Spring Workshop

Schnelleinstieg Spring

https://github.com/Michaeli71/AMTC_Spring_Workshop

Michael Inden

Freiberuflicher Consultant und Trainer

Speaker Intro

- Michael Inden, Jahrgang 1971
- Diplom-Informatiker, C.v.O. Uni Oldenburg

- ~8 ¼ Jahre **SSE** bei Heidelberger Druckmaschinen AG in Kiel
- ~6 ¾ Jahre **TPL, SA** bei IVU Traffic Technologies AG in Aachen
- ~4 ¼ Jahre **LSA / Trainer** bei Zühlke Engineering AG in Zürich
- ~3 Jahre **TL / CTO** bei Direct Mail Informatics / ASMIQ in Zürich
- **Freiberuflicher Consultant, Trainer und Konferenz-Speaker**
- Autor und Gutachter beim dpunkt.verlag

E-Mail: michael.inden@hotmail.ch
Blog: https://jaxenter.de/author/minden

Kurse: Bitte sprecht mich an!

https://github.com/Michaeli71/AMTC_Spring_Workshop

Agenda

Workshop Contents

- **PART 1: Einführung**
  - Geschichte von Spring
  - Spring Architecture & ApplicationContext
- **PART 2: Dependency Injection**
  - Grundlagen IoC / DI
  - Spring Beans
  - Configuration
    - XML
    - Annotation
    - Java Config
  - Arten der Dependency Injection
  - Dependency Resolution Strategies

https://github.com/Michaeli71/AMTC_Spring_Workshop

Workshop Contents

- **PART 3: Bean Initialization / Scopes + Lifecycle**
  - Bean Initialization & Laziness
  - Zirkuläre Abhängigkeiten
  - Bean Scopes
  - Bean Lifecycle Callbacks

- **PART 4: Spring MVC**
  - Grundlagen DispatcherServlet & Controller
  - Beispiele Simple Rest Controller
  - Beispiele mit Thymeleaf

- **PART 5: Spring Tool Suite**

https://github.com/Michaeli71/AMTC_Spring_Workshop

PART 1:
Einführung

Vater von Spring

Rod Johnson

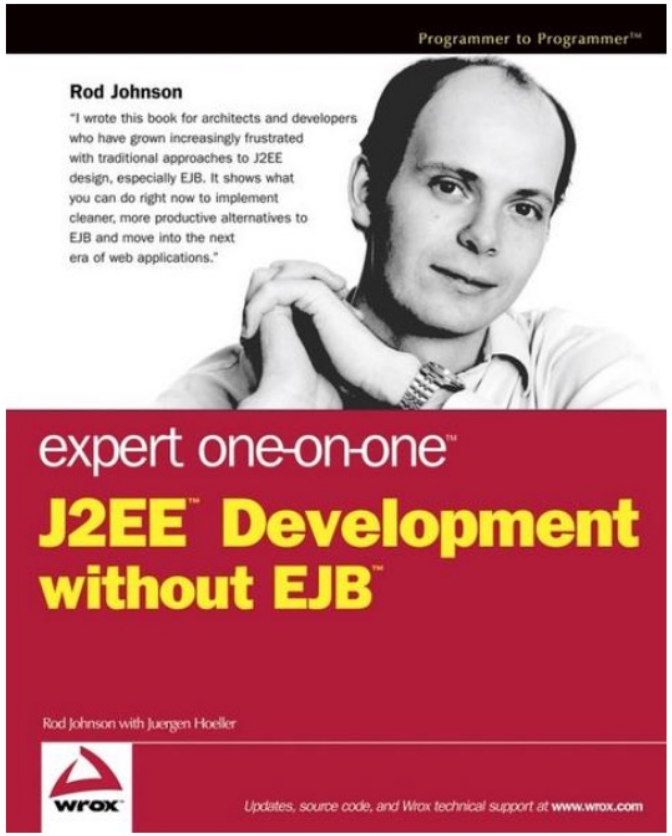
@springrod

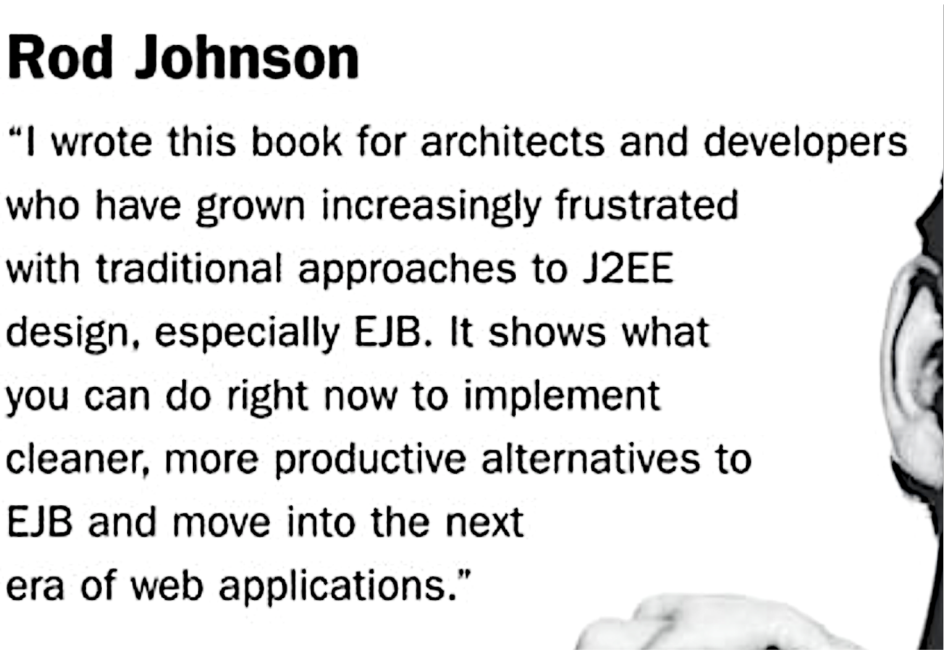
CEO, Atomist. Creator of Spring, Cofounder/CEO at SpringSource, Investor, Author

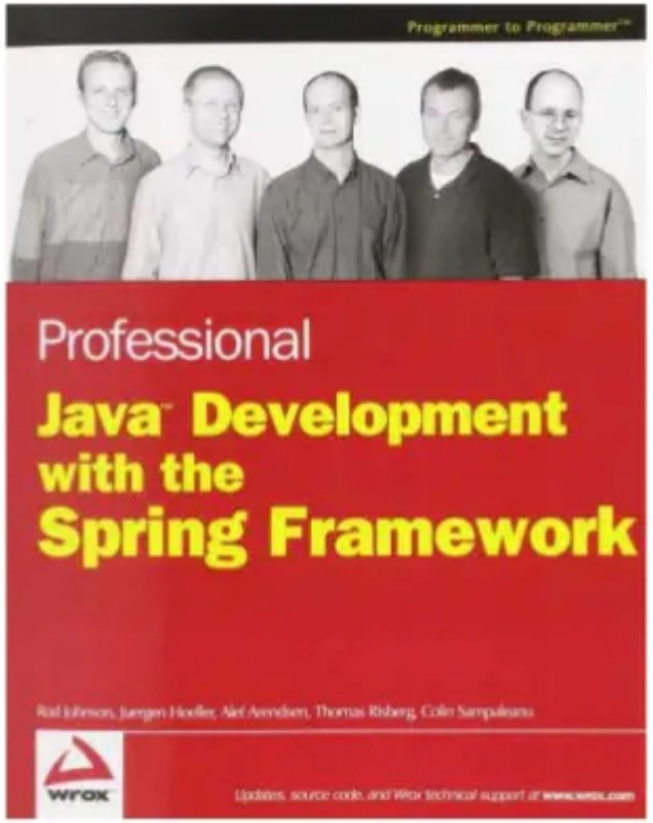
Sydney / Bay Area

atomist.com

Spring‘s Entstehung







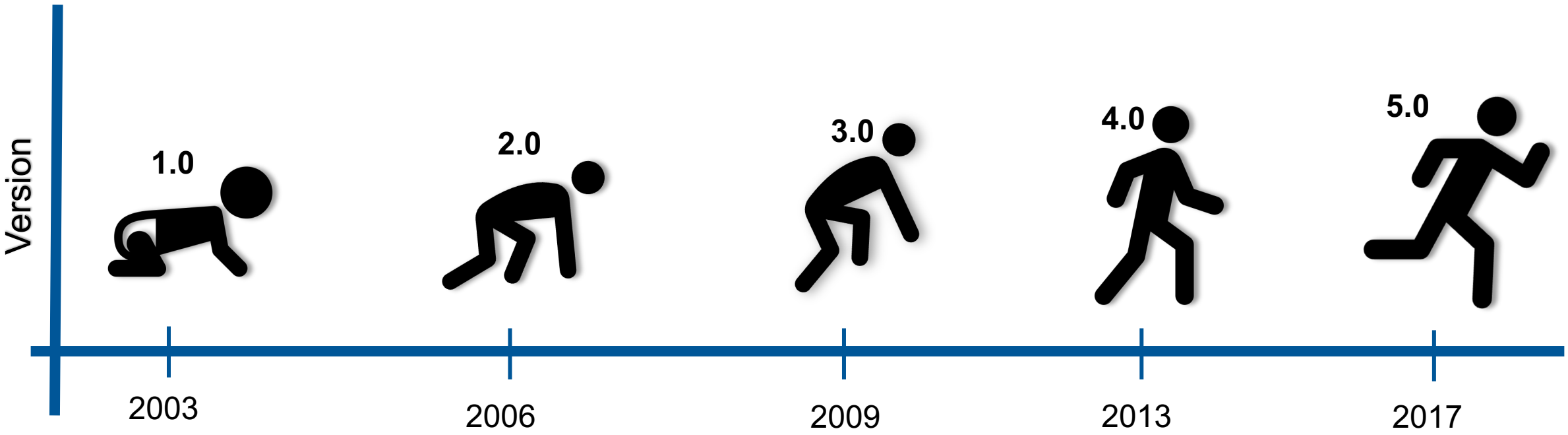
Spring‘s Entstehung

V 0.9

2002

INTERFACE21
Spring from the source

Spring‘s Werdegang



Core
Context
Web MVC

XML-Config &
JDBC, AspectJ-
Support, Java 5
Annotation

Java 5 Features,
`ExecutorService`,
SpEL, REST,
Embedded DBs

Java 8 Support
Getting Started
Guides,Hibernate 5
Updates von Libraries

JMS 2.0, JPA 2.1
Spring Web Flow
Reactive Streams
JUnit 5

Spring‘s Werdegang

- Spring 1.0
  - Mit offizieller Dokumentation
  - Aufgeteilt in wesentliche Module
    - Core
    - Context
    - Web MVC
- Spring 2.0
  - Neuorganisation der Module
  - Vereinfachte XML-Konfiguration
  - Vereinfachtes JDBC-Handling
  - AspectJ-Support
  - Java 5 Annotation Support

Spring‘s Werdegang

- Spring 2.5
  - Neue Annotations: @Repository, @Component, @Service, @Controller
  - @Autowired und Unterstützung von JSR-250-Annotations
  - Classpath Scanning

- Spring 3.0
  - Nutzt viele Java 5 Features wie Generics
  - Unterstützung für `ExecutorService`, `Callables` usw.
  - Einführung Spring Expression Language (SpEL)
  - Guter REST-Support
  - Unterstützung für Embedded Databases (H2)

Spring‘s Werdegang

- Spring 3.1
  - Cache Abstraction
  - Hibernate 4
  - `Viele neue Annotations: @ComponentScan,@EnableTransaactionManagement, …`
- Spring 3.2
  - JSON 2 Support
  - @DateTimeFormat ohne JODA-Time
  - Unterstützung für `ExecutorService`

