Spring Workshop

Quick Start / Intro

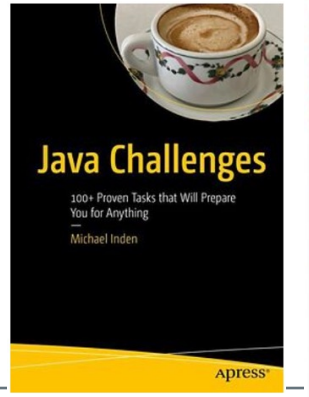
https://github.com/Michaeli71/ADC_BOOTCAMP_SPRING

Michael Inden

Speaker Intro

- Michael Inden, Year of Birth 1971
- Diploma Computer Science, C.v.O. Uni Oldenburg
- ~8 ¼ Years **SSE** at Heidelberger Druckmaschinen AG in Kiel
- ~6 ¾ Years **TPL, SA** at IVU Traffic Technologies AG in Aachen
- ~4 ¼ Years **LSA / Trainer** at Zühlke Engineering AG in Zurich
- ~3 Years **TL / CTO** at Direct Mail Informatics / ASMIQ in Zurich
- Independent **Consultant, Conference Speaker and Trainer**
- **Since January 2022 Head of Development at Adcubum in Zurich**
- Author @ dpunkt.verlag and APress

E-Mail: michael_inden@hotmail.com
Blog: https://jaxenter.de/author/minden



https://github.com/Michaeli71/ADC_BOOTCAMP_SPRING

Agenda

Workshop Contents

- **PART 1: Introduction**
  - History of Spring
  - Spring Architecture
- **PART 2: Dependency Injection**
  - Basics of IoC / DI
  - Spring Beans
  - Configuration
  - Types of Dependency Injection
  - Dependency Resolution Strategies

https://github.com/Michaeli71/ADC_BOOTCAMP_SPRING

Workshop Contents

- **PART 3: Bean Initialization / Scopes + Lifecycle**
  - Bean Initialization & Laziness
  - Circular Dependencies
  - Bean Scopes
  - Bean Lifecycle Callbacks

- **PART 4: Spring MVC**

https://github.com/Michaeli71/ADC_BOOTCAMP_SPRING

PART 1:
Introduction

Father of Spring

Rod Johnson

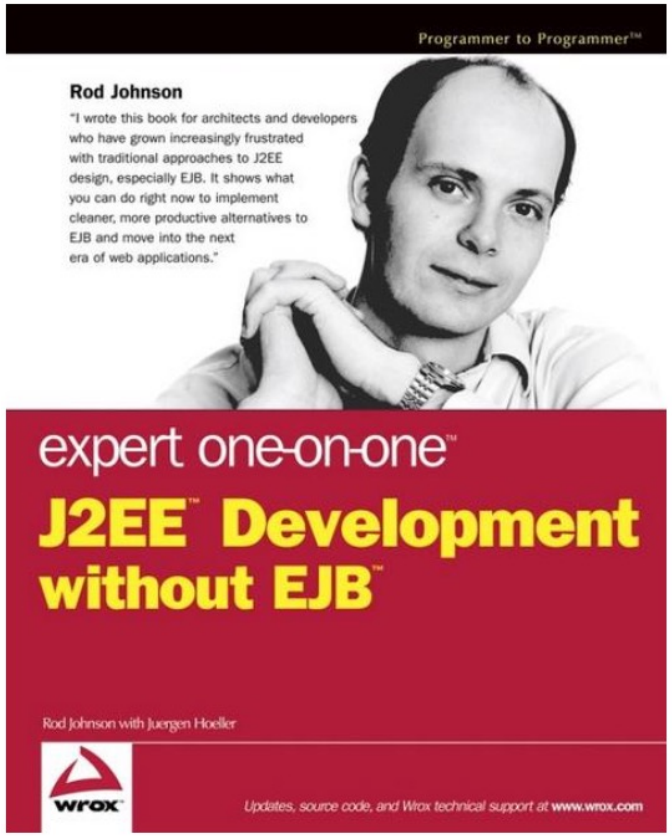
@springrod

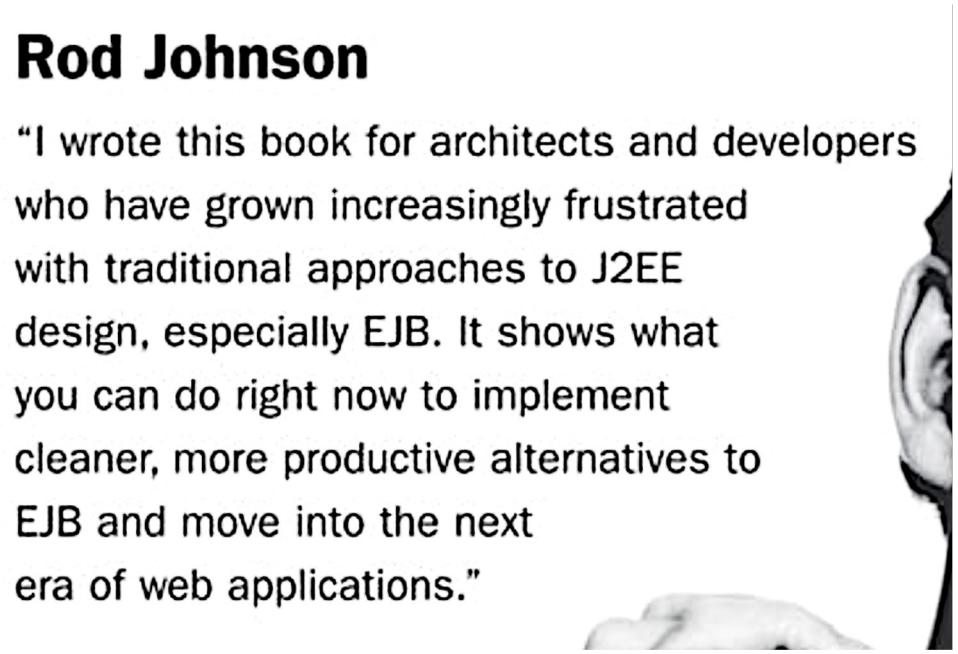
CEO, Atomist. Creator of Spring, Cofounder/CEO at SpringSource, Investor, Author

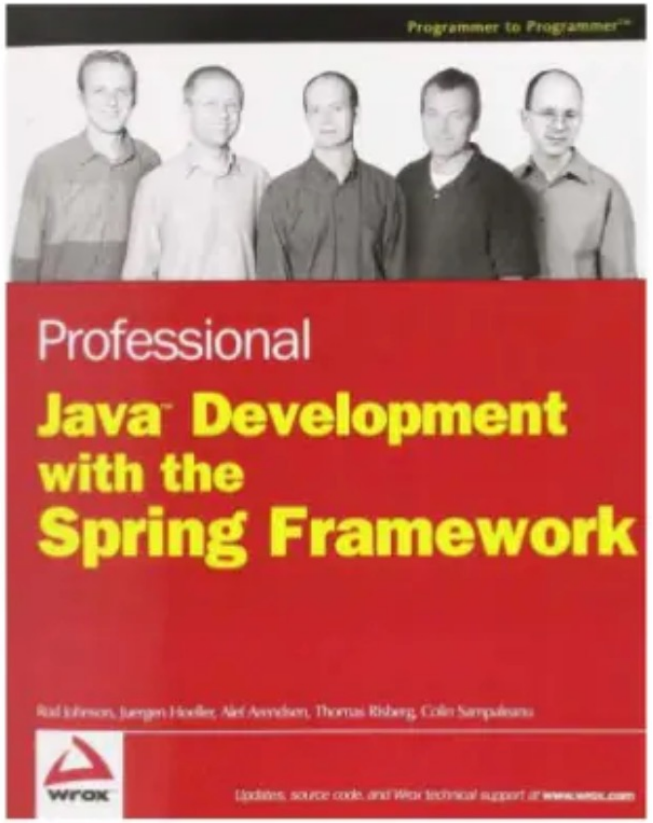
Sydney / Bay Area

atomist.com

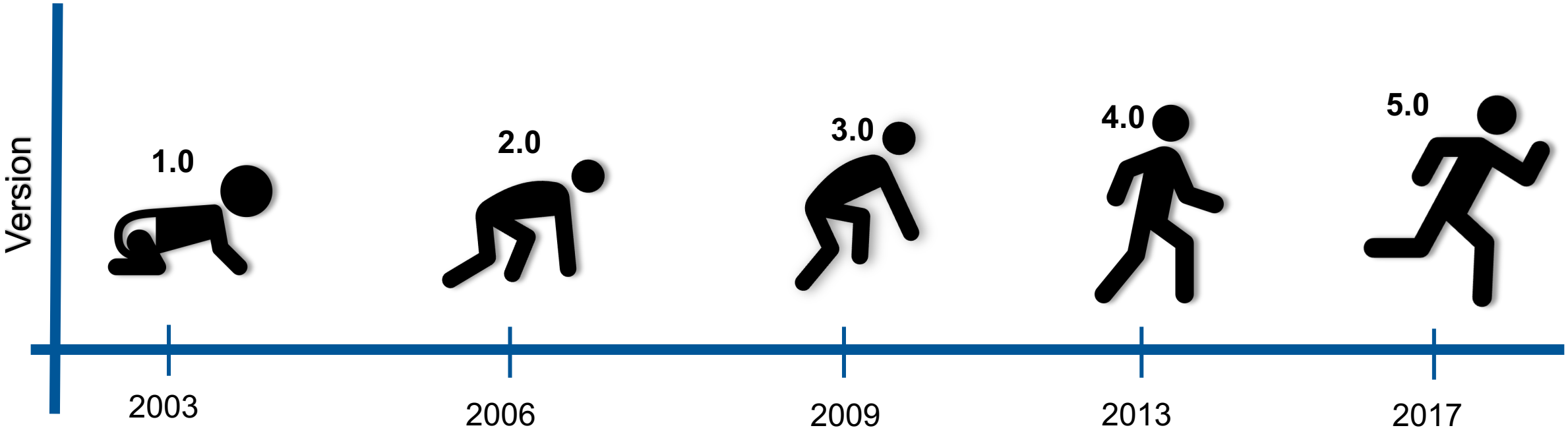
Spring‘s Origin







Spring‘s Emergence



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 of Libraries

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

History of the company

2002 2008 2009 2013

Advantages of Spring

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

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

https://spring.io/why-spring

Major Spring Projects

SPRING FRAMEWORK

SPRING BOOT

SPRING CLOUD

SPRING DATA

SPRING CLOUD DATA FLOW

SPRING INTEGRATION

SPRING BATCH

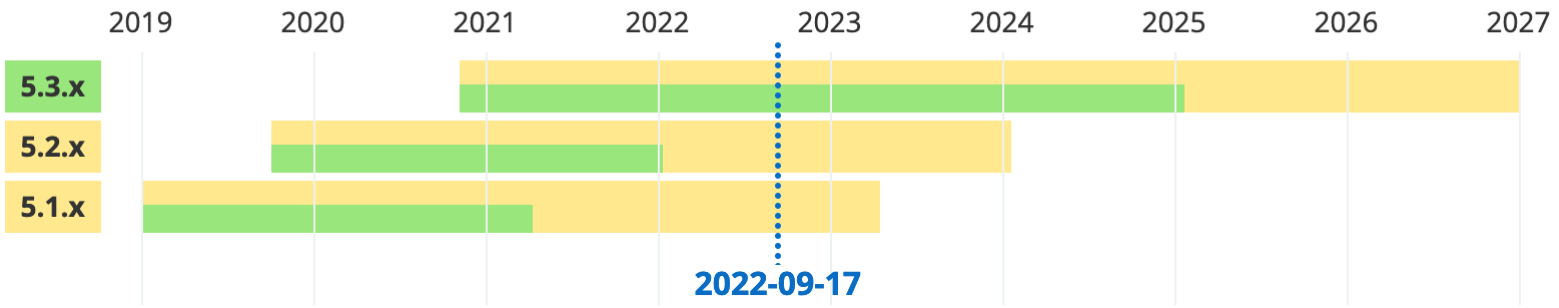
SPRING SECURITY

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	👎	👎	👍	👍	👍	👍	👍

Spring Framework – Version Overview

Branch	Initial Release	End of Support	End Commercial Support *
5.3.x	2020-10-27	2024-12-31	2026-12-31
5.2.x	2019-09-30	2021-12-31	2023-12-31
5.1.x	2018-09-21	2020-12-31	2022-12-31



OSS support

Free security updates and bugfixes with support from the Spring community. See VMware Tanzu OSS support policy.

Commercial support

Business support from Spring experts during the OSS timeline, plus extended support after OSS End-Of-Life.

Publicly available releases for critical bugfixes and security issues when requested by customers.

