

Objektorientierte Programmierung

Hochschule Bochum

WS 19/20

Dr.-Ing. Darius Malysiak

V. OOP Software Engineering

Spring:



[https://de.wikipedia.org/wiki/Spring_\(Framework\)#/media/Datei:Spring_Framework_Logo_2018.svg](https://de.wikipedia.org/wiki/Spring_(Framework)#/media/Datei:Spring_Framework_Logo_2018.svg)

- Entwickelt 2002 von Rod Johnson (Buch: Expert One-On-One J2EE Design and Development).
- 17 Jahre auf dem Markt, de facto Standard für viele Java Anwendungen.
- Lizenziert unter Apache License 2.0.
- Aktuelle Version 5.2.2.
- Derzeit entwickelt durch Joint Venture Pivotal (VMWare, EMC, General Electric)
- Framework mit extremen Fokus auf **Aspect Oriented Programming (AOP)** und **Dependency Injection**.
- Spring ist aufgeteilt in Subprojekte: Spring Boot, Spring Batch, Spring AMQP, Spring Data, Spring MVC,...
- Spring nutzt Beans zum Aufbau einer Anwendung!

V. OOP Software Engineering

Spring:

- Spring Boot (Convention over Configuration Framework)
- Spring AMQP (Messaging Framework)
- Spring Batch (Batch Processing Framework)
- Spring Cloud (Webservice Framework)

Basis für Spring



[https://de.wikipedia.org/wiki/Spring_\(Framework\)#/media/Datei:Spring_Framework_Logo_2018.svg](https://de.wikipedia.org/wiki/Spring_(Framework)#/media/Datei:Spring_Framework_Logo_2018.svg)

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.2.2.RELEASE</version>
</parent>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter</artifactId>
</dependency>
```

V. OOP Software Engineering

Spring Application Context

- Objekt, welches eine Spring Applikation initialisiert:
 - Lädt Config Files (yaml).
 - Erstellt annotierte Beans (z.B. Component).
 - Erstellt definierte Beans.
 - Vollzieht Constructor Injection.
 - Vollzieht Autowiring.

Generell gilt beim Einsatz von Spring folgende Strategie:

- Definiere Beans als Bausteine.
- Nutze Convention over Configuration.
- Baue Systeme aus Beans.




V. OOP Software Engineering

1001 ways to create a Java Bean I

```
@Configuration
public class BeanFactory {
    @Bean
    //@Scope("prototype")
    @Scope(value = ConfigurableBeanFactory.SCOPE_PROTOTYPE)
    public Object objectBeanPrototype(){
        return new Object();
    }

    @Bean
    @Scope("singleton")
    @Qualifier(„objectBeanSingleton“)
    public Object objectBeanSingleton(){
        return new Object();
    }
}
```

Default Scope



- Factory Klassen erlauben Bean Erzeugung.
 - Scopes definieren die Erzeugungsart:
 - singleton (...)
 - prototype (neue Instanzen bei Aufruf)
 - request
 - session
 - application
 - websocket
- Bei Webapplikationen

V. OOP Software Engineering

1001 ways to create a Java Bean II

```
@Component  
@Getter  
@Setter  
@NoArgsConstructor  
@RequiredArgsConstructor  
public class SimpleBean{  
    @NotNull  
    private String name;  
}
```

Hierdurch wird eine Bean
,simpleBean' deklariert.

Wird benötigt!
Reihenfolge wichtig!!


- Mit ,Component' annotierte Klassen werden durch einen sogenannten ComponentScan detektiert und instanziiert.
- Ein Default Konstruktor ist hierbei nötig.
- Problem: Initialisierung der Bean Attribute!

V. OOP Software Engineering

1001 ways to create a Java Bean III

Erstellt eine Bean ,xmlBean1' und scanned das Bean Package nach Components
-> erstellt eine zweite Bean ,simpleBean'.