Spring‘s Werdegang

- Spring 4.0
  - Java 8 Support
  - Getting Started Guides
- Spring 4.2 / 4.3
  - Hibernate 5
  - Updates von Libraries

Spring‘s Werdegang

- Spring 5.x
  - JMS 2.0
  - JPA 2.1
  - Spring Web Flow
  - Reactive Streams
  - JUnit 5

Werdegang der Firma

2002 2008 2009 2013

Vorteile von Spring

- **Leichtgewichtig**
  - Just POJO – need not implement/extend any framework interfaces/classes
- **Modular**
  - Include just the needed module to use the needed functionality.
  - For DI just include spring-context

- **Nicht-intrusive**
  - domain logic code generally has no dependencies on the framework itself

Major Spring Projects

SPRING FRAMEWORK

SPRING BOOT

SPRING CLOUD

SPRING DATA

SPRING CLOUD DATA FLOW

SPRING INTEGRATION

SPRING BATCH

SPRING SECURITY

Spring Framework - JDK Baseline

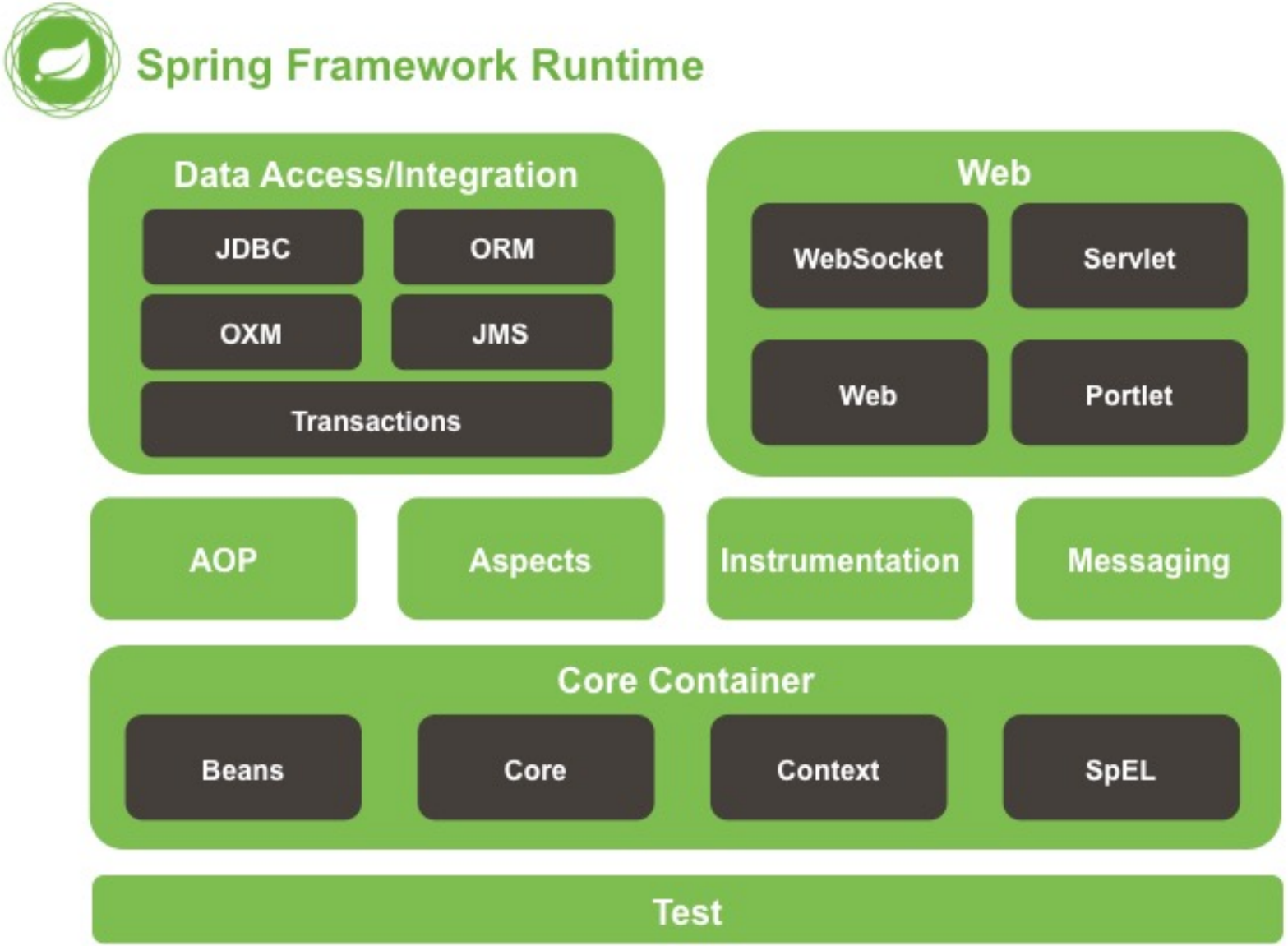
Version	JDK 6	JDK 7	JDK 8	JDK 9	JDK 10	JDK 11	JDK 12
4.3.x	👍	👍	👍	👎	👎	👎	👎
5.0.x	👎	👎	👍	👍	👍	👎	👎
5.1.x	👎	👎	👍	👍	👍	👍	👍

Arbeitet Spring mit dem Modulsystem?

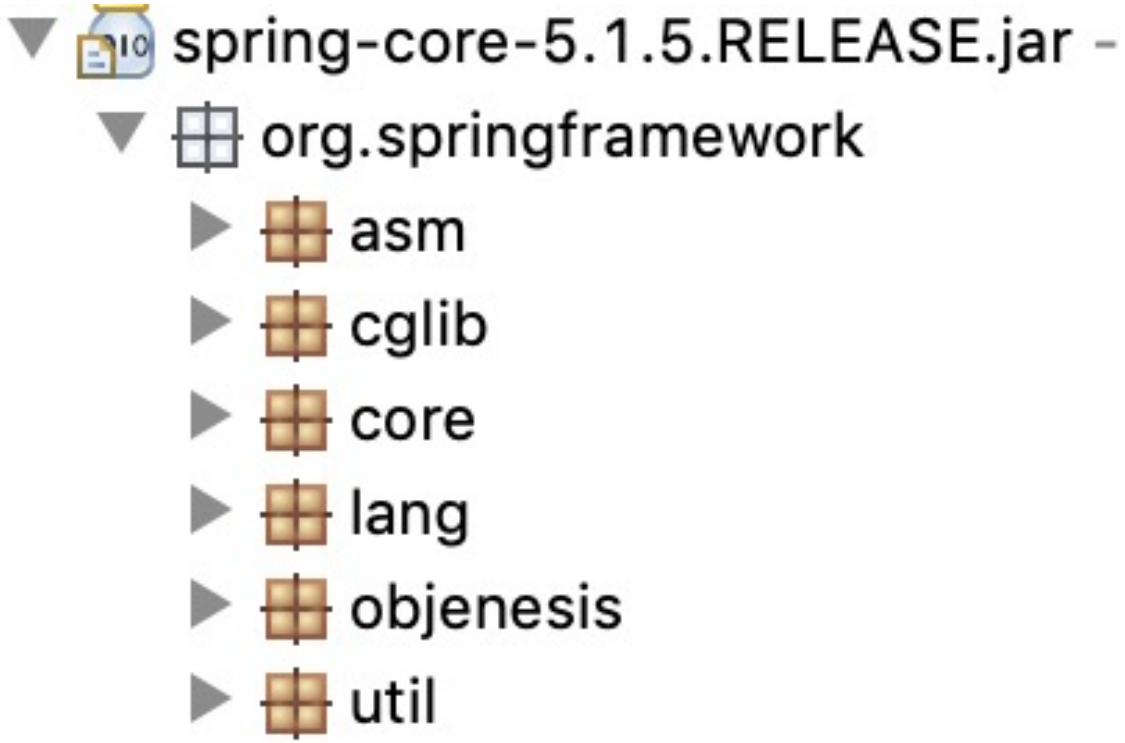
Yes, Spring Framework 5 ships with **automatic module name entries** in the manifests of our Spring Framework 5 jars. The public API surface of the Spring libraries remains unchanged.

Spring Architecture

Spring Architecture



Spring Core Container Details



Beans Modul

Contains necessary classes and interfaces for the manipulation of the java beans

Defines most important interface like **BeanFactory** for Bean Creation and Dependency Injection.

Includes utility functions for checking property types, copying bean properties, instantiating java beans.

Context und SpEL Module

Context

- Defines the interface ApplicationContext, which extends BeanFactory

SpEL = Spring Expression Language

- *“Language to query and manipulate the object graph at runtime.“*
  - #{object.property}
  - #{object.subobject.property}

Beans & Application Context

- Ein (Spring) Bean ist eine ganz normale Java-Klasse, die von Spring verwaltet werden soll (Instanziierung, Lifecycle usw.)

```
public class GreetingService {

    public void sayHello() {
        System.out.println("Welcome to Spring Workshop :-)");
    }
}
```

- Zugriff beispielsweise über Application Context

```
ClassPathXmlApplicationContext applicationContext =
                               new ClassPathXmlApplicationContext("greeting-app.xml");

GreetingService greetingService = applicationContext.getBean(GreetingService.class);
```

Inversion of Control

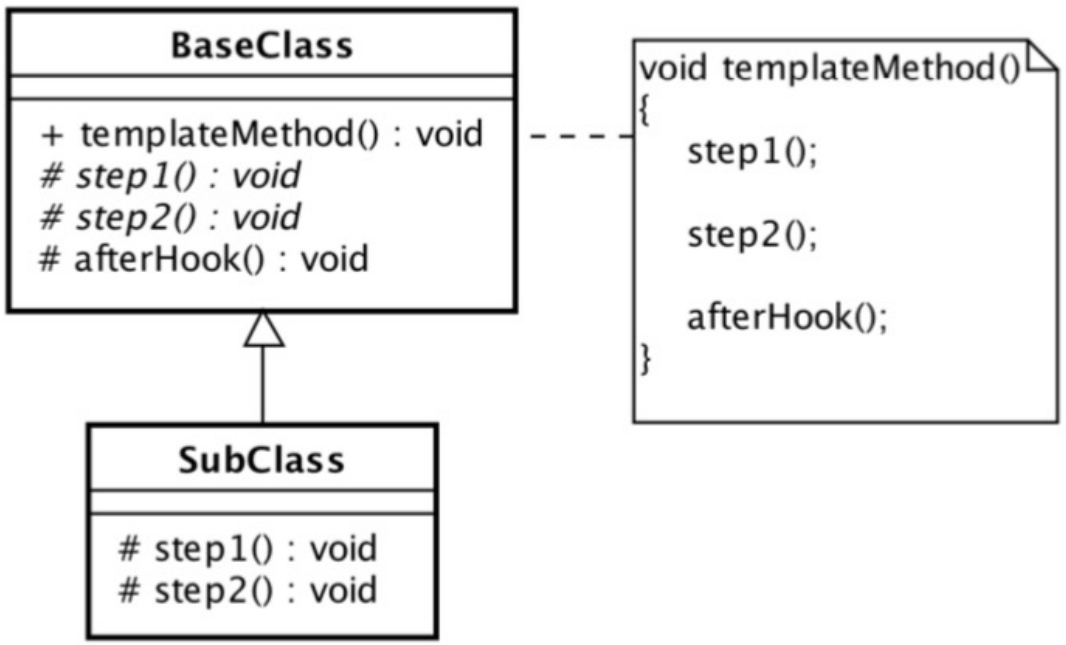
Was ist eigentlich Dependency Injection und Inversion of Control?

https://martinfowler.com/bliki/InversionOfControl.html

Inversion of Control

- Hollywood Principle («Don't call us, we'll call you»)

Inversion of Control: Template Method Pattern




```
public void paint(Graphics g)
{
    super.paint(g);

    final Graphics2D g2d = (Graphics2D) g;

    drawSheetAndBackground(g2d);

    if (showRuler)
         drawRuler(g2d);

    if (showGrid)
         drawGrid(g2d);

    drawContent(g2d);
}

public abstract drawContent(Graphics2D g2d);
```

- Der grundsätzliche Ablauf und **Algorithmus** ist in der Basisklasse `BaseClass` in der **Methode** `templateMethod()` **realisiert** und **umfasst** die **Teilschritte** 1 und 2 und einen abschließenden **Hook**. Diese **Funktionalität** wird durch die **korrespondierenden Methoden** `step1()`, `step2()` sowie `afterHook()` **realisiert**. Die Methode `templateMethod()` ist `final` und die Methoden `step1()` und `step2()` sind abstrakt. Die Methode `afterHook()` ist in der Basisklasse leer implementiert, kann aber in Subklassen redefiniert werden.