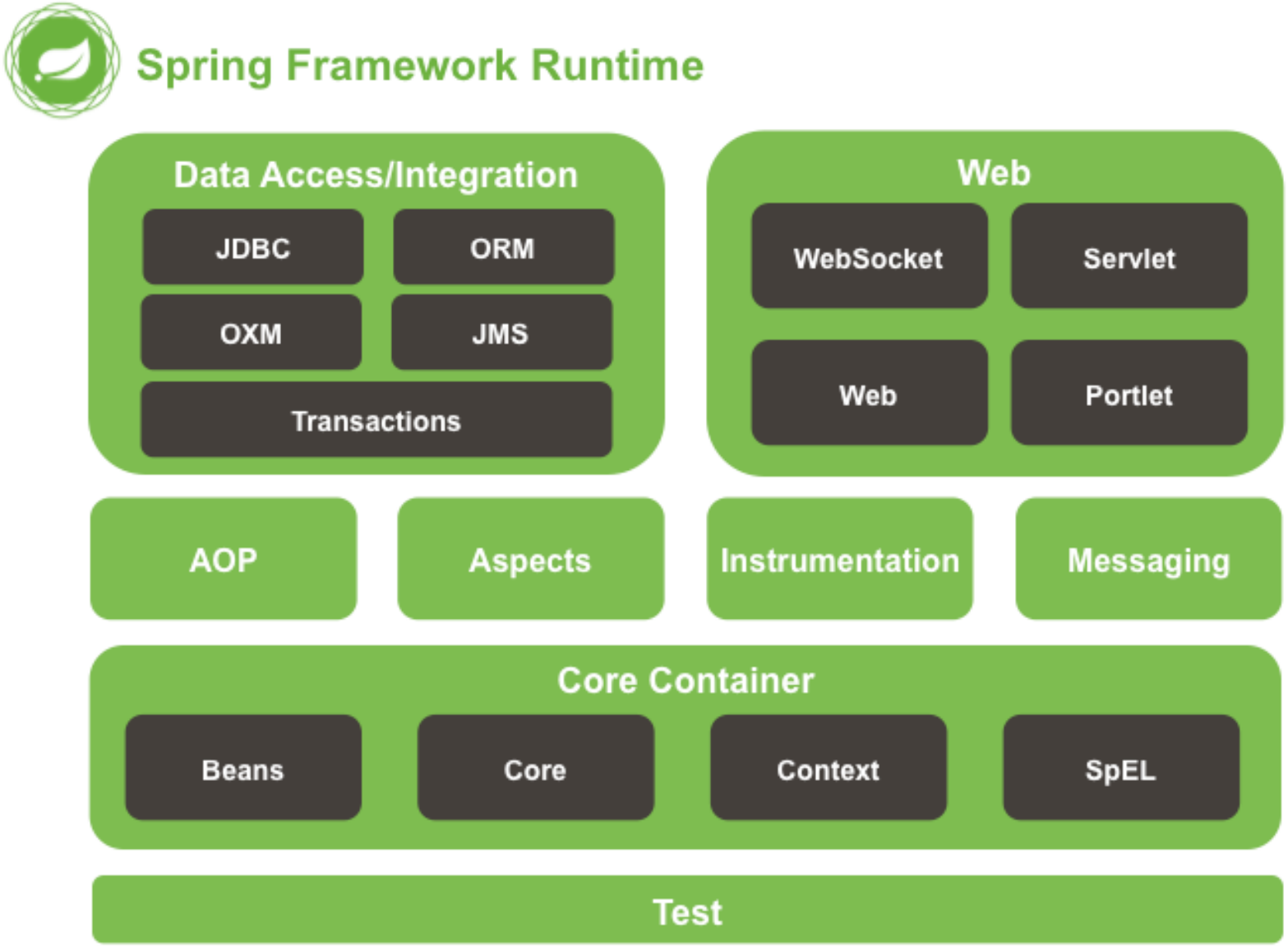
https://spring.io/projects/spring-framework#support

Is Spring working with the Java module system?

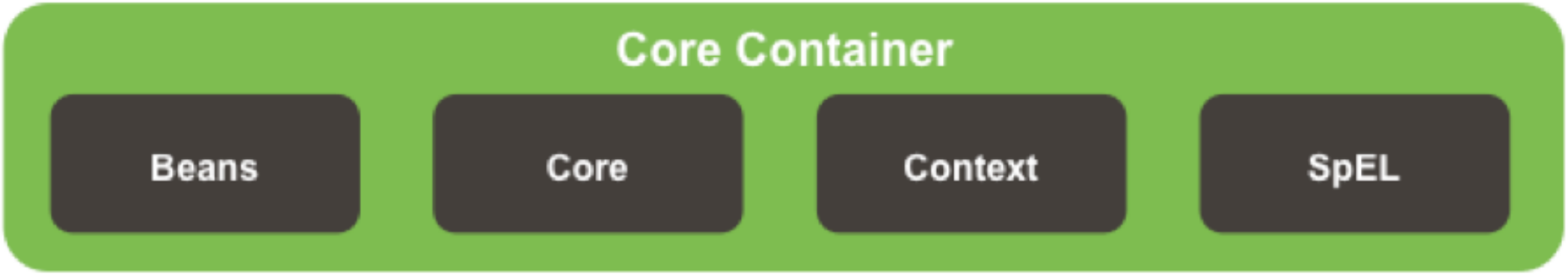
Yes, Spring Framework 5 ships with **automatic module name entries** in the manifests of our Spring Framework 5 jars. The public API surface of the Spring libraries remains unchanged.

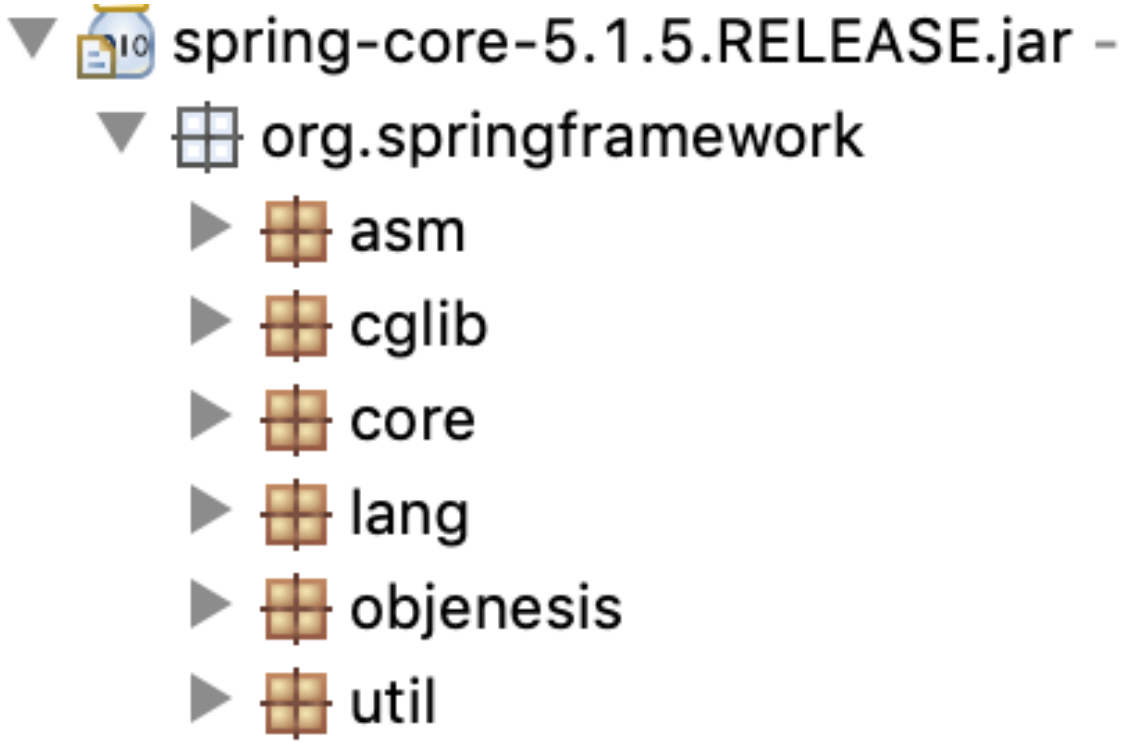
Spring Architecture

Spring Architecture



Spring Core Container Details



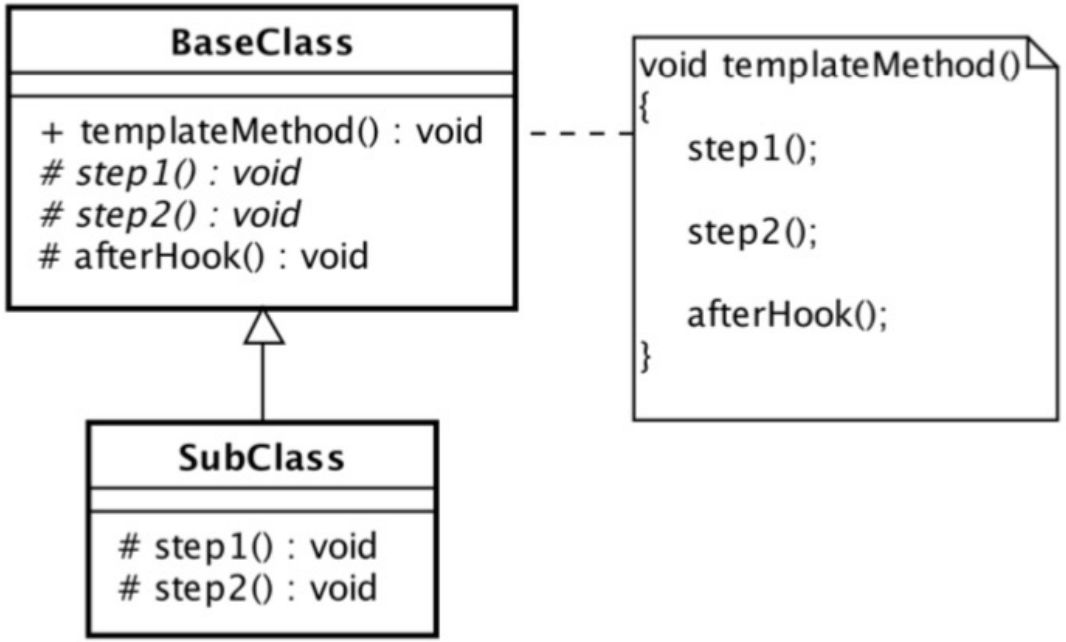


Inversion of Control

Inversion of Control

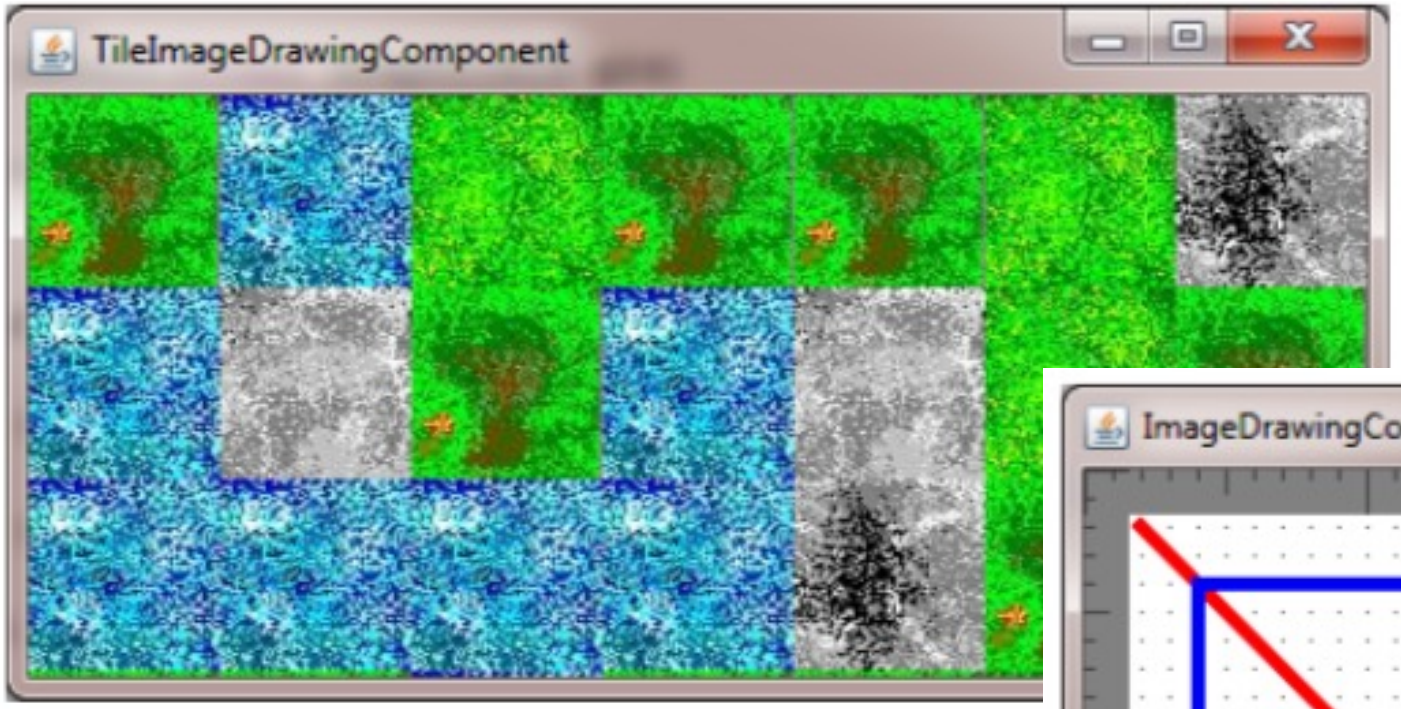
- Hollywood Principle («Don't call us, we'll call you»)

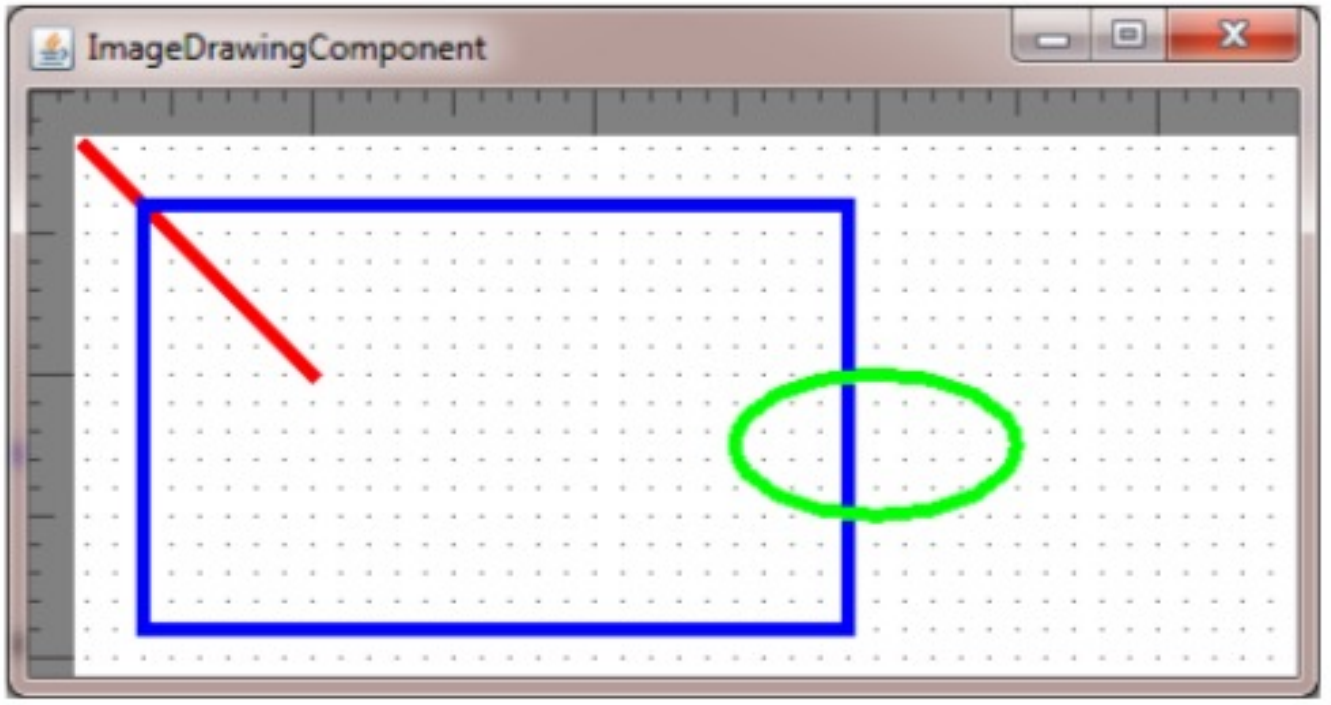
Inversion of Control: Template Method Pattern




```
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);
```

