

Spring Patterns

Evgeny Borisov

bsevgeny@gmail.com



Home mage Singletons



- Enum
- Intelij
- Lazy

We know how to make
singletons



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



я не Ленивый,
я энергосберегающий

```
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  
}
```

```
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

Use Spring



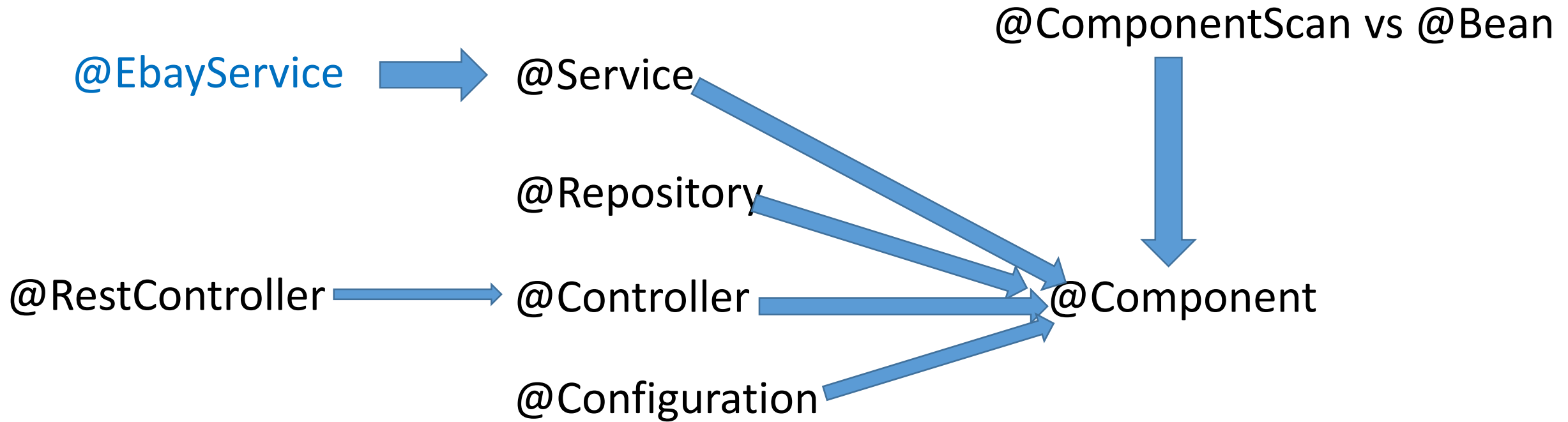
Writing singleton is antipattern



Singleton is a Whisky



Declaring singleton



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

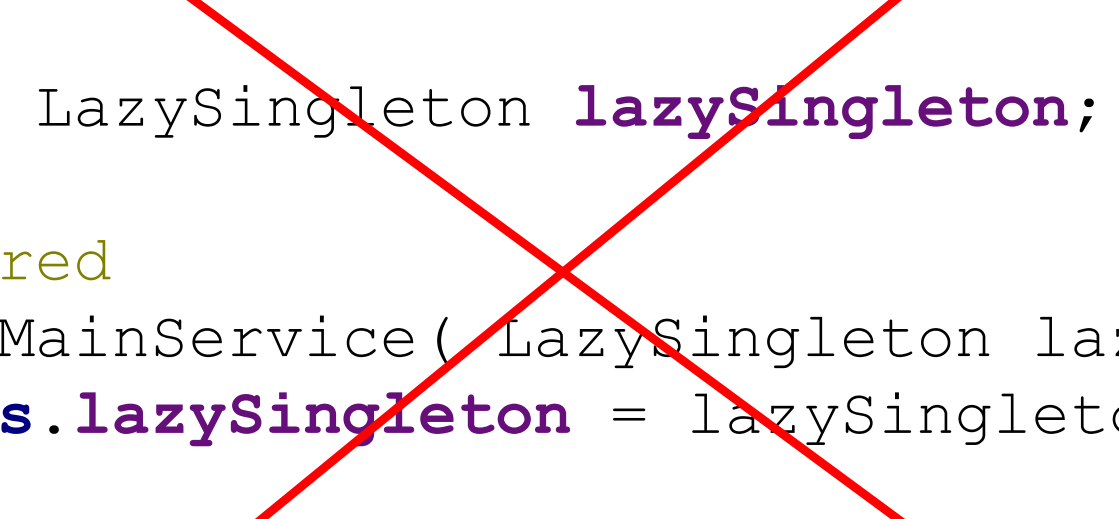
public MainService(LazySingleton lazySingleton) {

this.lazySingleton = lazySingleton;

}

@RequiredArgsConstructor(onConstructor_={@Lazy})

Will not affect



