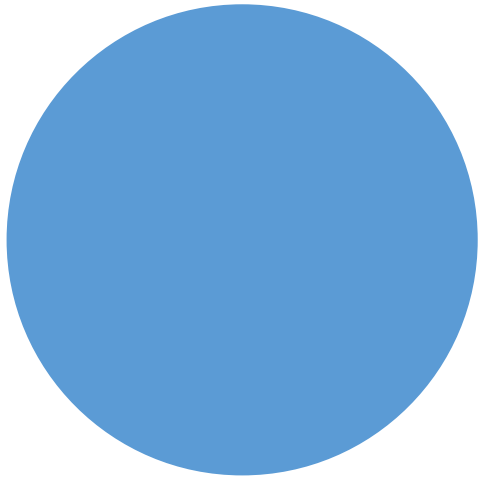
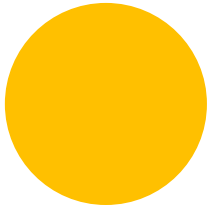



Spring workshop

Karthikeyan Bollu Ganesh
Michael Inden
ASMIQ Academy

Contents

Introduction

Spring Framework I

- Spring Architecture
- Dependency Injection
- DI/IoC in Spring
  - XML Configuration
  - Annotation Configuration
  - Java Configuration
- STS Suite / Plugin + Live Demo
- Exercises

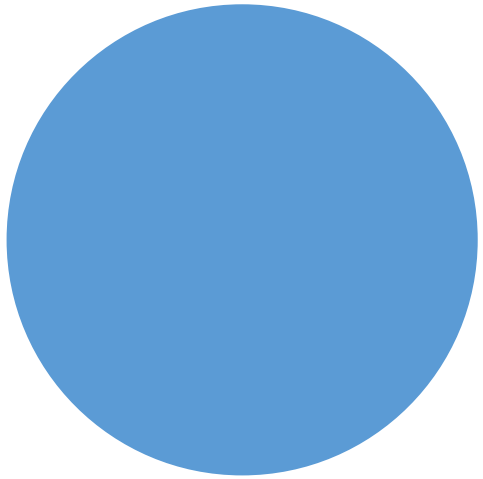
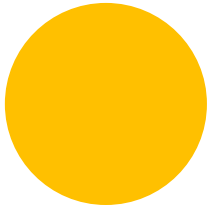
Contents

Spring Framework II

- Laziness
- Circular dependencies
- Bean Scopes
- Bean lifecycle callbacks
- Web-MVC
- Exercises

Spring Boot I

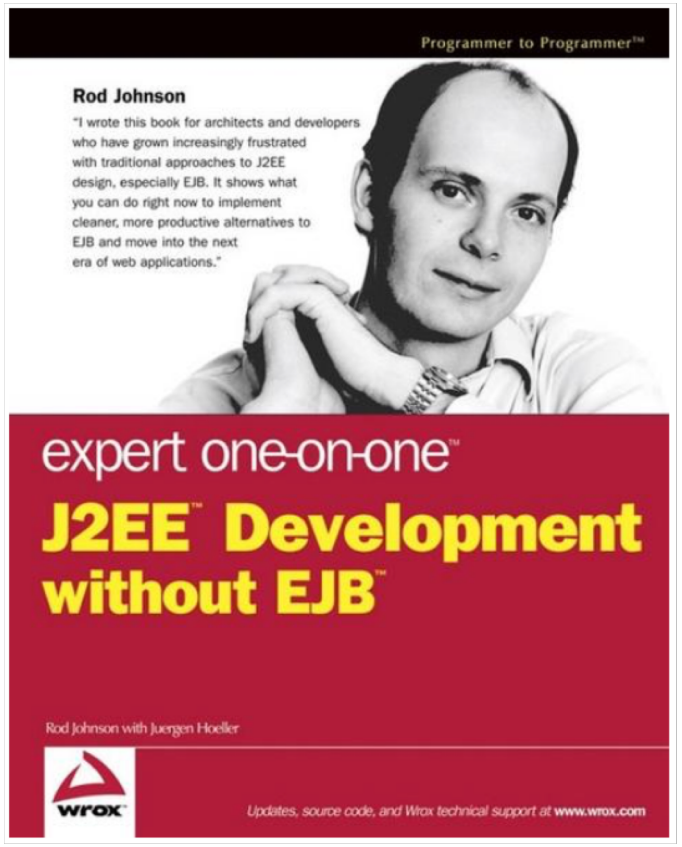
- Introduction
- Spring Initializr
- REST Service with Spring-Boot
- Spring Dev-tools Demo
- Testing
- Exercises

Introduction

Father of Spring

Spring‘s birth



Rod Johnson

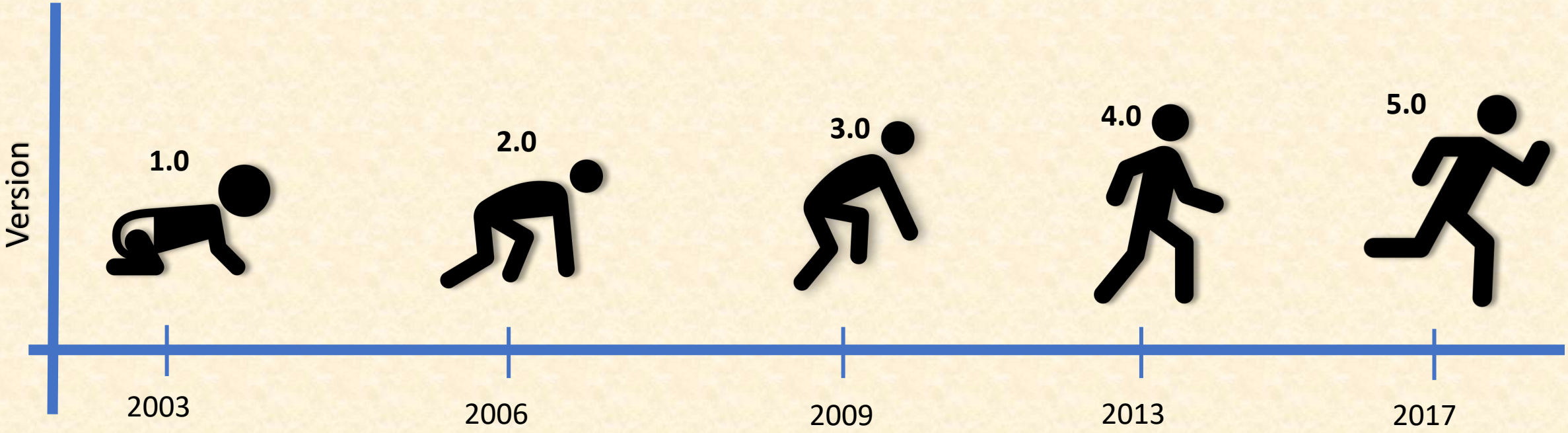
"I wrote this book for architects and developers who have grown increasingly frustrated with traditional approaches to J2EE design, especially EJB. It shows what you can do right now to implement cleaner, more productive alternatives to EJB and move into the next era of web applications."

Spring‘s birth

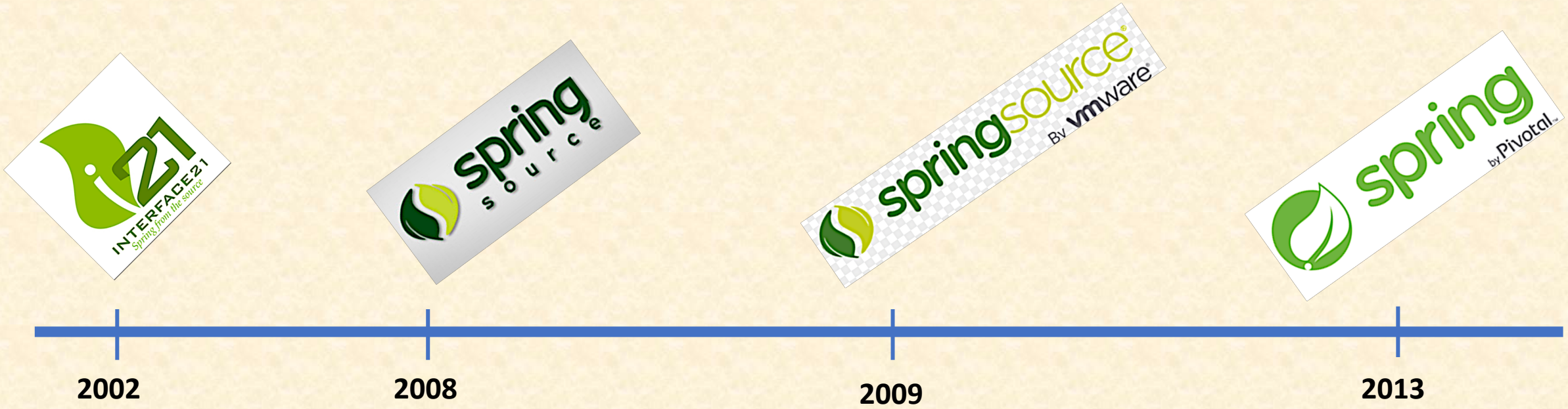
V 0.9

2002

Spring‘s growth



Spring company‘s growth



Spring - Benefits

Light weight

- Just POJO – need not implement/extend any framework interfaces/classes

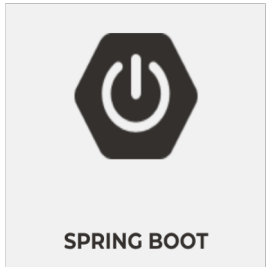
Modular

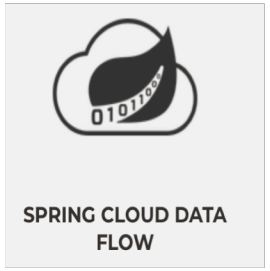
- Include just the needed module to use the needed functionality.
  - For DI just include spring-context

Non-intrusive

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

Major Spring Projects





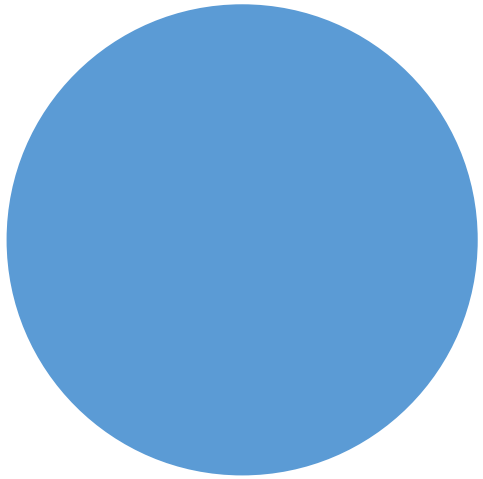
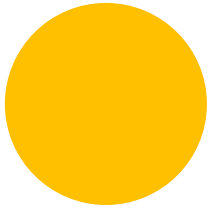
Spring Framework - JDK Baseline

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 5 + JDK 9 Module

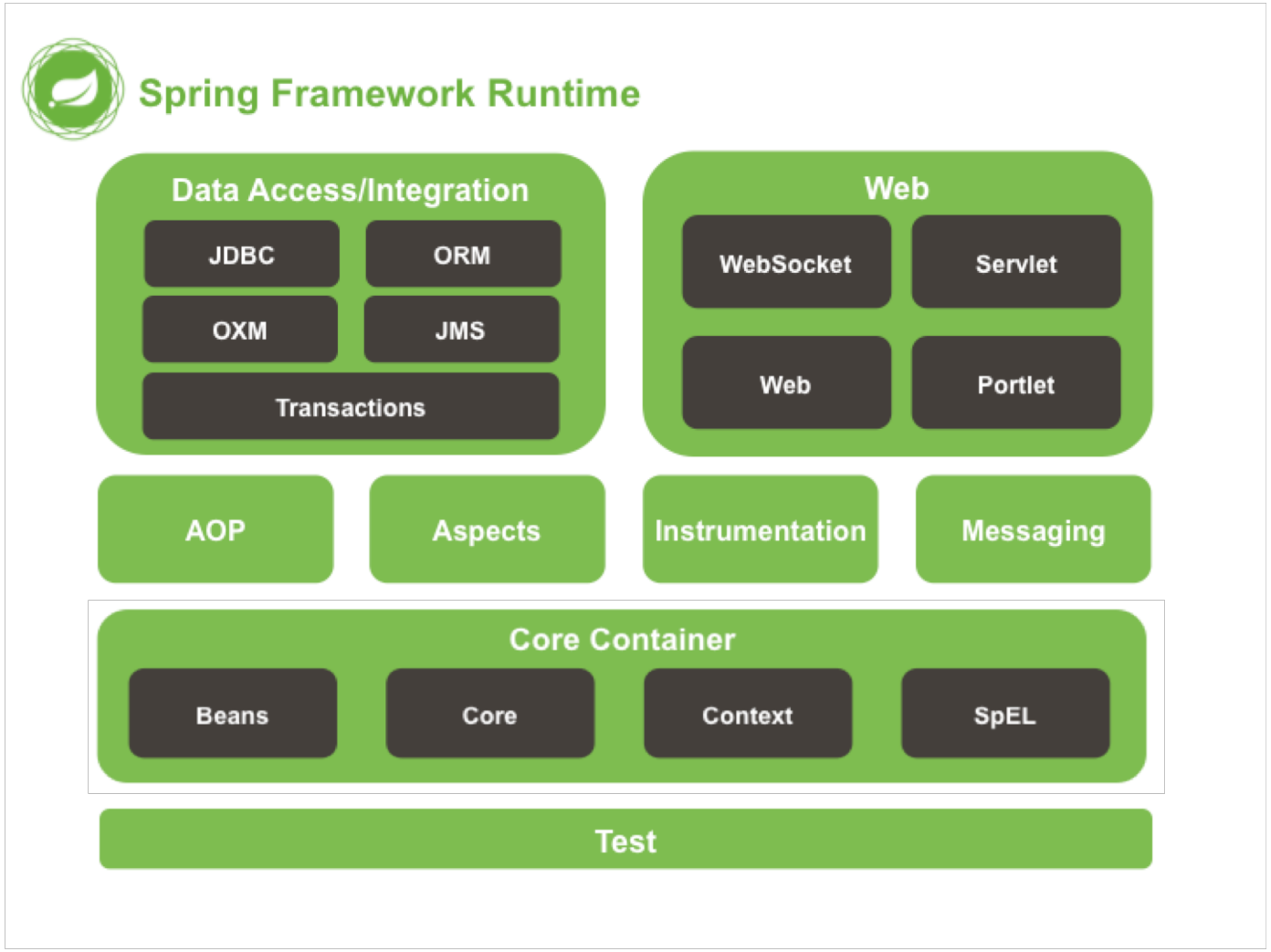
Does Spring Framework 5 work with the new Java 9 module system?

- 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.