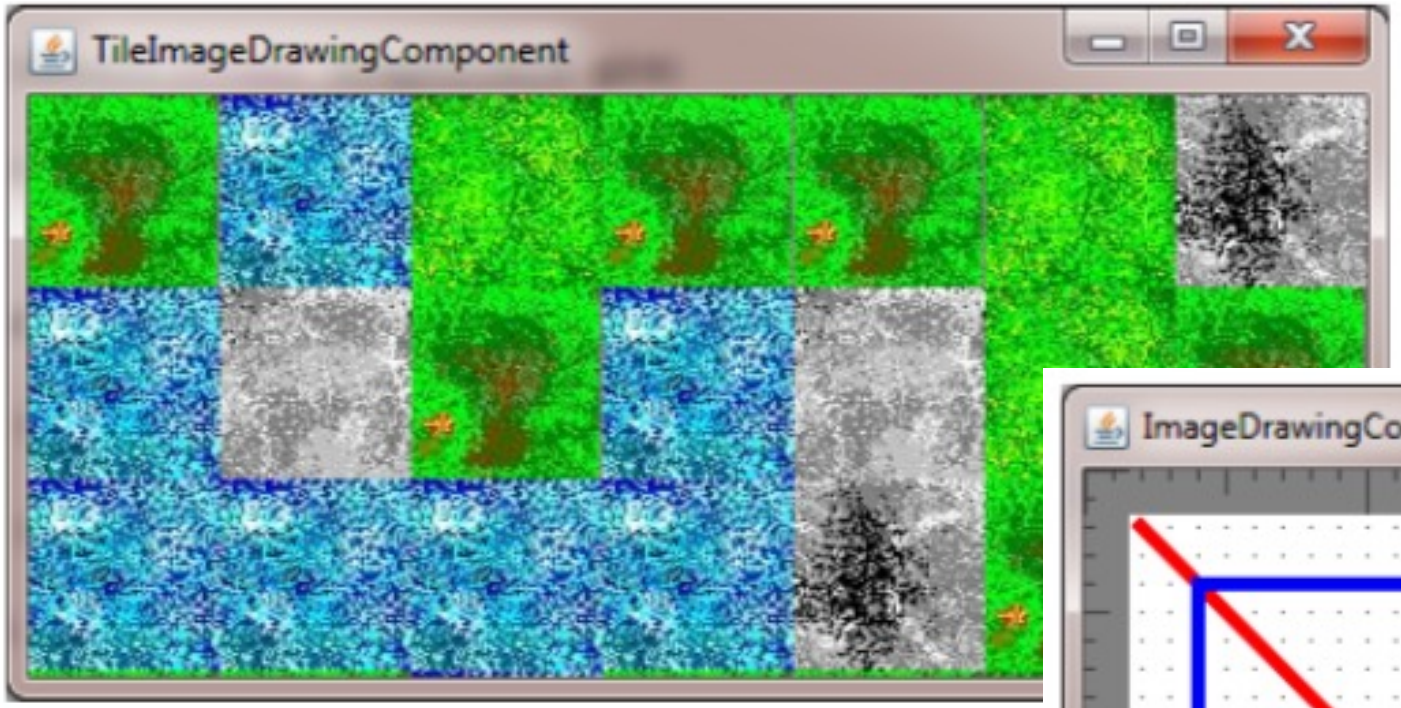
Motivation und Kurzbeschreibung Schablonenmethode

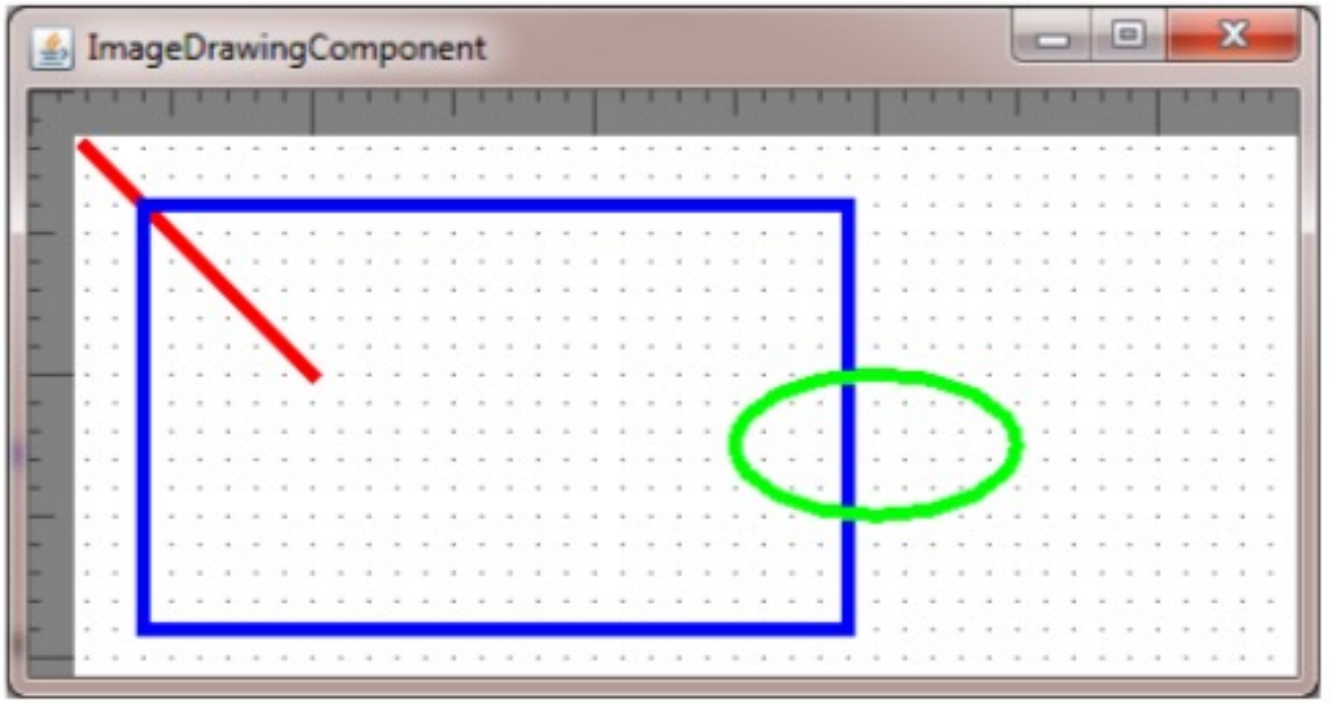
- Idee ist es, einen **Algorithmus** in verschiedene **Schritte** aufzuteilen
- **Grundzüge** des Algorithmus in einer Basisklasse definieren und es **Subklassen** erlauben, einige der **Berechnungsschritte** zu **implementieren**.
- Die Abfolge der Schritte wird in einer **speziellen Methode** der **Basisklasse** realisiert, die ***Schablonenmethode*** genannt wird. Sie ist in der Regel `final` definiert, um die grundsätzliche Abfolge zu schützen.
- **Einige Schritte** sind in **Basisklasse** noch undefiniert und **können** dann von **Subklassen ausformuliert** werden.

.

- Das beschriebene Vorgehen stellt sicher, dass die Struktur des Algorithmus unverändert bleibt, Subklassen jedoch Möglichkeiten der Einflussnahme gegeben wird.