```
<?xml version="1.0" encoding="UTF-8"?>
```



```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd">

  <bean id="xmlBean1" class="solutions.exercise13.beans.SimpleBean">
    <property name="name" value="Woohooo Spring"/>
  </bean>


  <context:component-scan base-package="solutions.exercise13.beans" />
</beans>
```

V. OOP Software Engineering

1001 ways to create a Java Bean III

Befindet sich im ,resources' Ordner

```
public static void main(String[] args){  
    ApplicationContext context =  
        new ClassPathXmlApplicationContext("/exercise13/beans.xml");  
  
    String[] beanDefinitionNames = context.getBeanDefinitionNames();  
    Arrays.stream(beanDefinitionNames).forEach(s -> {System.out.println(s);});  
}
```



```
00:23:34.665 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean  
'org.springframework.context.annotation.internalCommonAnnotationProcessor'  
00:23:34.678 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'xmlBean1'  
00:23:34.774 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'simpleBean'  
xmlBean1  
simpleBean  
org.springframework.context.annotation.internalConfigurationAnnotationProcessor  
org.springframework.context.annotation.internalAutowiredAnnotationProcessor  
org.springframework.context.annotation.internalCommonAnnotationProcessor
```


V. OOP Software Engineering

1001 ways to create a Java Bean IV (no XML)

Klasse mit ,@Configuration'



```
@Configuration
@ComponentScan(basePackages = "solutions.exercise13.beans")
public class Main {
    public static void main(String[] args){
        ApplicationContext context = new AnnotationConfigApplicationContext(Main.class);

        String[] beanDefinitionNames = context.getBeanDefinitionNames();
        Arrays.stream(beanDefinitionNames).forEach(s -> {System.out.println(s);});

        SimpleBean bean = (SimpleBean) context.getBean("simpleBean");
        //...
    }
}
```

Klasse mit ,@Configuration'



Bean zur Main Klasse (wegen ,@Configuration')

main

simpleBean

org.springframework.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.context.annotation.internalCommonAnnotationProcessor

V. OOP Software Engineering

Auch eine Config Klasse -> Nested Configurations

1001 ways to create a Java Bean V

```
@Configuration
@ComponentScan(basePackages = {"solutions.exercise13.beans", "solutions.exercise13.configuration"})
public class Main {
    public static void main(String[] args){
        ApplicationContext context = new AnnotationConfigApplicationContext(Main.class);
        Object bean2 = context.getBean("objectBeanPrototype");
        Object bean3 = context.getBean("objectBeanPrototype");
        System.out.println("prototype: "+bean2+" <-> "+bean3);
        bean2 = context.getBean("objectBeanSingleton");
        bean3 = context.getBean("objectBeanSingleton");
        System.out.println("singleton: "+bean2+" <-> "+bean3);
    }
}
```

prototype: java.lang.Object@4e7912d8 <-> java.lang.Object@53976f5c
singleton: java.lang.Object@2bfc268b <-> java.lang.Object@2bfc268b

V. OOP Software Engineering

1001 ways to create a Java Bean VI

Hier wird eine gleichnamige Bean injected! Vorsicht: Ohne Qualifier wird nur anhand des Typens selektiert -> evtl. Fehler bei mehreren Kandidaten

```
@Component
@Getter
@Setter
@NoArgsConstructor
@RequiredArgsConstructor
public class SimpleBean2 {
    @NonNull
    private String name;

    @Autowired
    @Qualifier("objectBeanSingleton")
    private Object objectBeanSingleton;
}
```

```
ApplicationContext context = new AnnotationConfigApplicationContext(Main.class);
SimpleBean2 bean2 = (SimpleBean2) context.getBean("simpleBean2");
Object singletonBean = context.getBean("objectBeanSingleton");

