public class Коньяк implements Лечение {
    @Override
    public void применить(Patient patient) {
        System.out.println("Дышать над коньячн
    }
}
```

1) ConflictingBeanDefinitionException
2) Водка will be injected
3) Compilation error
4) NoSuchBeanDefinitionException

```java
@Retention(RUNTIME)
@Component
@Qualifier
@Autowired
public @interface Treatment {
    String type();
}
```

```java
@Treatment(type = Лечение.АЛКОГОЛЬ)
public class Водка implements Лечение {
    @Override
    public void применить(Patient patient) {
        System.out.println("100 грамм водки вн
    }
}
```

```java
@Treatment(type = Лечение.АЛКОГОЛЬ)
public class Коньяк implements Лечение {
    @Override
    public void применить(Patient patient) {
        System.out.println("Дышать над коньячн
    }
}
```

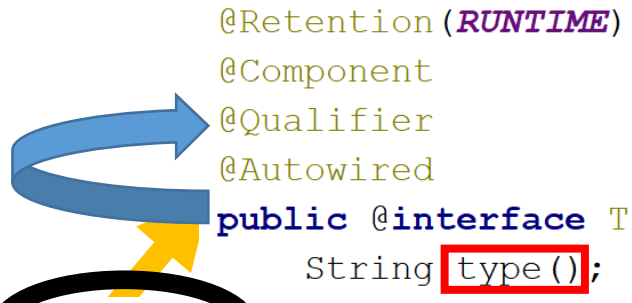
```java
@Component
public class Знахарь implements Целитель {
    @Treatment(type = Лечение.АЛКОГОЛЬ)
    private Лечение водка;

    @Override
    public void исцелять(Patient patient) {
        System.out.println("определяю лечение...");
        водка.применить(patient);
    }
}
```

1) ConflictingBeanDefinitionException

Водка will be injected

3) Compilation error

4) NoSuchBeanDefinitionException



```java
@Retention(RUNTIME)
@Component
@Qualifier
@Autowired
public @interface Treatment {
    String type();
}
```

```java
@Treatment(type = Лечение.АЛКОГОЛЬ)
public class Водка implements Лечение {
    @Override
    public void применить(Patient patient) {
        System.out.println("100 грамм водки вн
    }
}
```

```java
@Treatment(type = Лечение.АЛКОГОЛЬ)
public class Коньяк implements Лечение {
    @Override
    public void применить(Patient patient) {
        System.out.println("Дышать над коньячн
    }
}
```

```java
@Component
public class Знахарь implements Целитель {
    @Treatment(type = Лечение.АЛКОГОЛЬ)
    private Лечение водка;

    @Override
    public void исцелять(Patient patient) {
        System.out.println("определяю лечение...");
        водка.применить(patient);
    }
}
```

1) ConflictingBeanDefinitionException

2) Водка will be injected

3) Compilation error

4) NoSuchBeanDefinitionException

Chain of responsibility

```
public class MainHandler {
    public void handle(DataObject t) {
        handle1(t);
        handle2(t);
        handle3(t);
    }
```

Open Close principle broken

Chain of responsibility

```
@Service
public class MainHandler {
    @Autowired
    private List<Handler> handlers;

    public void handle(DataObject t) {
       handlers.forEach(handler -> handler.handle(t));
    }
```

Chain of responsibility

```
@Service
public class MainHandler {
    @Autowired
    private List<Handler> handlers;

    public void handle(DataObject t) {
        handlers.forEach(handler -> handler.handle(t));
    }
```

Choose Bean

handler1 (Handler1.java) spring-patterns
handler2 (Handler2.java) spring-patterns
handler3 (Handler3.java) spring-patterns

How to inject custom list?

- InjectList(Водка.class,Баня.class…)
- BeanPostProcessor ☺

Integrating spring with alternative frameworks

- Integration with legacy as example
- ImportBeanDefinitionRegistrar as a solution

More patterns

- Strategy & Command
- Never use switch

Наша больница

```java
@Service
public class HospitalImpl implements Hospital {
    @Autowired
    private Знахарь знахарь;
    @Autowired
    private Врач врач;

    @Override
    public void processPatient(Patient patient) {
        switch (patient.getMethod()) {
            case Целитель.ТРАДИЦИОННАЯ:
                врач.исцелять(patient);
                break;
            case Целитель.НАРОДНАЯ:
                знахарь.исцелять(patient);
                break;
        }
    }
}
```

```java
@Service
public class HospitalImpl implements Hospital {
    @Autowired
    private Знахарь знахарь;
    @Autowired
    private Врач врач;
    @Autowired
    private DefaultЦелитель defaultЦелитель;

    @Override
    public void processPatient(Patient patient) {
        switch (patient.getMethod()) {
            case Целитель.ТРАДИЦИОННАЯ:
                врач.исцелять(patient);
                break;
            case Целитель.НАРОДНАЯ:
                знахарь.исцелять(patient);
                break;
            default:
                defaultЦелитель.исцелять(patient);
        }
    }
}
```

