

# Einführung in das Spring-Framework



## **Spring Framework Reference Documentation**

- Voraussetzungen:
  - Java Basics
- Methode:
  - Präsentation, Diskussion, Übungen (> 30%)
- Werkzeuge:
  - Java 8 oder größer
  - Spring 5.x
  - STS (Eclipse- oder IntelliJ-basiert)
  - Apache Maven
  - JUnit
- Dauer:
  - 8 Einheiten je 90'

© Javacream

Javacream

Dr. Rainer Sawitzki

Alois-Gilg-Weg 6

81373 München

**Alle Rechte, einschließlich derjenigen des auszugsweisen Abdrucks, der fotomechanischen und elektronischen Wiedergabe vorbehalten.**

Spring Basics	6
Context und Dependency Injection	18
Test und Anwendungsstart	31
Weitere Features	38

1

# SPRING BASICS

## 1.1

# SETUP

- Spring ist ein sehr mächtiges Framework
  - mit sehr vielen Abhängigkeiten zu weiteren Bibliotheken der Community
- Spring Boot stellt vordefinierte Maven POMs zur Verfügung
  - Parent POM
  - "Starter" POMs definieren einen kompletten Technologie-Stack
    - Web applications
    - JPA
    - ...

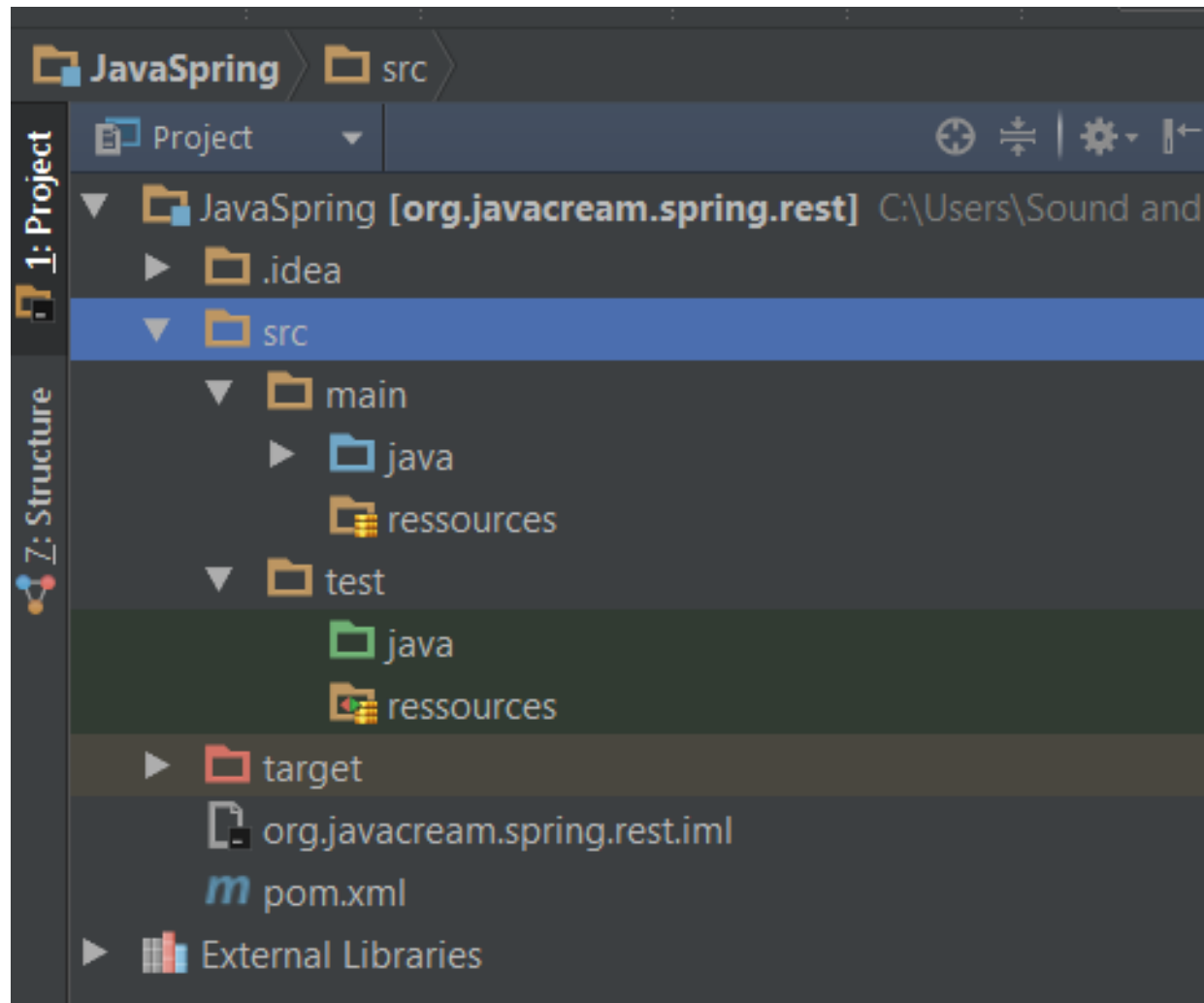


# Beispiel: Ein Auszug einer Spring-POM

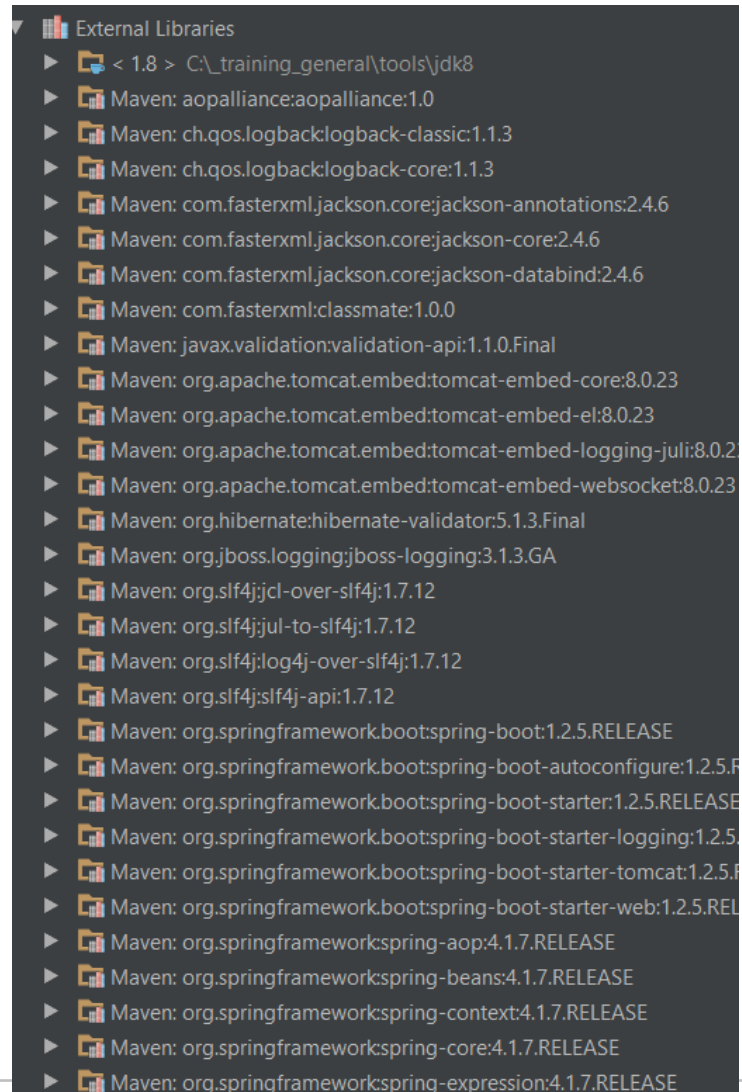
```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.javacream.training</groupId>
  <artifactId>org.javacream.training.spring.core.boot</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.3.RELEASE</version>
    <relativePath />
  </parent>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
    </dependency>
  </dependencies>
</project>
```

1.2

## **IDE UNTERSTÜTZUNG**



























# Maven-Dependencies in IntelliJ



- demo [boot]
  - src/main/java
    - org.javacream.training.webservices.jaxrs.spring
  - src/main/resources
  - src/test/java
  - JRE System Library [JavaSE-1.8]
  - Maven Dependencies
  - src
  - target
    - mvnw
    - mvnw.cmd
    - pom.xml