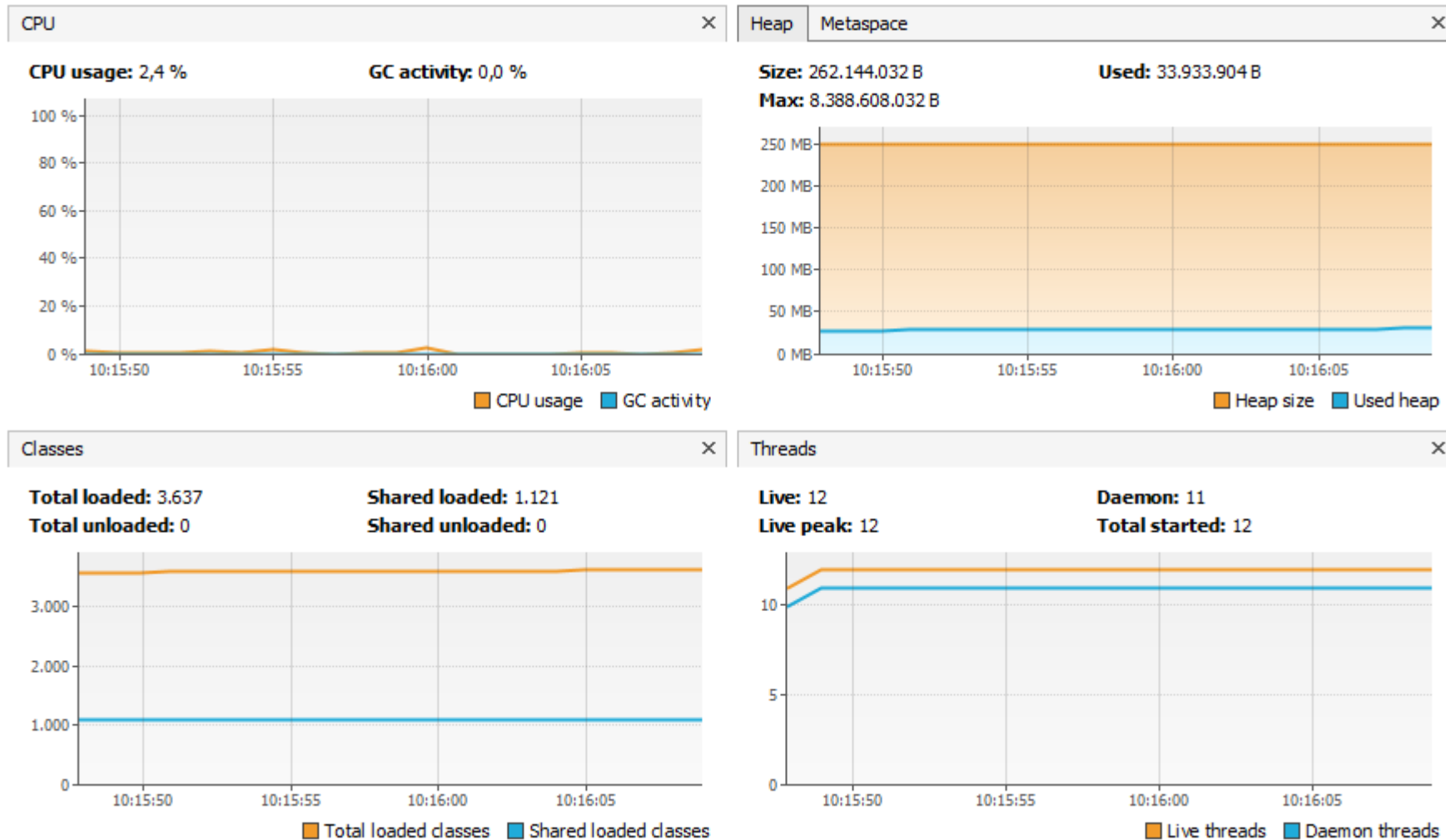
System.out.println("singleton: "+singletonBean+" <-> "+bean2.getObjectBeanSingleton());
```



singleton: java.lang.Object@5f9b2141 <-> java.lang.Object@5f9b2141

V. OOP Software Engineering

Spring Memory Footprint



V. OOP Software Engineering

Spring Boot

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
  <bean id="customerImpl" class="com.example.yamlbeanExample.Customer">
    <property name="firstName">
      <value>Rohit</value>
    </property>
    <property name="lastName">
      <value>jain</value>
    </property>
    <property name="phone">
      <value>203428304230</value>
    </property>
    <property name="address" ref="addressImpl"></property>
  </bean>
  <bean id="addressImpl" class="com.example.yamlbeanExample.Address">
    <property name="address1">
      <value>gyan nagar</value>
    </property>
    <property name="address2">
      <value>sector 4</value>
    </property>
    <property name="pinCode">
      <value>313001</value>
    </property>
    <property name="city">
      <value>Pune</value>
    </property>
    <property name="state">
      <value>Maharastra</value>
    </property>
    <property name="country" ref="countryImpl"></property>
  </bean>
  <bean id="countryImpl" class="com.example.yamlbeanExample.Country">
    <property name="countryName">
      <value>India</value>
    </property>
    <property name="countryCode">
      <value>+91</value>
    </property>
    <property name="currency">
      <value>INR</value>
    </property>
  </bean>
</beans>
```

XML



YAML

```
customerImpl:
  class: com.example.yamlbeanExample.Customer
  properties:
    firstName: Rohit
    lastName: jain
    phone: 203428304230
    address: ref::addressImpl
addressImpl:
  class: com.example.yamlbeanExample.Address
  properties:
    address1: gyan nagar
    address2: sector 4
    pinCode: 313001
    city: pune
    state: Maharastra
    country: ref::countryImpl
countryImpl:
  class: com.example.yamlbeanExample.Country
  properties:
    countryName: India
    countryCode: 91
    currency: INR
```

V. OOP Software Engineering

Spring Boot

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-configuration-processor</artifactId>
  <optional>true</optional>
</dependency>
```

Bean, welche die Einstellungen aus der YML Datei nutzt

```
@Configuration
@EnableConfigurationProperties
@ConfigurationProperties("simple-bean-3")
@Getter
@Setter
public class SimpleBean3 {
    private String name;
}
```

,application.yml' im resources Ordner

```
simple-bean-3:
  name: "justABean3"
```


V. OOP Software Engineering

Spring Boot

Was passiert hier?