Example

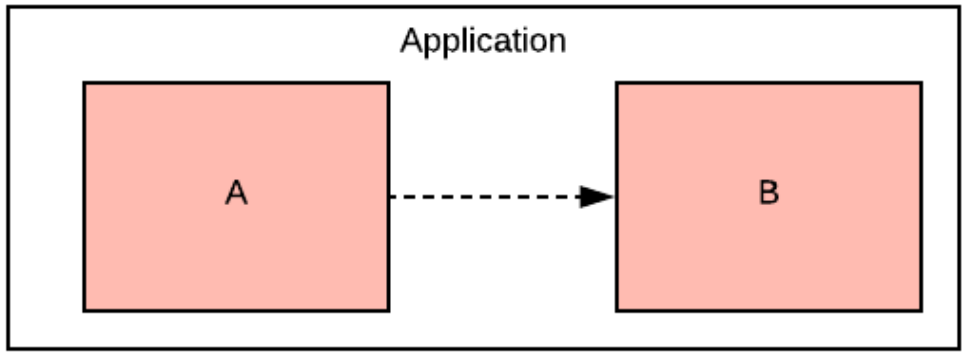


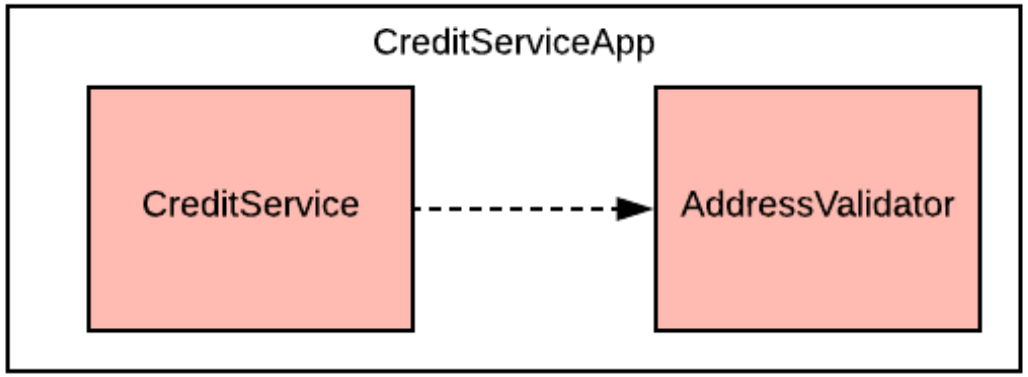


PART 2:
Dependency Injection

Dependencies – Short Recap

- Dependency ⇔ one class uses another





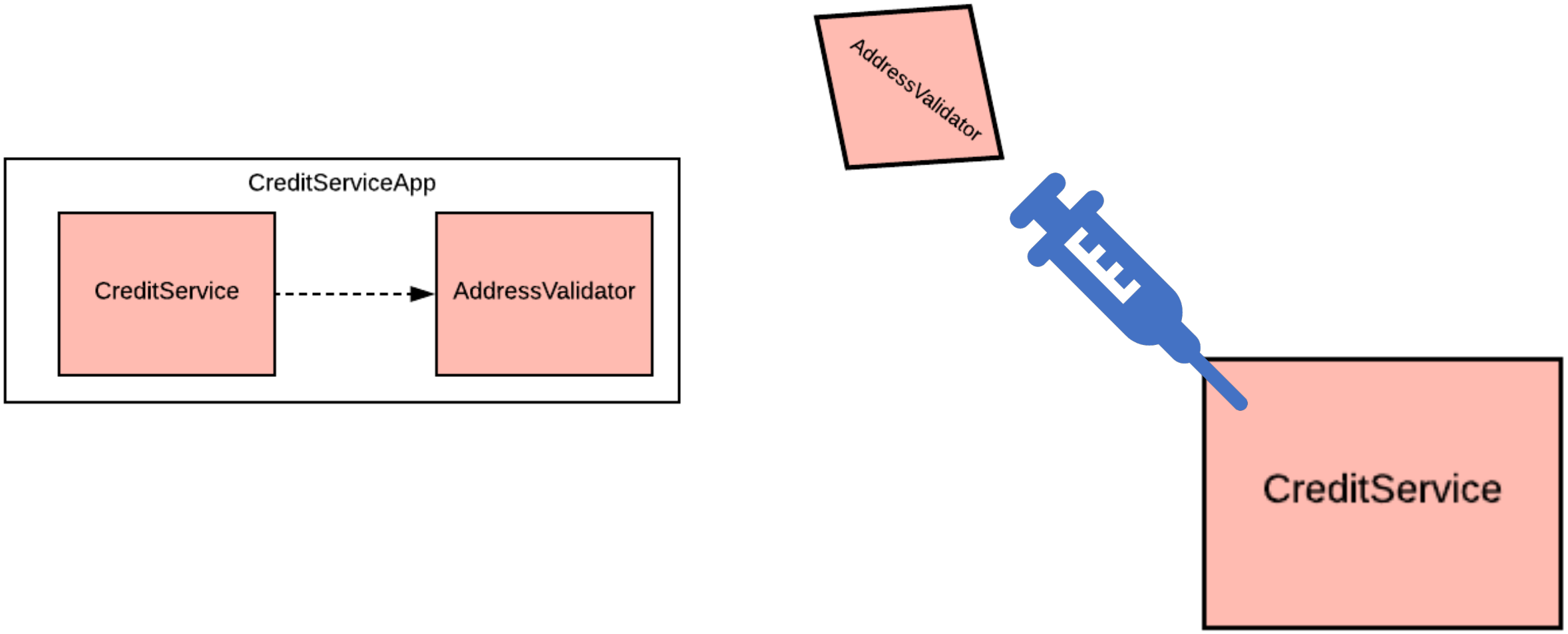
Central question: How do we get the dependency?

In the past often: By calling 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();
        }

}
```

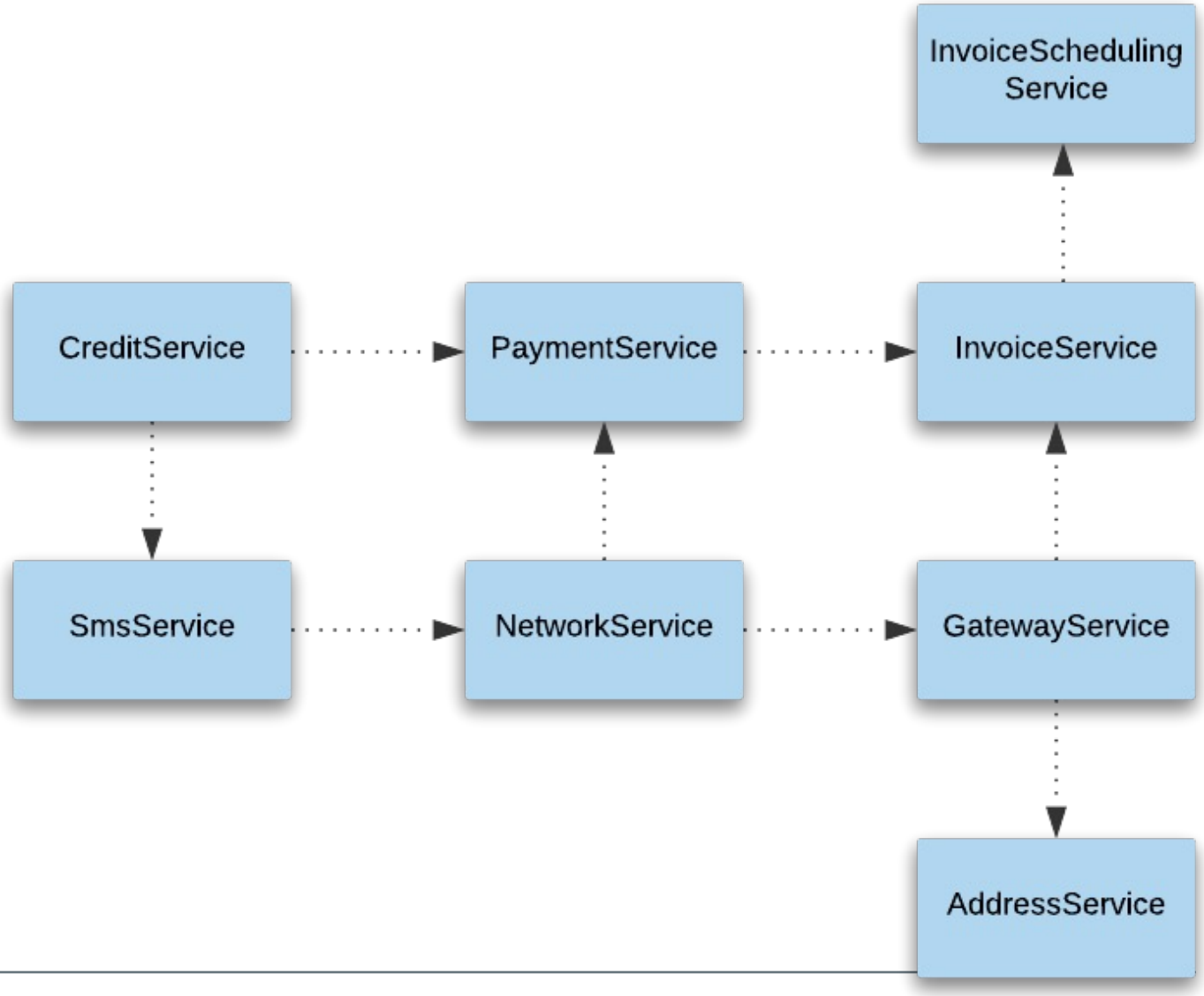
Manuelle Dependency Injection

Manual DI