## ▼ Maven Dependencies

- ▶  spring-boot-starter-data-jpa-2.0.6.RELEASE.jar - /home/rainer/.m2/repos
- ▶  spring-boot-starter-2.0.6.RELEASE.jar - /home/rainer/.m2/repository/org
- ▶  spring-boot-2.0.6.RELEASE.jar - /home/rainer/.m2/repository/org/spring
- ▶  spring-boot-autoconfigure-2.0.6.RELEASE.jar - /home/rainer/.m2/reposit
- ▶  spring-boot-starter-logging-2.0.6.RELEASE.jar - /home/rainer/.m2/reposit
- ▶  logback-classic-1.2.3.jar - /home/rainer/.m2/repository/ch/qos/logback/l
- ▶  logback-core-1.2.3.jar - /home/rainer/.m2/repository/ch/qos/logback/lo
- ▶  log4j-to-slf4j-2.10.0.jar - /home/rainer/.m2/repository/org/apache/logg
- ▶  log4j-api-2.10.0.jar - /home/rainer/.m2/repository/org/apache/logging/lc
- ▶  jul-to-slf4j-1.7.25.jar - /home/rainer/.m2/repository/org/slf4j/jul-to-slf4j/
- ▶  javax.annotation-api-1.3.2.jar - /home/rainer/.m2/repository/javax/annol
- ▶  snakeyaml-1.19.jar - /home/rainer/.m2/repository/org/yaml/snakeyaml/
- ▶  spring-boot-starter-aop-2.0.6.RELEASE.jar - /home/rainer/.m2/repository
- ▶  spring-aop-5.0.10.RELEASE.jar - /home/rainer/.m2/repository/org/spring
- ▶  aspectjweaver-1.8.13.jar - /home/rainer/.m2/repository/org/aspectj/asp
- ▶  spring-boot-starter-jdbc-2.0.6.RELEASE.jar - /home/rainer/.m2/repositor
- ▶  HikariCP-2.7.9.jar - /home/rainer/.m2/repository/com/zaxxer/HikariCP/2
- ▶  spring-jdbc-5.0.10.RELEASE.jar - /home/rainer/.m2/repository/org/spring
- ▶  javax.transaction-api-1.2.jar - /home/rainer/.m2/repository/javax/transa
- ▶  hibernate-core-5.2.17.Final.jar - /home/rainer/.m2/repository/org/hibern
- ▶  jboss-logging-3.3.2.Final.jar - /home/rainer/.m2/repository/org/jboss/log
- ▶  hibernate-jpa-2.1-api-1.0.2.Final.jar - /home/rainer/.m2/repository/org/hi
- ▶  javassist-3.22.0-GA.jar - /home/rainer/.m2/repository/org/javassist/javas
- ▶  antlr-2.7.7.jar - /home/rainer/.m2/repository/antlr/antlr/2.7.7

1.3

## **EIN ERSTES BEISPIEL**

# Eine Spring Boot Rest Application

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class RestApplication {

    @RequestMapping("/hello")
    public String doHello(){
        return "Hello!";
    }