```
@ComponentScan(basePackages = {"solutions.exercise13.beans", "solutions.exercise13.configuration"})
@SpringBootApplication
public class MainBoot{
    public static void main(String[] args){
        SpringApplication.run(MainBoot.class, args);
    }

    @Bean
    public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
        return args -> {
            SimpleBean3 bean = (SimpleBean3) ctx.getBean("simpleBean3");
            System.out.println(bean.getName());
        };
    }
}
```



justABean3

V. OOP Software Engineering

Spring Boot

Allgemein können alle existierenden Beans über die Datei ,application.yml' konfiguriert werden!

```
spring:
  profiles: test
name: test-YAML
environment: test
servers:
  - www.abc.test.com
  - www.xyz.test.com
```

← Spring Beans

← Custom Beans

Werte können explizit zugewiesen werden (d.h. Trennung von YAML Parameter und Bean Klasse)

```
@Configuration
@Getter
@Setter
public class SimpleBean4 {
    @Value("${myConfig.Bean4.name:a default value}")
    private String name;
}
```

```
simple-bean-3:
  name: "justABean3"
myConfig:
  Bean4:
    name: "just a name"
```

Ohne Spring Boot:
a default value

Mit Spring Boot:
just a name

V. OOP Software Engineering

Spring Boot

```
"C:\Program Files\JetBrains\IntelliJ IDEA 2019.2\jbr\bin\java.exe" ...
```

```
  ____
 /  _ \   / \   / \   / \   / \   / \   / \   / \   / \
(  )_/   /  \ /  \ /  \ /  \ /  \ /  \ /  \ /  \ /  \
 \  __/   /  \ /  \ /  \ /  \ /  \ /  \ /  \ /  \ /  \
  \___/   \___/   \___/   \___/   \___/   \___/   \___/
:: Spring Boot ::      (v2.2.2.RELEASE)
```

```
2019-12-16 10:01:51.446 INFO 4456 --- [main] solutions.exercise13.MainBoot
2019-12-16 10:01:51.449 INFO 4456 --- [main] solutions.exercise13.MainBoot
2019-12-16 10:01:52.226 INFO 4456 --- [main] solutions.exercise13.MainBoot
```

```
Let's inspect the beans provided by Spring Boot:
applicationTaskExecutor
beanFactory
commandLineRunner
mainBoot
mbeanExporter
```

Angabe der aktiven Profile

Angabe der Main-Klasse, Startzeit