Anwendungsbeispiel

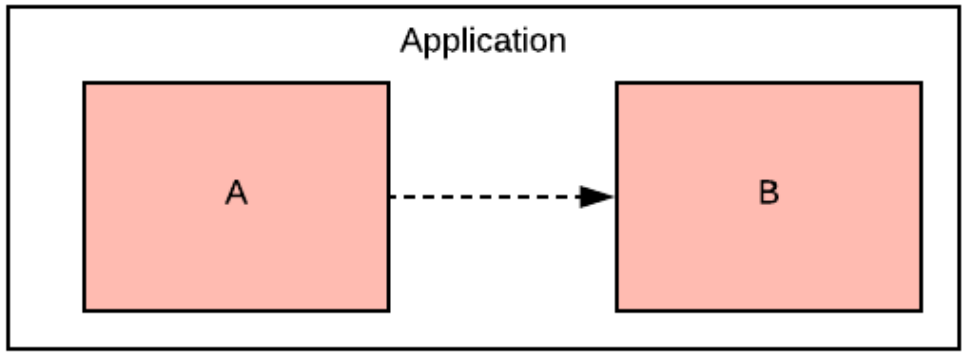


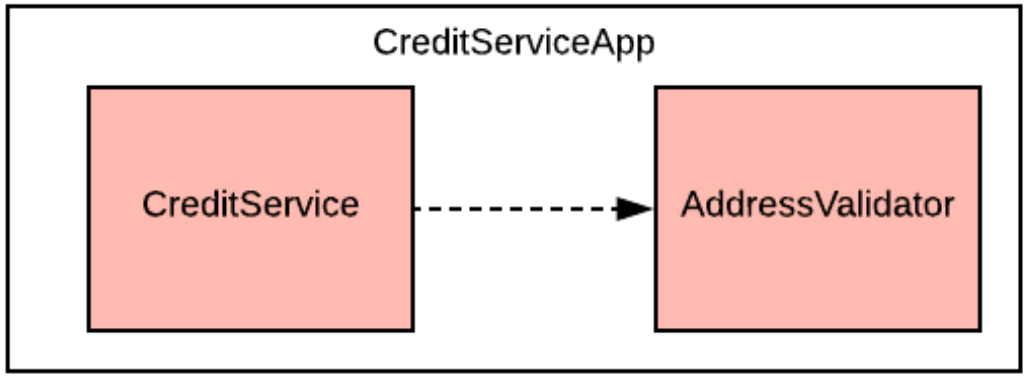


PART 2:
Dependency Injection

Dependencies – Short Recap

- Dependency = Abhängigkeit ⇔ Eine Klasse basiert / nutzt eine andere





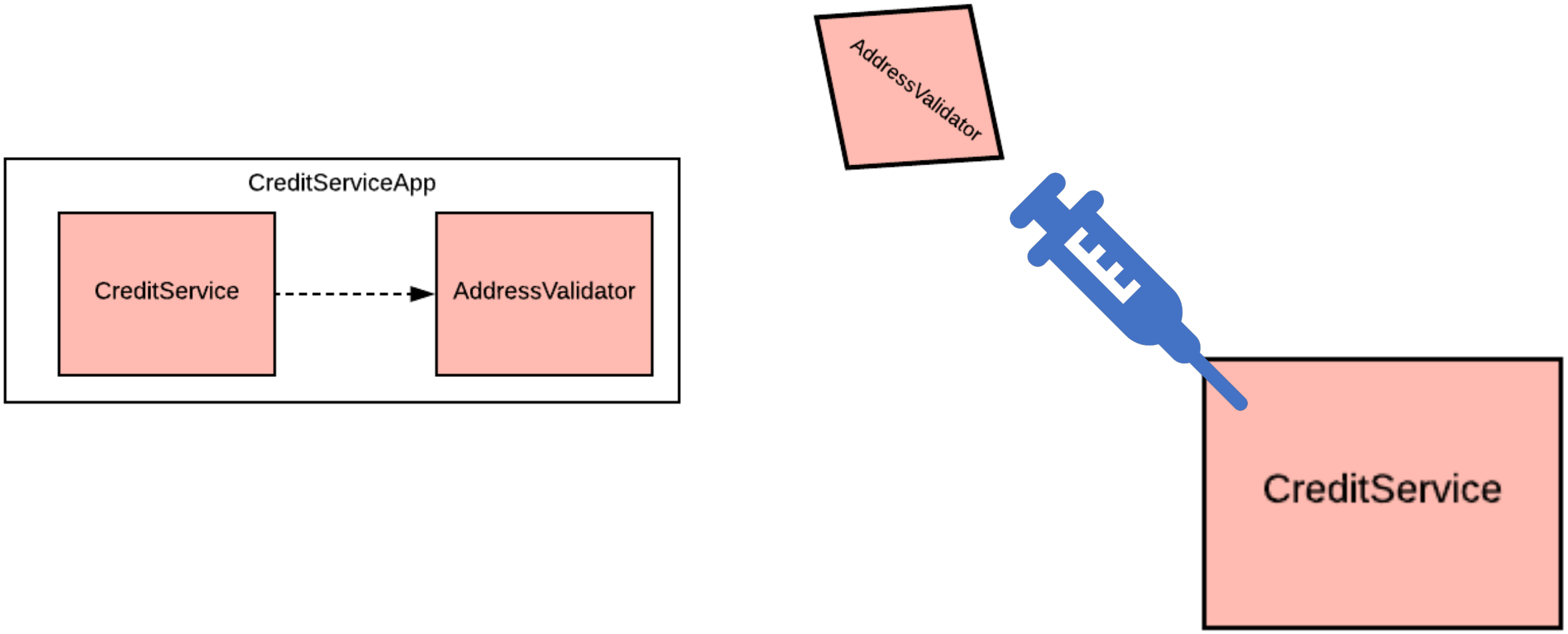
Zentrale Frage: Wie komme ich an die Dependency?

Früher oft: Durch Aufruf von new.

Dependency Injection

The process of introducing the dependency to the dependent object is called dependency injection

Dependency Injection in Action



Dependency Injection

```
public class CreditServiceApp {

        public static void main(String[] args) {
                var addressValidator = new AddressValidator();
                var creditService = new CreditService(addressValidator);
                creditService.dispatchCredit();
        }

}
```

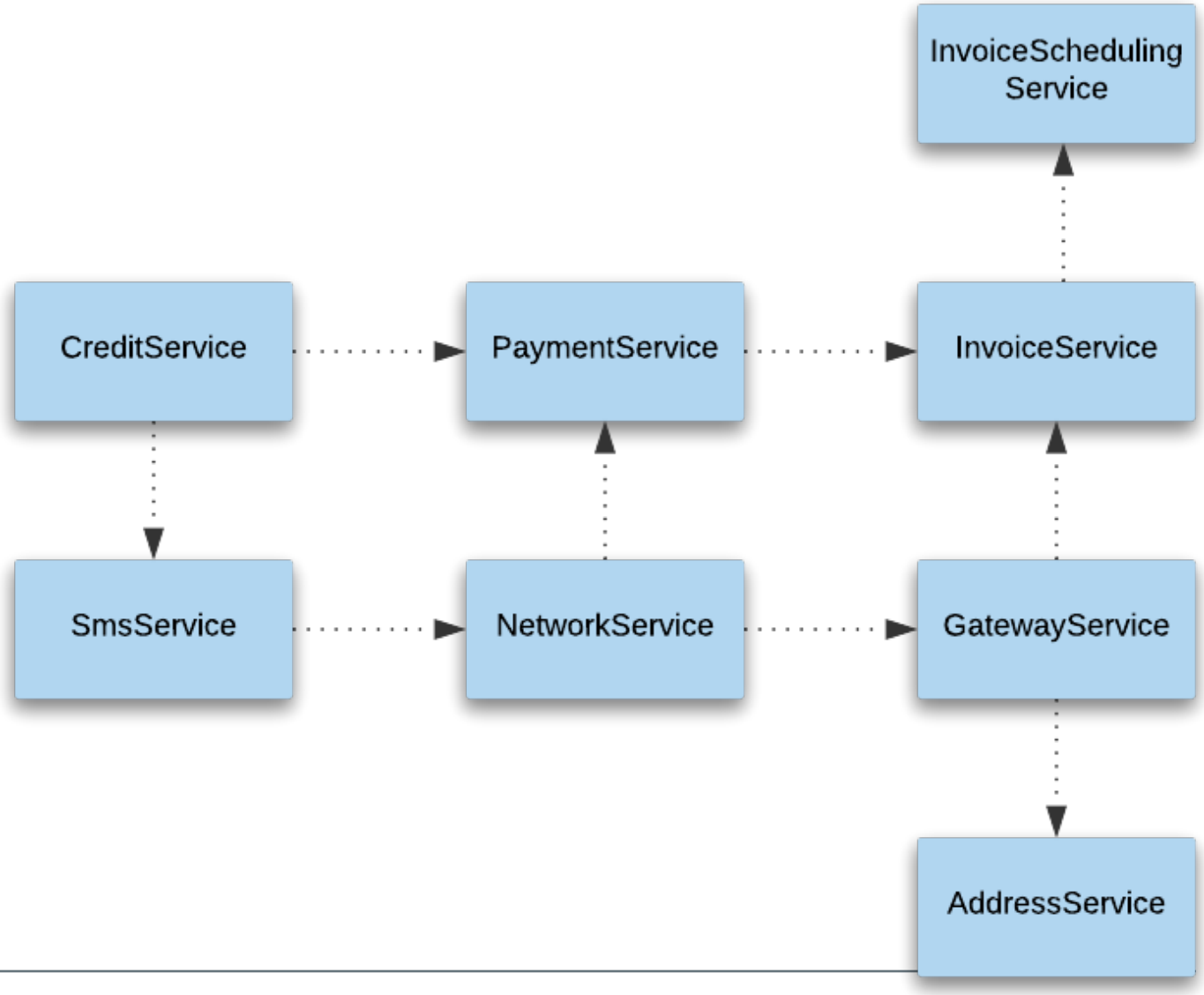
Manuelle Dependency Injection

Manual DI

```
public class CreditServiceApp {

        public static void main(String[] args) {
                var addressValidator = new AddressValidator();
                var creditService = new CreditService(addressValidator);
                creditService.dispatchCredit();
        }

}
```

Was ist ein Hauptproblem am Aufruf von new?

Was ist bei einem solchen Objektgraphen?



Probleme der manuellen DI

- Aufwendig
- Viele manuelle Objektkonstruktionen & Bolierplate Code
- Enge Kopplung
- Schlechtere Lesbarkeit
- Manchmal wird interne Aufbau nach außen sichtbar
- Mitunter schwierig zu testen (Objektkonstruktionen, Abläufe, …)

Kommen wir nochmal zur vorherigen Frage: Was ist eigentlich Dependency Injection und Inversion of Control?

Den Prozess, die Kontrolle über die Instanziierung abhängiger Objekte und die Konfiguration des Objektgraphen abzugeben, nennt man **Inversion of Control (IoC).**

Die Auflösung / Bereitstellung von Abhängigkeiten wird **Dependency Injection (DI)** genannt.

Dependency Injection in Spring

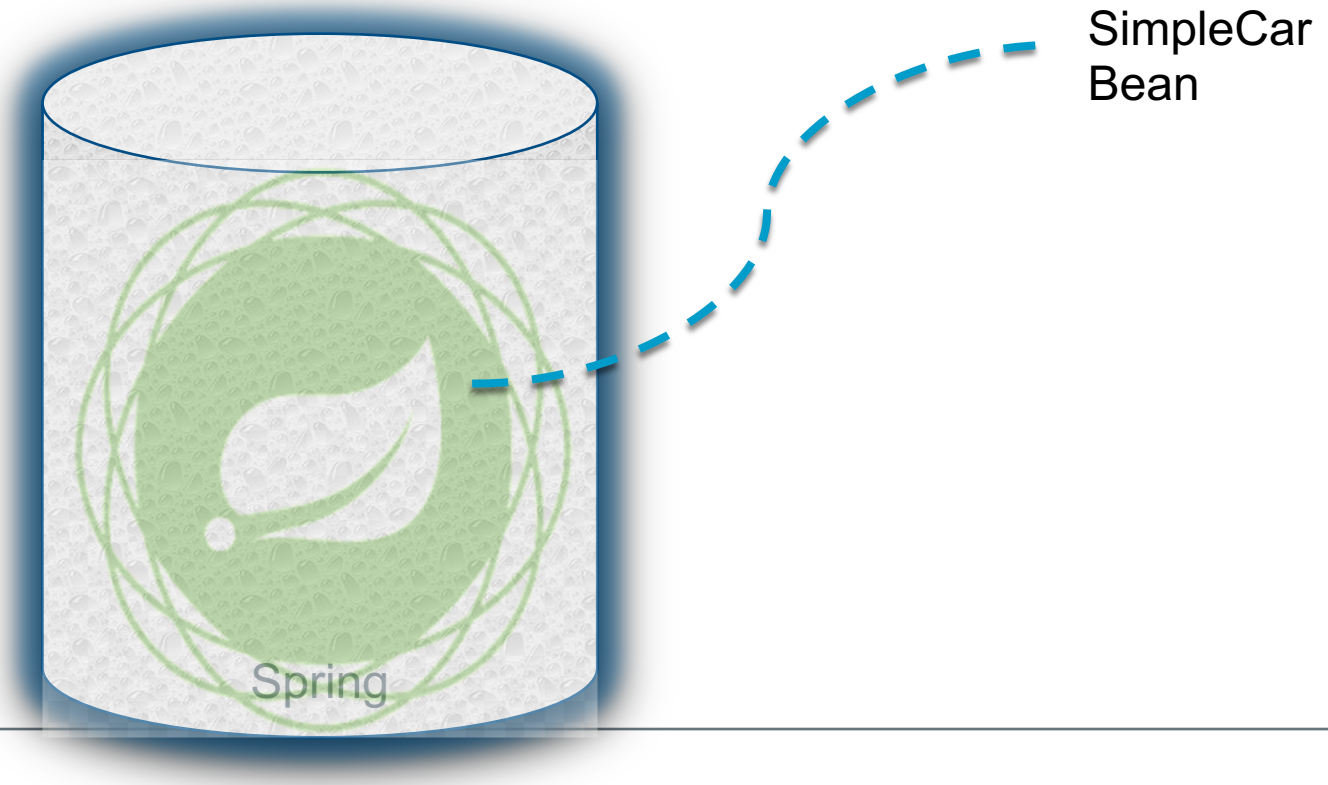
IoC Container

- Ist auch (nur) ein Objekt
- Verwaltet andere Objekte
- Ermöglicht Dependency Injection
- Baut den Object Graph auf