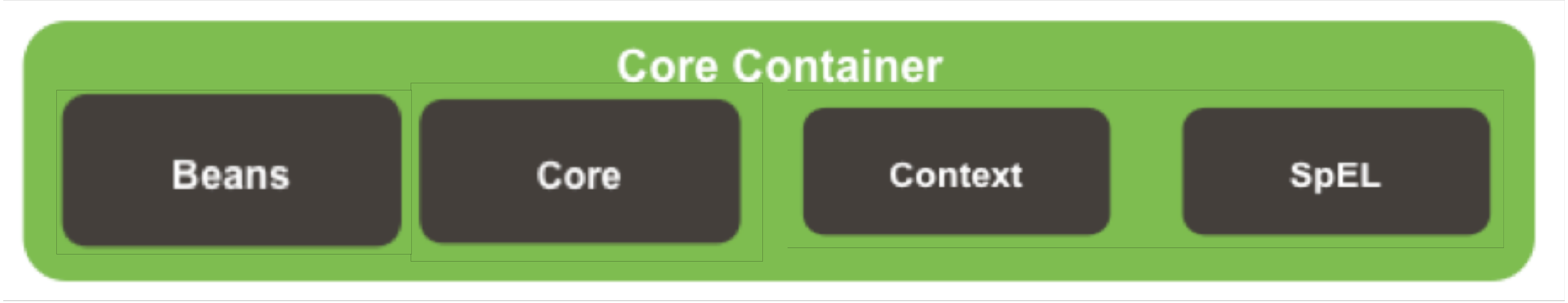



Spring Framework I

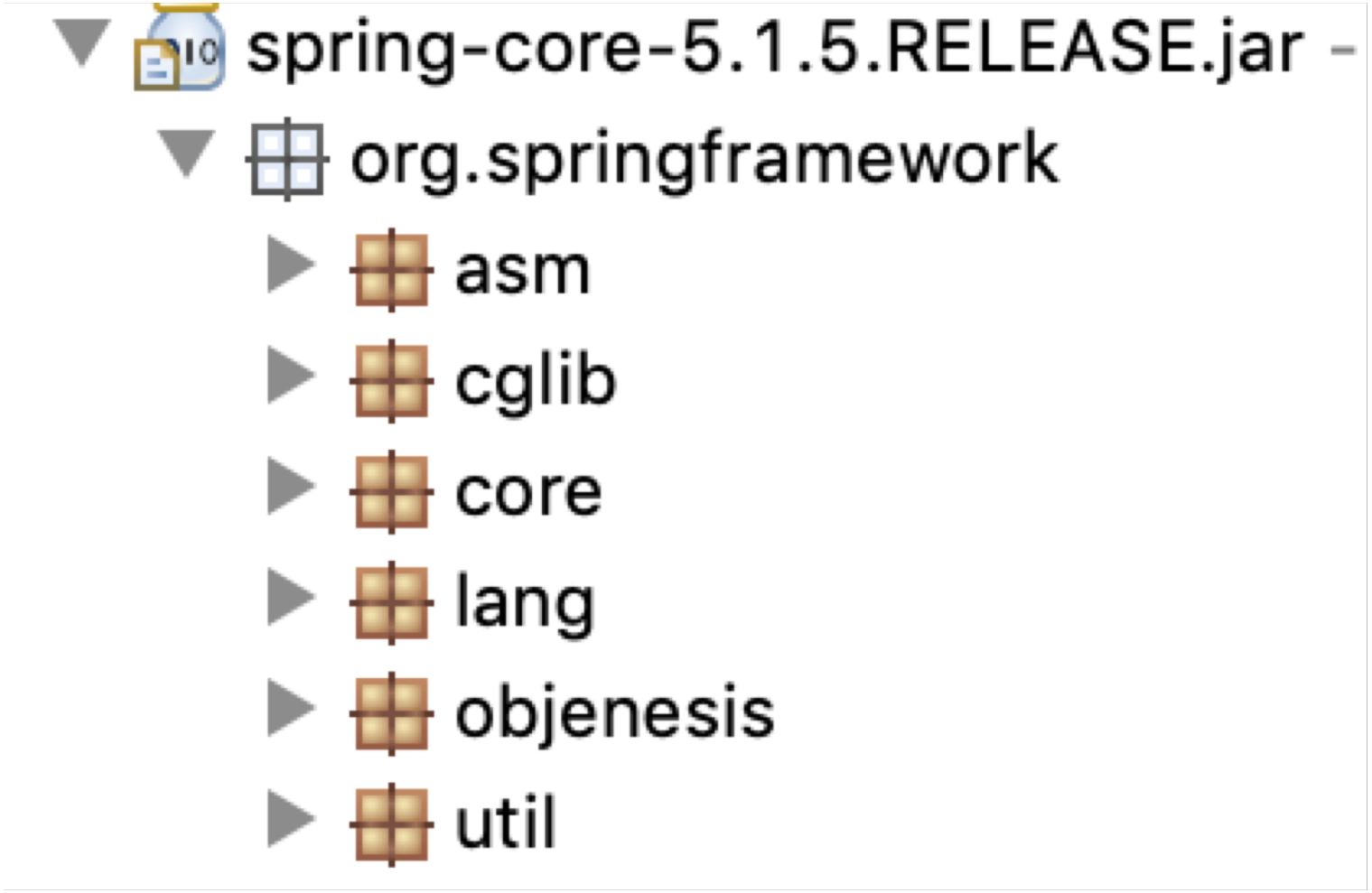
Spring Architecture



Spring Core Container Details



Core Module


spring-core-5.1.5.RELEASE.jar -
org.springframework
asm
cglib
core
lang
objenesis
util

Beans Module

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

Defines most important interface like **BeanFactory** for Dependency Injection.

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

Context and SpEL Modules

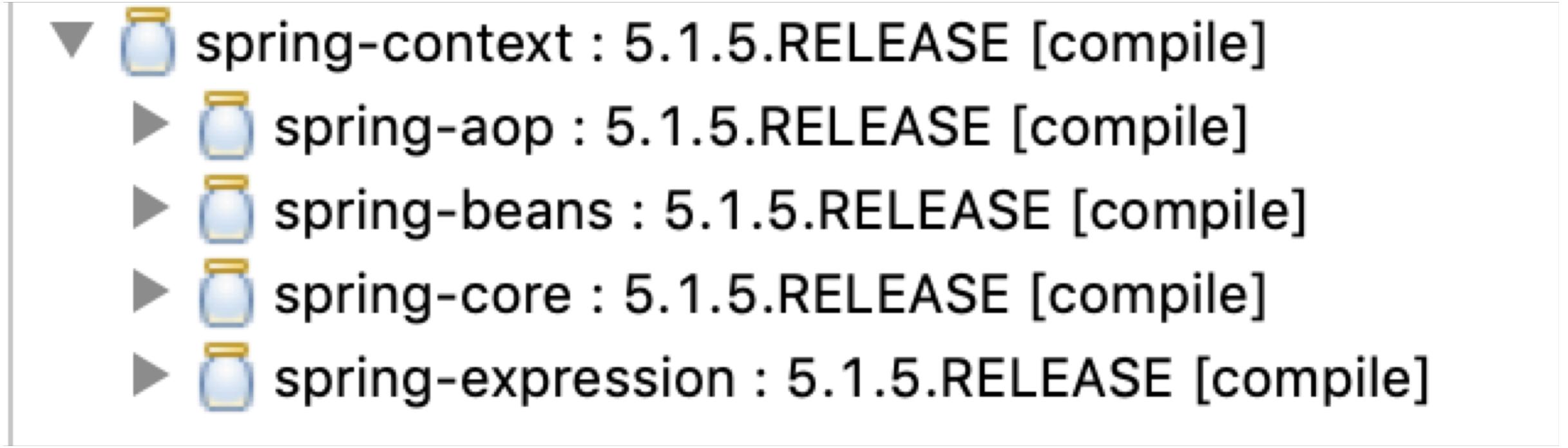
Context

- Defines the interface ApplicationContext, which extends BeanFactory

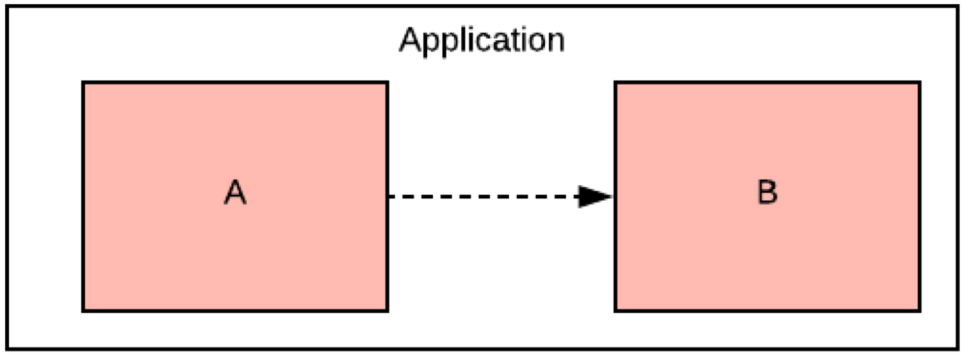
Spring Expression Language

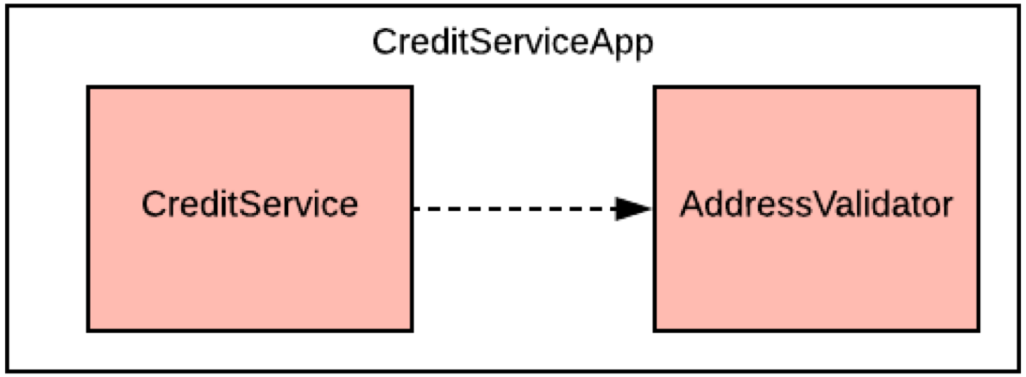
- *"Language to query and manipulate the object graph at runtime."*
  - #{object.property}
  - #{object.subobject.property}

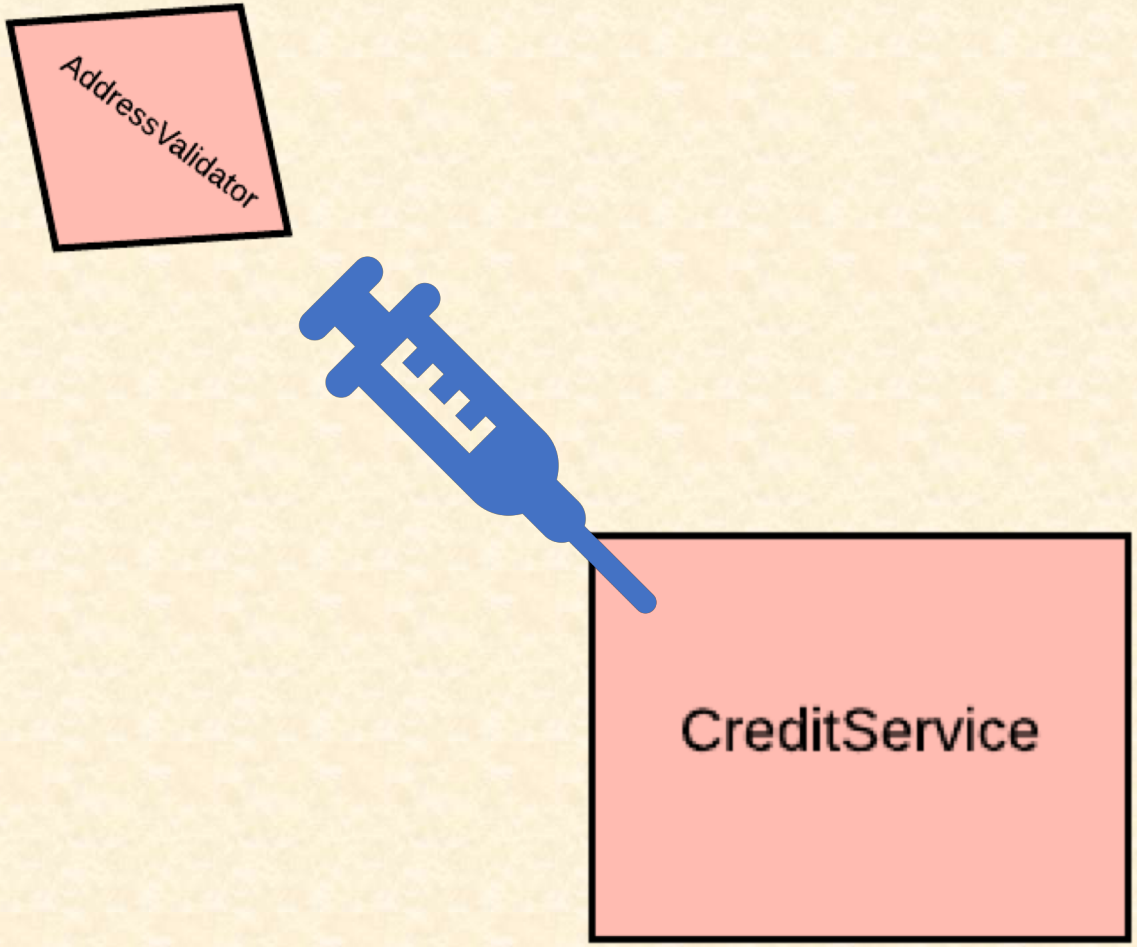
Core Container Module



Dependency


Application
A
B


CreditServiceApp
CreditService
AddressValidator


AddressValidator
CreditService

Dependency Injection

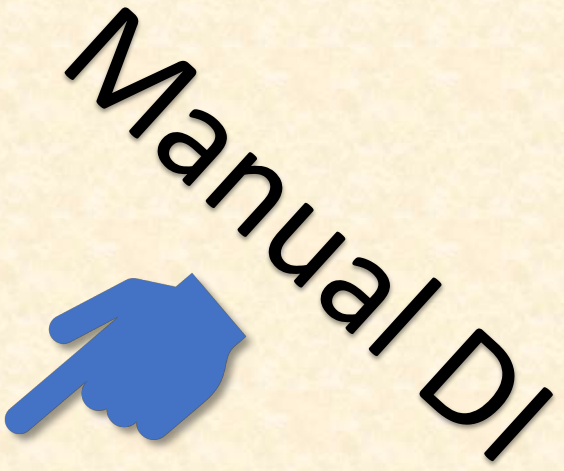
- The process of introducing the dependency to the dependent object is called dependency injection