```
: Starting MainBoot on DESKTOP-S0N4GIB with
: No active profile set, falling back to default
: Started MainBoot in 1.341 seconds (JVM run
```

Start-Output einer Spring Boot Anwendung

V. OOP Software Engineering

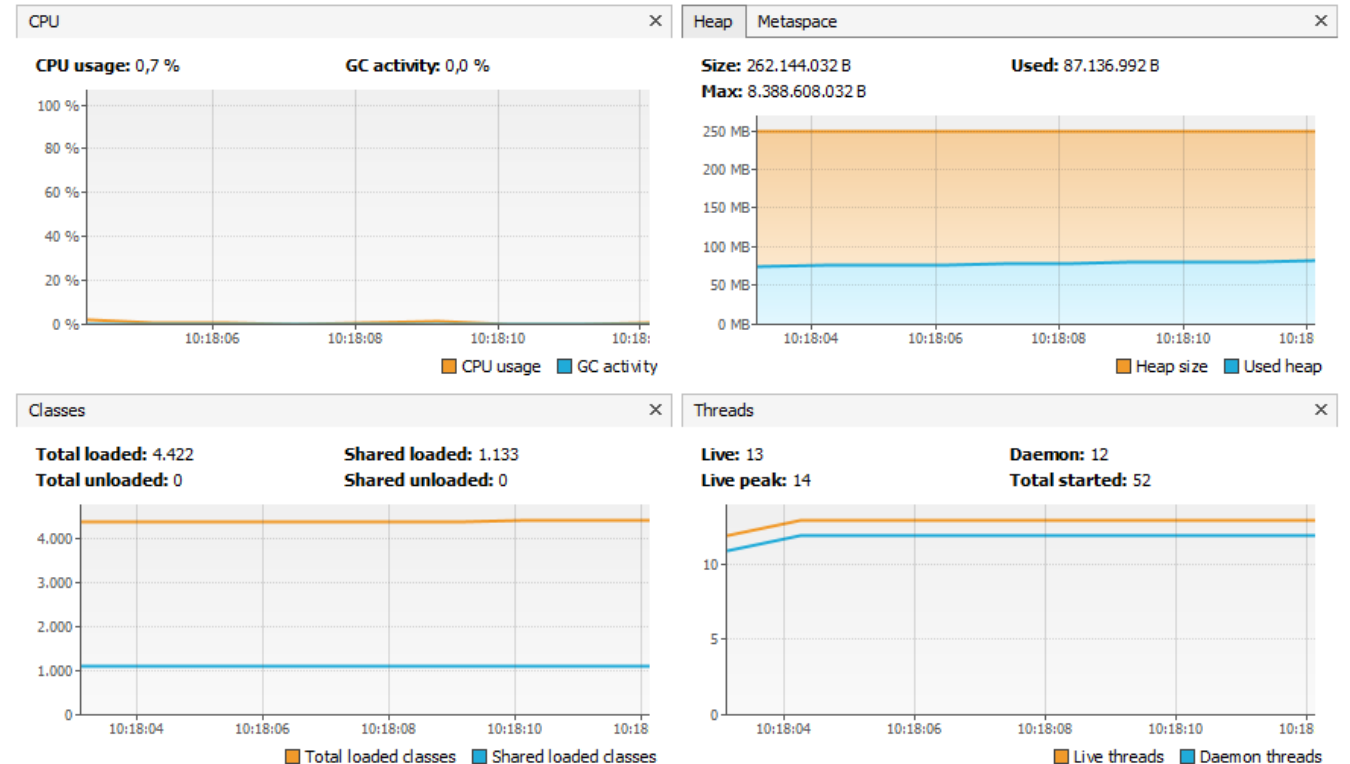
Spring Boot

```
applicationTaskExecutor
beanFactory
commandLineRunner
mainBoot
mbeanExporter
mbeanServer
objectBeanPrototype
objectBeanSingleton
objectNamingStrategy
org.springframework.aop.config.internalAutoProxyCreator
org.springframework.boot.autoconfigure.AutoConfigurationPackages
org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration$ClassProxyingConfiguration
org.springframework.boot.autoconfigure.context.ConfigurationPropertiesAutoConfiguration
org.springframework.boot.autoconfigure.context.PropertyPlaceholderAutoConfiguration
org.springframework.boot.autoconfigure.info.ProjectInfoAutoConfiguration
org.springframework.boot.autoconfigure.internalCachingMetadataReaderFactory
org.springframework.boot.autoconfigure.jmx.JmxAutoConfiguration
org.springframework.boot.autoconfigure.task.TaskExecutionAutoConfiguration
org.springframework.boot.autoconfigure.task.TaskSchedulingAutoConfiguration
org.springframework.boot.autoconfigure.transaction.jta.JtaAutoConfiguration
org.springframework.boot.context.internalConfigurationPropertiesBinder
org.springframework.boot.context.internalConfigurationPropertiesBinderFactory
org.springframework.boot.context.properties.ConfigurationBeanFactoryMetadata
org.springframework.boot.context.properties.ConfigurationPropertiesBeanDefinitionValidator
org.springframework.boot.context.properties.ConfigurationPropertiesBindingPostProcessor
org.springframework.boot.context.annotation.internalAutowiredAnnotationProcessor
org.springframework.boot.context.annotation.internalCommonAnnotationProcessor
org.springframework.boot.context.annotation.internalConfigurationAnnotationProcessor
org.springframework.context.event.internalEventListenerFactory
org.springframework.context.event.internalEventListenerProcessor
propertySourcesPlaceholderConfigurer
simpleBean
simpleBean2
simpleBean3
spring.info-org.springframework.boot.autoconfigure.info.ProjectInfoProperties
spring.task.execution-org.springframework.boot.autoconfigure.task.TaskExecutionProperties
spring.task.scheduling-org.springframework.boot.autoconfigure.task.TaskSchedulingProperties
springApplicationAdminRegistrar
taskExecutorBuilder
taskSchedulerBuilder
justABean3
```

Process finished with exit code 0

Spring-Beans des Minimalbeispiels:

- Anzahl der Beans beeinflusst die Startzeit direkt.
- Jede Bean als Objekt erhöht den Memory Footprint.



V. OOP Software Engineering

Spring Boot - Profiles