```
public class CreditServiceApp {

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

}
```

What is a main problem with calling new?

What about such an object graph?



Problems of manual DI

- Complex
- Many manual object constructions & bolierplate code
- Tight coupling
- Poor readability
- Sometimes internal construction is visible to the outside
- Sometimes difficult to test (object constructions, flows, ...)

Dependency Injection in Spring

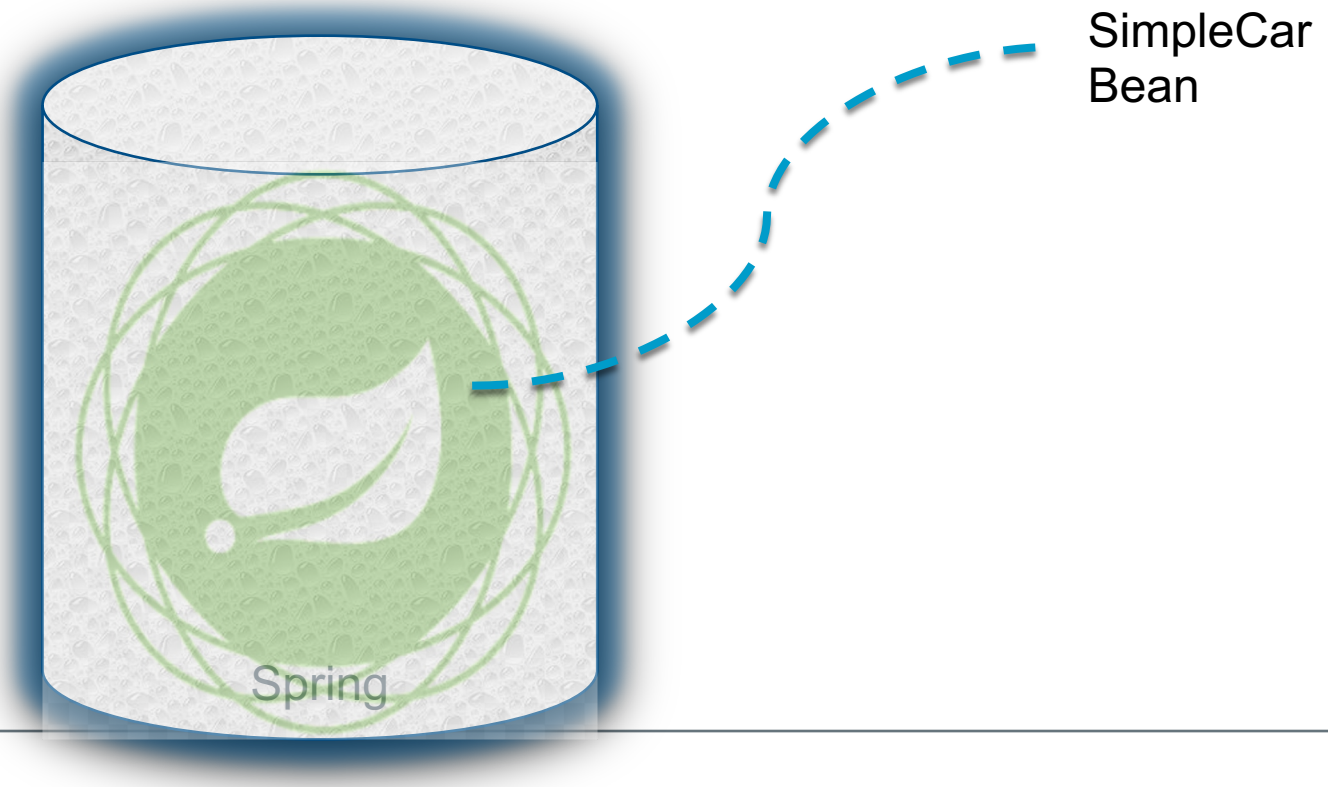
IoC Container

- Enables Dependency Injection
- Manages other objects
- Builds the object graph
- Is also (only) an object

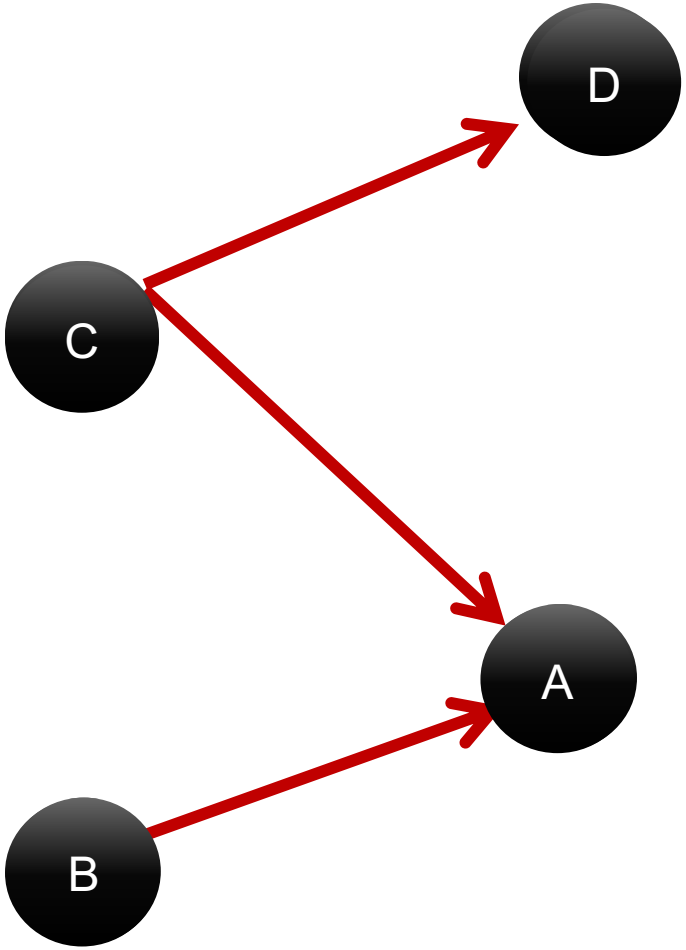
Spring Beans

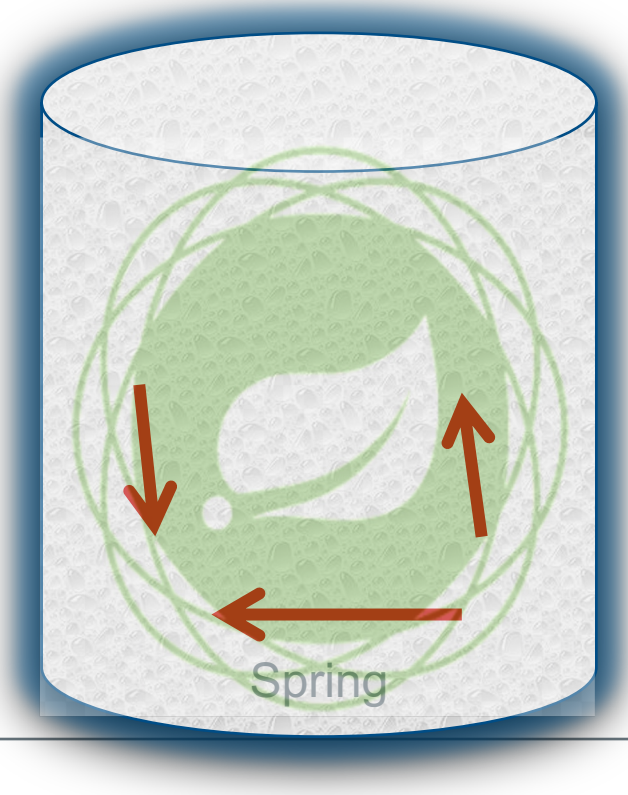
- Plain Old Java Object (POJO) which is/will be **managed** by the Spring is called **Spring Bean**.

SimpleCar



IoC Container – Topological sorting and object creation





How does the IoC container know how the beans need to be configured and are related?

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

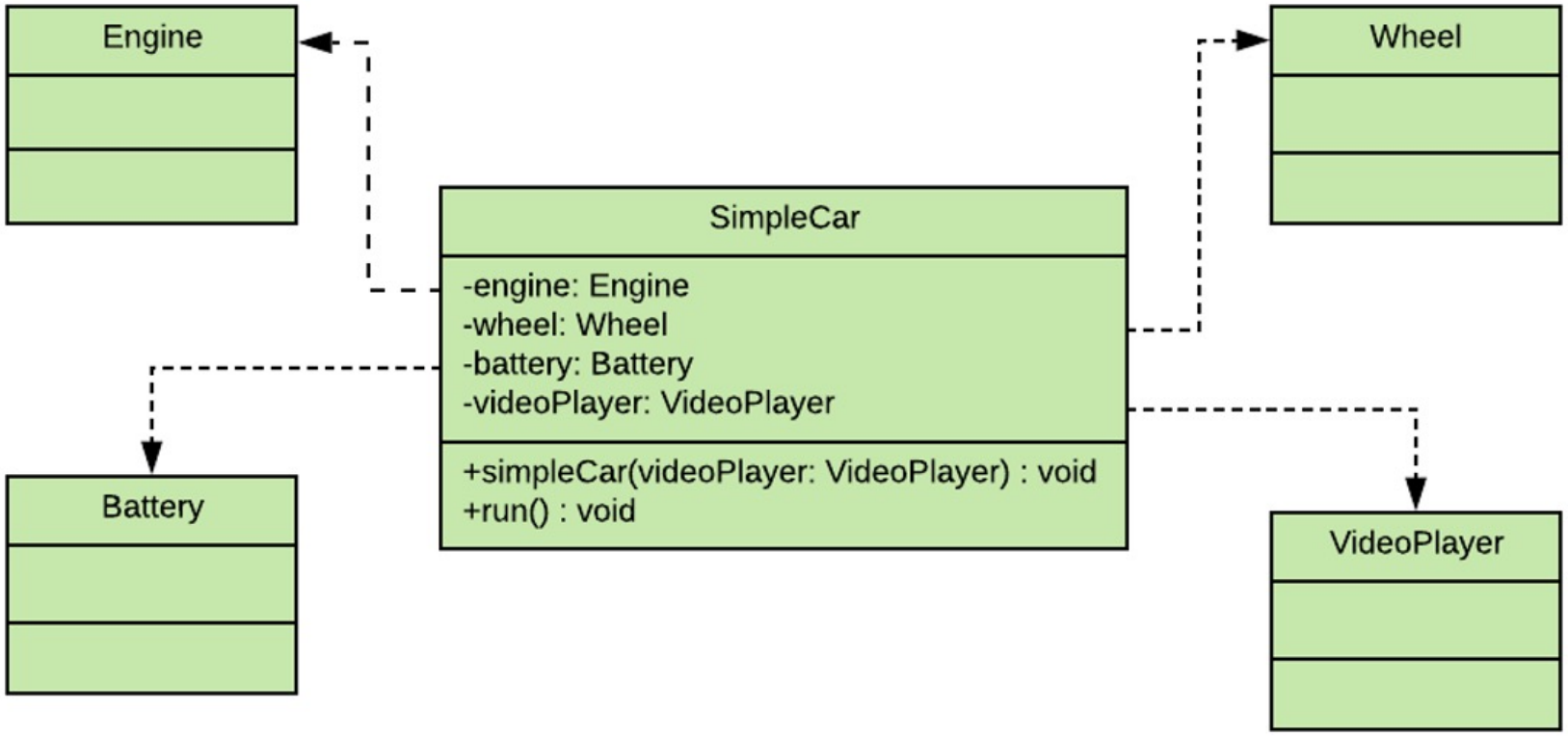
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:c="http://www.springframework.org/schema/c"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans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

+

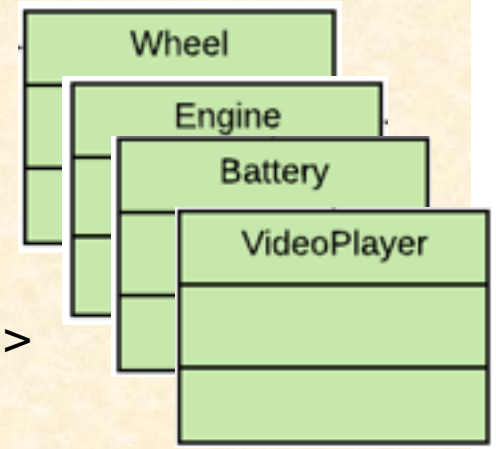
```java
@Component
public class Person {

...

}
```

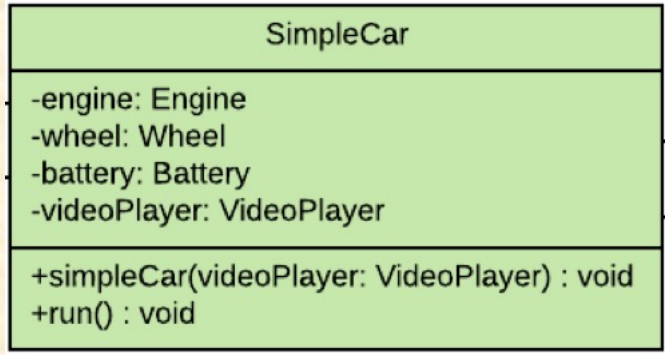
Example for XML-Konfiguration

XML Configuration



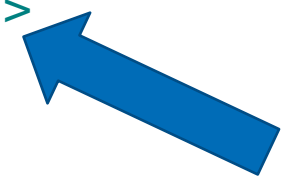
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

        <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>
```





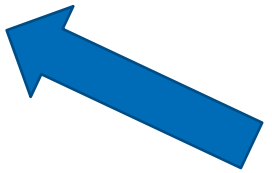
XML Configuration Special feature with construction

```
<?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 Special feature with construction

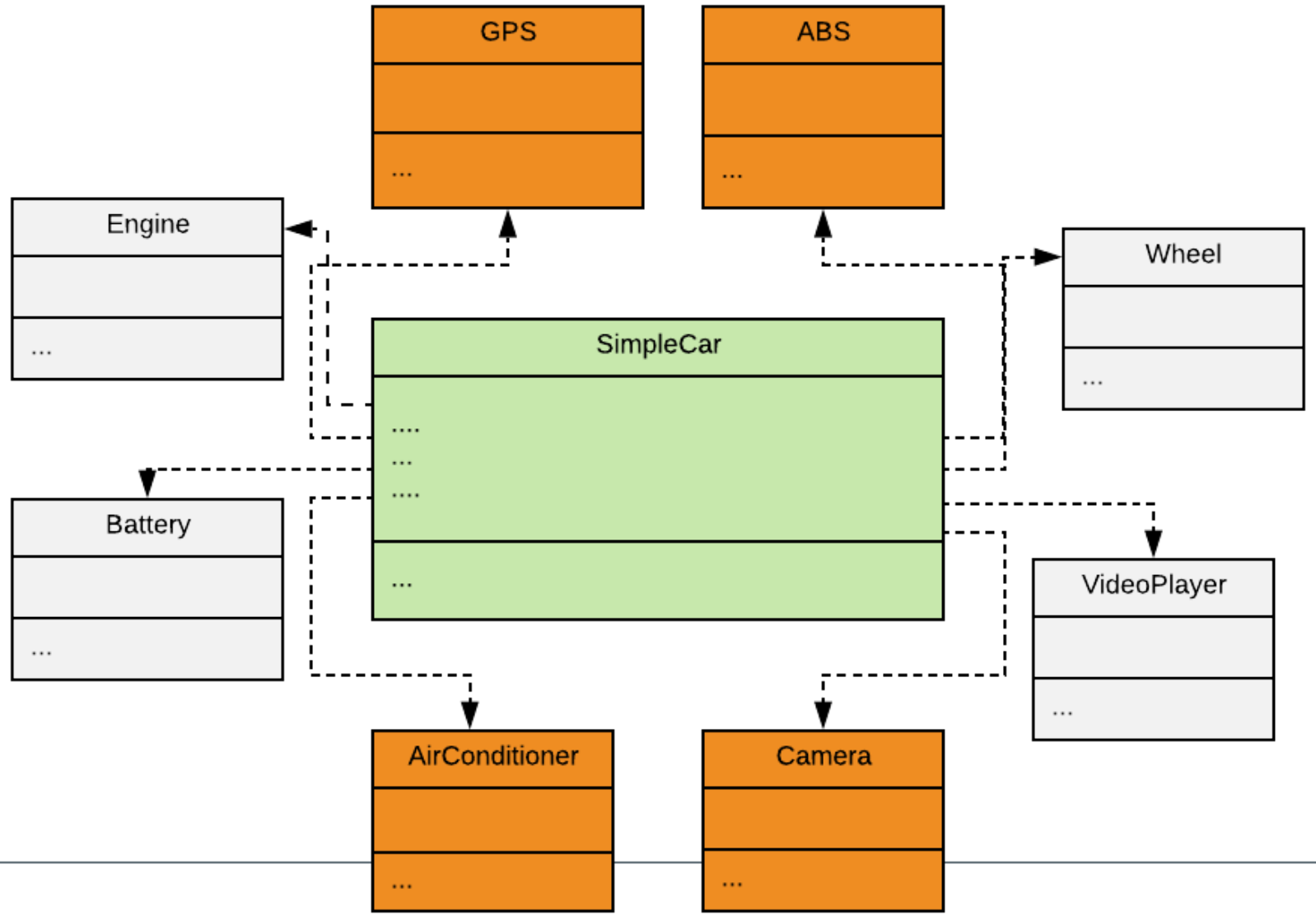
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:c="http://www.springframework.org/schema/c"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    …
    <bean id="emailService" class="ch.asmiq.EMailService" />
    <bean id="smsService" class="ch.asmiq.SmsService" />

    <bean id="academyService" class="ch.asmiq.AcademyService"
        c:emailService-ref="emailService">
        <constructor-arg name="smsService" ref="smsService" />
    </bean>

</beans>
```



Demo

`SimpleXmlConfigApp`
`ServiceXmlConfigApp`

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>
```

Wouldn't it be nice if Spring could take some of the work off our hands?

```
<?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>
```

Can you please make it a little easier?

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`

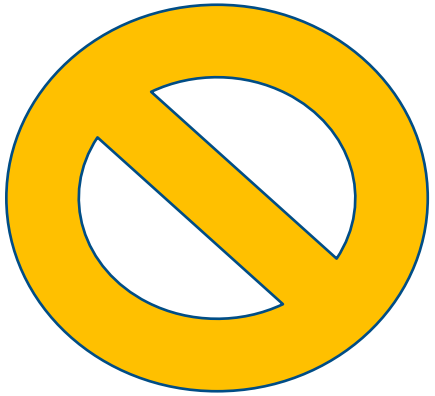
- `@Controller`
- `@RestController`

- `@Configuration`

How it is possible to define the dependencies in Java?

@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;
        }
}
```



Can we do without XML entirely?