Dependency Injection

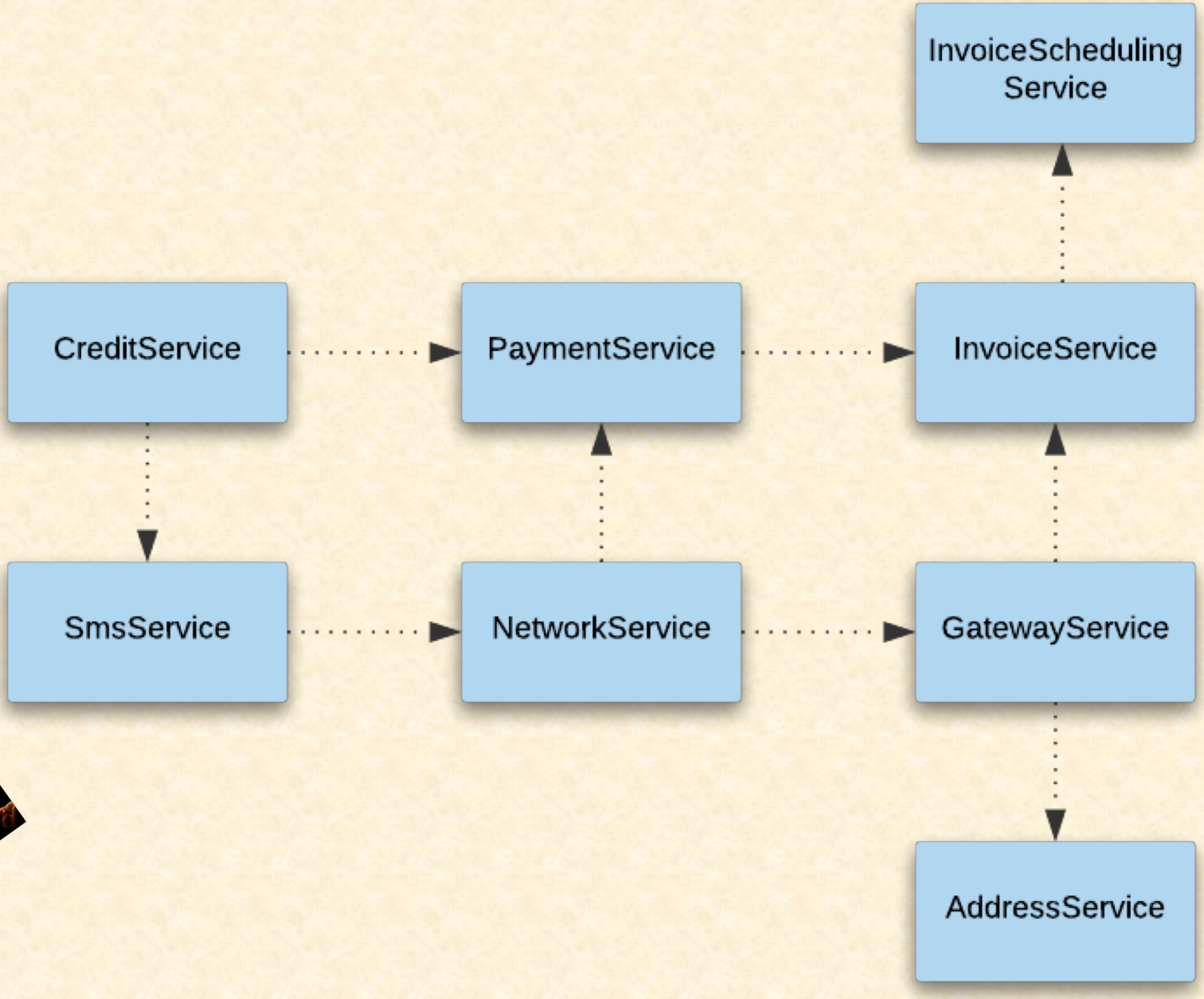
```
public class CreditServiceApp {

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

}
```

Manual Dependency Injection

```
public class CreditServiceApp {

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

}
```



What if ?


InvoiceScheduling Service
CreditService
PaymentService
InvoiceService
SmsService
NetworkService
GatewayService
AddressService


NO!

Manual DI Problems

- Tight Coupling
- Not easily testable
- A depender knows the internal details of the dependee
- Hinders the code readability
- Lot of manual object instantiation & boiler-plate codes

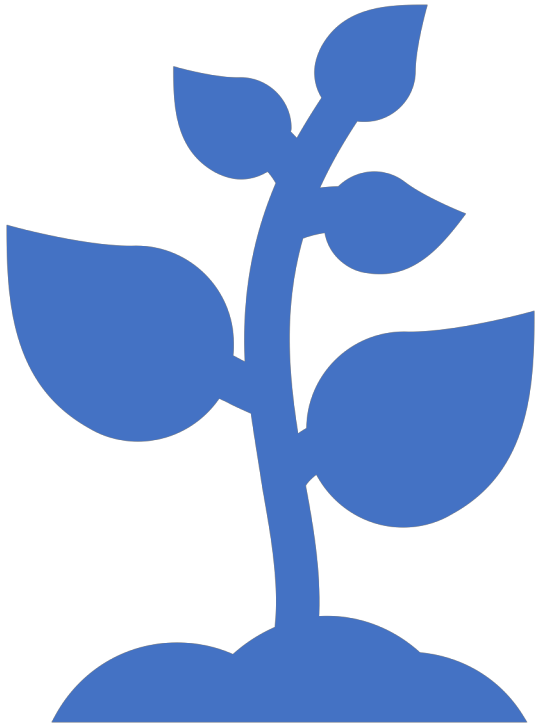
make it all in Orange

Solution for manual DI Problems

- having an **agent/factory/framework**
  - to instantiate the dependee objects
  - resolve the dependencies

IoC / DI

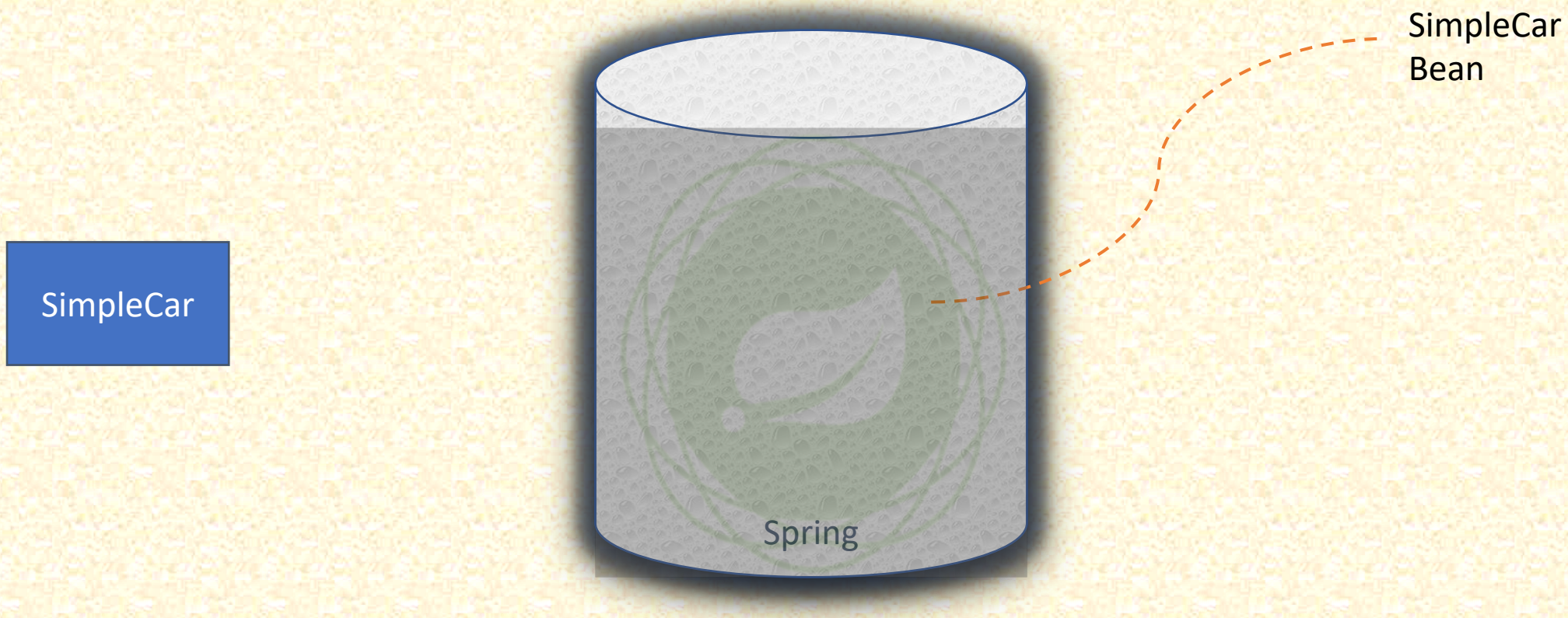
- The process
  - of letting out our control in the instantiation of dependent objects and configuring the object graph is **Inversion of Control (IoC)**
  - of resolving the dependencies is called **Dependency Injection (DI)**



Dependency Injection in Spring

Spring Bean

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



IoC Container

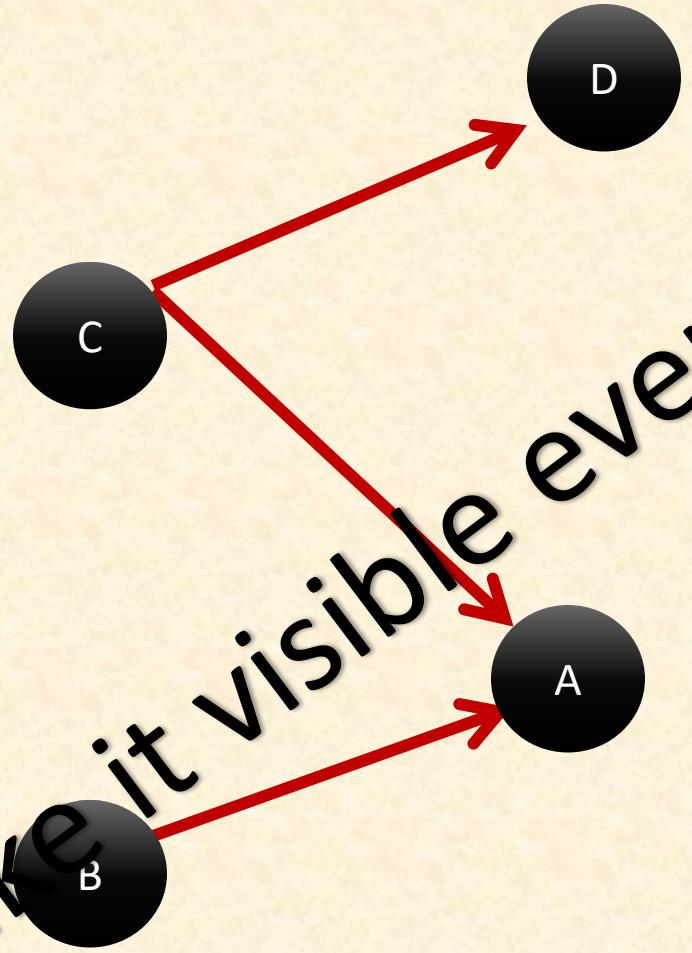
Is also an object

manages other objects

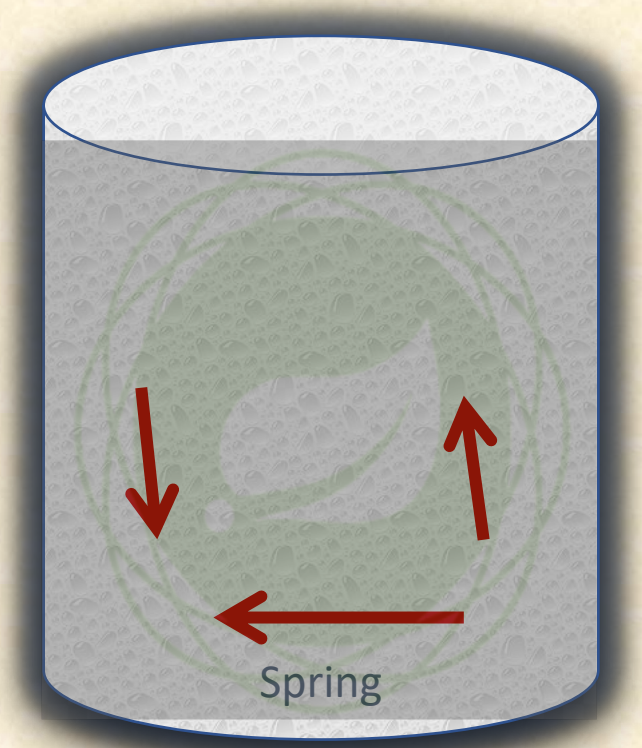
enables Dependecy Injection

builds Object Graphs

IoC Container



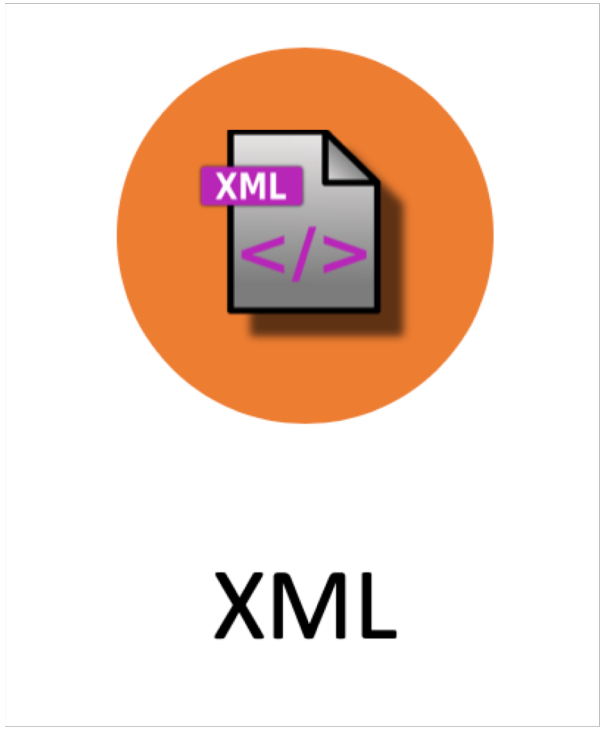
make it visible even during animi.