```
@Lazy
private LazySingleton lazySingleton;

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


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();
    }
}
```

@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 Service123HeЗнаюКомуОнНуженНоБезНегоПадает 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
```


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



```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = MockConf.class)
public class ConfTest {
    @MockBean
    SomeService1 someService1;
    @MockBean
    SomeService2 someService2;
    @MockBean
    SomeService3 someService3;
    @MockBean
    SomeService4 someService4;
```

If we don't need to configure mocks

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = MockConf.class)
@MockBean(SomeService1.class)
@MockBean(SomeService2.class)
@MockBean(SomeService3.class)
@MockBean(SomeService4.class)
public class ConfTest {
```

How can scan for all packages, and configurations
without creating only needed beans

To many mocks... Use lazy scanning

Since 4.1

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

@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  Primary bean injection
`private RestTemplate restTemplate;`

@LoadBalanced  restTemplateWithZuul injection
`private RestTemplate restTemplate;`

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

Chain of responsibility

- Chain of responsibility pattern
- Spring implementation
- With Streams / Optional

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

Open Close principle broken

Atbildības ķēde

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

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

Мережа відповідальності

```
@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

Strategy / Command / never use switch

Different delivery types



```
@PostMapping("/send")
public String distribute(@RequestBody Message message) {
    String type = message.getDistributionType();
    switch (type) {
        case SMS:
            return sendSms(message);
        case WHATS_APP:
            return sendWhatsApp(message);
        default:
            return "distribution code "+type+" not supported";
    }
}
```

```
@PostMapping("/send")
public String distribute(@RequestBody Message message) {
    String type = message.getDistributionType();
    switch (type) {
        case SMS:
            return sendSms(message);
        case WHATS_APP:
            return sendWhatsApp(message);
        case VIBER:
            return viberSender.send(message);
        default:
            return "distribution code "+type+" not supported";
    }
}
```

We love you, Switch ...

```
public DistribHandler resolve(Integer valueOfFac) {
    switch (Integer.valueOfFac) {
        case PDF_STORAGE:
            fileContainer = new PdfRecordFile();
            getPdfFromStorage(fileContainer, documentObject.getDocument());
            break;
        case PDF_SRC:
            fileContainer = new PdfRecordFile();
            fillObjectsForPdf(fileContainer, documentObject.getDocument());
            break;
        case PDF_WS:
            fileContainer = new WsPdfRecordFile();
            getPdfFromPdfWs(fileContainer, documentObject.getDocument());
            break;
        case LIS:
            fileContainer = new PdfRecordFile();
            getPdfFromLisDocument(j, fileContainer);
            break;
        case IMAGE:
            fileContainer = new PdfRecordFile();
            getPdfFromImageDocument(j, fileContainer);
            break;
        case FORM:
            fileContainer = new PdfRecordFile();
            getPdfFromFormDocument(fileContainer, documentObject.getDocument());
            break;
    }
}

switch (value.getNumericValue()) {
    case 1:
        text = MessageFormat.format("{0} הצעת פוליטת", emailRequest.getSubject());
        break;
    case 2:
        text = MessageFormat.format("{0} רבישת פוליטת", emailRequest.getSubject());
        break;
    case 3:
        text = MessageFormat.format("{0} חידוש פוליטת", emailRequest.getSubject());
        break;
    case 4:
        text = MessageFormat.format("{0} שינויים בפוליטת", emailRequest.getSubject());
        break;
    case 5:
        text = MessageFormat.format("{0} מכתב מ", brandHebName);
        break;
    case 6:
        text = MessageFormat.format("{0} מכתב חשוב עבור מ", brandHebName);
        break;
    case 7:
        text = MessageFormat.format("{0} רבישת ביטוח נסיעות -", brandHebName);
        break;
    case 8:
        text = MessageFormat.format("{0} - {1}", emailRequest.getSubject(), brandHebName);
        break;
    case 9:
        text = emailRequest.getSubject();
        break;
    case 10:
        text = MessageFormat.format("{0} - {1}", emailRequest.getSubject(), brandHebName);
        break;
    case 11:
        text = emailRequest.getSubject();
        break;
    case 12:
        if (!resources.getBrandKey().isBituhYashir()) {
            text = format("{0} - תודה מ-", brandHebName);
        } else {
            text = format("{0} - תודה מ-", brandHebName);
        }
        break;
    case 13:
        text = MessageFormat.format("{0} - השקט הנפשו שלך -", brandHebName);
        break;
    case 14:
        text = MessageFormat.format("{0} - השקט הנפשו שלך -", brandHebName);
        break;
    case 15:
        text = MessageFormat.format("{0} - השקט הנפשו שלך -", brandHebName);
        break;
    case 16:
        text = MessageFormat.format("{0} - השקט הנפשו שלך -", brandHebName);
        break;
}
```

```
icyDatFileConsumer.class);
if (item.getAddressCode().length() >= 1
    de()) ? item.getAddressCode().trim() : "0";
dresser(address);
currentPath, basket, dataConsumer);
```