Java Configuration + @ComponentScan

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

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

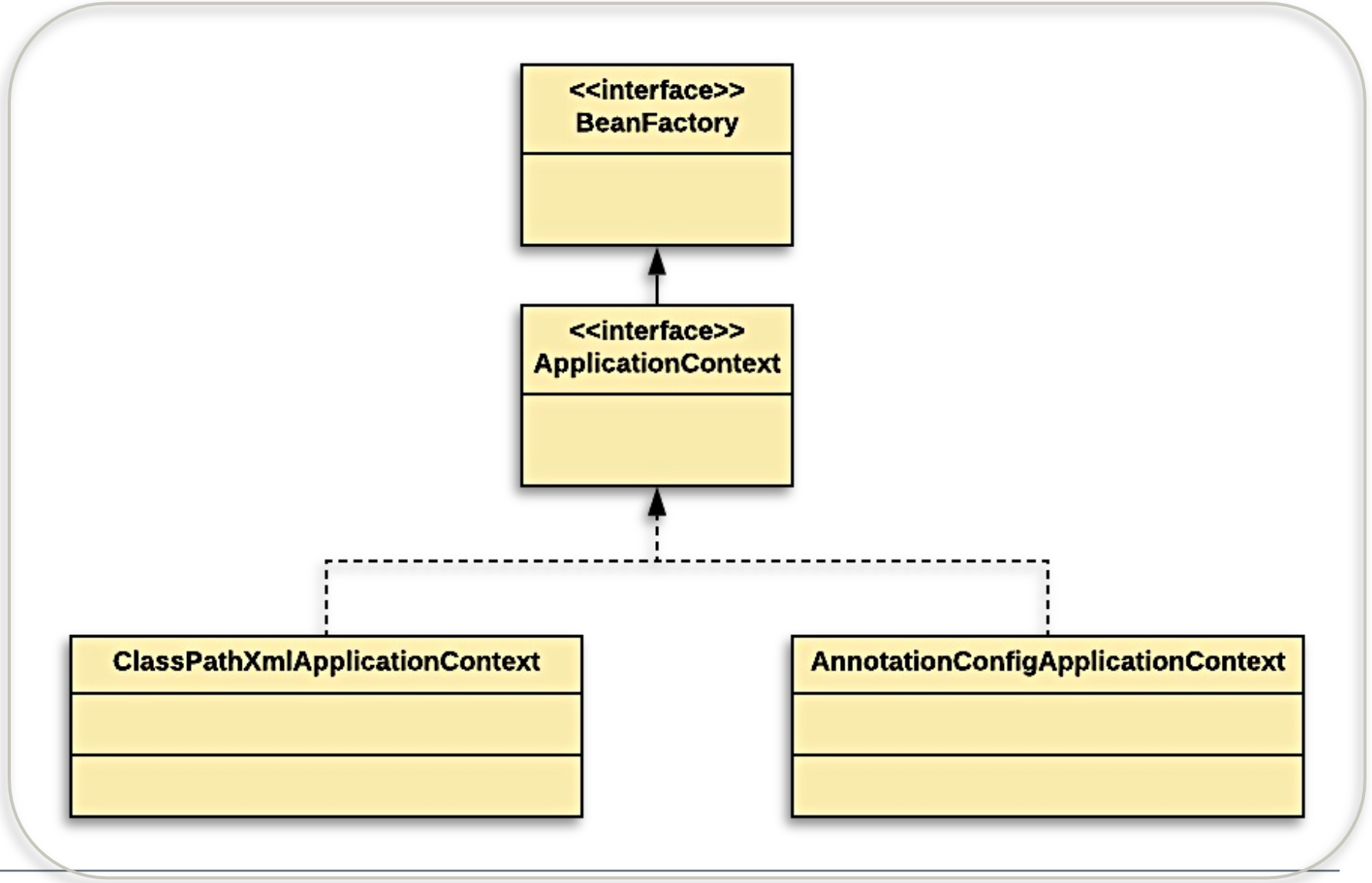
```
<bean id="engine" class="ch.karthi.Engine"></bean>
…
```

```
@Component
public class SimpleCar {

        @Autowired
        private Engine engine;
}
```

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

IoC Container instantiation & usage

IoC Container in Spring



XML

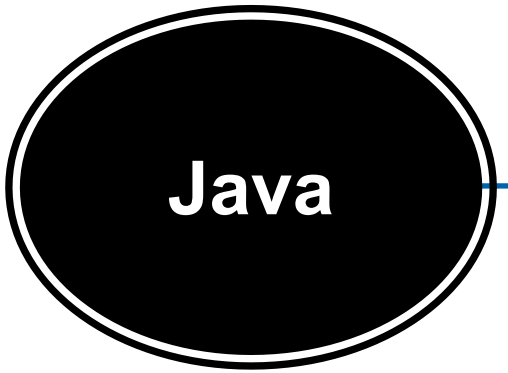
```
var container = new ClassPathXmlApplicationContext("config.xml");

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>
```



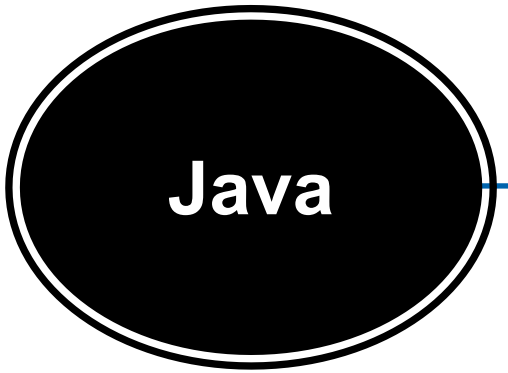
@ComponentScan("ch.karthi")
public class AppConfig {…}

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

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

// start the simpleCar
```

@Configuration
public class AppConfig
{…}

Is it possible to combine multiple configurations?

```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

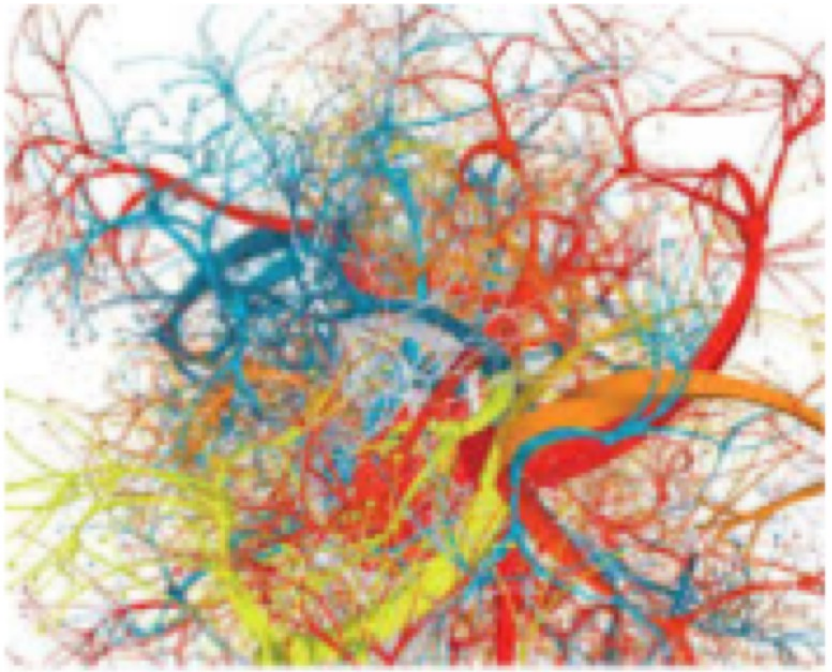
`MultipleAnnotationConfigDemo`

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

Types of Dependency Injection



Types of Dependency Injection

- Constructor-Injection
- Setter / Method-Injection
- Field-Injection

Constructor Injection

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

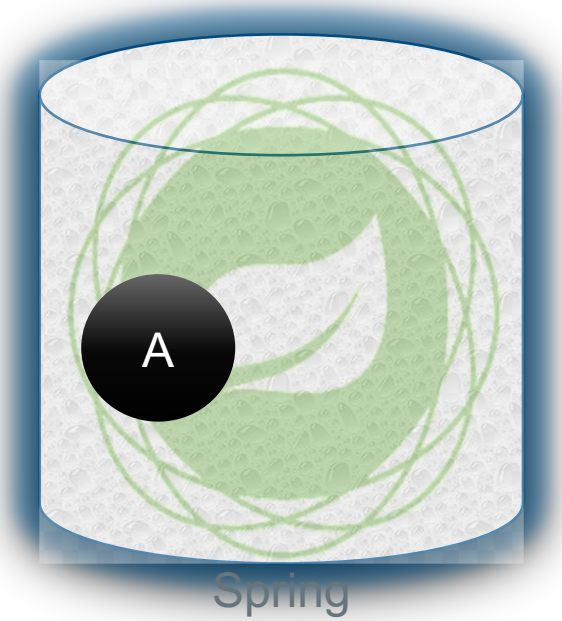
Rule: @Autowired should be present at least on one constructor, when more than one constructor has been declared.

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

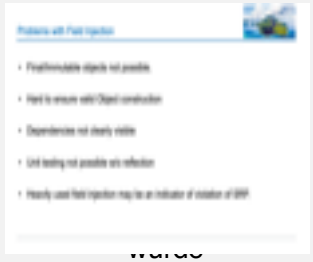
```
public class SimpleCar {

        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



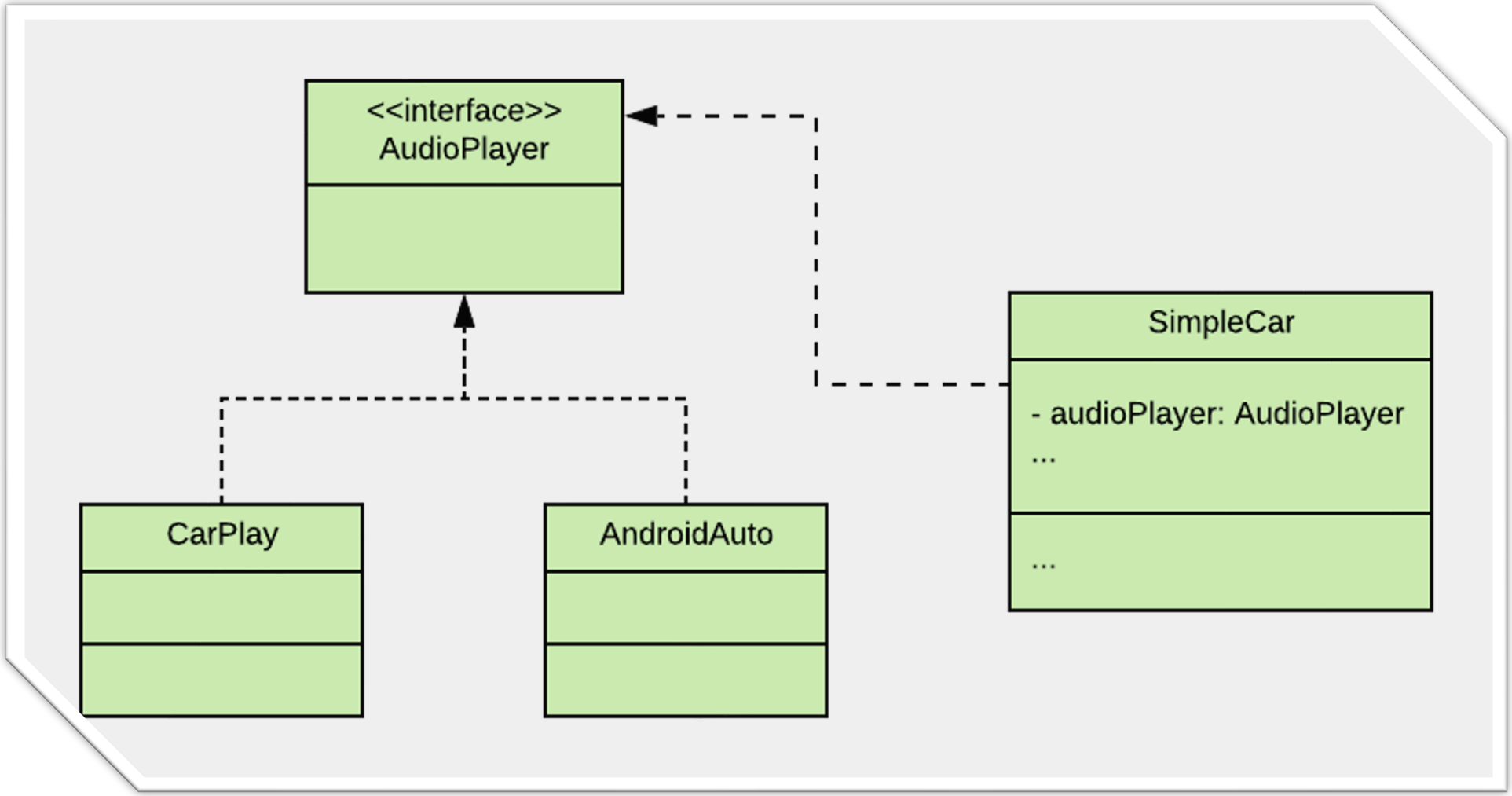
Field injection should be mostly avoided

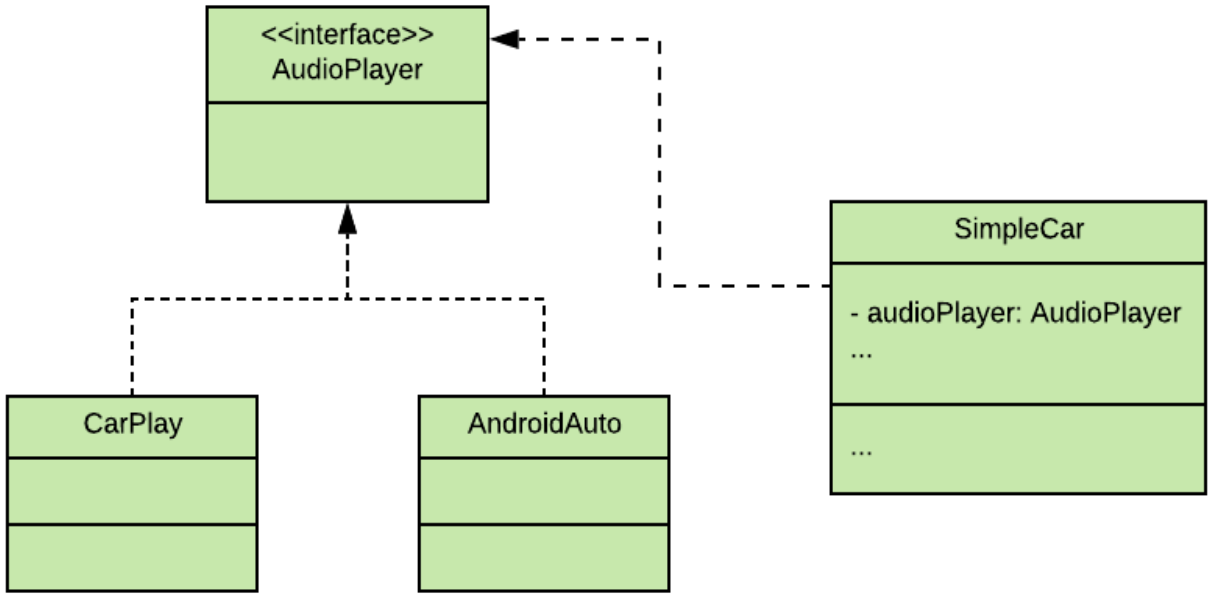
Problems with Field Injection

- 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.

Dependency Resolution

SimpleCar with AudioPlayer