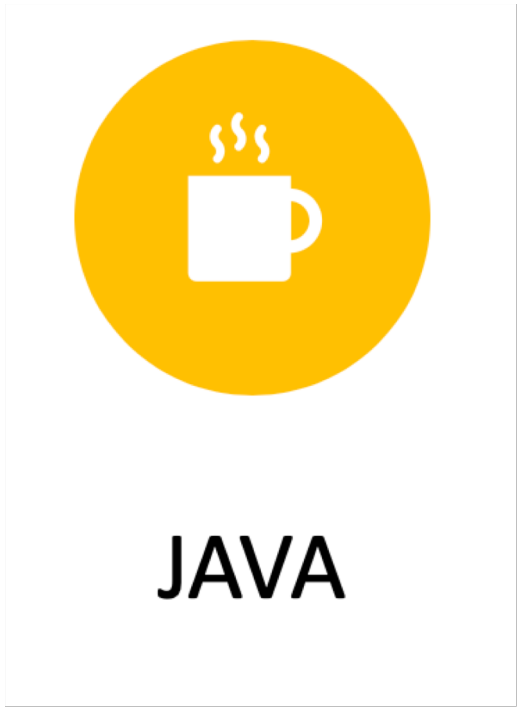


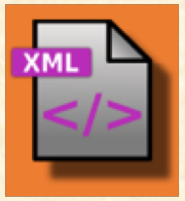
IoC configuration metadata

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

Configuration metadata types







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;
        }


}
```

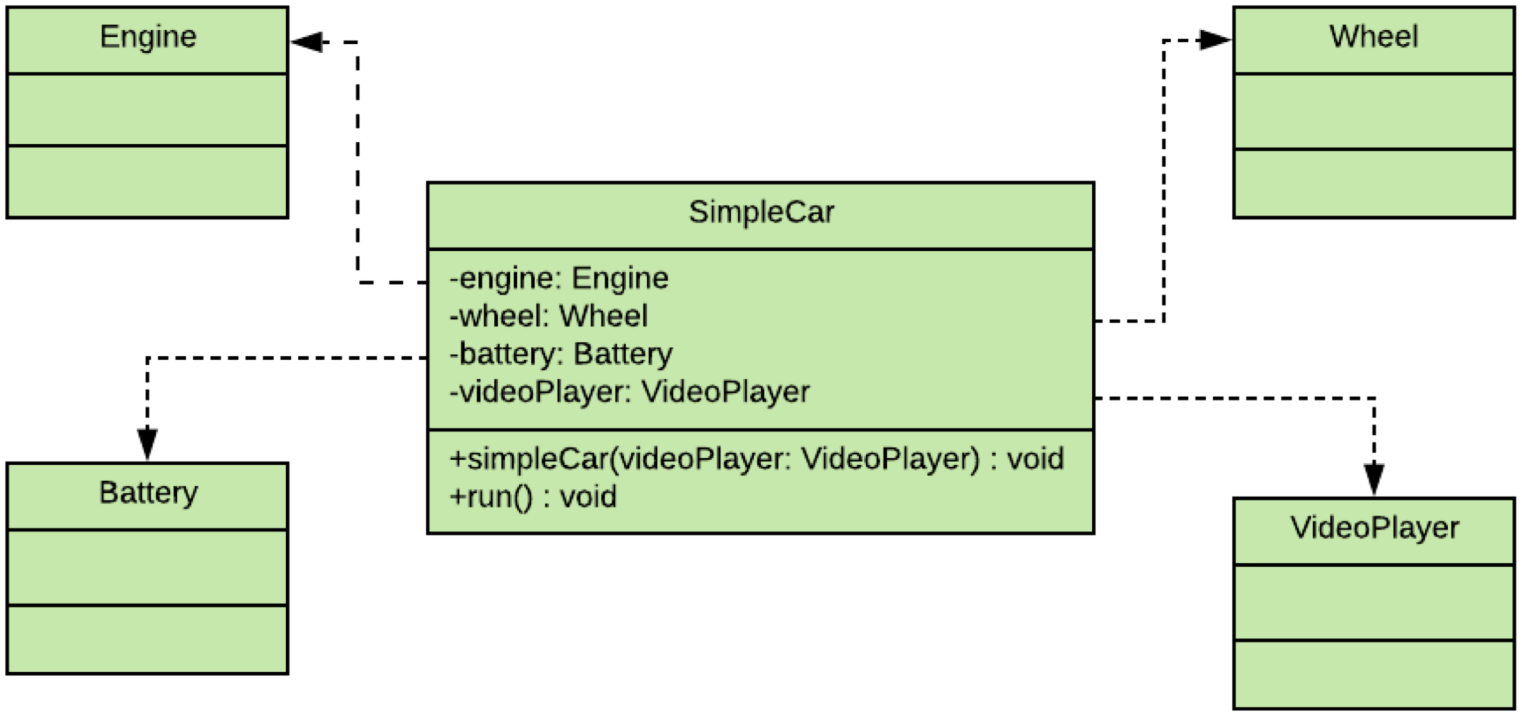
Annotation Configuration

```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"
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>
```

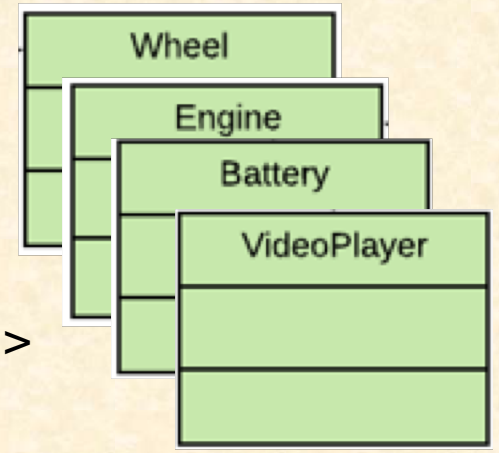
+

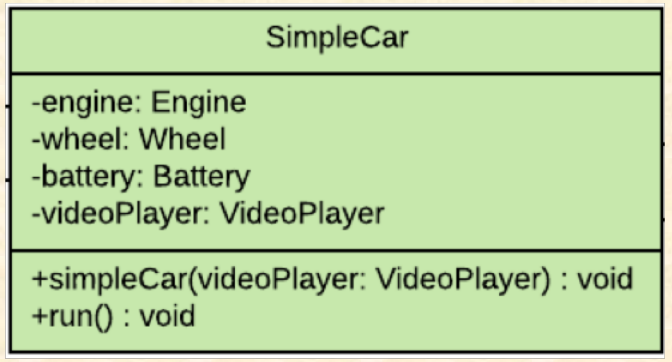
```java
@Component
public class Person {

...


}
```

XML Configuration

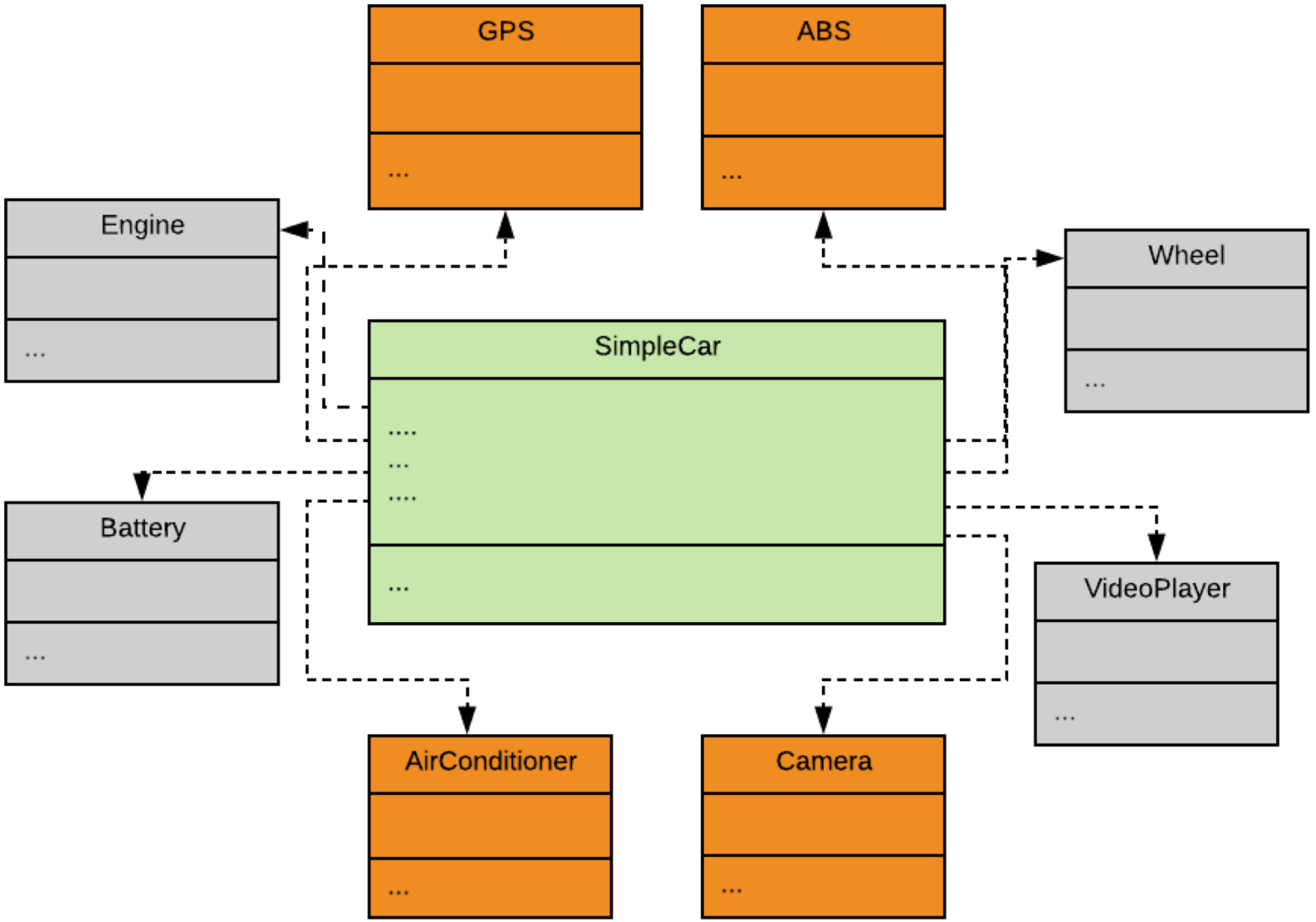


```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

        <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



```
<?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>
```

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

      <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= … />
            ...
```

Karthi : *Hey Spring! Could you please do the bean definitions for me ?*

Spring : *Yes with pleasure. For that I need some infos too*

```
      <bean id="simpleCar" class="ch.karthi.SimpleCar">
            …
      </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"
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>
```

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
- @ControllerAdvice
- @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
 http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
```

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

Karthi : *Hey Spring! is it possible to configure these dependencies in java class itself?*

Spring : *Yes with pleasure. see @Autowired*

```
    <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>
```

@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;
        }
```

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

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

Karthi : *Hey Spring! is it possible to configure this in java class itself?*

Spring : *Yes with pleasure. Use @ComponentScan and remove this XML File* ☺

```
</beans>
```

Java Configuration

```
@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;

}
```

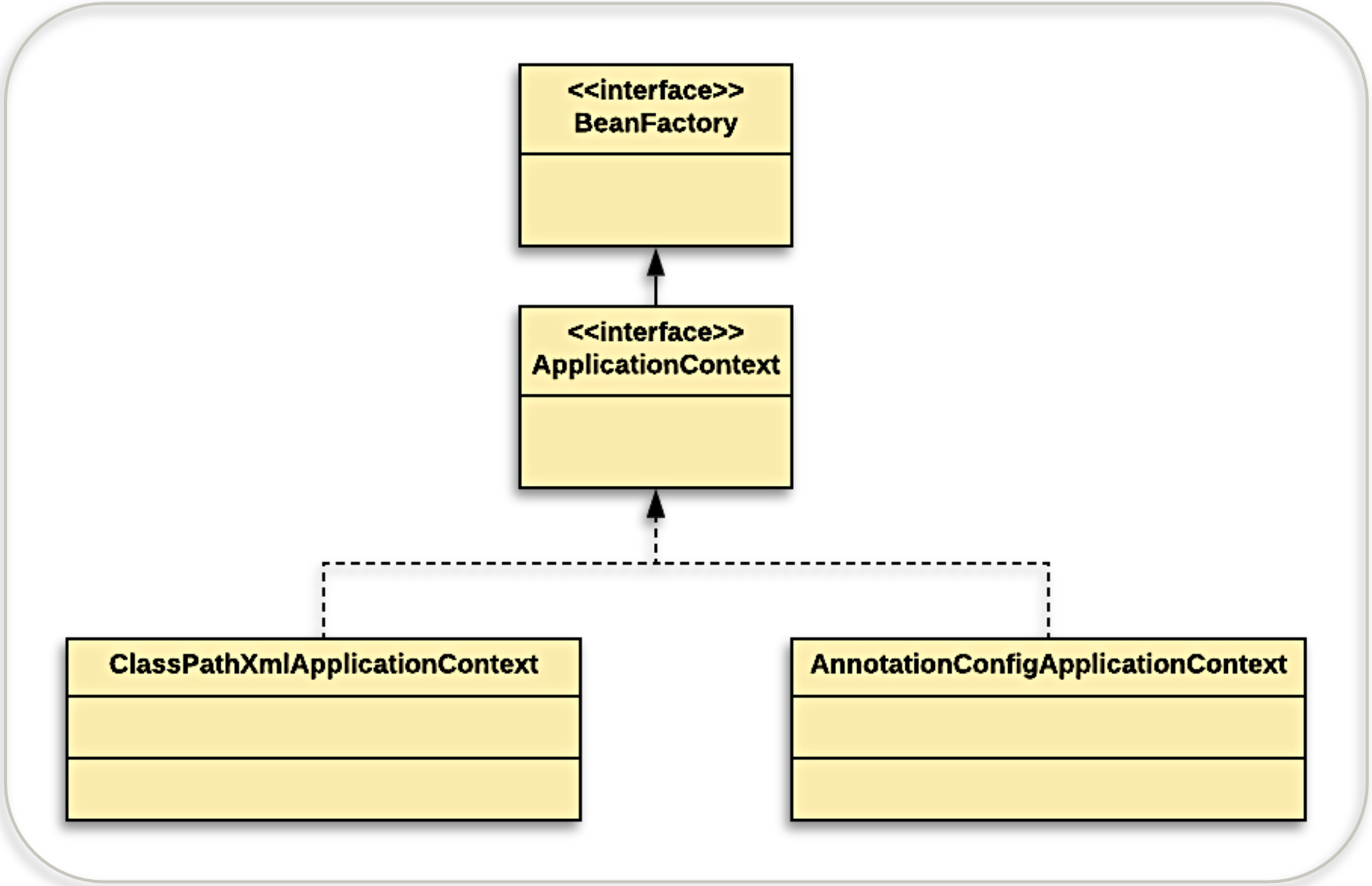
```
@Configuration
public class AppConfig {

        @Bean
        public Battery battery() {
                return new Battery();
        }

        //further bean definitions


        @Bean
        public SimpleCar simpleCar() {
                SimpleCar simpleCar = new SimpleCar(videoPlayer());
                simpleCar.setBattery(battery());
                // set other properties
                return simpleCar;
        }

}
```

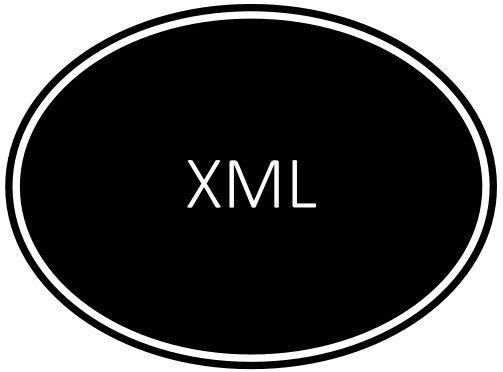
Autowiring byName

```
@Configuration
public class AppConfig {

        @Bean
        public Battery battery() {
                return new Battery();
        }

        //further bean definitions

        @Bean
        public SimpleCar simpleCar(Battery battery, Engine engine, Wheel wheel, VideoPlayer videoPlayer){
                SimpleCar simpleCar = new SimpleCar(videoPlayer);
                simpleCar.setBattery(battery);
                // set other properties
                return simpleCar;
        }


}
```

IoC Container instantiation & usage

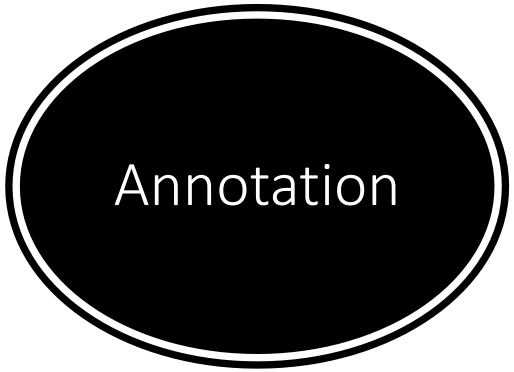
IoC Container in Spring





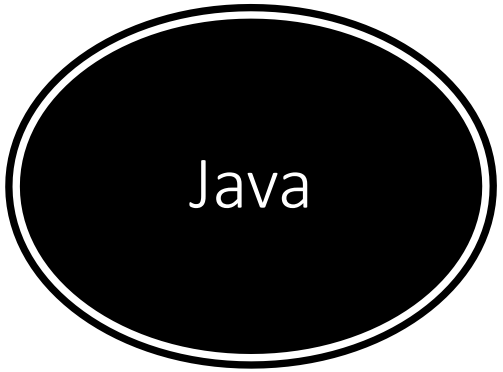

```
var container = new ClassPathXmlApplicationContext("config.xml");

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= … />
        ...
```




```
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
 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 {…}

Dependency Injection Types

Constructor

Setter

Field

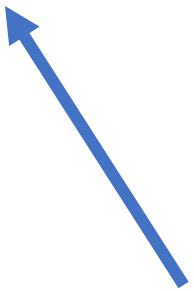
Constructor Injection

```
public SimpleCar(VideoPlayer videoPlayer){

        this.videoPlayer = videoPlayer;

}
```