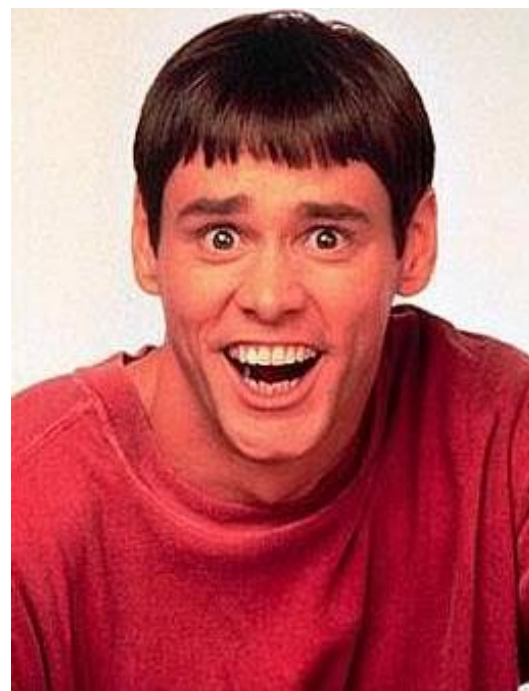
```
icyDetailsConfConsumer.class);
currentPath, basket, dataConsumer);
```

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

```
icyCompulsoryConsumer.class);
setSeqNr(item.getCompSeqNr());
currentPath, basket, dataConsumer);
```

Do you have good solutions





```
@Retention(RUNTIME)
@Qualifier
public @interface Comedy{}
```

```
@Retention(RUNTIME)
@Qualifier
public @interface Action{}
```

```
@Retention(RUNTIME)
@Qualifier
public @interface Melodrama{}
```

```
@Retention(RUNTIME)
@Qualifier
@Comedy @Action @Melodrama
public @interface AnyGenre{}
```

```
public interface Actor {
    void play();
}
```



```
@Component @Comedy
public class Jim implements Actor {
```



```
@Component @Action
public class Tobey implements Actor {
```



```
@Component @Action @Comedy
public class Stallone implements Actor {
```



```
@Component @Melodrama
public class Julia implements Actor {
```



```
@Component @AnyGenre
public class Chuck implements Actor {
```



```
@Component
public class Katy implements Actor {
```



What will be injected to list?

```
@Service
public class Film {
    @Autowired
    @Comedy
    @Action
    private List<Actor> actors;
}
```

1. Jim, Tobey
2. Jim, Tobey, Stallone, Chuck
3. Chuck, Katy
4. Chuck, Stallone
5. Some problems will happen

What about me??



```
@Component @Comedy
public class Jim implements Actor {
```



```
@Component @Action
public class Tobey implements Actor {
```



```
@Component @Action @Comedy
public class Stallone implements Actor {
```



```
@Component @Melodrama
public class Julia implements Actor {
```



```
@Component @AnyGenre
public class Chuck implements Actor {
```



```
@Component
public class Katy implements Actor {
```



What will be injected to list?

```
@Service
public class Film {
    @Autowired
    @Comedy
    @Action
    private List<Actor> actors;
}
```

1. Jim, Tobey
2. Jim, Tobey, Stallone, Chuck
3. Chuck, Katy
4. Chuck, Stallone
5. Some problems will happen

BeanDefinition registration with custom qualifier

- We have classes, from other system not managed by spring
- We need to register them as beans
- We need to have possibility to inject only them even if we have regular spring beans which implement the same interface