```
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

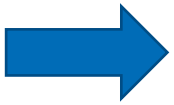
```
public interface AudioPlayer {...}

@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

PART 3:
Bean Initialization / Scopes + Lifecycle

- Bean Initialization & Laziness
- Circular Dependencies
- Bean Scopes
- Bean Lifecycle Callbacks

Bean Initialization

Bean Initialization

- No thoughts so far
- always used the default
- by default, beans are initialized DIRECTLY AT STARTUP
- **helps to avoid dependency configuration errors**

Example class

```
@Component
public class SimpleCar {

    public void run() {
          //start battery
          //start engine
          //start the wheel
    }

    public void playMusic(){
          audioPlayer.playMusic();
    }

    @Autowired
    public void setAudioPlayer(AudioPlayer pl){
       this.audioPlayer = pl;
    }

    //setter injected battery,engine,
    //wheel & audioplayer
}
```

Usage in Application

```
@Component
public class AndroidAuto implements AudioPlayer {

        public AndroidAuto() {
          //check the internet connection (3 secs)
          //google account login (2 secs)
          //access for microphone and camera (4 secs)
          //load the playlists (2 secs)
          //cache the playlists (5 secs)
        }
}

var context = new AnnotationConfigApplicationContext(App.class);
SimpleCar simpleCar = context.getBean(SimpleCar.class);
simpleCar.run();

//later if user press the play button
simpleCar.playMusic();
```

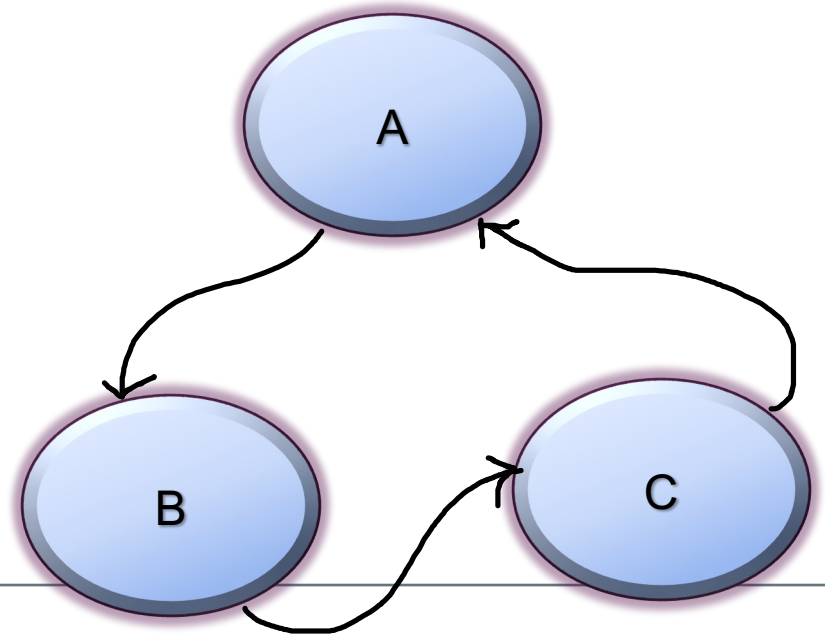
Laziness is good ☺

Lazy Init as Remedy

```
@Component
public class SimpleCar {

  public void run() {
     //start battery
     //start engine
     //start the wheel
   }

   public void playMusic(){
      if (audioPlayer == null)
          audioPlayer = context.getBean(AudioPlayer.class);
          audioPlayer.playMusic();
      }

    //setter injected battery,engine,wheel
}
```

```
@Lazy
@Component
public class AndroidAuto implements AudioPlayer {

        public AndroidAuto() {
           //check the internet connection (3 secs)
           //google account login (2 secs)
           //access for microphone and camera (4 secs)
           //load the playlists (10 secs)
           //cache the playlists (5 secs)
        }
}
```

Circular Dependencies



Circular Dependency Example

```
@Component
public class Student {

  private Department department;

  @Autowired
  public Student(Department department) {
      ...
  }

}
```

```
@Component
public class Department {

  private Student student;

  @Autowired
  public Department(Student student) {
      ...
  }

}
```

Caused by: org.springframework.beans.factory.BeanCurrentlyInCreationException: Error creating bean with name 'department': Requested bean is currently in creation: Is there an unresolvable circular reference?

Circular dependencies solution

Modify some dependencies to be **injected through setters** => after conctruction

```
@Component
public class Student {

  private Department department;

  public Student(Department department) {
       ...
  }

  @Autowired
  public void setDepartment(Department dpt){
       this.department = dpt;
  }

}
```

Circular dependencies solution

Lazy initialize certain dependencies => Proxy gets created (*Interface as Indirection)

```
@Component
public class Student {

    private Department department;

    public Student(@Lazy Department department) {
       ...
    }
}
```

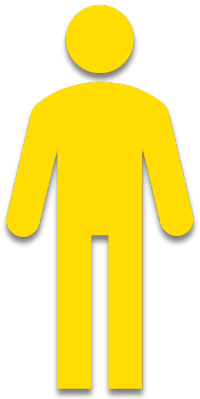
```
@Component
public class Student {

    private Department department;

    private int age=36;

    public Student(@Lazy Department department) {
        this.department = department;
    }

    public int getAge() {
        return age;
    }
```

department= Department$$EnhancerBySpringCGLIB$$f030f2cb (id=38)
CGLIB$BOUND= false
CGLIB$CALLBACK_0= CglibAopProxy$DynamicAdvisedInterceptor (id=58)
CGLIB$CALLBACK_1= CglibAopProxy$DynamicUnadvisedInterceptor (id=63)

Bean Scopes

Bean Scopes

Default

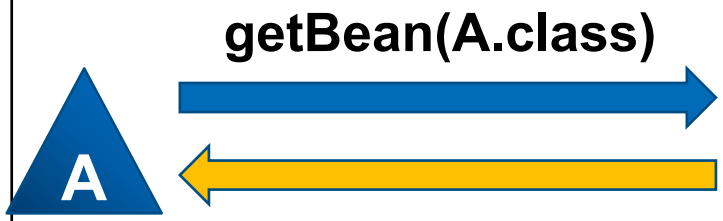


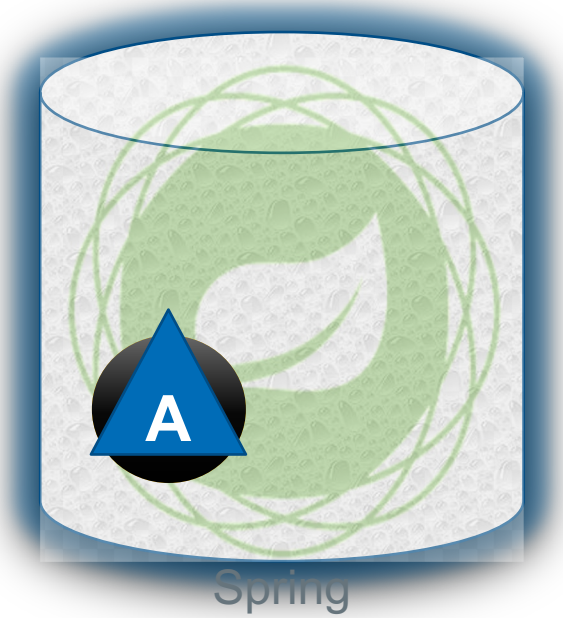
Singleton

Prototype

Singleton

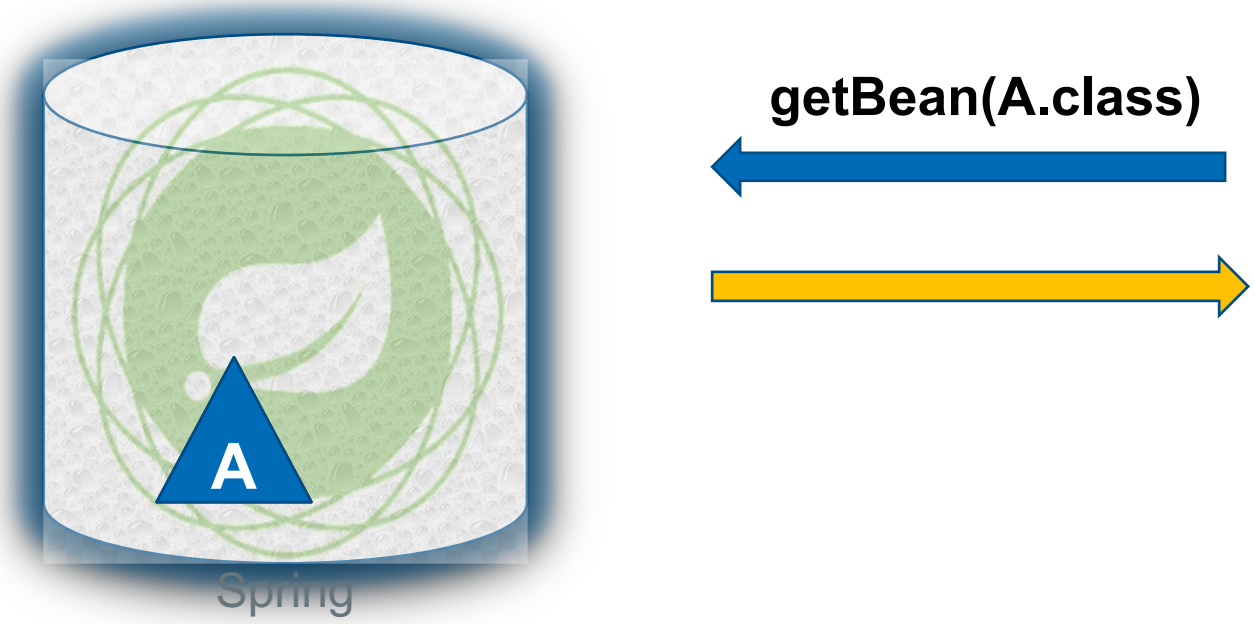
Das Bild kann nicht angezeigt werden.





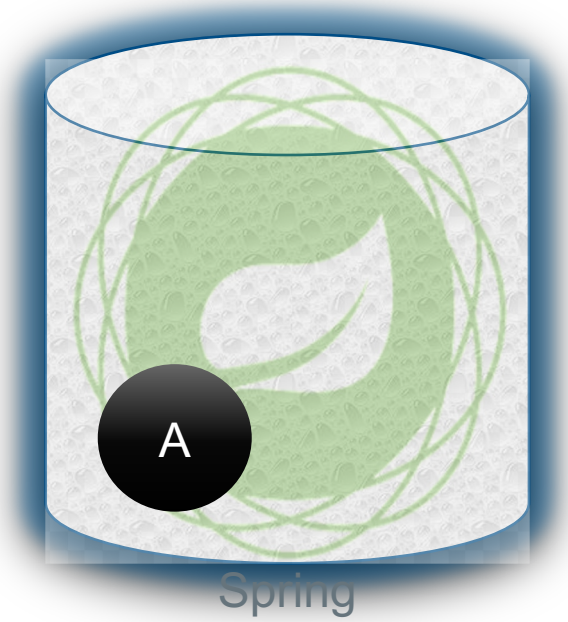
Singleton

Das Bild kann nicht angezeigt werden.



Das Bild kann nicht angezeigt werden.

Prototype

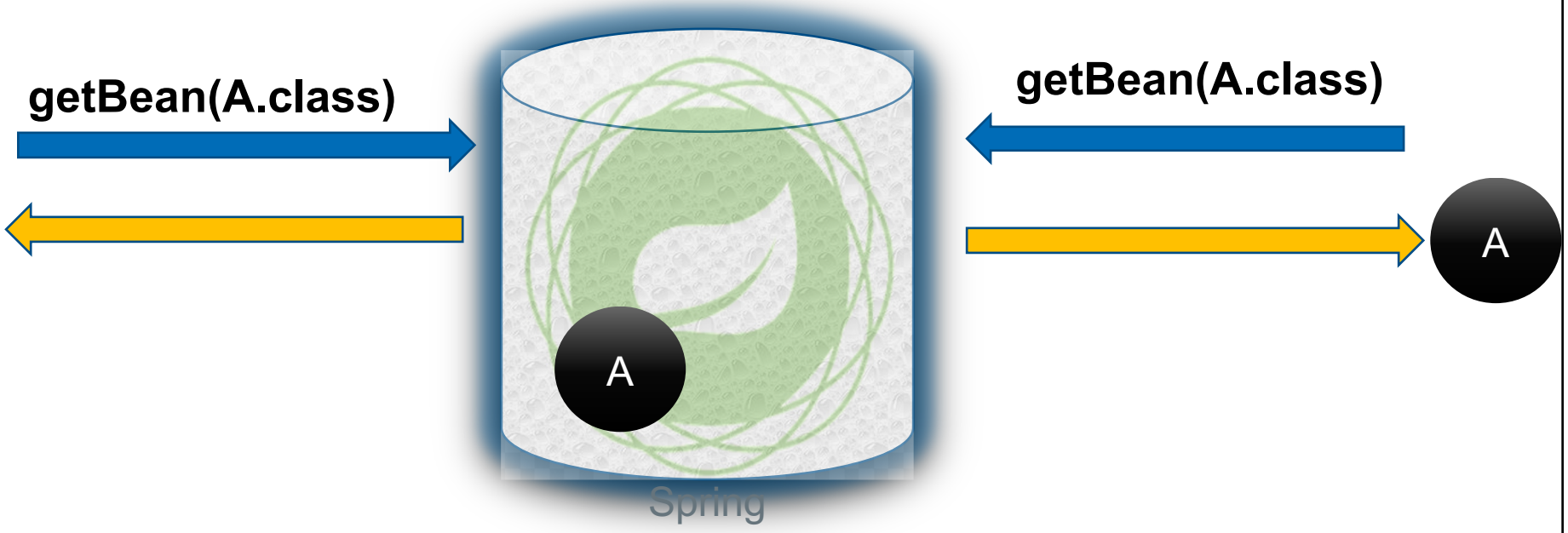


getBean(A.class)

Das Bild kann nicht angezeigt werden.

Prototype

Das Bild kann nicht angezeigt werden.



Das Bild kann nicht angezeigt werden.

Singleton Example