Constructor Injection

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

```
public SimpleCar() {}

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



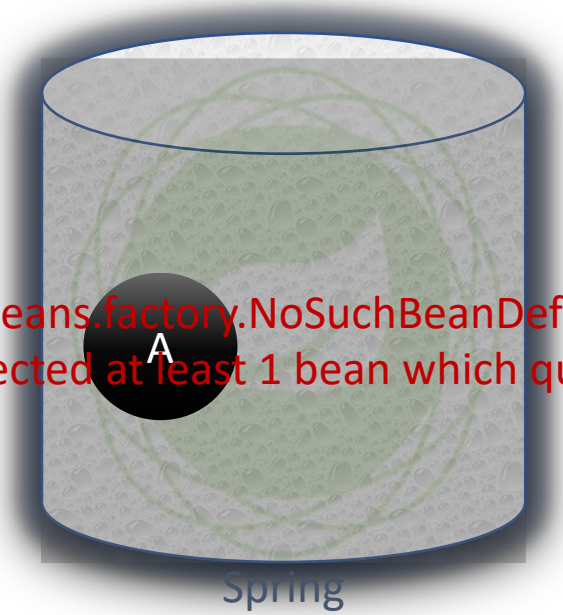
Exception in thread "main" java.lang.NullPointerException

Constructor Injection

```
@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;



}
```

Constructor /Setter/Field Injection ?

Use constructor for mandatory fields

Use setter for optional fields

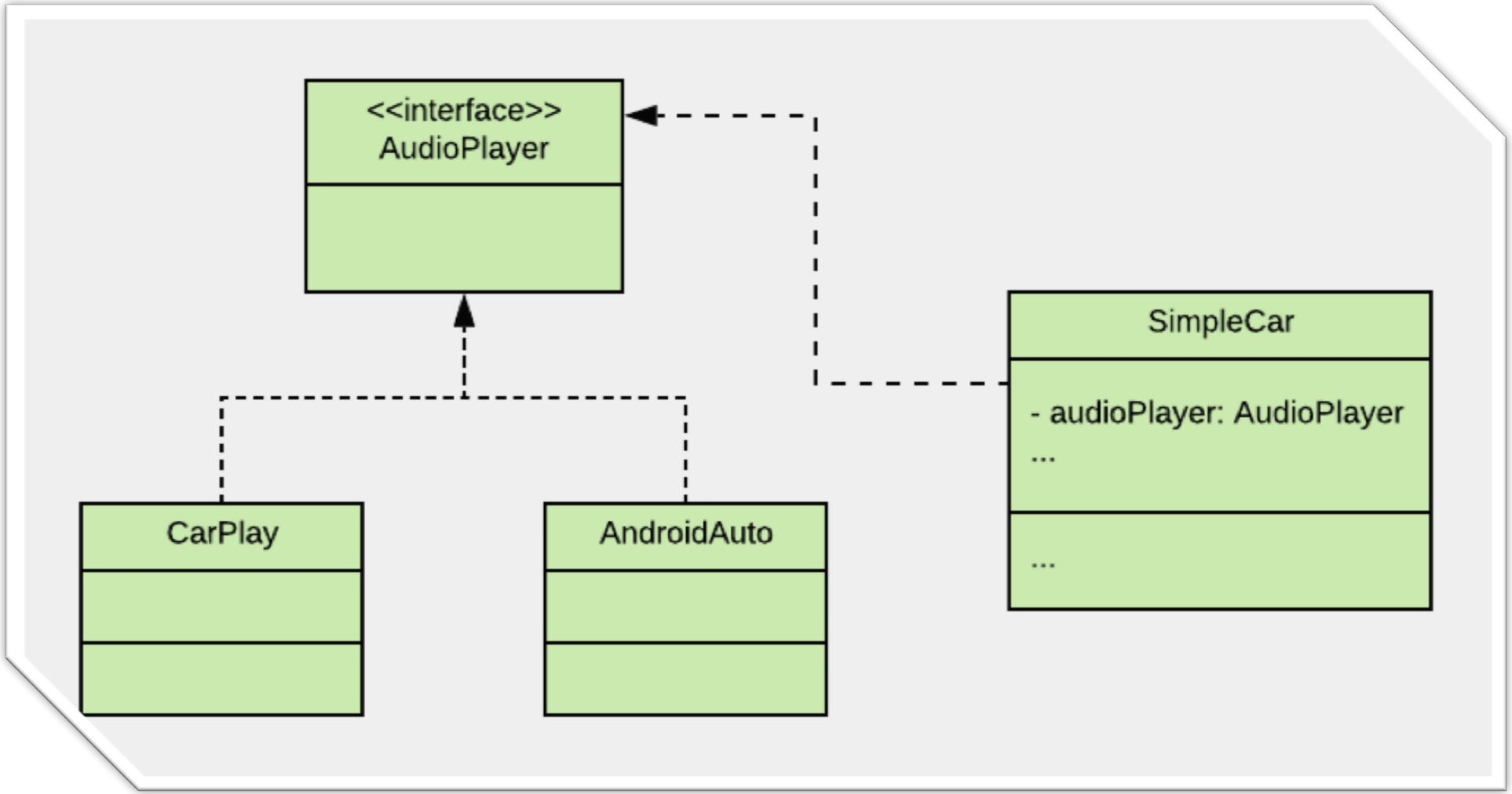
Field injection should be mostly avoided

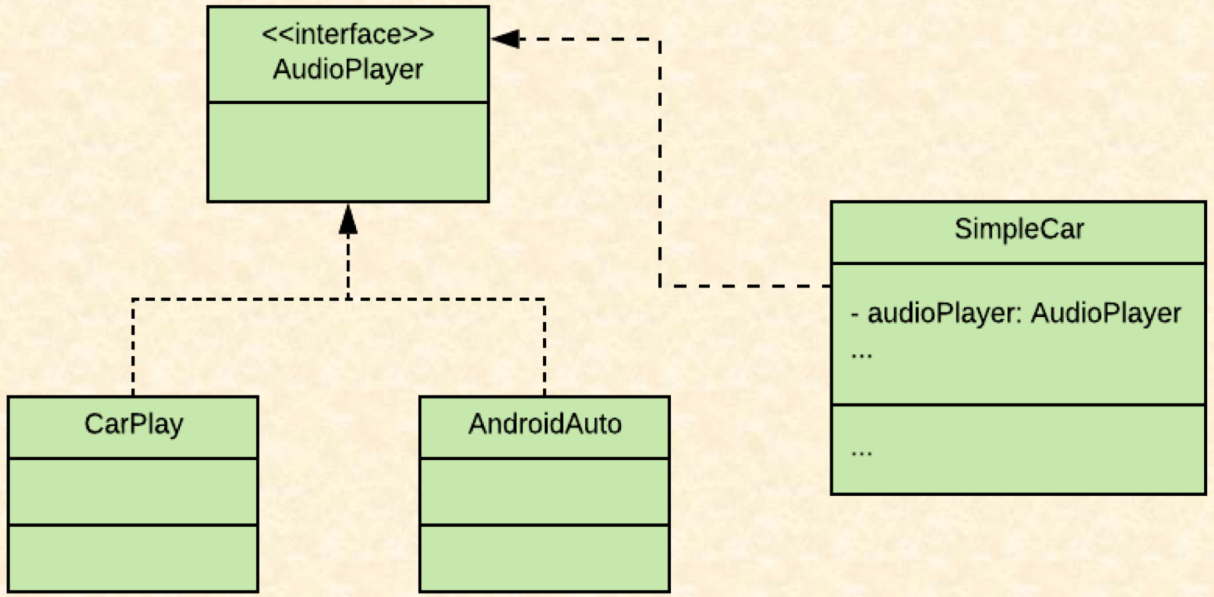
Problems with Field Injection

- Final/Immutable objects not possible.
- could easily violate the SRP
- Unit testing not possible w/o reflection
- Tight coupling to the DI-Framework

Dependency Resolution in conflicts

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

Dependency Resolution Modes

- @Primary
- @Qualifier
- byName
- byType

@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
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

byName

```
public interface AudioPlayer {...}


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



@Component(“carplay”)
public class CarPlay implements AudioPlayer { ... }

@Component
public class SimpleCar {

        @Autowired
        private AudioPlayer carplay;

}
```

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

byName
[fallback]

```
public interface AudioPlayer {...}


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



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

@Component
public class SimpleCar {

        @Autowired
        private AudioPlayer carPlay;

}
```

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

byType

```
public interface AudioPlayer {...}


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



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

@Component
public class SimpleCar {

        @Autowired
        private CarPlay 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;

...

}
```

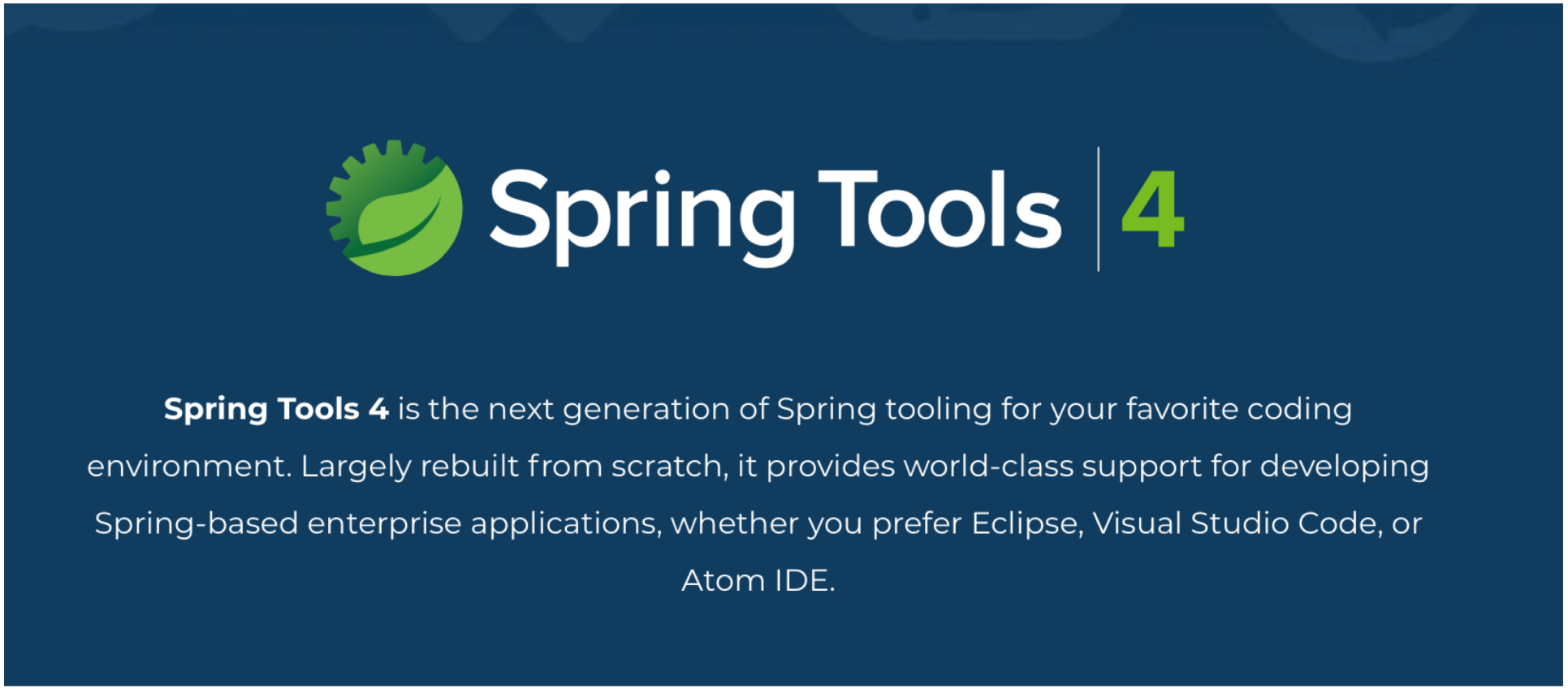
src/main/resources/car.properties

```
wheel.radius=2.34
```

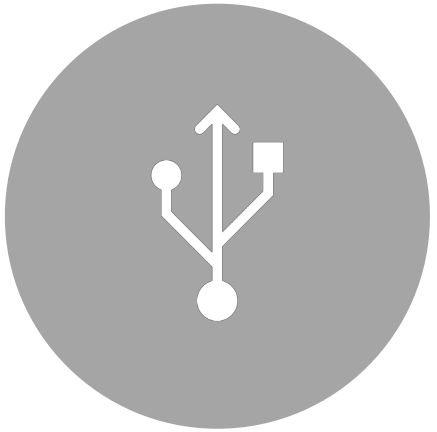
```
@Value("${wheel.radius:2.34}")
private double radius = -1.0d;
```

Tooling

Spring Tool Suite (STS)

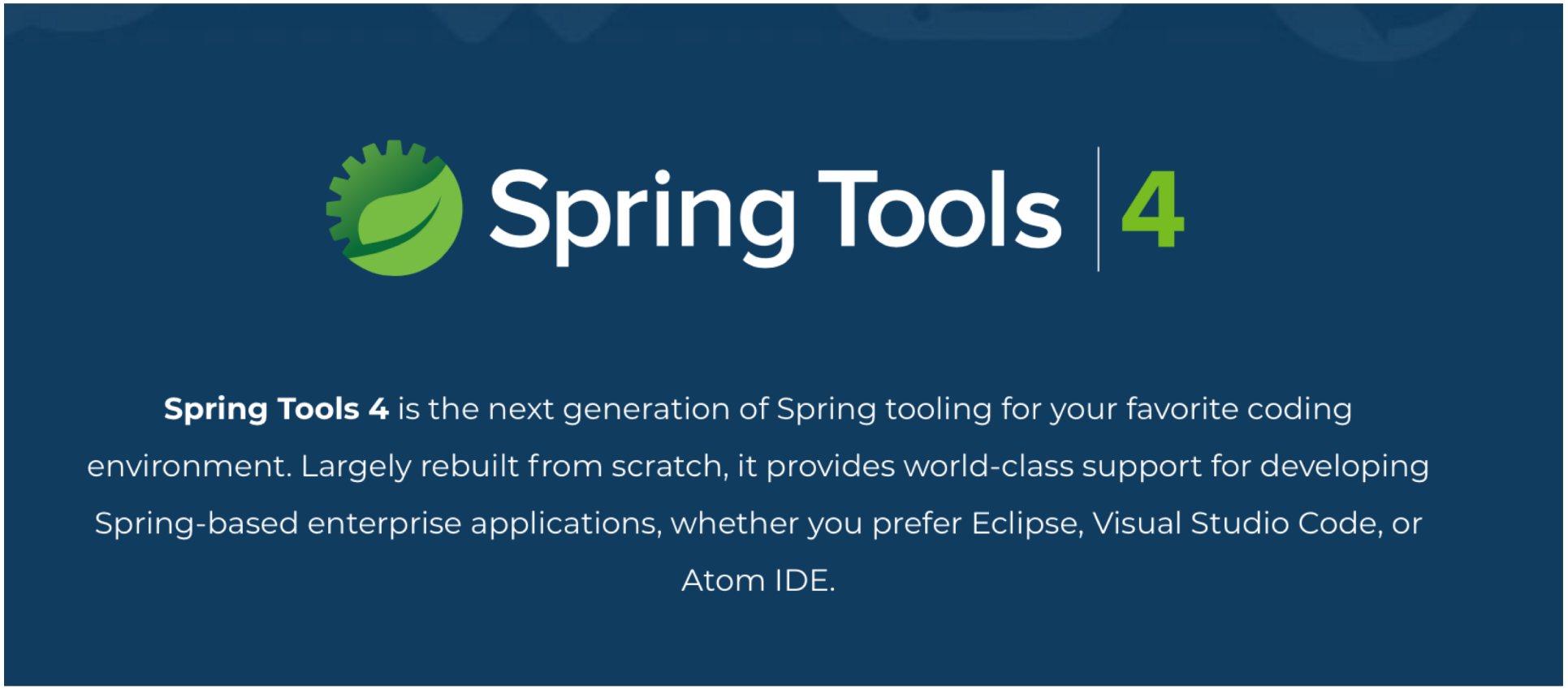


STANDALONE



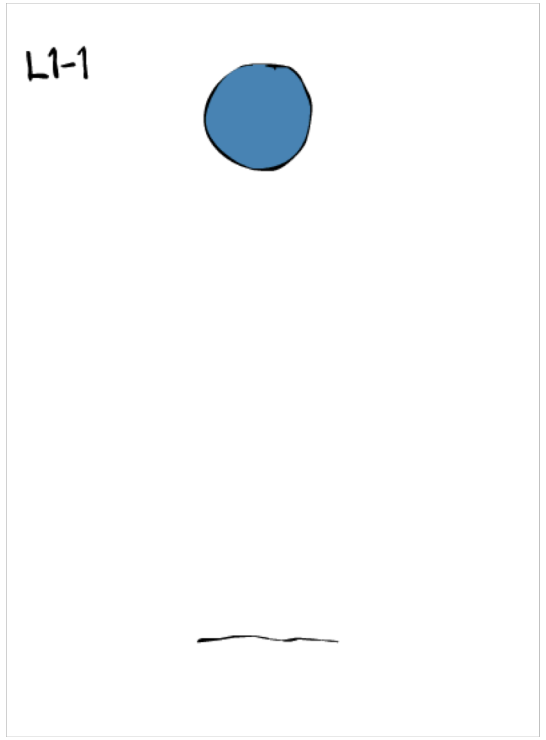
PLUGIN

Spring Tool Suite (STS)

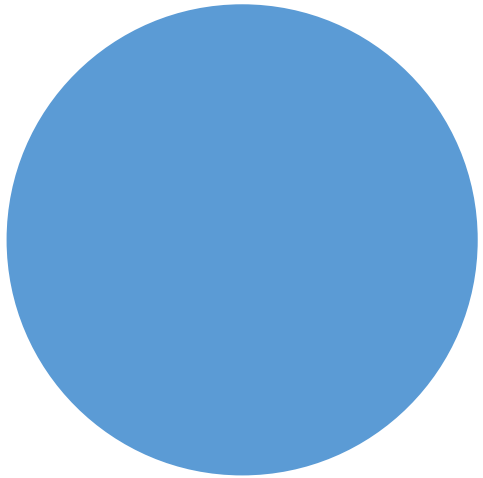
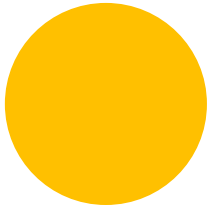


STS Demo

bean-graph



Exercises – Spring Framework I Questions 1 to 8

Spring Framework II

LAZINESS

Eager initialisation

- by default beans are initialised at startup
- is good to prevent errors in dependency configuration

Eagerness