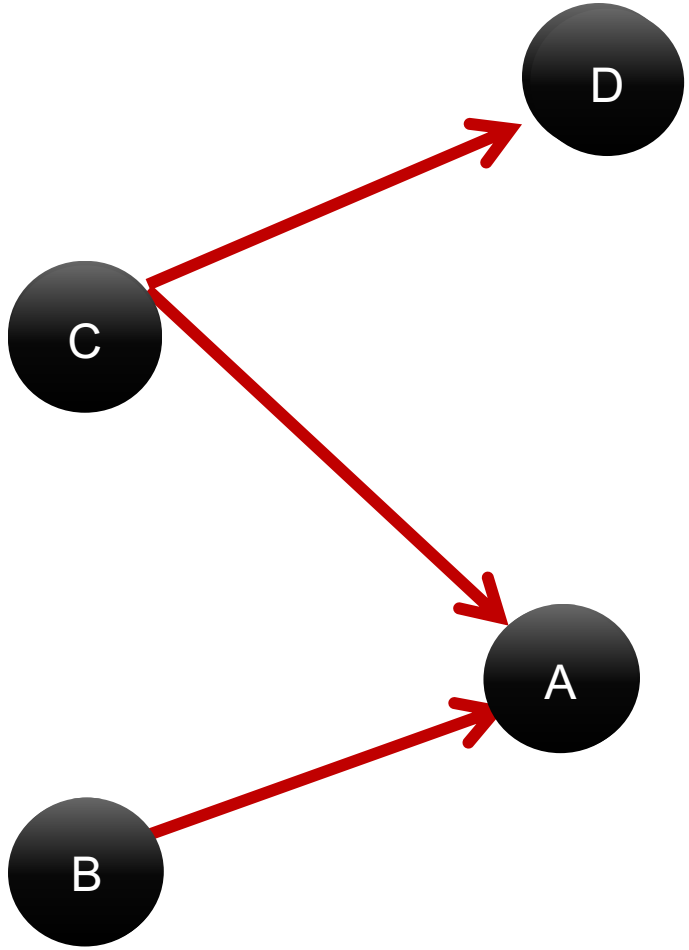
Spring Beans

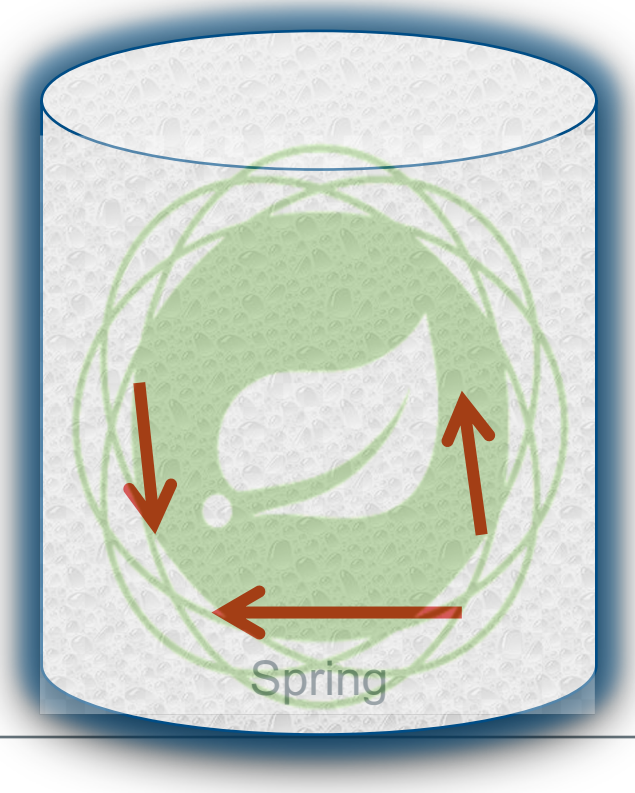
- Plain Old Java Object (POJO) which is/will be **managed** by the Spring is called **Spring Bean**.

SimpleCar



IoC Container – Topologische Sortierung und Objekterzeugung





Woher weiß der IoC Container, wie die Beans konfiguriert werden müssen und zusammenhängen?

Configuration

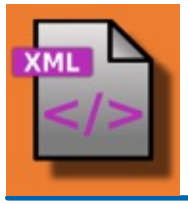
Configuration Metadata

The information needed to configure the IoC Container is **Configuration Metadata**

XML

ANNOTATION

JAVA



XML Configuration

```
<?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 id=“person" class=“ch.karthi.Person">
                <property name=“name“ value=“karthi“ />
                <property name=“age“  value=“22“ />
        </bean>

</beans>
```

Java Configuration

```
@Configuration
public class MyConfig{

        @Bean
        public Person createPerson(){
                Person person = new Person();
                person.setName(“karthi“);
                person.setAge(“22“);
                return person;
        }
}
```

Annotation Configuration

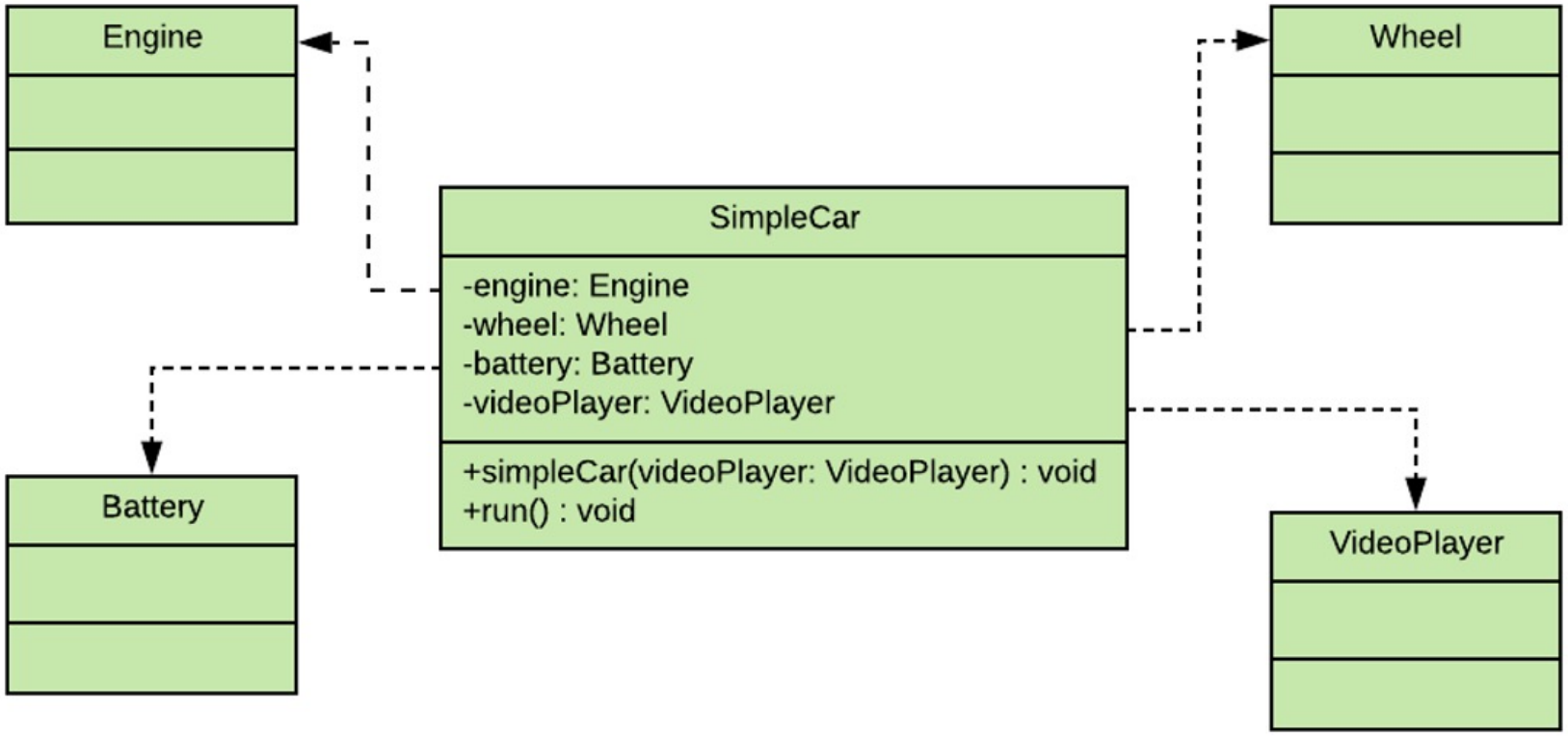
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:c="http://www.springframework.org/schema/c"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

  <context:component-scan base-package="ch.javaprofi_academy" />

</beans>
```

+

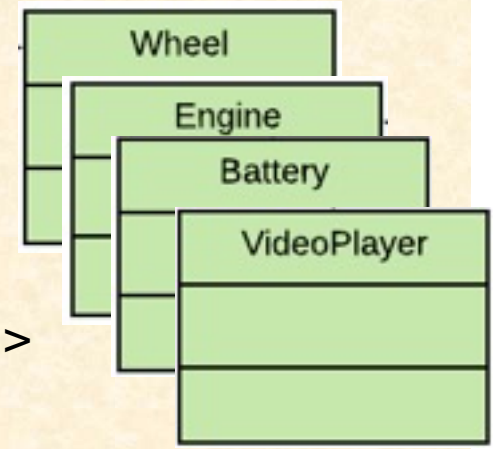
```java
@Component
public class Person {

...

}
```

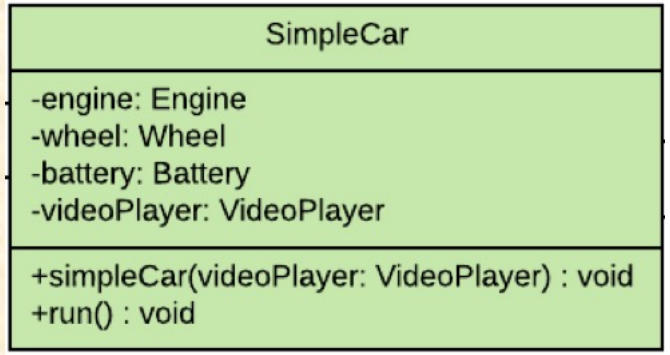
Beispiel für XML-Konfiguration

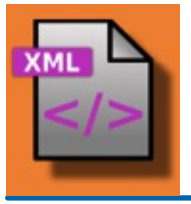
XML Configuration



```
<?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 id="wheel" class="ch.karthi.Wheel"/>
        <bean id="engine" class="ch.karthi.Engine"/>
        <bean id="battery" class="ch.karthi.Battery"/>
        <bean id="videoPlayer" class="ch.karthi.VideoPlayer"/>


        <bean id="simpleCar" class="ch.karthi.SimpleCar">
            <constructor-arg ref="videoPlayer" name="videoPlayer"/>
            <property name="engine" ref="engine" />
            <property name="wheel" ref="wheel"/>
            <property name="battery" ref="battery"/>
        </bean>


</beans>
```





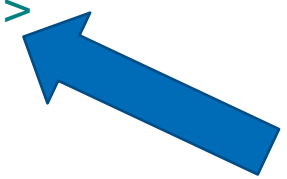


XML Configuration Besonderheit bei Konstruktion

```java
public class Customer {

    private String name;
    private String address;

    public Customer(String name, String address) {
        this.name = name;
        this.address = address;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setAddress(String address) {
        this.address = address;
    }
```

XML Configuration Besonderheit bei Konstruktion

```
<?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:c="http://www.springframework.org/schema/c"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean name="customer1" class="ch.javaprofi_academy.Customer">
        <constructor-arg name="name" value="Michael" />
        <constructor-arg name="address" value="Zürich" />
    </bean>


    <bean name="customer2" class="ch.javaprofi_academy.Customer"
          c:name="Tim" c:address="Kiel" />