```
@Bean
@Profile("default")
public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
    return args -> {
        SimpleBean4 bean = (SimpleBean4) ctx.getBean("simpleBean4");
        System.out.println("bean4: "+bean.getName());
    };
}

@Bean
@Profile("lecture")
public CommandLineRunner commandLineRunner2(ApplicationContext ctx) {
    return args -> {
        SimpleBean4 bean = (SimpleBean4) ctx.getBean("simpleBean4");
        System.out.println("bean4: "+bean.getName());
    };
}
```

application.yml

```
simple-bean-3:
  name: "justABean3"
myConfig:
  Bean4:
    name: "just a name"

spring:
  profiles:
    active: "lecture"
    #active: "default"
  main:
    banner-mode: "off"
```

Profile „default“

bean4: just a name

application-lecture.yml

```
simple-bean-3:
  name: "justABean3 :-)"
myConfig:
  Bean4:
    name: "i am groot"
```

Profile „lecture“

bean4: i am groot

- Application.yml wird vor dem Laden der Main Methode ausgewertet!
- Je nach Profil werden unterschiedliche Beans für dieses Profil erstellt.
- Für jedes Profil kann eine zusätzliche Application.yml erstellt werden, für den Namen gilt:
application-PROFILNAME.yml

V. OOP Software Engineering

Spring Boot – Application Server

Application Server allgemein:

- Hat allgemein nichts mit Spring etc. zu tun.
- Erlaubt das serverseitige Starten sowie Ausführen einer Anwendung.
- Stellt der Anwendung Funktionen zur Verfügung:
 - HTTP Schnittstellen / HTTP Funktionen
 - Datenbank Adapter
 - Präprozessor Dienste (JSP)
- Ermöglicht die Skalierung der Anwendung (z.B. parallel laufende HTTP Schnittstellen)
- Erlaubt das Monitoring der Anwendung.
- Verbessert bzw. erleichtert das Deployment der Anwendung.

Funktionaler Umfang



https://upload.wikimedia.org/wikipedia/commons/thumb/9/95/JBoss_logo.svg/214px-JBoss_logo.svg.png



Spring
Standard

<https://upload.wikimedia.org/wikipedia/commons/thumb/7/7b/Tomcat-logo.svg/80px-Tomcat-logo.svg.png>



https://upload.wikimedia.org/wikipedia/commons/thumb/4/4c/Jetty_Logo.png/150px-Jetty_Logo.png

Memory Footprint
Startup Time

V. OOP Software Engineering

Spring Boot – Application Server

```
@Controller
@RequestMapping("/beans")
public class WebController {

    @Autowired
    private SimpleBean bean;

    @GetMapping(path="/{id}", produces = "application/json")
    public @ResponseBody
    SimpleBean getBook(@PathVariable int id,
    @RequestParam(name="name", required=false, defaultValue="World")
    String name) {
        bean.setName(name+id);
        return bean;
    }
}
```

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-tomcat</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```



Einfügen der Web Komponenten, ersetzen von Tomcat durch Jetty.

V. OOP Software Engineering

Spring Boot – Application Server

```
@ComponentScan(basePackages =  
{"solutions.exercise13.beans", "solutions.exercise13.configuration", "solutions.exercise13.web"})  
@SpringBootApplication  
@EnableWebMvc  
public class MainBootWeb {  
    public static void main(String[] args){  
        SpringApplication.run(MainBootWeb.class, args);  
    }  
}
```

V. OOP Software Engineering

Spring Boot – Application Server

```
@Controller
@RequestMapping(„/beans“)
public class WebController {

    @Autowired
    private SimpleBean bean;

    @GetMapping(path="/{id}", produces = "application/json")
    public @ResponseBody
    SimpleBean getBook(@PathVariable int id, @RequestParam(name="name", required=false, defaultValue="World") String name) {
        bean.setName(name+id);
        return bean;
    }
}
```

<http://localhost:8080/beans/5?name=test>



{"name": "test5"}

V. OOP Software Engineering

Nächste Vorlesung:

Praxisbeispiel:

AWS

Java Webservice mit Spring

Build Pipeline

Docker Deployment



Nicht klausurrelevant 😊