```
@Component
public class SimpleCar {

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

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

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

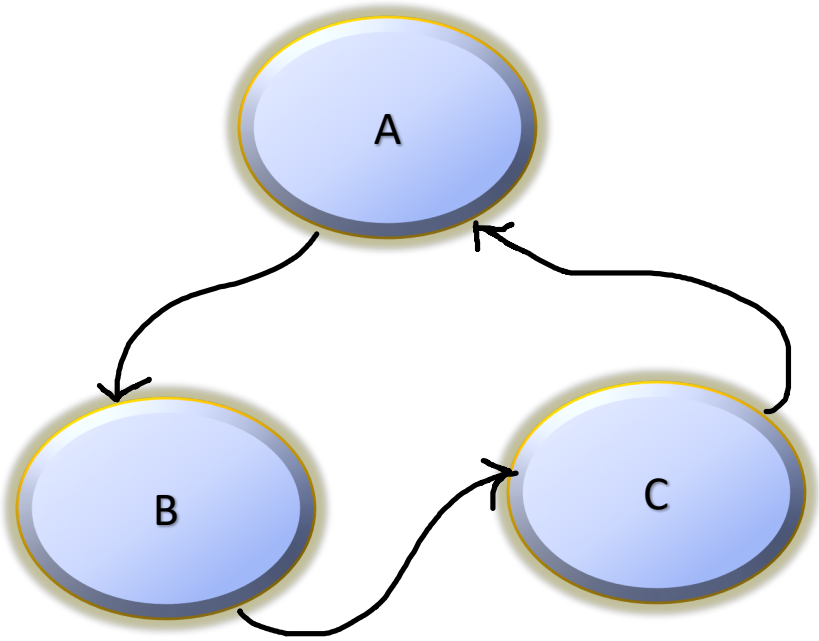
```
@Component
public class AndroidAuto implements AudioPlayer {

        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

```
@Component
public class SimpleCar {

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

  public void playMusic(){
     if(Objects.isNull(audioPlayer))
         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;

public Student(Department department) {
      ...
}

}
```

```
@Component
public class Department {

private Student student;

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

```
@Component
public class Student {

private Department department;

public Student(Department department) {
        ...
}

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

}
```

Circular dependencies solution

- Lazy initialise certain dependencies

```
@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;
    }
```

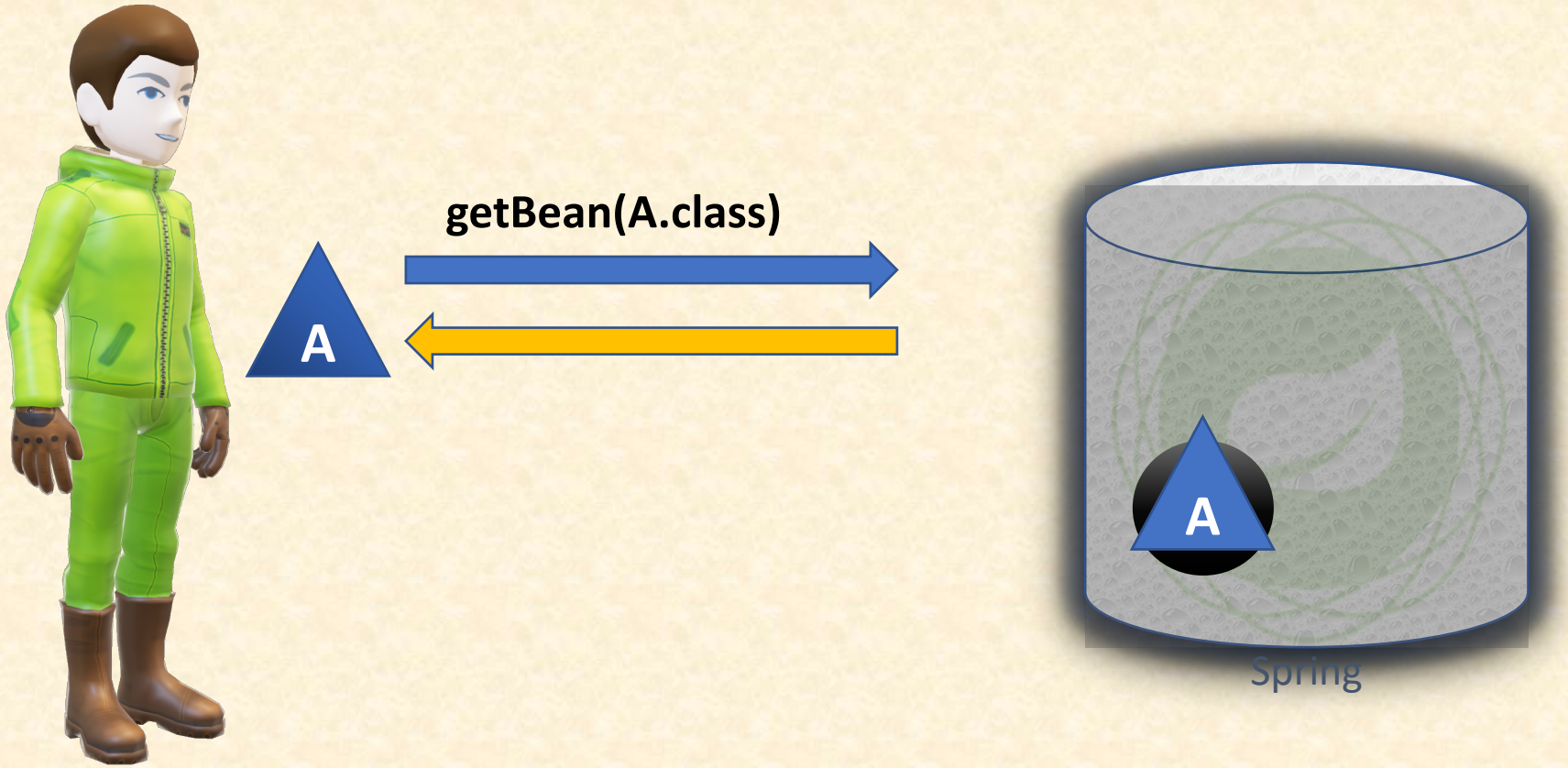
```
department= Department$$EnhancerBySpringCGLIB$$f030f2cb  (id=38)
    CGLIB$BOUND= false
    CGLIB$CALLBACK_0= CglibAopProxy$DynamicAdvisedInterceptor  (id=58)
    CGLIB$CALLBACK_1= CglibAopProxy$DynamicUnadvisedInterceptor  (id=63)
```

Bean Scopes

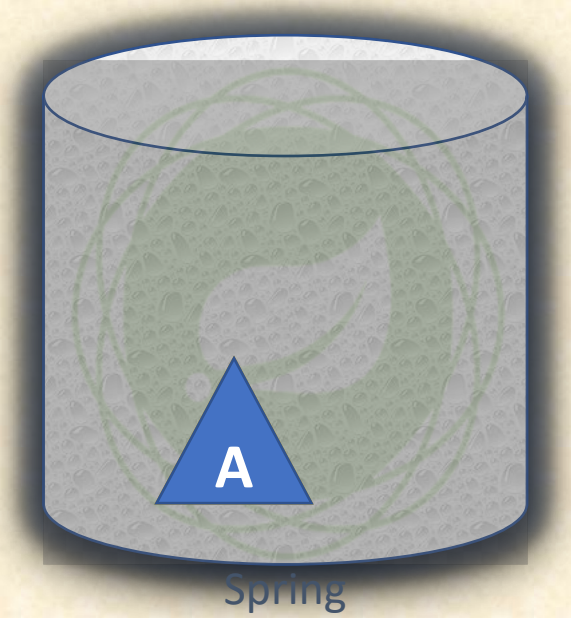
Singleton

Prototype

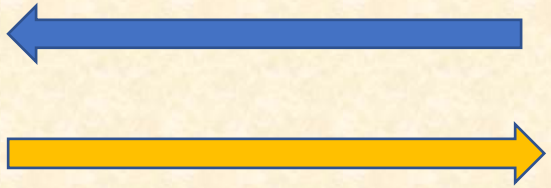
Singleton



Singleton

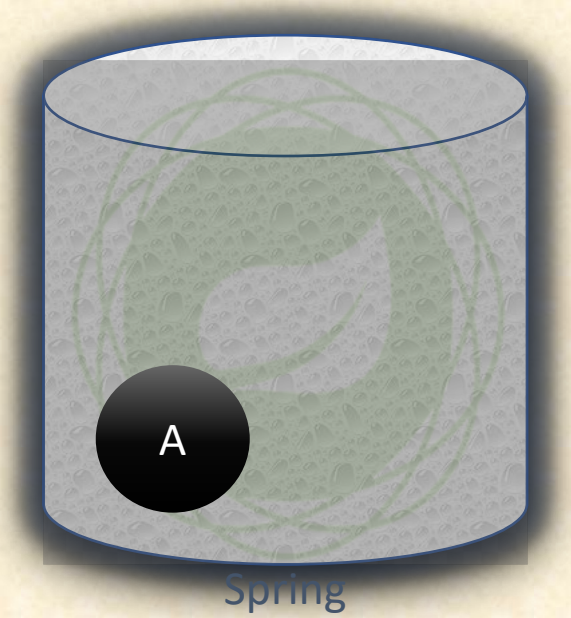


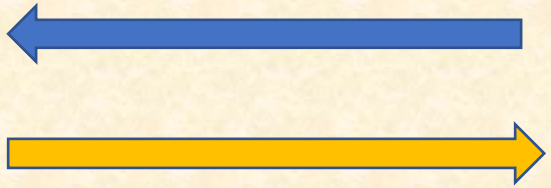
getBean(A.class)



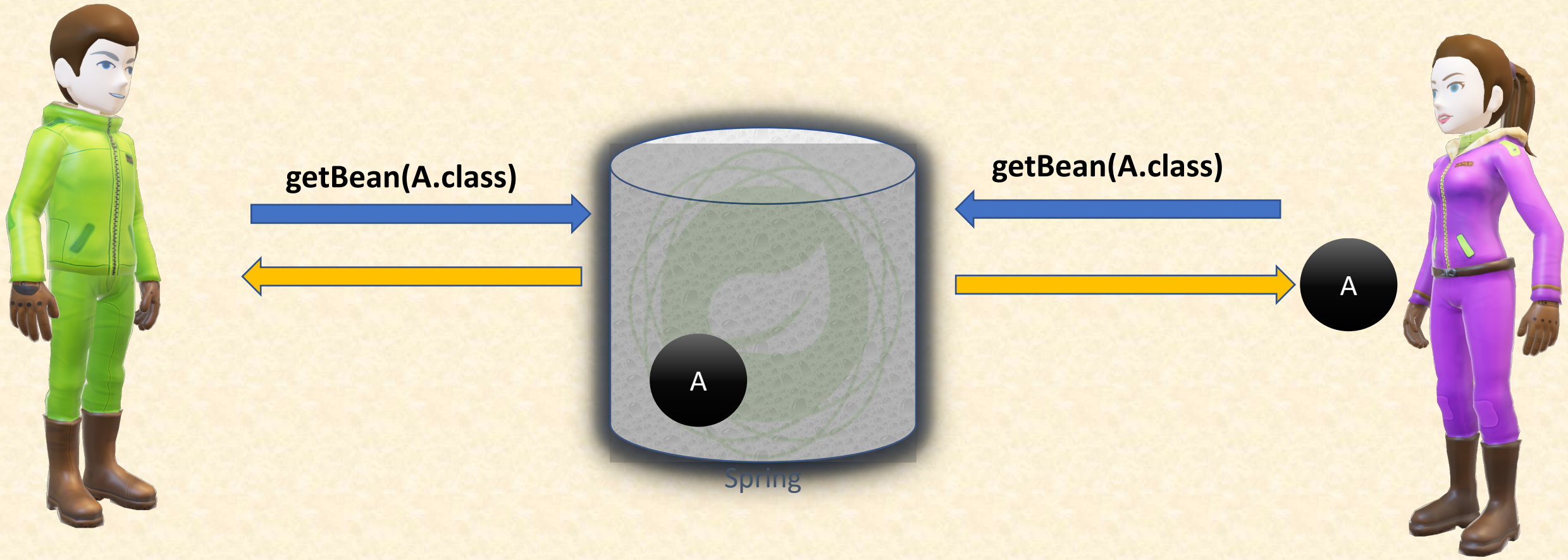
Prototype

In contrast to the other scopes, Spring does not manage the complete lifecycle of a prototype bean. The container instantiates, configures, and otherwise assembles a prototype object and hands it to the client, with no further record of that prototype instance.