</beans>
```



XML Configuration Besonderheit bei Konstruktion

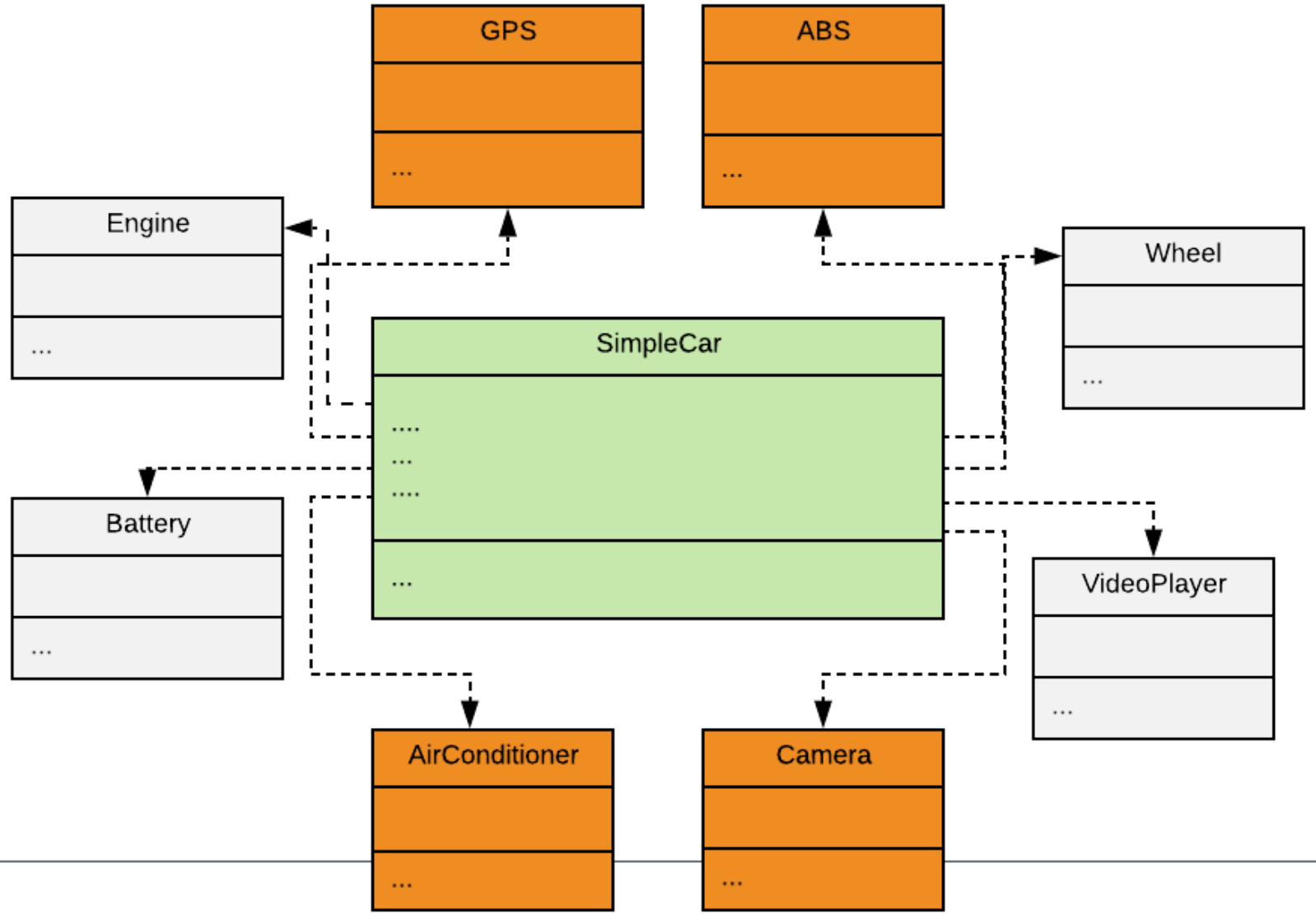
```
public class Application {

    public static void main(String[] args) {

        var ctx = new GenericXmlApplicationContext("bean-def.xml");

        var cust1 = ctx.getBean("customer1");
        var cust2 = ctx.getBean("customer2");

        System.out.println(cust1);
        System.out.println(cust2);

        ctx.close();
    }
}
```

```
Customer [name=Michael, address=Zürich]
Customer [name=Tim, address=Kiel]
```

Demo

`SimpleXmlConfigApp`

XML Configuration



```
<?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 id="wheel" class="ch.karthi.Wheel"/>
        <bean id="engine" class="ch.karthi.Engine"/>
        <bean id="battery" class="ch.karthi.Battery"/>
        <bean id="videoPlayer" class="ch.karthi.VideoPlayer"/>
        <bean id="gps" class="ch.karthi.GPS"/>
        <bean id="camera" class="ch.karthi.Camera"/>
        <bean id= … />
        ...
        <bean id="simpleCar" class="ch.karthi.SimpleCar">
                <constructor-arg ref="videoPlayer" name="videoPlayer"/>
                <property name="engine" ref="engine" />
                <property name="wheel" ref="wheel"/>
                <property name="battery" ref="battery"/>
                <property …..
        </bean>

</beans>
```

Wäre es nicht schön, wenn uns Spring da einige Arbeit abnehmen würde?

```
<?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.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

        <context:component-scan base-package="ch.karthi" />


        <bean id="simpleCar" class="ch.karthi.SimpleCar">
                <constructor-arg ref="videoPlayer" name="videoPlayer"/>
                <property name="engine" ref="engine" />
                <property name="wheel" ref="wheel"/>
                <property name="battery" ref="battery"/>
                <property …..
        </bean>

</beans>
```

```
<?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.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

        <context:component-scan base-package="ch.karthi" />


        <bean id="simpleCar" class="ch.karthi.SimpleCar">
                        <constructor-arg ref="videoPlayer" name="videoPlayer"/>
                        <property name="engine" ref="engine" />
                        <property name="wheel" ref="wheel"/>
                        <property name="battery" ref="battery"/>
                        <property …..
        </bean>

</beans>
```

Geht es bitte noch etwas einfacher?

Annotation configuration

Stereotype Annotations

```
@Component
public class Wheel {
```

```
@Component
public class SimpleCar {
```

```
@Component
public class Engine {
```

```
@Component
public class VideoPlayer {
```

```
@Component
public class Battery {
```

Stereotype Annotations

- `@Component`

- `@Service`
- `@Repository`

- `@Controller`
- `@RestController`

- `@Configuration`

Können wir die Abhängigkeiten in Java definieren?

@Autowired

```
@Component
public class SimpleCar {
        @Autowired
        private Engine engine;
        @Autowired
        private Wheel wheel;
        @Autowired
        private Battery battery;

        // this is autowired through constructor
        private VideoPlayer videoPlayer;
        …
        @Autowired
        public SimpleCar(VideoPlayer videoPlayer) {
                this.videoPlayer = videoPlayer;
        }
}
```

Können wir ganz auf XML verzichten?

Java Configuration + @ComponentScan

```
@ComponentScan("ch.karthi")
public class AppConfig {…}
```

```
@Component
public class Engine {…}
```

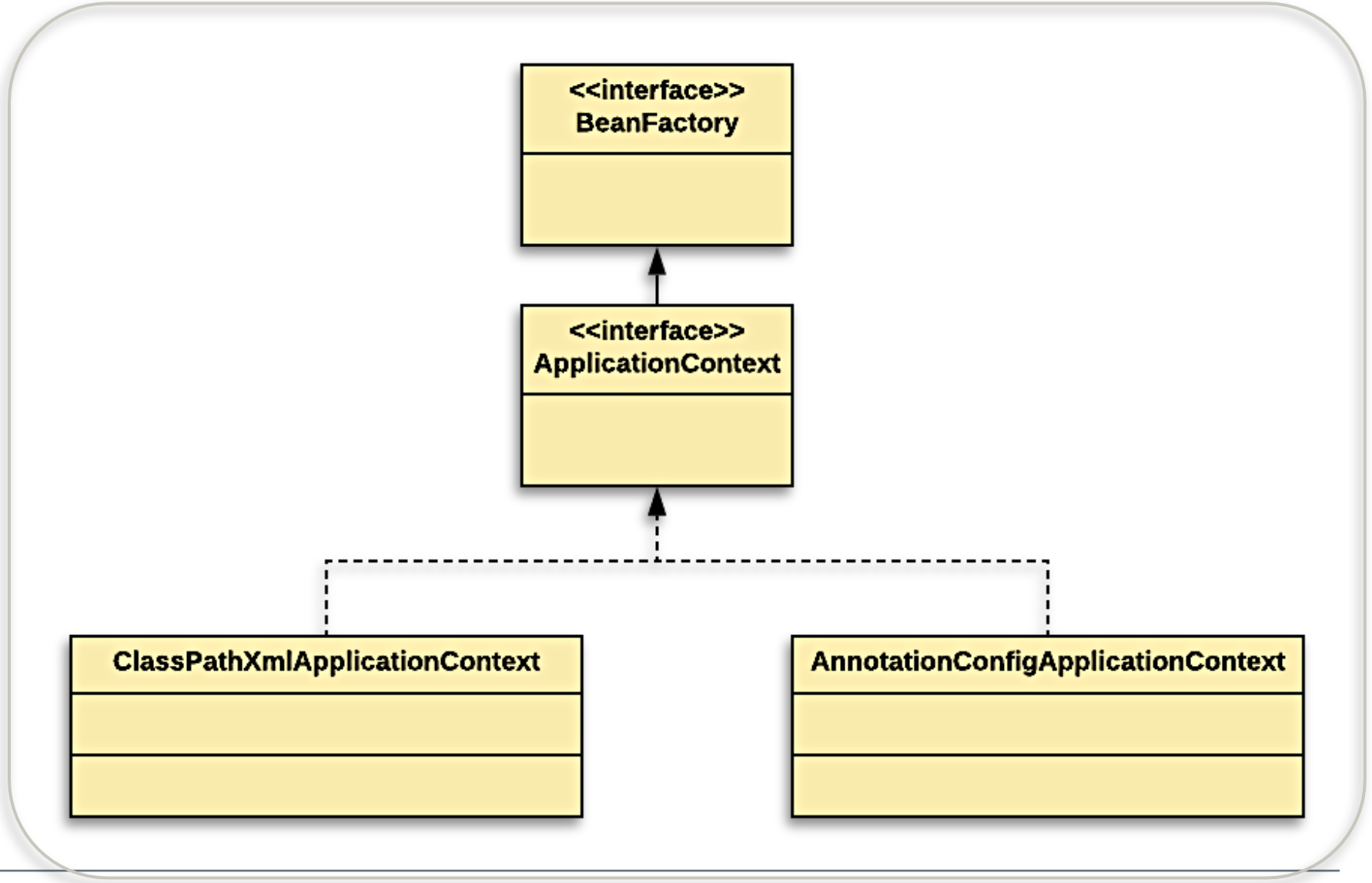
```
<bean id="engine" class="ch.karthi.Engine"></bean>
…
```

```
<bean id="simpleCar" class="ch.karthi.SimpleCar">
        <constructor-arg ref="videoPlayer"
name="videoPlayer"></constructor-arg>
        <property name="engine" ref="engine"></property>
        …
</bean>
```

```
@Component
public class SimpleCar {

        @Autowired
        private Engine engine;
}
```

IoC Container instantiation & usage

IoC Container in Spring



XML

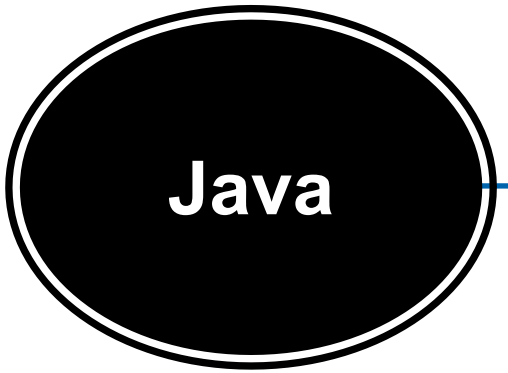
```
var container = new ClassPathXmlApplicationContext("config.xml");

var simpleCar = container.getBean(SimpleCar.class);

// start the simpleCar
```

```
<?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 id="wheel" class="ch.karthi.Wheel"/>
        <bean id="engine" class="ch.karthi.Engine"/>
        <bean id="battery" class="ch.karthi.Battery"/>
        <bean id="videoPlayer" class="ch.karthi.VideoPlayer"/>
        <bean id="gps" class="ch.karthi.GPS"/>
        <bean id="camera" class="ch.karthi.Camera"/>
        <bean id= … />
        ...
```

Annotation