```
@Component
public class Shape {

   private String color = "yellow";

   //setter

}
```

```
@ComponentScan("ch.javaprofi_academy")
public class BeanScopesApp {
    public static void main(String[] args) {

        ClassPathXmlApplicationContext context =
                        new ClassPathXmlApplicationContext("bean-def.xml");

        var shape1 = context.getBean(Shape.class);
        System.out.println(shape1);

        shape1.setColor("blue");
        System.out.println(shape1);

        var shape2 = context.getBean(Shape.class);
        System.out.println(shape2);

        context.close();
    }
}
```

Shape [color=yellow]

Shape [color=blue]

Shape [color=blue]

Scope Configuration

@Scope(scopeName=ConfigurableBeanFactory.SCOPE_PROTOTYPE)

@Scope(scopeName=ConfigurableBeanFactory. SCOPE_SINGLETON)

@Scope(“prototype“)

@Scope(“singleton“)

@Scope(“somethingwrong“)

Caused by: java.lang.IllegalStateException: No Scope registered for scope name 'somethingwrong'

Prototype Example

```
@ComponentScan("ch.javaprofi_academy")
public class BeanScopesApp {
    public static void main(String[] args) {

        ClassPathXmlApplicationContext context =
                        new ClassPathXmlApplicationContext("bean-def.xml");

        var shape1 = context.getBean(Shape.class);
        System.out.println(shape1);

        shape1.setColor("blue");
        System.out.println(shape1);

        var shape2 = context.getBean(Shape.class);
        System.out.println(shape2);

        context.close();
    }
}
```

Shape [color=yellow]

Shape [color=blue]

Shape [color=yellow]

```
@Component
@Scope("prototype")
public class Shape {

   private String color = "yellow";

   //setter

}
```

What does singleton mean for Java? Where is the uniqueness given?

Singleton is per container and not per classloader (~~JVM~~)

```
@Component
public class Shape {

   // shape

}
```

```
@ComponentScan("ch.karthi")
public class App {

    public static void main(String[] args) {
        var container = new AnnotationConfigApplicationContext(App.class);
        var container2 = new AnnotationConfigApplicationContext(App.class);

        var shape1 = container.getBean(Shape.class);
        LOG.info(shape1);

        shape1.setColour("blue");
        LOG.info(shape1);

        var shape2 = container2.getBean(Shape.class);
        LOG.info(shape2);

    }
}
```

INFO: Shape [color=yellow]

INFO: Shape [color=blue]

INFO: Shape [color=yellow]

other singleton pitfalls

```
<?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="scopeTest" class="ch.javaprofi_academy.Shape" scope="singleton">
        <property name="color" value="blue"/>
    </bean>

    <bean id="scopeTestDuplicate" class="ch.javaprofi_academy.Shape"
          scope="singleton">
        <property name="color" value="red"/>
    </bean>
</beans>
```

other singleton pitfalls

```
public class SingletonSurprise {
    public static void main(String[] args) {
        var ctx = new ClassPathXmlApplicationContext("shapes.xml");

        Shape shape1 = ctx.getBean("scopeTest", Shape.class);
        Shape shape2 = ctx.getBean("scopeTestDuplicate", Shape.class);

        System.out.println(shape1 == shape2);
        System.out.println(shape1 + "::" + shape2);
    }
}
```

What is the output?

other singleton pitfalls

```
public class SingletonSurprise {
    public static void main(String[] args) {
        var ctx = new ClassPathXmlApplicationContext("shapes.xml");

        Shape shape1 = ctx.getBean("scopeTest", Shape.class);
        Shape shape2 = ctx.getBean("scopeTestDuplicate", Shape.class);

        System.out.println(shape1 == shape2);
        System.out.println(shape1 + "::" + shape2);
    }
}
```

```
false
Shape [color=blue]::Shape [color=red]
```

other singleton pitfalls

```
public class SingletonSurprise {
    public static void main(String[] args) {
        var ctx = new ClassPathXmlApplicationContext("shapes.xml");

        Shape shape1 = ctx.getBean("scopeTest", Shape.class);
        Shape shape2 = ctx.getBean("scopeTestDuplicate", Shape.class);

        System.out.println(shape1 == shape2);
        System.out.println(shape1 + "::" + shape2);

        Shape shape3 = ctx.getBean("scopeTest", Shape.class);
        Shape shape4 = ctx.getBean("scopeTestDuplicate", Shape.class);
        System.out.println(shape1 == shape3);
        System.out.println(shape2 == shape4);
        System.out.println(shape3 + "::" + shape4);
    }
}
```

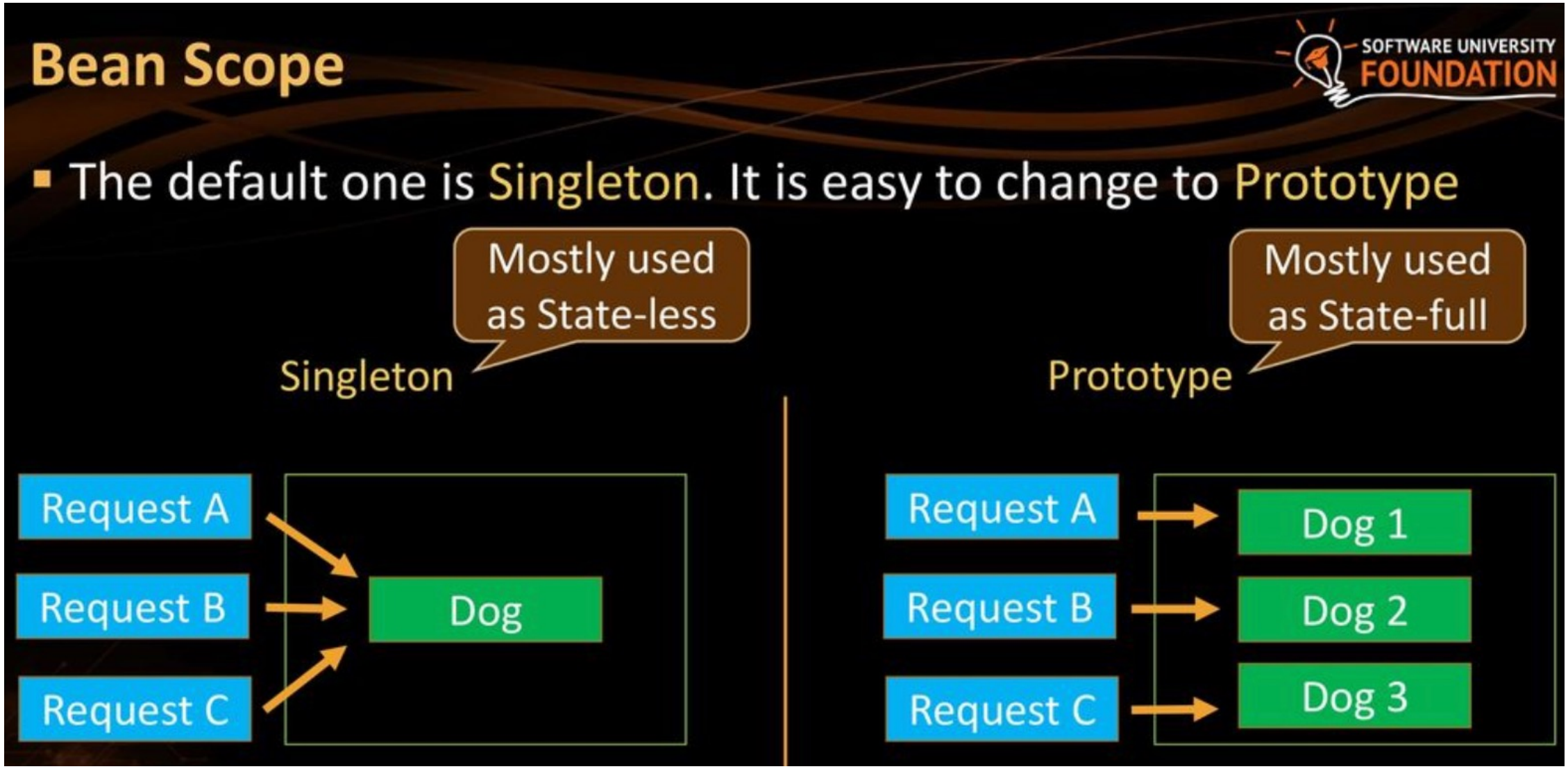
What changes now?

other singleton pitfalls

```
public class SingletonSurprise {
    public static void main(String[] args) {
        var ctx = new ClassPathXmlApplicationContext("shapes.xml");

        Shape shape1 = ctx.getBean("scopeTest", Shape.class);
        Shape shape2 = ctx.getBean("scopeTestDuplicate", Shape.class);

        System.out.println(shape1 == shape2);
        System.out.println(shape1 + "::" + shape2);

        Shape shape3 = ctx.getBean("scopeTest", Shape.class);
        Shape shape4 = ctx.getBean("scopeTestDuplicate", Shape.class);
        System.out.println(shape1 == shape3);
        System.out.println(shape2 == shape4);
        System.out.println(shape3 + "::" + shape4);
    }
}
```

```
false
Shape [color=blue]::Shape [color=red]

true
true
Shape [color=blue]::Shape [color=red]
```

Conclusion Singleton vs Prototype

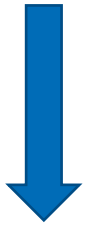


https://slideplayer.com/slide/13335057/

Life Cycle

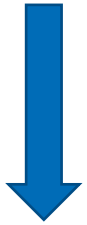
Lifecycle Callbacks

- **Initialization Callbacks**
  - `@PostConstruct`
  - InitializingBean
  - initMethod in `@Bean`



Precedence

- **Destruction Callback**
  - `@PreDestroy`
  - DisposableBean
  - destroyMethod in `@Bean`



@PostConstruct

```
@Component
public class VideoPlayer {

	private AsmiqAcademyStream asmiqAcademyStream;
	private YoutubeStream youtubeStream;

	private String youtubeApiKey = "y7k8nd9d92dl2m";
	private String asmiqAcademyApiKey = "y7k8nd9d92dl2m";

	@PostConstruct
	private void initStreams() {
		asmiqAcademyStream =
		 AsmiqAcademyStream.builder().withApiKey(asmiqAcademyApiKey).build();
		youtubeStream = YoutubeStream.builder().withApiKey(youtubeApiKey).build();
	}
}
```

```
<dependency>
<groupId>javax.annotation</groupId>
<artifactId>javax.annotation-api</artifactId>
<version>1.3.2</version>
</dependency>
```

InitializingBean

```
@Component
public class VideoPlayer implements InitializingBean {

	private AsmiqAcademyStream asmiqAcademyStream;
	private YoutubeStream youtubeStream;

	private String youtubeApiKey = "y7k8nd9d92dl2m";
	private String asmiqAcademyApiKey = "y7k8nd9d92dl2m";

	@Override
	public void afterPropertiesSet() throws Exception {
		initStreams();
	}

	private void initStreams() {
		asmiqAcademyStream =
		 AsmiqAcademyStream.builder().withApiKey(asmiqAcademyApiKey).build();
		youtubeStream = YoutubeStream.builder().withApiKey(youtubeApiKey).build();
	}
}
```

initMethod in @Bean

```
@Component
public class VideoPlayer {

    private AsmiqAcademyStream asmiqAcademyStream;
    private YoutubeStream youtubeStream;

    private String youtubeApiKey = "y7k8nd9d92dl2m";
    private String asmiqAcademyApiKey = "y7k8nd9d92dl2m“;

    private void initStreams() {
        asmiqAcademyStream =
                AsmiqAcademyStream.builder().withApiKey(asmiqAcademyApiKey).build();
        youtubeStream = YoutubeStream.builder().withApiKey(youtubeApiKey).build();
    }
}
```

```
@Configuration
public class AppConfig {
    @Bean(initMethod = "initStreams")
    public VideoPlayer videoPlayer() {
        return new VideoPlayer();
    }
}
```

@Predestroy

```
@Component
public class VideoPlayer {

	private AsmiqAcademyStream asmiqAcademyStream;
	private YoutubeStream youtubeStream;

	private String youtubeApiKey = "y7k8nd9d92dl2m";
	private String asmiqAcademyApiKey = "y7k8nd9d92dl2m";

	@PreDestroy
	private void signOffStreams() {
		asmiqAcademyStream.signOff();
		youtubeStream.signOff();
	}

	public void run() {
		// start any of the stream
	}
}
```

DisposableBean

```
@Component
public class VideoPlayer implements DisposableBean {

        private AsmiqAcademyStream asmiqAcademyStream;
        private YoutubeStream youtubeStream;

        private String youtubeApiKey = "y7k8nd9d92dl2m";
        private String asmiqAcademyApiKey = "y7k8nd9d92dl2m";

        @Override
        public void destroy(){
                signOffStreams ();
        }

        private void signOffStreams() {
                asmiqAcademyStream.signOff();
                youtubeStream.signOff();
        }
}
```

destroyMethod in @Bean