Prototype



Scope Configuration

- @Scope(scopeName=ConfigurableBeanFactory.SCOPE_PROTOTYPE)
- @Scope(scopeName=ConfigurableBeanFactory. SCOPE_SINGLETON)
- @Scope(“prototype“)
- @Scope(“singleton“)

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

Singleton Example

```
@Component
public class Shape {

private String colour="yellow";

//setter

}
```

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

public static void main(String[] args) {
	var container = //annotationcontext

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

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

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

	}
}
```

INFO: Shape [colour=yellow]

INFO: Shape [colour=blue]

INFO: Shape [colour=blue]

Prototype Example

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

private String colour="yellow";

//setter

}
```

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

public static void main(String[] args) {
	var container = //annotationcontext

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

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

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

	}
}
```

INFO: Shape [colour=yellow]

INFO: Shape [colour=blue]

INFO: Shape [colour=yellow]

Singleton is per container and not per 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 [colour=yellow]

INFO: Shape [colour=blue]

INFO: Shape [colour=yellow]

Singleton vs Prototype

- Singleton => Stateless beans
- Prototype => Stateful beans

Thread safeness

- Spring beans are not thread-safe.
- We have to make sure they are thread-safe by:
    - by not using instance variables in multithreaded environment.

OR

    - Use synchronized block/keyword on methods wherever the instance variables are modified

Bean Lifecycle Callbacks

- Initialization callbacks
- Destruction callbacks

Initialization callbacks

- @PostConstruct
- InitializingBean
- initMethod 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();
        }


}
```

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();
        }

}
```

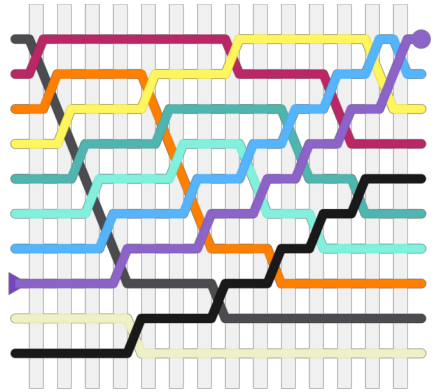
initMethod in @Bean

```
@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();
}

}
```

Initialization callbacks precedence

- @PostConstruct
- InitializingBean
- initMethod in @Bean



Destruction callbacks

- @Predestroy
- DisposableBean
- destroyMethod in @Bean

@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();
        }


}
```

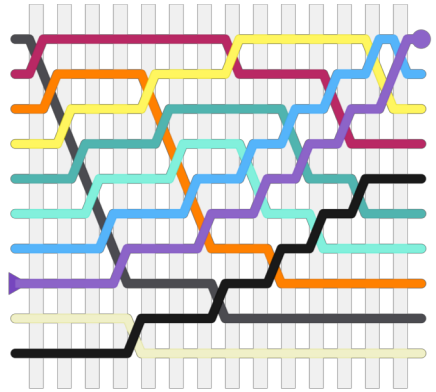
destroyMethod in @Bean

```
@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();
}

}
```

Destruction callbacks precedence

- @PreDestroy
- DisposableBean
- destroyMethod in @Bean



Web-MVC

Refresh …

Servlet

- Java program that runs on server.

Servlet container

- A software that provides runtime for the servlets. (eg) Tomcat, Jetty

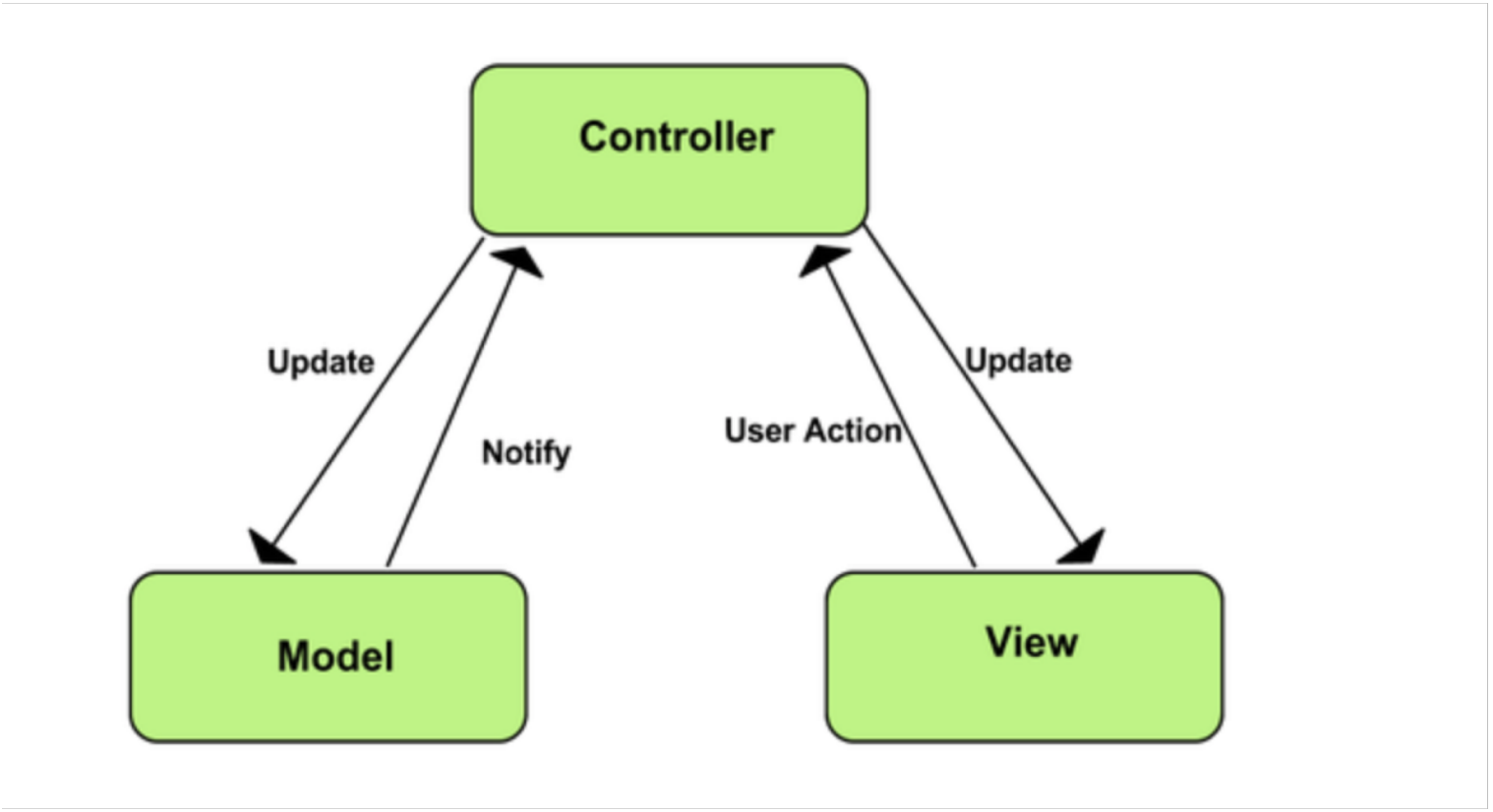
Servlet – An example

```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
      response.setContentType("text/html");
      request.authenticate(response);
      PrintWriter writer = response.getWriter();
      writer.println("<html>");
      writer.println("<head>");
      writer.println("this is simple serlvet example");
      writer.println("</head>");
      writer.println("<body>");
      writer.println("Hello HBT :-)");
      writer.println("</body>");
      writer.println("</html>");
}
```

Servlet – An example

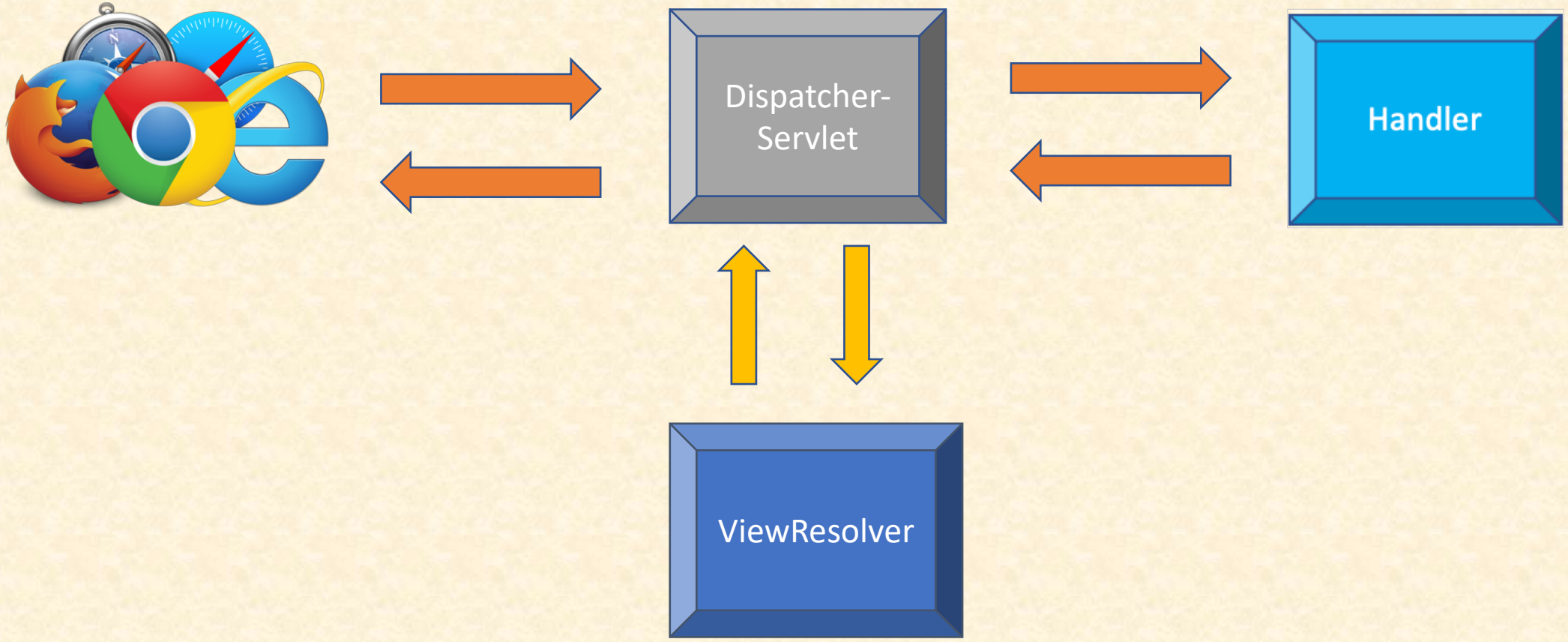
```
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

      authenticate(response);

      createView(response);
}
```

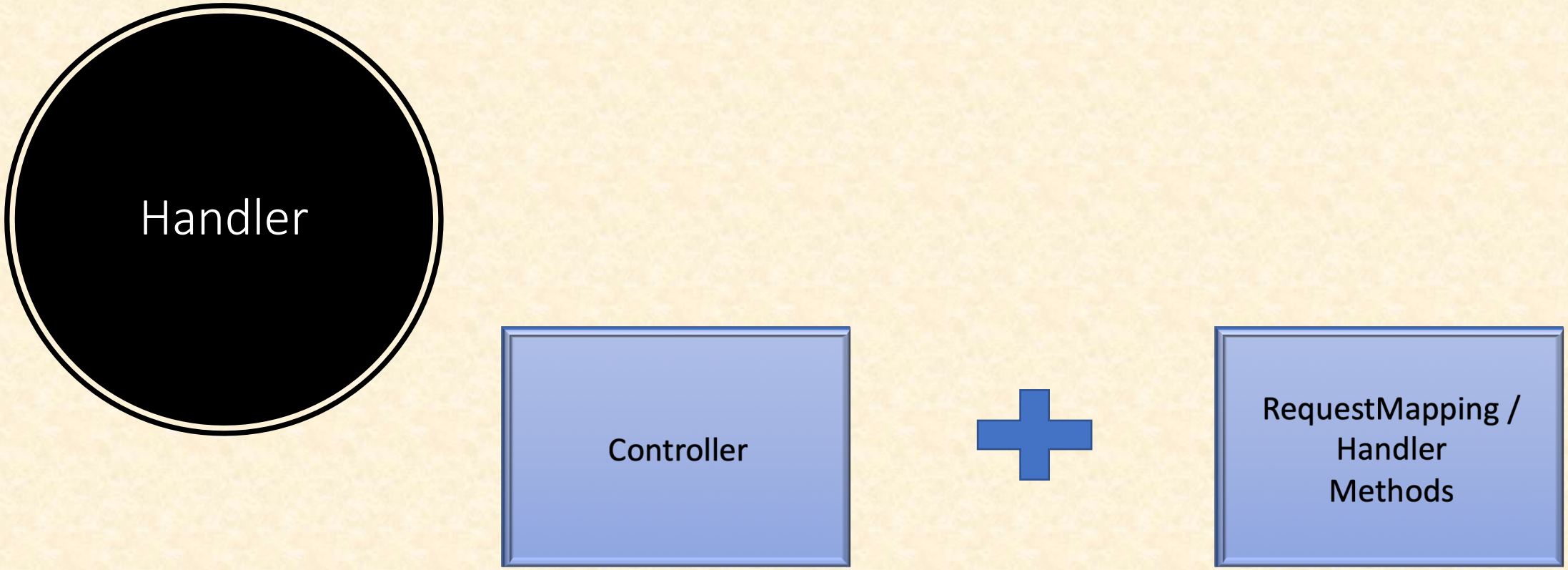
MVC



https://developer.chrome.com/apps/app_frameworks#mvc

Spring Web-MVC – FrontController Pattern




Handler
Controller
RequestMapping /
Handler
Methods

Controller

@Controller

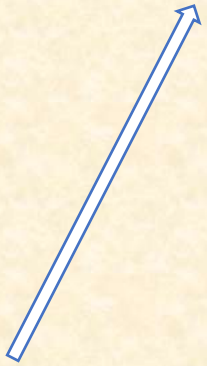
@RestController

@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 getItems(){
            //return the view name for the shopping lists
    }


}
```

<Request mapping method>>

@RestController

@Controller @RestController

```
public class ShoppingBasket{


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

}
```

@RestController

```
@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

```
@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(…)

@RequestMapping is both class level as well as method level annotation
@GetXXX,@PutXX etc are method level annotations

Elements of @XXXMapping

- consumes
- produces
- method
- params
- headers

to narrow down the request mappings

produces => accept header

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

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

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

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

MediaType

Field Summary

Fields

Modifier and Type	Field and Description
static **MediaType**	**ALL** Public constant media type that includes all media ranges (i.e.
static java.lang.String	**ALL_VALUE** A String equivalent of **ALL**.
static **MediaType**	**APPLICATION_ATOM_XML** Public constant media type for `application/atom+xml`.
static java.lang.String	**APPLICATION_ATOM_XML_VALUE** A String equivalent of **APPLICATION_ATOM_XML**.
static **MediaType**	**APPLICATION_FORM_URLENCODED** Public constant media type for `application/x-www-form-urlencoded`.
static java.lang.String	**APPLICATION_FORM_URLENCODED_VALUE** A String equivalent of **APPLICATION_FORM_URLENCODED**.
static **MediaType**	**APPLICATION_JSON** Public constant media type for `application/json`.
static **MediaType**	**APPLICATION_JSON_UTF8** Public constant media type for `application/json;charset=UTF-8`.
static java.lang.String	**APPLICATION_JSON_UTF8_VALUE** A String equivalent of **APPLICATION_JSON_UTF8**.
static java.lang.String	**APPLICATION_JSON_VALUE** A String equivalent of **APPLICATION_JSON**.
static **MediaType**	**APPLICATION_OCTET_STREAM** Public constant media type for application/octet-stream

consumes => content-type header