```java
@Service
public class HospitalImpl implements Hospital {
    @Autowired
    private Знахарь знахарь;
    @Autowired
    private Врач врач;
    @Autowired
    private DefaultЦелитель defaultЦелитель;
    @Autowired
    private Священник священник;
    @Autowired
    private Шаман шаман;

    @Override
    public void processPatient(Patient patient) {
        switch (patient.getMethod()) {
            case Целитель.ТРАДИЦИОННАЯ:
                врач.исцелять(patient);
                break;
            case Целитель.НАРОДНАЯ:
                знахарь.исцелять(patient);
                break;
            case "магия":
                шаман.исцелять(patient);
                break;
            case "религия":
                священник.исцелять(patient);
                break;
            default:
                defaultЦелитель.исцелять(patient);
        }
    }
```

We love you, Switch …

```
public DistribHandler resol
    switch (Integer.valueOfac
switch (documentObject.getDocumen
  case PDF_STORAGE:
    fileContainer = new PdfRecordFi
    getPdfFromStorage(fileContainer,
    break;
  case PDF_SRC:
    fileContainer = new PdfRecordFi
    fillObjectsForPdf(fileContainer, d
    break;
  case PDF_WS:
    fileContainer = new WsPdfRecor
    getPdfFromPdfWs(fileContainer, 
    break;
  case LIS:
    fileContainer = new PdfRecordFi
    getPdfFromLisDocument(j, fileCc
    break;
  case IMAGE:
    fileContainer = new PdfRecordFi
    getPdfFromImageDocument(j, fil
    break;
  case FORM:
    fileContainer = new PdfRecordFi
    getPdfFromFormDocument(fileCc
    break;
}
```

```
switch (value.getNumericValue()) {
  case 1:
    text = MessageFormat.format("{0} הצעה לפוליסת", emailRequest.getSubject());
    break;
  case 2:
    text = MessageFormat.format(" {0} רבישת פוליסת", emailRequest.getSubject());
    break;
  case 3:
    text = MessageFormat.format(" {0} חידוש פוליסת", emailRequest.getSubject());
    break;
  case 4:
    text = MessageFormat.format("{0} שינויים בפוליסת", emailRequest.getSubject());
    break;
  case 5:
  case 8:
    text = MessageFormat.format("{0}מבתב מ", brandHebName);
    break;
  case 6:
  case 7:
    text = MessageFormat.format("{0}מבתב חשוב עבורך מ", brandHebName);
    break;
  case 9:
  case 17:
    text = MessageFormat.format("{0} - רבישת ביטוח נסיעות", brandHebName);
    break;
  case 10:
  case 13:
    text = MessageFormat.format("{0} - {1}", emailRequest.getSubject(), brandHebName);
    break;
  case 11:
    text = emailRequest.getSubject();
    break;
  case 12:
    if (!resources.getBrandKey().isBituhYashir()) {
      text = format("s%-תודה מ", brandHebName);
    } else {
      text = format("s%תודה מ", brandHebName);
    }
    break;
  case 14:
    text = MessageFormat.format("{0} - ביטוח תאונות אישיות - השקט הנפשי שלך", brandHebName);
    break;
  case 15:
    text = MessageFormat.format("{0} בקשה לצירת קשר לחידוש ביטוח", emailRequest.getSubject());
    break;
  case 16:
```

```
icyDatFileConsumer.class);
ll && item.getAddresseeCode().length() >= 1
de())) ? item.getAddresseeCode().trim() : "0";
dressee(addressee);
urrentPath, basket, dataConsumer);

icyDetailsConfConsumer.class);
urrentPath, basket, dataConsumer);

icyLetterConsumer.class);
lr(basket.getMetaDBasket().getPrtNr());
urrentPath, basket, dataConsumer);