    @RequestMapping("/exit")
    public void doExit(){
        System.exit(0);
    }

    public static void main(String[] args){
        SpringApplication.run(RestApplication.class, args);
    }
}
```



- Einfach die Starter-Klasse aufrufen
  - es ist tatsächlich so einfach
- Öffnen eines Browsers mit
  - <http://localhost:8080/hello>



Hello!

## 2

# CONTEXT UND DEPENDENCY INJECTION

2.1

## ÜBERBLICK

- Der Context ist verantwortlich dafür, "relevante" Objekte zu erzeugen und zu verwalten
  - Die Lebensdauer ist abhängig vom angegebenen "Scope"
    - singleton/application
    - prototype/request
    - session
    - ...
  - In erster Näherung ist der Context eine (ziemlich) smarte Map
  -

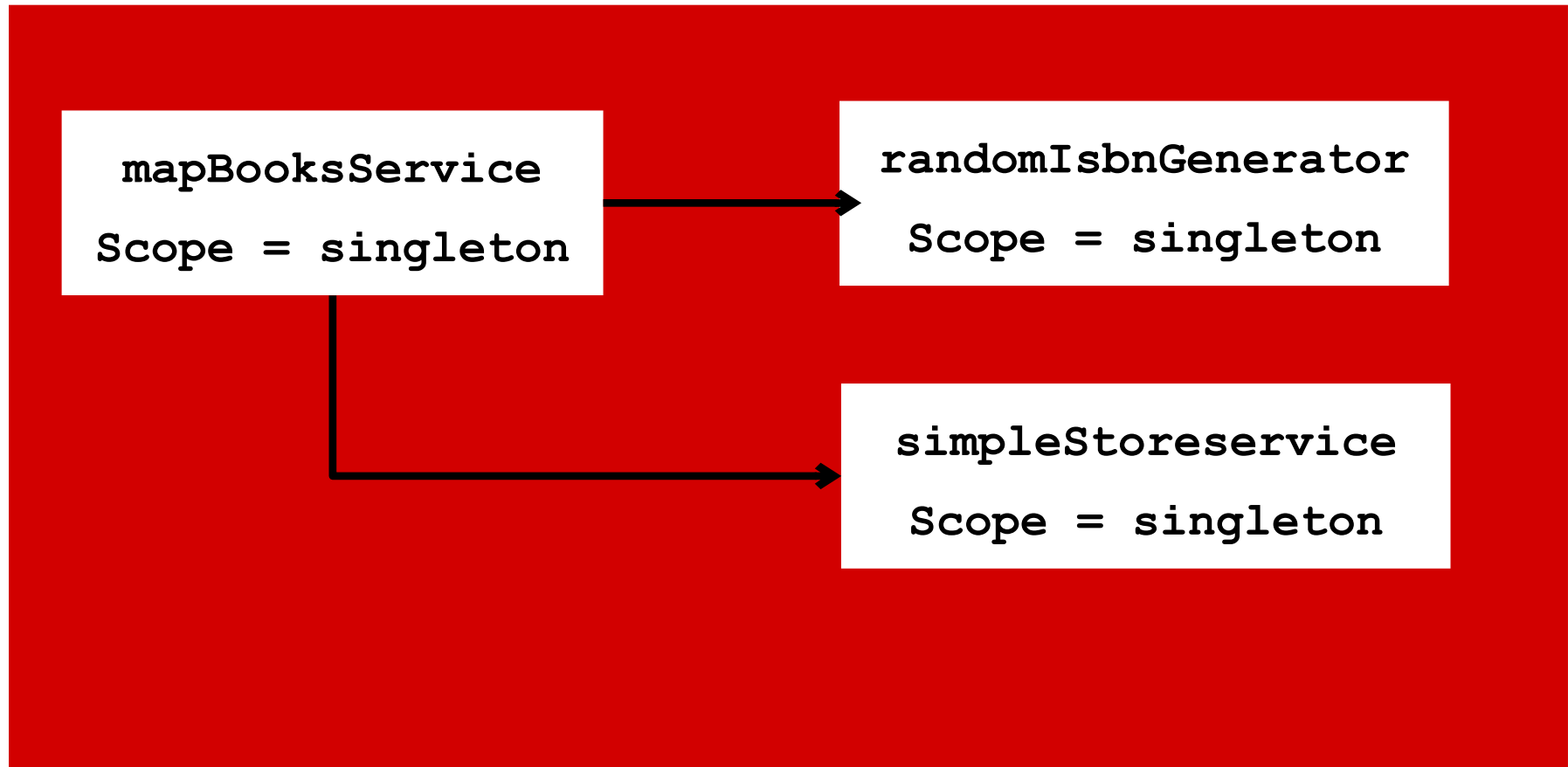
```
mapBooksService  
Scope = singleton
```