```
@Component
public class VideoPlayer {

        private AsmiqAcademyStream asmiqAcademyStream;
        private YoutubeStream youtubeStream;

        private String youtubeApiKey = "y7k8nd9d92dl2m";
        private String asmiqAcademyApiKey = "y7k8nd9d92dl2m";

        private void signOffStreams() {
                asmiqAcademyStream.signOff();
                youtubeStream.signOff();
        }

        public void run() {
                // start any of the stream
        }
}
```

```
@Configuration
public class AppConfig {
    @Bean(destroyMethod = "signOffStreams")
    public VideoPlayer videoPlayer() {
        return new VideoPlayer();
    }
}
```

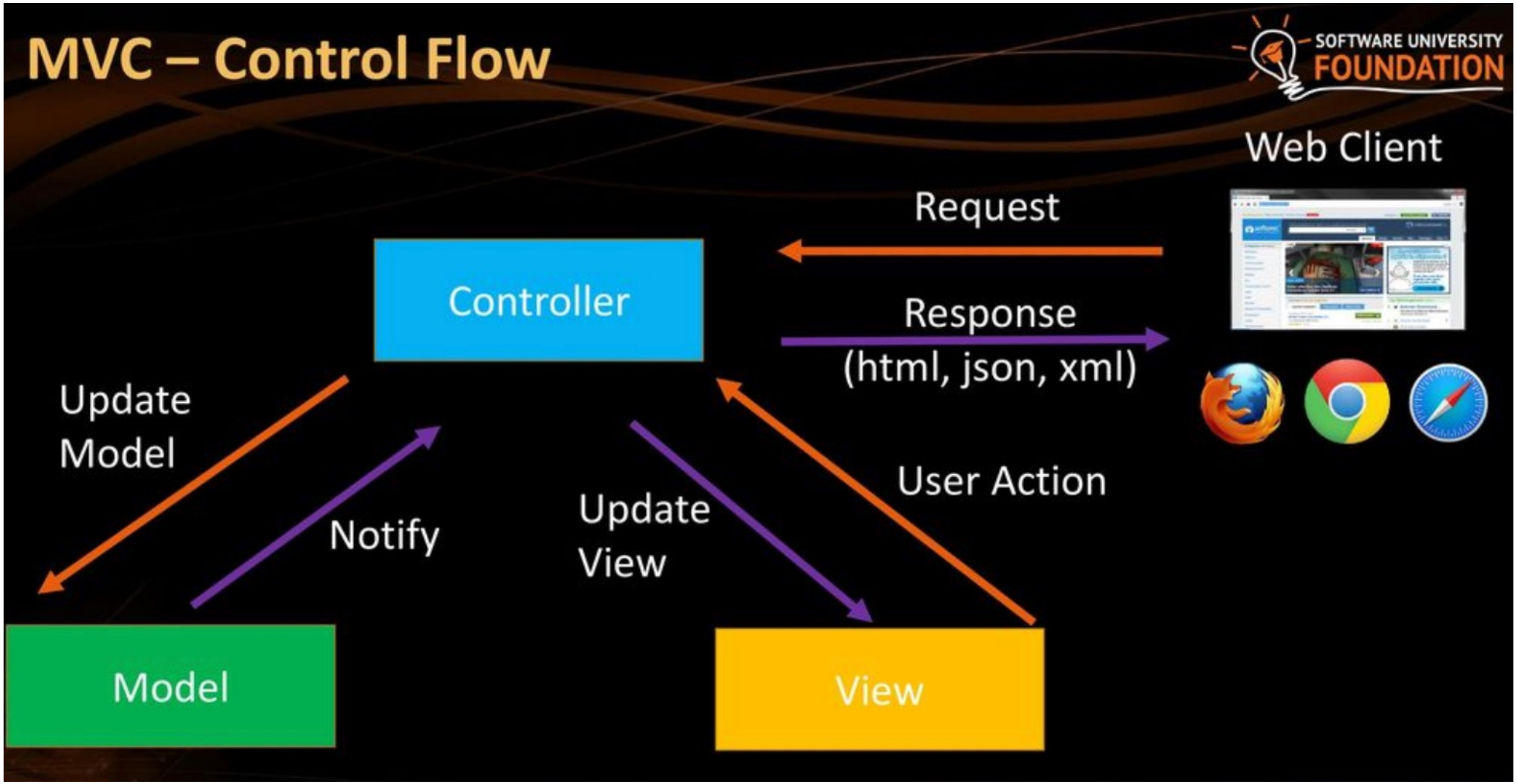
*Aware-Interfaces for special application cases

- Another way to get into the lifecycle is to implement XyzAware interfaces:

```
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.BeanNameAware;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
import org.springframework.stereotype.Component;

@Component
class MySpringBean implements BeanNameAware, ApplicationContextAware {

    @Override
    public void setBeanName(String name) {
        // ...
    }

    @Override
    public void setApplicationContext(ApplicationContext applicationContext)
                throws BeansException {
        // ...
    }
}
```

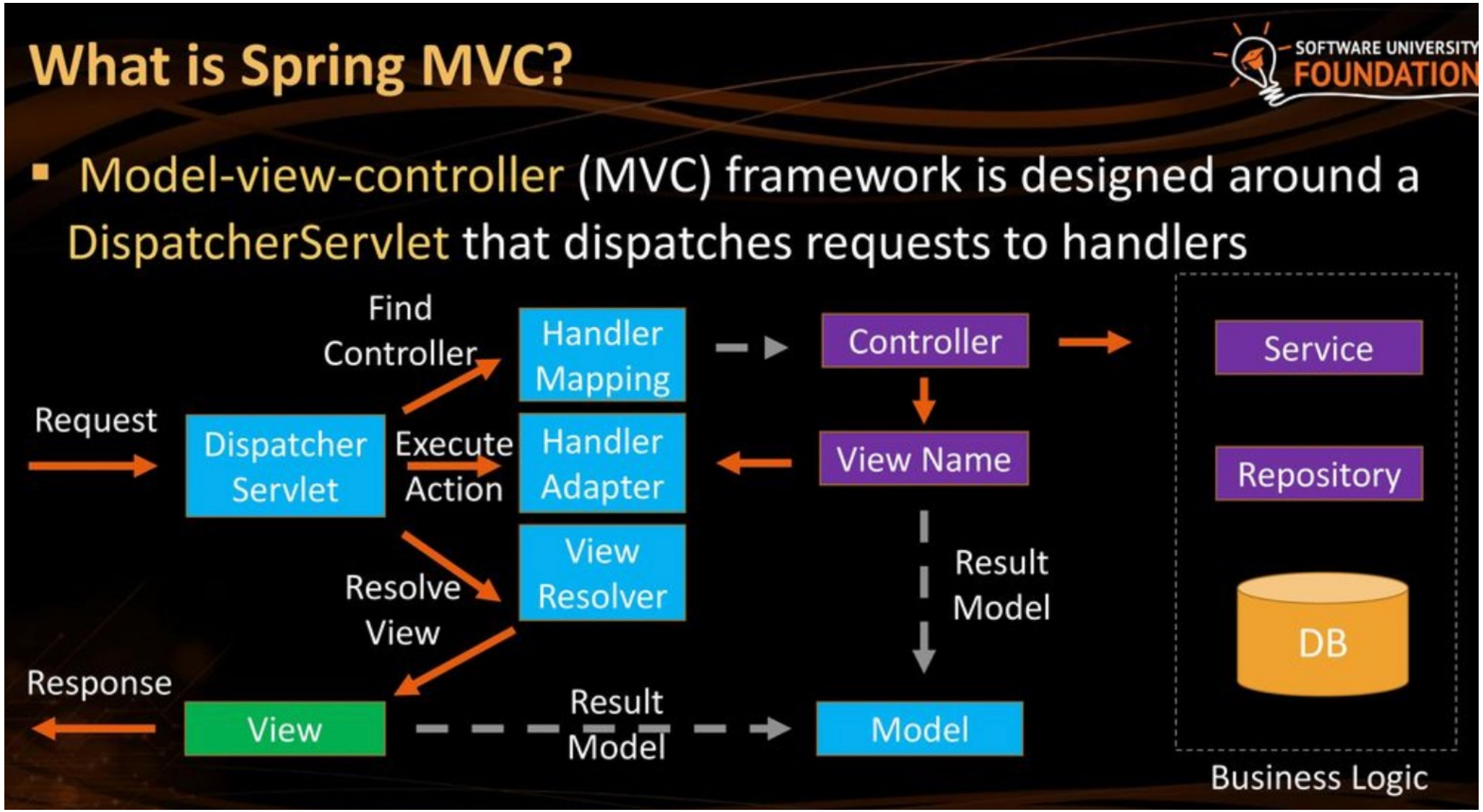
PART 4:
Spring MVC

MVC – Model View Controler – Logical Model



https://slideplayer.com/slide/13335057/

Spring Web-MVC – FrontController Pattern – View in detail



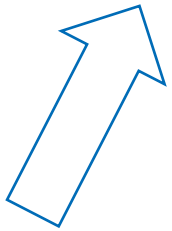
https://slideplayer.com/slide/13335057/

@Controller

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Component
public @interface Controller {


    /**
    * The value may indicate a suggestion for a logical component name,
    * to be turned into a Spring bean in case of an autodetected component.
    * @return the suggested component name, if any (or empty String otherwise)
    */
    @AliasFor(annotation = Component.class)
    String value() default "";


}
```

@Controller

```
public enum RequestMethod {
        GET, HEAD, POST, PUT, DELETE, ...
}
```



```
@Controller
public class ShoppingBasket {

    @RequestMapping(name="/items", method=RequestMethod.GET)
    public String items() {
        // return the view name for the shopping lists
    }

    <<More request mapping methods>>
}
```

What about REST?

REST-Controller based on Controller

```
@Controller
public class ShoppingBasket{


        @RequestMapping(name="/items”, method=RequestMethod.GET)
        @ResponseBody
        public List<ShoppingItem> items(){
                //return the shopping lists
        }

}
```

@RestController

Das Bildelement mit der Beziehungs-ID rId2 wurde in der Datei nicht gefunden.

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Controller
@ResponseBody
public @interface RestController {


	/**
	* The value may indicate a suggestion for a logical component name,
	* to be turned into a Spring bean in case of an autodetected component.
	* @return the suggested component name, if any (or empty String otherwise)
	* @since 4.0.1
	*/
	@AliasFor(annotation = Controller.class)
	String value() default "";

}
```

@RestController

@Controller @RestController

```
public class ShoppingBasket{


        @RequestMapping(name="/items”, method=RequestMethod.GET)
        @ResponseBody
        public List<ShoppingItem> items(){
                //return the shopping lists
        }

}
```

@RestController

```
@RestController
public class ShoppingBasket{

        @GetMapping("/items")
        @RequestMapping(name="/items", method=RequestMethod.GET)
        public List<ShoppingItem> items(){
                //return the shopping lists
        }

}
```

Shortcut Handler Methods

@RequestMapping(… , method=RequestMethod.GET)	@GetMapping(..).
@RequestMapping(… , method=RequestMethod.POST)	@PostMapping(…).
@RequestMapping(… , method=RequestMethod.PUT)	@PutMapping(…).
@RequestMapping(… , method=RequestMethod.DELETE)	@DeleteMapping(…)

produces => accept header

```
curl http://localhost:8080/items
```

```
@GetMapping("/items")
public List<ShoppingItem> items(){
	return items;
}
```

```
curl -H "accept: application/json" http://localhost:8080/items
```

```
@GetMapping(value="/items", produces=MediaType.APPLICATION_JSON_VALUE)
public List<ShoppingItem> itemsInJSON(){
	return items;
}
```

produces => accept header

```
curl -H "accept: application/xml" http://localhost:8080/items
```

```
@GetMapping(value="/courses", produces= { MediaType.APPLICATION_JSON_VALUE,
                                          MediaType.APPLICATION_XML_VALUE })
public List<ShoppingItem> getItems(){
       return items;
}
```

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.12.5</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.12.5</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.12.5</version>
</dependency>
```

Get a particular item

```
@GetMapping("/items/{itemId}")
public ShoppingItem getItemById(@PathVariable String itemId){
      //return item with itemId
}
```

http://host:port/items/1

```
@GetMapping("/items")
public ShoppingItem getItemById(@RequestParam String itemId){
      //return item with itemId
}
```

http://host:port/items?itemId=1

Questions?

Hilfe

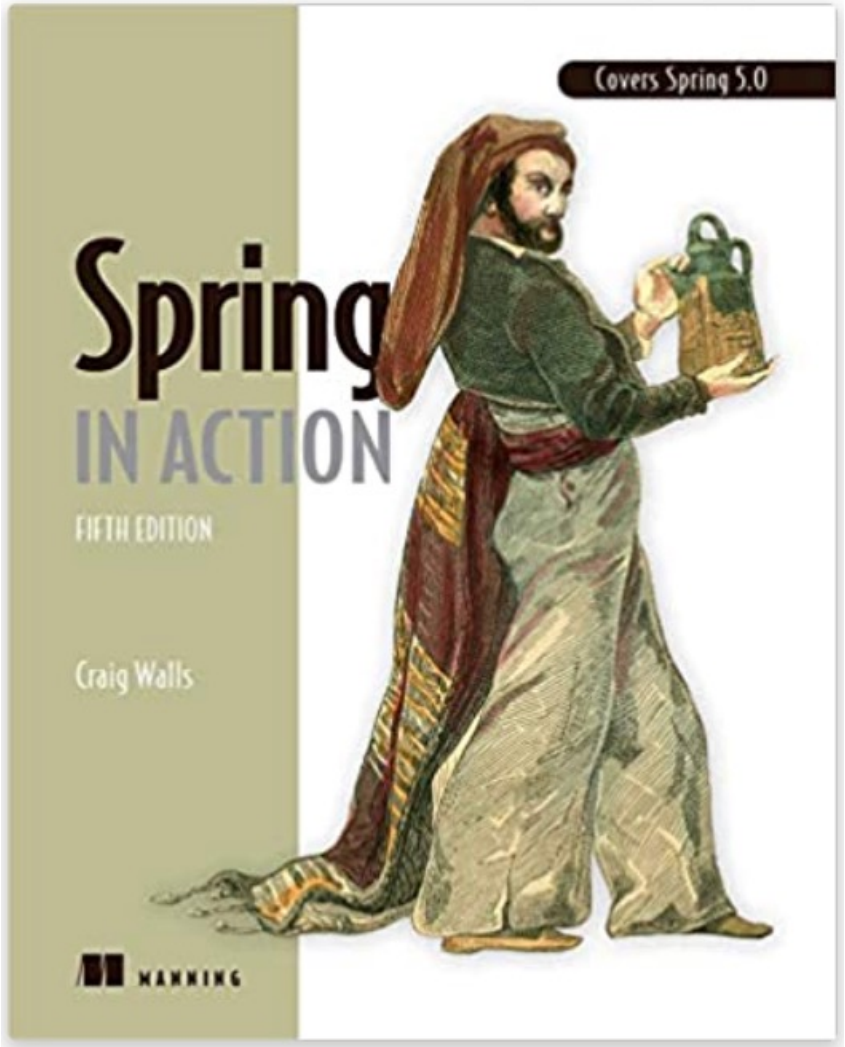

HELP
IS ON THE
WAY


5.
Auflage
Michael Inden
Der Weg zum
Java-Profi
Konzepte und Techniken für die
professionelle Java-Entwicklung
dpunkt.verlag


Michael Inden
Der Java-Profi:
Persistenzlösungen
und REST-Services
Datenaustauschformate,
Datenbankentwicklung
und verteilte Anwendungen
dpunkt.verlag


Java
Michael Inden
Challenge
Fit für das Job-Interview und die Praxis –
mit mehr als 100 Aufgaben
und Musterlösungen
dpunkt.verlag

Recommended books



Thank You