```
@PutMapping(value="/items", consumes= {MediaType.APPLICATION_JSON_VALUE})
public List<ShoppingItem> updateItem(@RequestBody ShoppingItem item){
      //update Item
}
```

```
curl -H "Content-Type: application/json" -X PUT -d <JSON_DATA>
http://localhost:8080/items
```

headers

```
@GetMapping(value="/items", headers="quantity=3")
public List<ShoppingItem> itemsWithHeaderFiltered(){

	//return items
}
```

```
curl -H "quantity=3" http://localhost:8080/items
```

params

```
@GetMapping(value="/items", params="quantity=2")
public List<ShoppingItem> itemsWithParamterFiltered(){
	//return items
}
```

```
curl http://localhost:8080/items?quantity=2
```

Get a particular item

https://localhost:8080/items/1

```
@GetMapping("/items/1")
public List<ShoppingItem> items(){
        //return first item
}
```

https://localhost:8080/items/2

```
@GetMapping("/items/2")
public List<ShoppingItem> items(){
        //return second item
}
```

https://localhost:8080/items/3

```
@GetMapping("/items/3")
public List<ShoppingItem> items(){
        //return third item
}
```

```
@GetMapping("/items/{itemiId}")
public List<ShoppingItem> items(@PathVariable String itemId){
        //return first item
}
```

Get a particular item

Get a particular item

Bean Scopes

Singleton

Prototype

Bean Scopes

Singleton

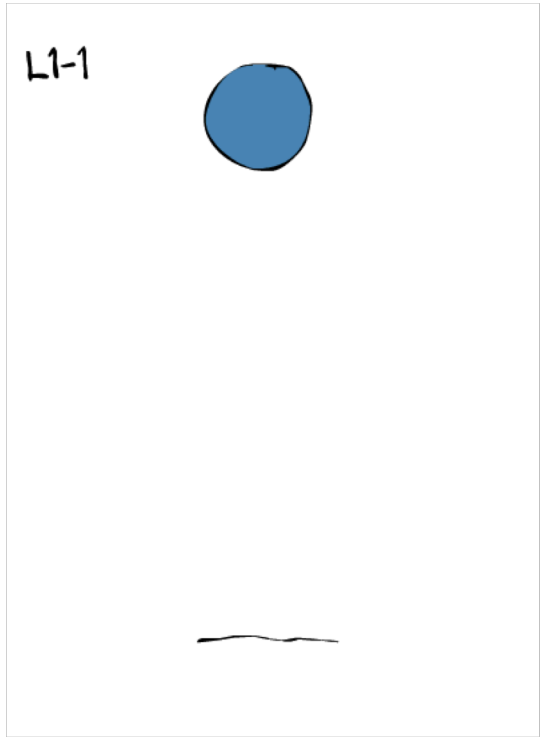
Prototype

Request

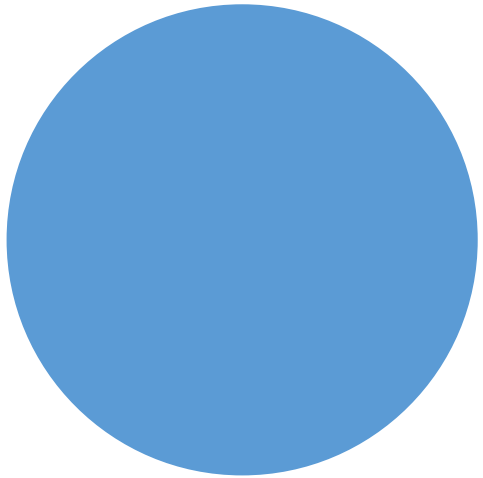
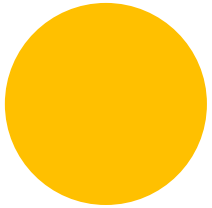
Session

Application

WebSocket

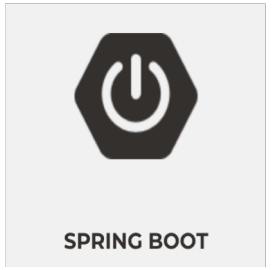


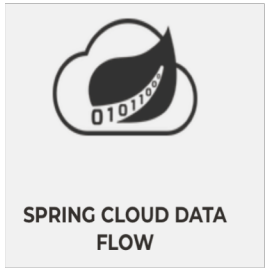
Exercises – Spring Framework I & II
Questions 9 to 15

Spring Boot I

Major Spring Projects





Major Spring Projects

WebApp in Spring Framework (The hard way)

```
public class WebInitializer implements WebApplicationInitializer {


@Override
public void onStartup(ServletContext servletContext) {
        //configure dispatcher servlet
}

}
```

```
public static void startTomcat() {
        final var tomcat = new Tomcat();
        final var connector = new Connector();
        //configure tomcat
}
```

```
@Configuration
@EnableWebMvc
@ComponentScan("ch.karthi")
public class WebAppConfig {


@Bean
public MappingJackson2HttpMessageConverter jsonMessageConverter() {
    return new MappingJackson2HttpMessageConverter();
}

}
```

WebApp in SpringBoot 1

```
@SpringBootApplication
public class App {


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


}
```

WebApp in SpringBoot 2

```
@SpringBootApplication
public class HBTApp {


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


}
```

Hello HBT !

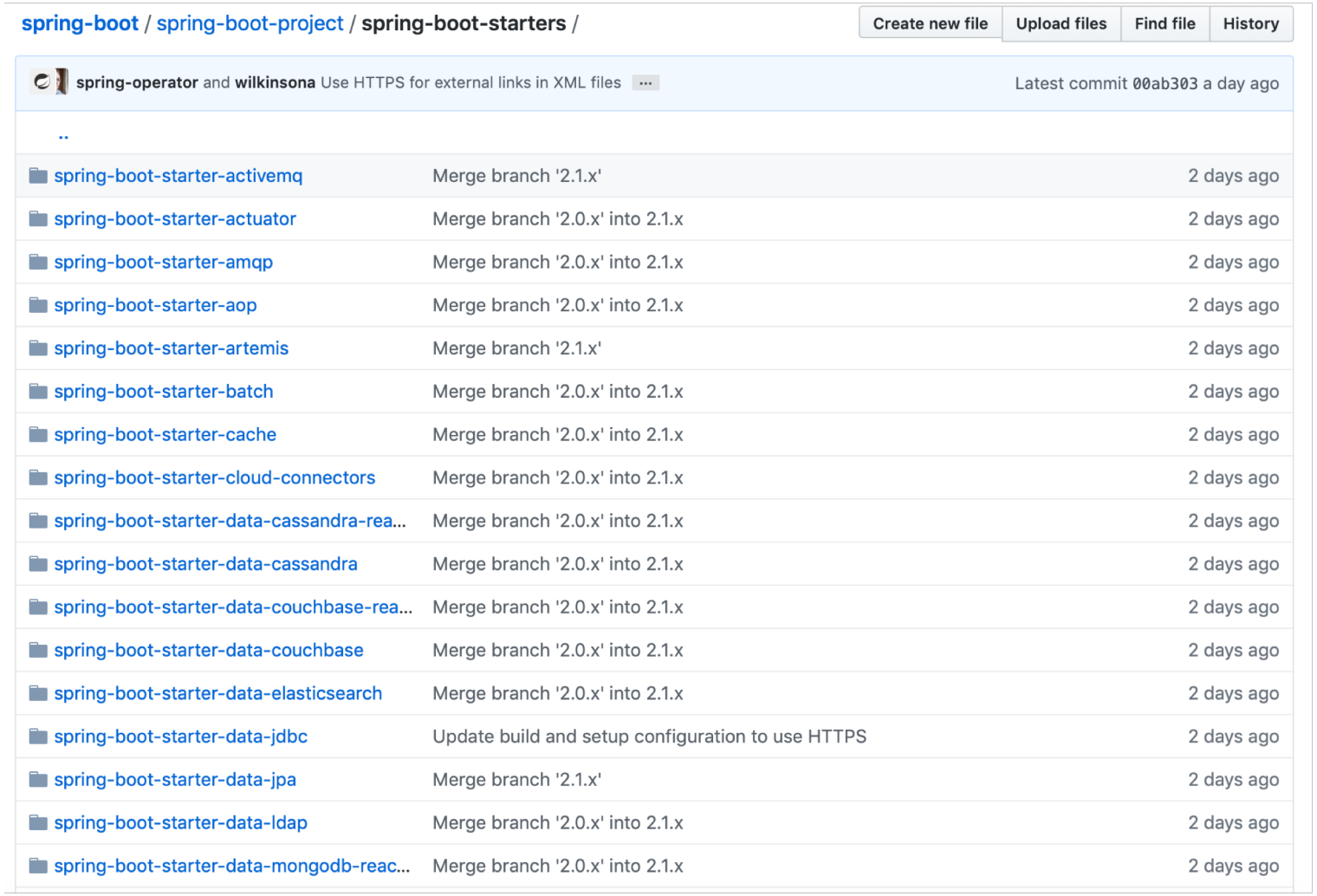
Spring Boot

- Convention over configuration
- RAD (Rapid Application Development)
- Microservice
- Cloud Native

Starters

- “ one-stop shop“ for all the dependencies
- Need not have to hunt for the dependencies
- just add the starter dependency and start writing business code

Starter Projects



spring-boot / spring-boot-project / spring-boot-starters /

Create new file | Upload files | Find file | History

spring-operator and wilkinsona Use HTTPS for external links in XML files ... Latest commit 00ab303 a day ago

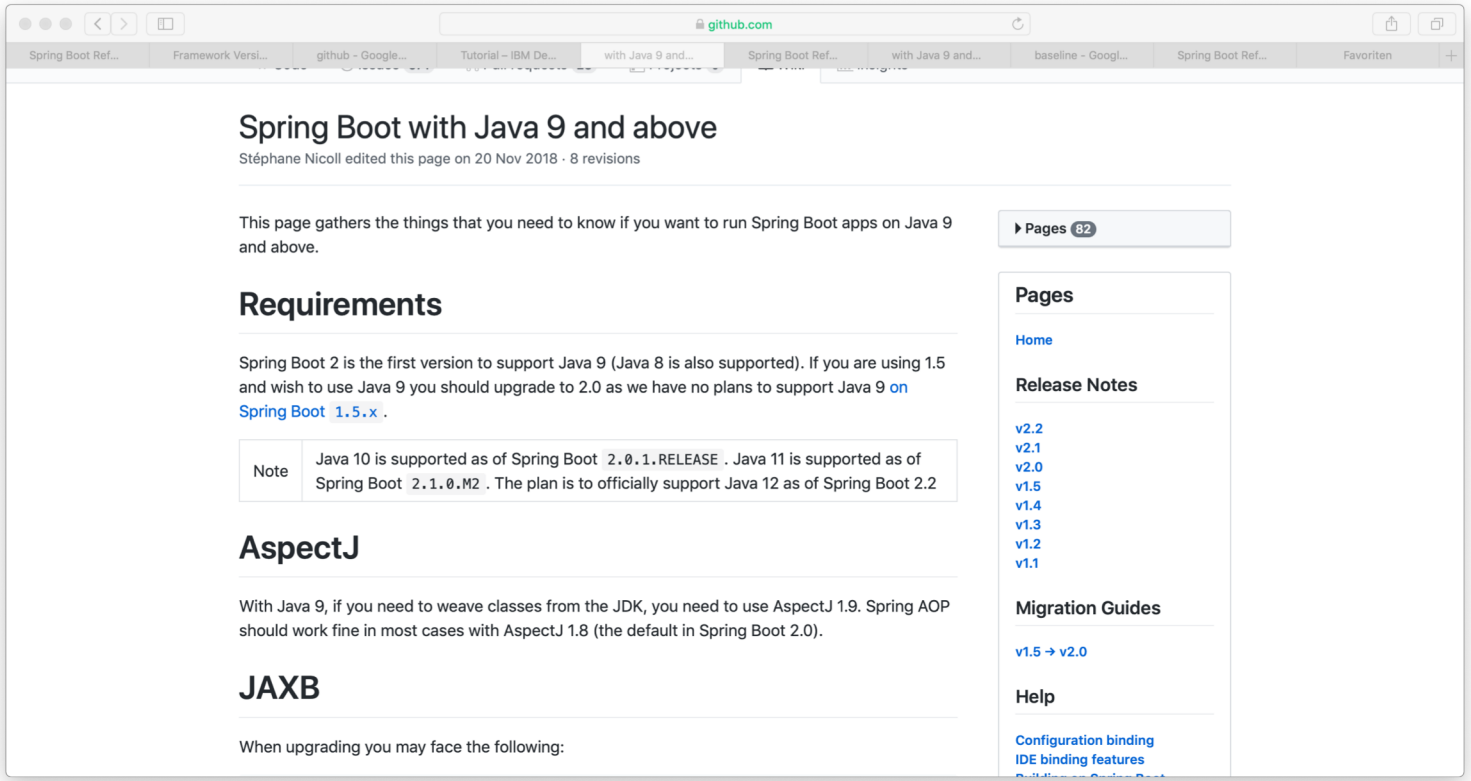
..

spring-boot-starter-activemq	Merge branch '2.1.x'	2 days ago
spring-boot-starter-actuator	Merge branch '2.0.x' into 2.1.x	2 days ago
spring-boot-starter-amqp	Merge branch '2.0.x' into 2.1.x	2 days ago
spring-boot-starter-aop	Merge branch '2.0.x' into 2.1.x	2 days ago
spring-boot-starter-artemis	Merge branch '2.1.x'	2 days ago
spring-boot-starter-batch	Merge branch '2.0.x' into 2.1.x	2 days ago
spring-boot-starter-cache	Merge branch '2.0.x' into 2.1.x	2 days ago
spring-boot-starter-cloud-connectors	Merge branch '2.0.x' into 2.1.x	2 days ago
spring-boot-starter-data-cassandra-rea...	Merge branch '2.0.x' into 2.1.x	2 days ago
spring-boot-starter-data-cassandra	Merge branch '2.0.x' into 2.1.x	2 days ago
spring-boot-starter-data-couchbase-rea...	Merge branch '2.0.x' into 2.1.x	2 days ago
spring-boot-starter-data-couchbase	Merge branch '2.0.x' into 2.1.x	2 days ago
spring-boot-starter-data-elasticsearch	Merge branch '2.0.x' into 2.1.x	2 days ago
spring-boot-starter-data-jdbc	Update build and setup configuration to use HTTPS	2 days ago
spring-boot-starter-data-jpa	Merge branch '2.1.x'	2 days ago
spring-boot-starter-data-ldap	Merge branch '2.0.x' into 2.1.x	2 days ago
spring-boot-starter-data-mongodb-reac...	Merge branch '2.0.x' into 2.1.x	2 days ago

Spring Boot - JDK Baseline

Version	JDK 6	JDK 7	JDK 8	JDK 9	JDK 10	JDK 11	JDK 12
1.5.x							
2.1.X							
2.2.x							

Spring Boot with JDK 9 and above



Spring Initializr