```
randomIsbnGenerator  
Scope = singleton
```

```
simpleStoreservice  
Scope = singleton
```

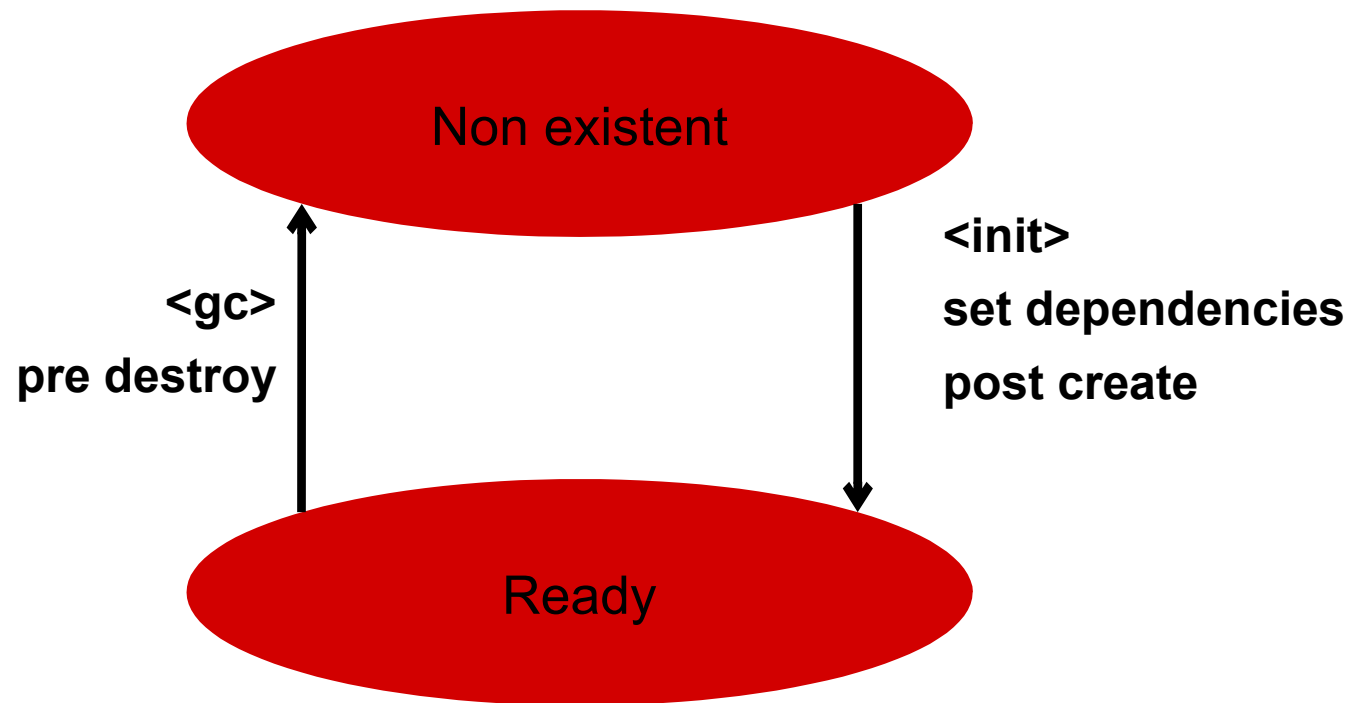
- Ein relevantes Objekt definiert neben dem Scope auch Abhängigkeiten auf andere Objekte
  - Im OOP-Modell ist dies eine Assoziation
- Der Context ist ebenfalls verantwortlich:
  - Dependencies zu identifizieren und
  - diese zu setzen

# Ein "Live" Context nach Dependency Injection



- Every business object has a defined lifecycle
  - instantiation
  - dependency injection
  - post create
  - pre destroy
  - destroy (Garbage collection)





2.2

## **SPRING CORE**

- Spring definiert Scope und Lifecycle mit
  - XML
  - Java Annotations
  - JavaConfig
    - eine Factory-Klasse
  - Eine Mischung aus allen VErfahren ist möglich

# Context-Definition: Spring XML

```
<?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.xsd">

  <bean class="org.javacream.store.business.SimpleStoreService" id="simpleStoreService">
    <property name="stock" value="42"></property>
  </bean>
  <bean class="org.javacream.keygeneration.business.RandomKeyGeneratorImpl"
    id="randomKeyGeneratorImpl" init-method="initTheKeyGenerator" destroy-method="destroyTheKeyGenerator">
    <property name="countryCode" value="-de">
    </property>
    <property name="prefix" value="ISBN:"></property>
  </bean>
  <bean class="org.javacream.books.warehouse.business.MapBooksService"
    id="mapBooksService">
    <property name="keyGenerator" ref="randomKeyGeneratorImpl"></property>
    <property name="storeService" ref="simpleStoreService"></property>
  </bean>
</beans>
```