```
var container = new ClassPathXmlApplicationContext("config.xml");

var simpleCar = container.getBean(SimpleCar.class);

// start the simpleCar
```

```
<?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.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
    <context:component-scan base-package="ch.karthi" />
</beans>
```

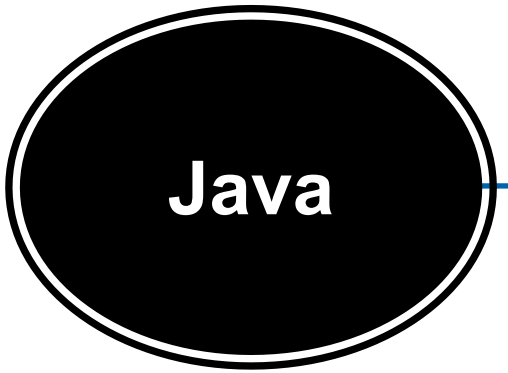


@ComponentScan("ch.karthi")
public class AppConfig {…}

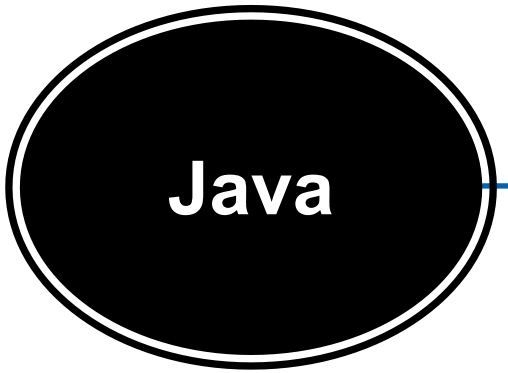
```
var container = new AnnotationConfigApplicationContext(AppConfig.class);

var simpleCar = container.getBean(SimpleCar.class);

// start the simpleCar
```

@Configuration
public class AppConfig
{…}

Lassen sich mehrere Konfigurationen kombinieren? Warum wäre das praktisch?



```java
@Configuration
public class CustomerConfig {
    @Bean
    public Customer newMikeCustomer() {
       return new Customer("Michael", "Zürich");
    }

    @Bean
    public Customer newTimCustomer() {
       return new Customer("Tim", "Kiel");
    }
}

@Configuration
public class ServiceConfig {
    @Bean
    public ServiceBean serviceBean() {
        return new ServiceBean();
    }
}
```



```java
@Configuration
@Import({CustomerConfig.class, ServiceConfig.class})
public class ImportBeansConfig {
  @Bean
  public ExampleBean exampleBean() {
    return new ExampleBean();
  }
}
```

```java
public static void main(String[] args) {
    AnnotationConfigApplicationContext context =
        new AnnotationConfigApplicationContext(ImportBeansConfig.class);

    ExampleBean exampleBean = context.getBean(ExampleBean.class);
    Customer customerBean = context.getBean("newMikeCustomer", Customer.class);

    System.out.println(exampleBean);
    System.out.println(customerBean);
    context.close();
}
```

Demo

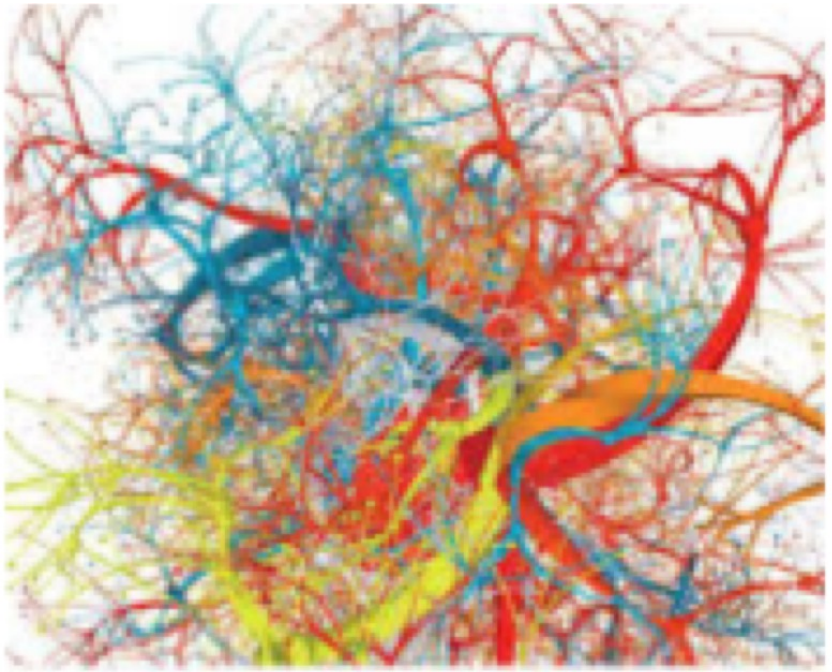
MultipleAnnotationConfigDemo

Spring Documentation

https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-java-composing-configuration-classes

https://docs.spring.io/spring-javaconfig/docs/1.0.0.m3/reference/html/modularizing-configurations.html

Arten der Dependency Injection



Arten der Dependency Injection

- Konstruktor-Injection
- Setter / Method-Injection
- Field-Injection

Constructor Injection

```
public SimpleCar(VideoPlayer videoPlayer){
        this.videoPlayer = videoPlayer;
}
```

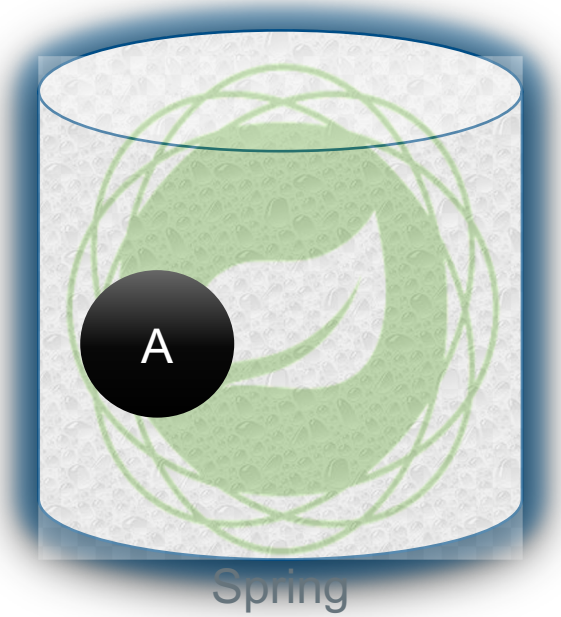
Rule: @Autowired should be present at least on one constructor, when more than one constructor has been declared.

Constructor Injection

```
public SimpleCar() {}

@Autowired
public SimpleCar(VideoPlayer videoPlayer) {
        this.videoPlayer = videoPlayer;
}
```

Exception in thread "main"
java.lang.NullPointerException

Constructor Injection Special Case




```
@Autowired
public MyBean(A a) {
```

```
@Autowired
public MyBean(A a, B b) {
```

```
@Autowired
public MyBean(A a,
              @Autowired(required=false) B b) {
```

Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type 'ch.karthi.B' available: expected at least 1 bean which qualifies as autowire candidate. Dependency annotations: {}

Setter Injection

```
public class SimpleCar {

        private Engine engine;

        private Wheel wheel;

        @Autowired
        public void setEngine(Engine engine) {
                this.engine = engine;
        }

        @Autowired
        public void setWheel(Wheel wheel) {
                this.wheel = wheel;
        }
}
```

Field Injection

```
public class SimpleCar {

        @Autowired
        private Engine engine;

        @Autowired
        private Wheel wheel;
}
```

Use constructor for mandatory fields

Use setter for optional fields

Field injection should be mostly avoided

in der

Problems with Field Injection

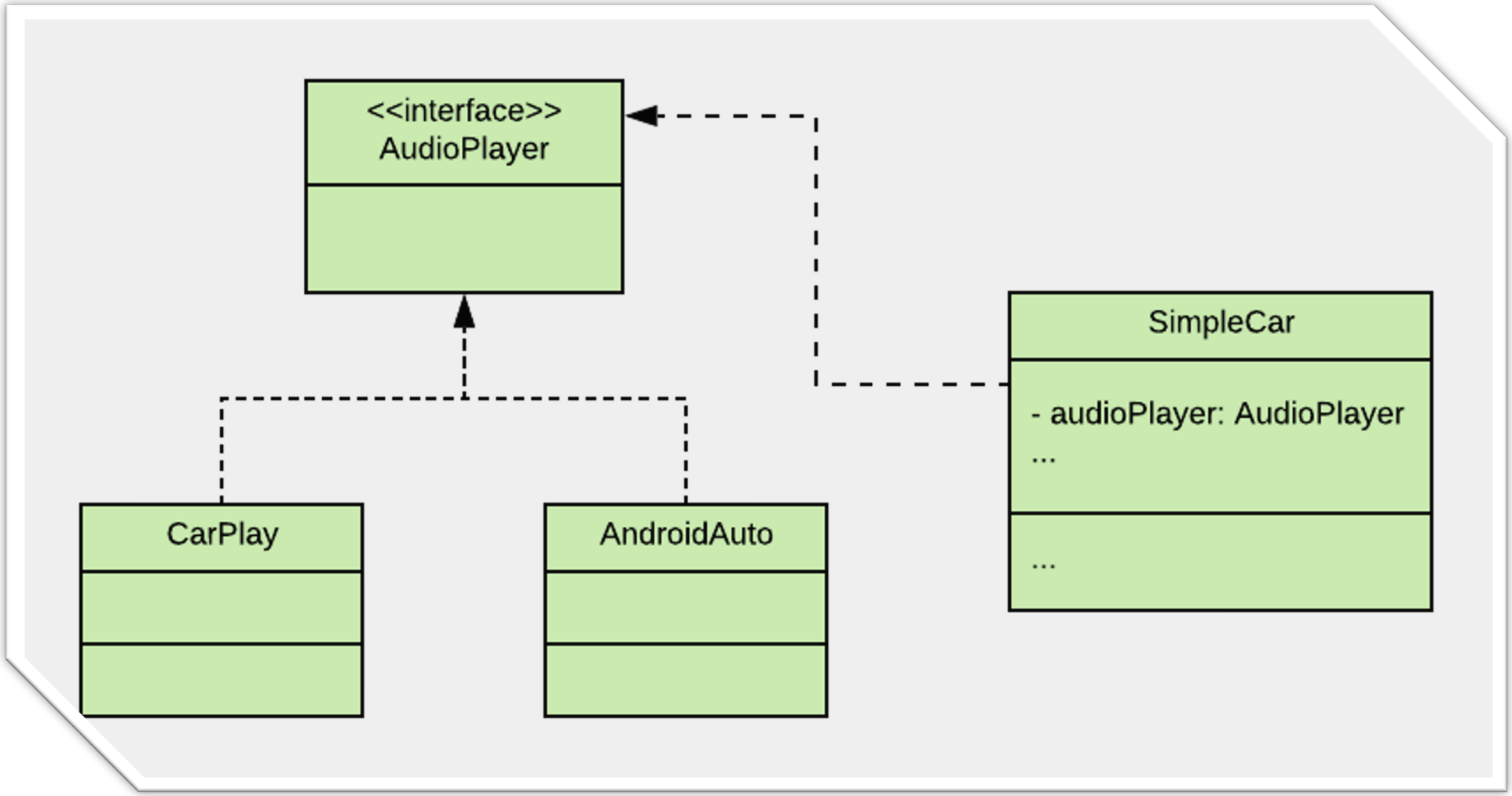
- Final/Immutable objects not possible.
- Hard to ensure valid Object construction
- Dependencies not clearly visible
- Unit testing not possible w/o reflection
- Heavily used field injection may be an indicator of violation of SRP.

Exercises 1 – 4

https://github.com/Michaeli71/AMTC_Spring_Workshop

Dependency Resolution

SimpleCar with AudioPlayer



Dependency Resolution Modes

Vor allem für XML-Config (aber auch programmatisch falls ohne `@Autowired`)

- byType
- byName
- constructor

Für Konflikte (gleich mehr) …

- @Primary
- @Qualifier

byName