icyCompulsoryConsumer.class);
etSeqNr(item.getCompSeqNr());
urrentPath, basket, dataConsumer);
```

Чем ковыряться в гавнокоде,
Пиши, как будто, ты, сеньёр

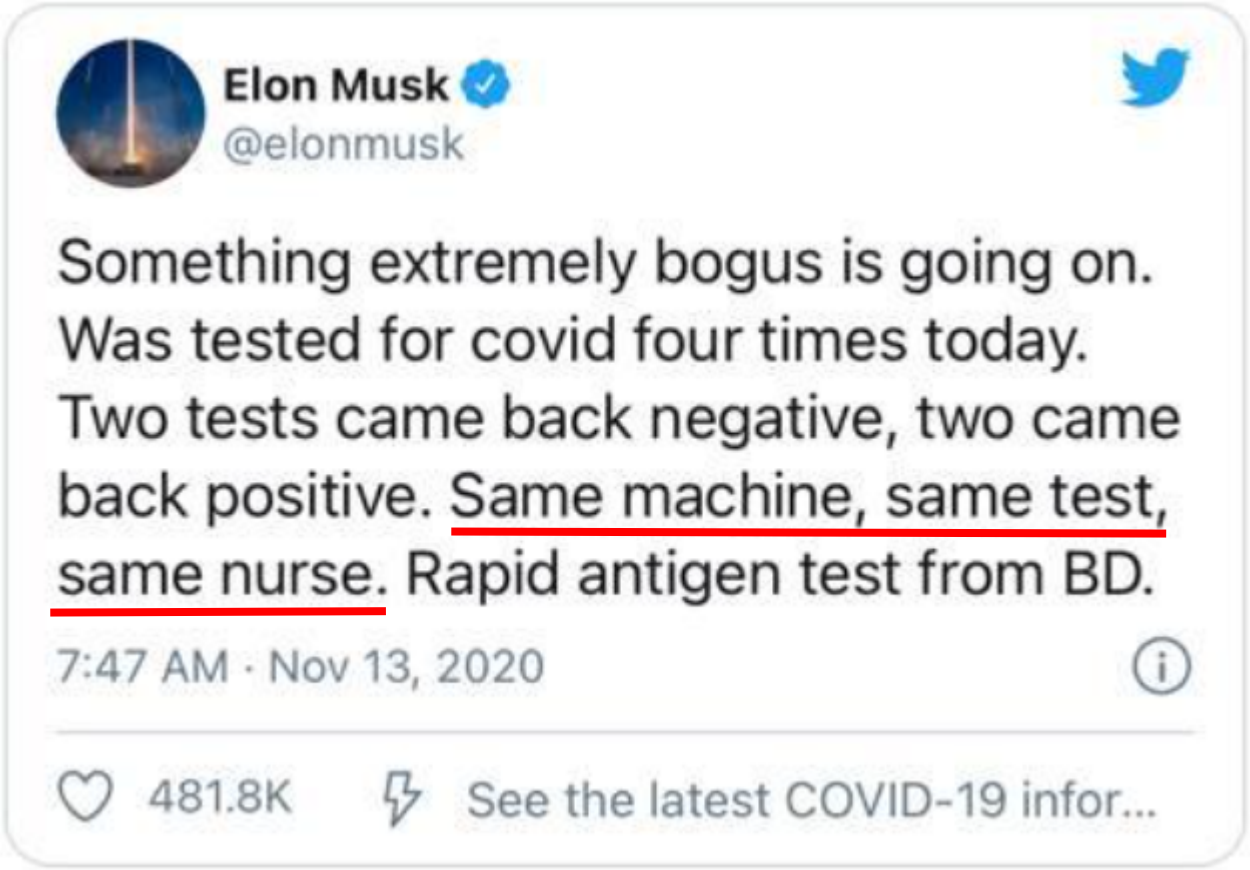
Будем делать

- Strategy
- Command
- Registry

```
interface Worker{

    @PostConstruct
    default void init(){

    }

    @Scheduled(fixedDelay = 1000)
    default void doWork(){

    }

    @EventListener(ContextRefreshedEvent.class)
    default void doOnce() {

    }

    @Autowired
    default void register(Registry registry) {
        registry.bind(this.getClass().getName(),this);
    }

}
```

Новые возможности

А допустим, что это правда



Тогда наш PSRService будет таким...

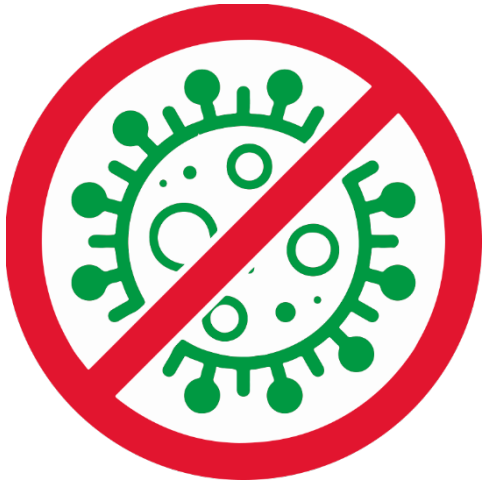
- Смотрим код...

Handling Exceptions with AOP

- Static pointcuts
- Dynamic pointcuts

А вы знали, что...

```
     * @since 1.5
     */
    public static boolean parseBoolean(String s) {
        return "true".equalsIgnoreCase(s);
    }
```



```
{
   "name": "Jeka",
   "age": 33,
   "method": "folk"
}
```

```
{
   "value": {
      "name": "Jeka",
      "age": 33,
      "method": "folk"
   },
   "corona": false
}
```

Patterns which are ~~simple~~ easy with spring

- Singleton (Lazy/Eager)
- Chain of responsibility
- Strategy
- Command
- Registry
- Observer
- Proxy

Spring techniques and mechanisms

- Ленивое сканирование
- Primary / Qualifier / Bean name
- Custom injection - BeanPostProcessor
- Declaring beans from other framework – ImportBeanDefinitionRegistrar
- Bean Registration after context was refreshed
- Dynamic pointcuts
- AOP for controllers - ControllerAdvice

Думайте об этом

- Расширение возможностей Spring - свои аннотации, свои конвенции
- Правильное использование aop
- Default methods + Spring annotations = новые возможности для интерфейсов
- Мой стартер всегде со мной

"Расставашки -
всегда пичалька".

(с) Сократик