```
@Repository
public class MapBooksService implements BooksService {
    @Autowired
    @Qualifier("sequence")
    private KeyGenerator randomKeyGeneratorImpl;

    @Autowired
    private StoreService storeService;

    private Map<String, BookValue> books;

    {
        books = new HashMap<String, BookValue>();
    }
}
```

# Context-Definition: Spring JavaConfig

```
@Configuration
public class BooksWarehouseConfig {

    @Bean public BooksService booksService() {
        MapBooksService mapBooksService = new MapBooksService();
        mapBooksService.setKeyGenerator(keyGenerator());
        mapBooksService.setStoreService(storeService());
        return mapBooksService;
    }

    @Bean public OrderService orderService() {
        OrderServiceImpl orderServiceImpl = new OrderServiceImpl();
        orderServiceImpl.setBooksService(booksService());
        orderServiceImpl.setStoreService(storeService());
        orderServiceImpl.setKeyGenerator(keyGenerator());
        return orderServiceImpl;
    }

    @Bean public StoreService storeService() {
        SimpleStoreService simpleStoreService = new SimpleStoreService();
        simpleStoreService.setStock(42);
        return simpleStoreService;
    }

    @Bean public KeyGenerator keyGenerator() {
        RandomKeyGeneratorImpl randomKeyGeneratorImpl = new RandomKeyGeneratorImpl();
        randomKeyGeneratorImpl.setPrefix("ISBN:");
        randomKeyGeneratorImpl.setCountryCode("-is");
        return randomKeyGeneratorImpl;
    }
}
```

3

## **TEST UND ANWENDUNGSSTART**

3.1

## **SPRING UNIT TESTS**



```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("/books-service.xml")
public class BooksServiceSpringTest {

    @Autowired
    private BooksService booksService;

    @Test
    public void testSpring() {
        TestActor.doTest(booksService);
    }
}
```

- Durch die Angabe des Runners wird der angegebene Spring-Kontext geladen
- Das zu testende Objekt steht dann über normale Dependency Injection zur Verfügung
  - Hier mit Autowiring
- Weitere Features
  - Profiles definieren nur die Objekte, die dem angegebenen Profil entsprechen

3.2

**MAIN**

```
public static void main(String[] args){  
    ClasspathXmlApplicationContext context = new  
ClasspathXmlApplicationContext("/books-service.xml");  
    BooksService bs context.getBean(BooksService.class);  
}
```

```
public static void main(String[] args){  
    SpringApplication.run(BooksWarehouseConfig.class,  
        args) ;  
}
```

4

## **WEITERE FEATURES**

4.1

## **SPRING AOP**

- Aspect Oriented Programmimg führt "Cross Cutting Concerns" ein
- Beispiel:
  - "Jeder Aufruf einer Business-Funktion muss eine Audit-Meldung generieren"
  - Authentication
  - Transaction Management
  - Profiling
  - Tracing
- Spring stellt ein ausgefeiltes AOP Framework zur Verfügung
  - XML Konfiguration
    - Der `aop`-namespace
  - Annotations
    - `@AspectJ` und `@Around`



4.2

## KONFIGURATION

- Die Konfiguration von Spring Applications ist einfach
  - Lesen von Properties Files und System Properties
  - Bei Spring Boot wird automatisch die application.properties gelesen
    - auch das YAML-Format wird unterstützt
- Zum Zugriff auf die Properties wird die Spring Expression Language genutzt
  - in den meisten Fällen genügt eine einfache Expression
    - `${propertyKey}`
  - Ein komplexeres Beispiel
    - `${(bean.list[i5] + bean2.foo.goo) > 42}`
- Property-Values werden injected
  - value-Attribut im XML
  - @Value-Annotation

## 4.3

# UTILITIES UND WEITERE WEITERE BIBLIOTHEKEN

- Die Spring Distribution enthält umfangreiche Zusatzbibliotheken
  - Eingebunden jeweils über den entsprechenden Starter
- Beispiele
  - `JdbcTemplate` für direkten Datenbankzugriff
  - JPA-Integration
  - Das Web Framework Spring MVC
    - Klassische Web-Anwendungen
    - RESTful Web Services
  - Monitoring mit dem Spring Actuator
  - Developer Tools

- Spring Data
- Spring Security
- Spring Batch
- Spring Integration
- ...