```
<?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">

   <!-- Definition for textEditor bean -->
   <bean id = "textEditor" class = "com.tutorialspoint.TextEditor">
      <property name = "spellChecker" ref = "spellChecker" />
      <property name = "name" value = "Generic Text Editor" />
   </bean>

   <!-- Definition for spellChecker bean -->
   <bean id = "spellChecker" class = "com.tutorialspoint.SpellChecker"></bean>

</beans>
```

https://www.tutorialspoint.com/spring/spring_autowiring_byname.htm

byName

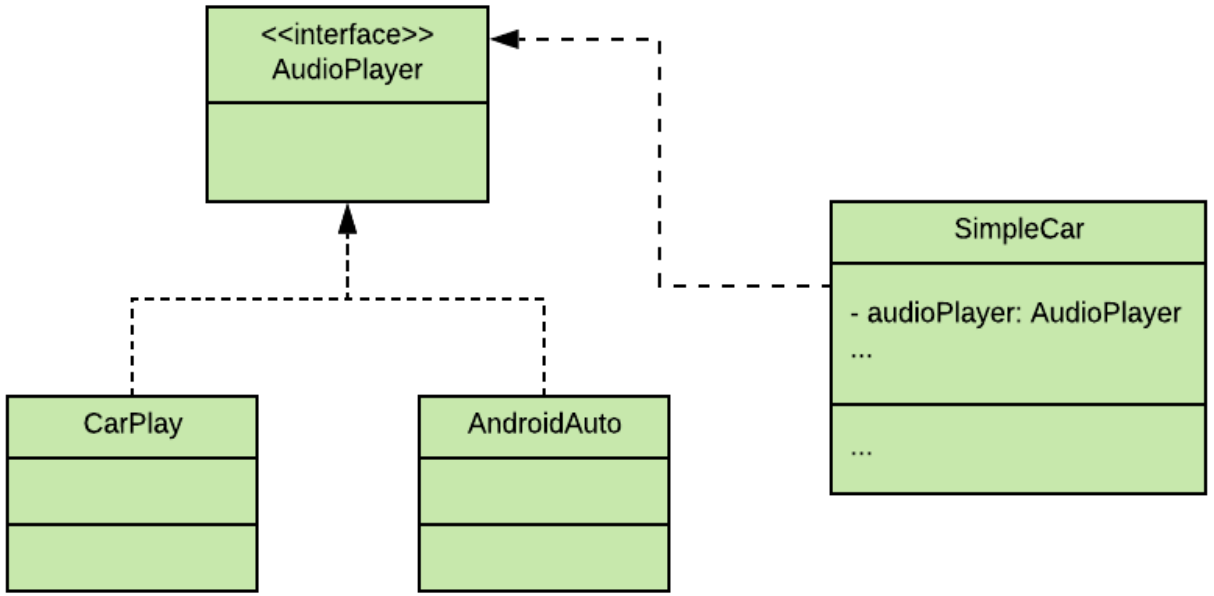
```xml
<?xml version = "1.0" encoding = "UTF-8"?>

<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-3.0.xsd">

   <!-- Definition for textEditor bean -->
   <bean id = "textEditor" class = "com.tutorialspoint.TextEditor"
      autowire = "byName">
      <property name = "name" value = "Generic Text Editor" />
   </bean>

   <!-- Definition for spellChecker bean -->
   <bean id = "spellChecker" class = "com.tutorialspoint.SpellChecker"></
      bean>

</beans>
```

https://www.tutorialspoint.com/spring/spring_autowiring_byname.htm

Demo

AutowiringDemo

Dependency Resolution in conflicts




```
public interface AudioPlayer {...}

@Component
public class AndroidAuto implements AudioPlayer { ... }

@Component
public class CarPlay implements AudioPlayer { ... }


@Component
public class SimpleCar {

        @Autowired
        private AudioPlayer audioPlayer;
}
```

org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type 'ch.karthi.AudioPlayer' available: expected single matching bean but found 2: androidAuto,carPlay

@Primary

```
public interface AudioPlayer {...}

@Primary
@Component
public class AndroidAuto implements AudioPlayer { ... }

@Primary
@Component
public class CarPlay implements AudioPlayer { ... }

@Component
public class SimpleCar {

        @Autowired
        private AudioPlayer audioPlayer;
}
```

org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type 'ch.karthi.AudioPlayer' available: expected single matching bean but found 2: androidAuto,carPlay

@Qualifier

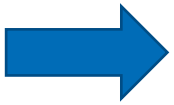
```
public interface AudioPlayer {...}

@Component("androidPlay")
public class AndroidAuto implements AudioPlayer { ... }


@Component("carPlay")
public class CarPlay implements AudioPlayer { ... }


@Component
public class SimpleCar {

        @Autowired
        @Qualifier("carPlay")
        private AudioPlayer audioPlayer;
}
```

org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type 'ch.karthi.AudioPlayer' available: expected single matching bean but found 2: androidAuto,carPlay

DI of scalar values

@Value

```
@Component
public class Wheel {

    private double radius = 2.34d;

...

}
```



```
@Component
@PropertySource("car.properties")
public class Wheel {

@Value("${wheel.radius}")
private double radius = -1.0d;

...

}
```

src/main/resources/car.properties

```
wheel.radius=2.34
```

```
@Value("${wheel.radius:2.34}")
private double radius = -1.0d;
```

Demo

DependencyConflictResolutionApp

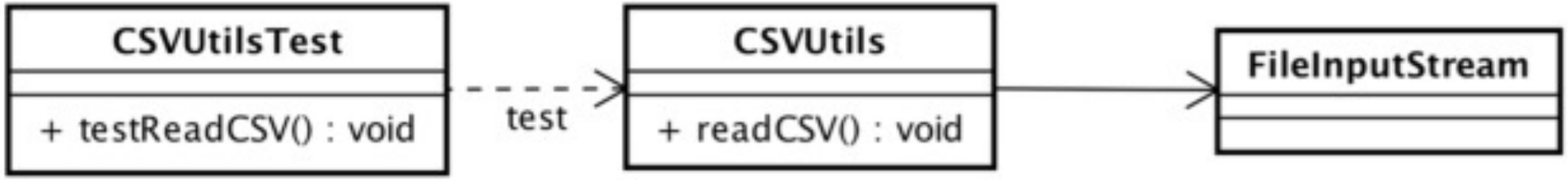
Sollbruchstellen / Injection Points für DI / Testing

Sollbruchstellen – Injection Points

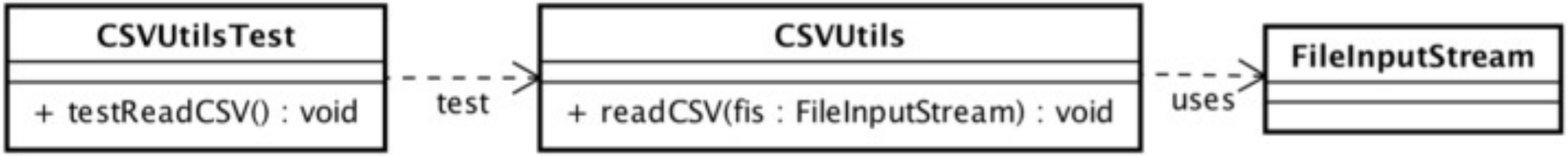
- Oftmals erschweren direkte Abhängigkeiten das Testen und auch DI
- Besser gegen Abstraktion arbeiten, also
  - Abstrakte Klasse
  - Interface
- Zum Teil gibt es diese und sie sind nur geeignet in das Design einzufügen
- Manchmal muss erst eine Abstraktion erzeugt und dann genutzt werden (**Dependency Injection**)

Sollbruchstellen – Injection Points

- Oftmals erschweren direkte Abhängigkeiten das Testen und auch DI

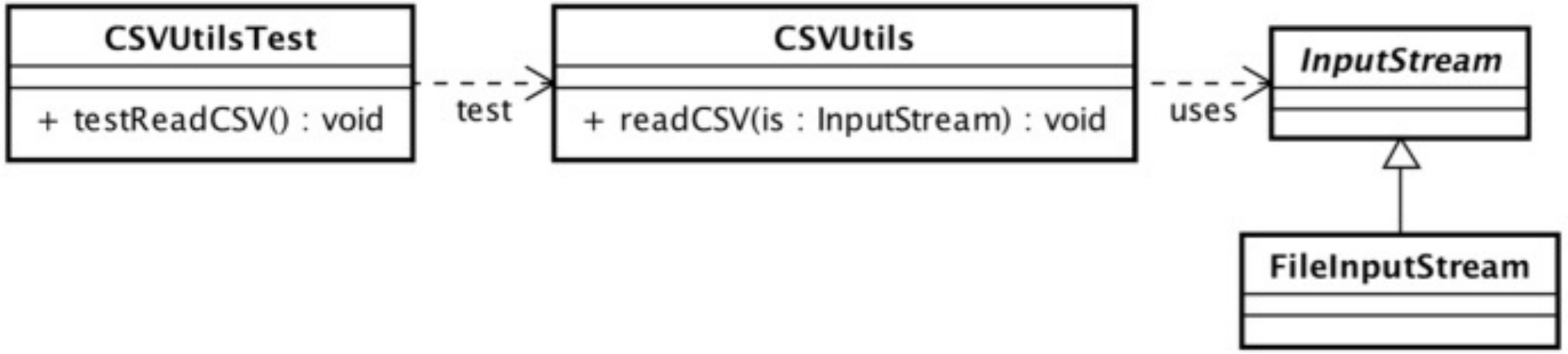


- Indirektion nutzen: Method Injection

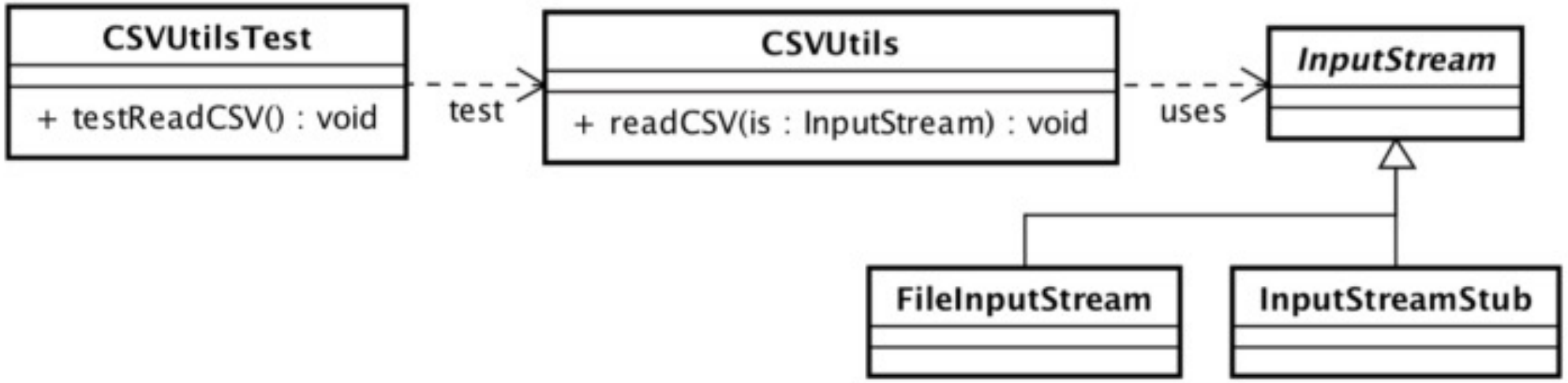


Sollbruchstellen – Injection Points

- Abstraktion nutzen



- Stub zur Testbarkeit nutzen



Exercises 5 – 8

https://github.com/Michaeli71/AMTC_Spring_Workshop