Spring

Why Spring:

- Alternative to EJB.
- EJB Applications are heavy weight and tightly coupled. (App server dependency)
- Spring applications are light weight and loosely coupled.(doesn't use app server)
- Spring reduces investment cost as we don't need to buy services from servers
- Spring is free source.
- Spring works with POJO classes.

Spring recommends to use associations instead of inheritance
Eg: if class D needs the functionality of class B and C then same can be achieved as below

```
Class D {                    class B{
        B b = new B ();      }
        C c = new C();       class C{
                             }
}
```

Spring Implementation is based on 2 principles *Association* and *Runtime polymorphism.*
Instead of passing Runtime arguments through command line we can pass via XML.
For working with XML we need Container support. IOC is the required Spring container.

Spring Container:

IOC has Core, J2EE containers.
Spring MVC has web containers build on top of above 2

<u>IOC:</u>
Core – Bean factory (Interface)→XMLBeanFactory(class)
J2EE- ApplicationContext(Interface)→ConfigurableApplicationContext(interface)→ ClasspathXMLApplicationContext(Class)

Core container creates object on calling getBean() method→ lazy container
J2EE Container creates while loading xml→eager container

<u>MVC:</u>
Web- webApplicationContext(Interface)→WebApplicationContextUtils

<u>Containers features:</u>
- Read xml file
- Create Instances of XML declarations (java classes (POJO) or bean classes)
- Manage life cycle of bean classes

- Dynamic parameter to bean classes from XML file ***(*dependency injections)***

DI(dependency injection) Helps to create loose coupling

Steps needed for starting containers:

1. Driver class (any class containing main method)
2. Create Object of Container classes

```
Eg: Class Test{
PSVM(string args[]){
        New XMLBeanFacory();
        New ClasspathXMLApplicationContext();
        WebApplicationContextUtils.getObject();
```

For web application it should be written in init()

Main components for Spring Application:

HelloWorld

1. POJO class
2. XML file
3. Driver class

Pojo class

```
Class Test{
Public static void hello(){
Sop("hello world")
}
}
```

Spring.xml

```
Dtd/xsd
<beans>
<bean class="Test" id="t" singleton="true"></bean>
</beans>
```

Driver Class:

```
Class Client{
Psvm(String args[]){
Resource rs = new ClasspathResource("Spring.xml")
BeanFactory factory = new XMLBeanFactory(r);
Test t = (Test) Factory.getBean("t");
Test t1 = (Test) Factory.getBean("t");

t.hello();
}
```

}

Note:- Spring dtd can be found at spring-beans.jar(org,springframework.core.factory.xml.spring-beans – version.dtd)
In the above scenario although we are using t and t1 Spring will create only one object.
If we make singleton=false it will have multiple objects

Spring scope: singleton and prototype / request, session, context

Spring Environment Setup:

1. Create Eclipse project
2. Download Spring Jar files
3. Add jar files to Eclipse project (builpath)

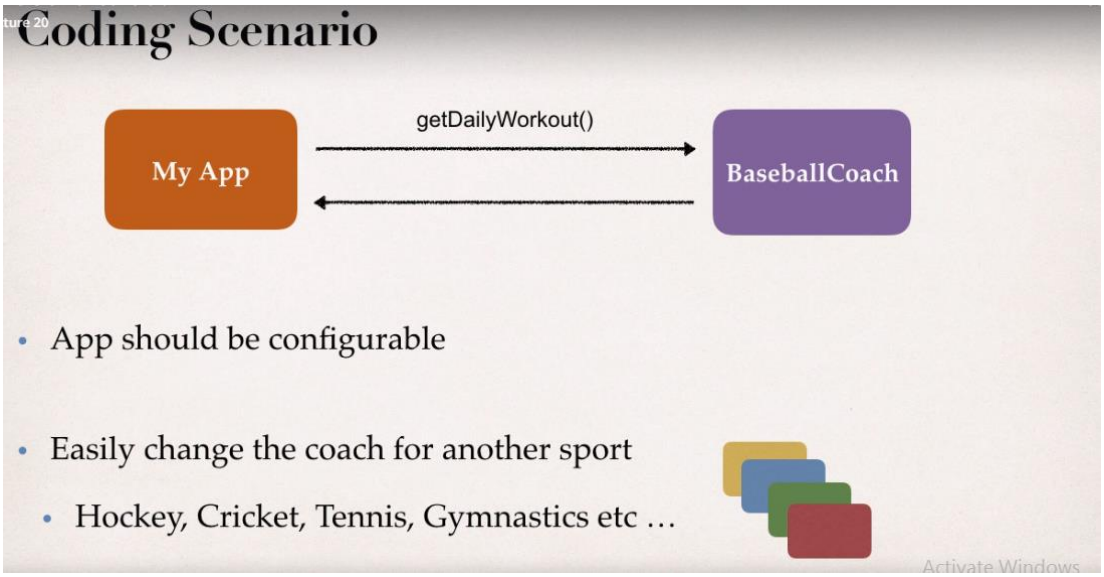
Steps:

- create a simple java project in eclipse
- download spring jar - www.luv2code.com/downloadspring
- copy all jars under lib folder
- Right click project >properties > java buildpath > add jar from lib folder

Spring IoC:

The process of outsourcing the construction and management of Objects to an Object Factory is termed as Inversion of Control (IoC)

Coding Scenario

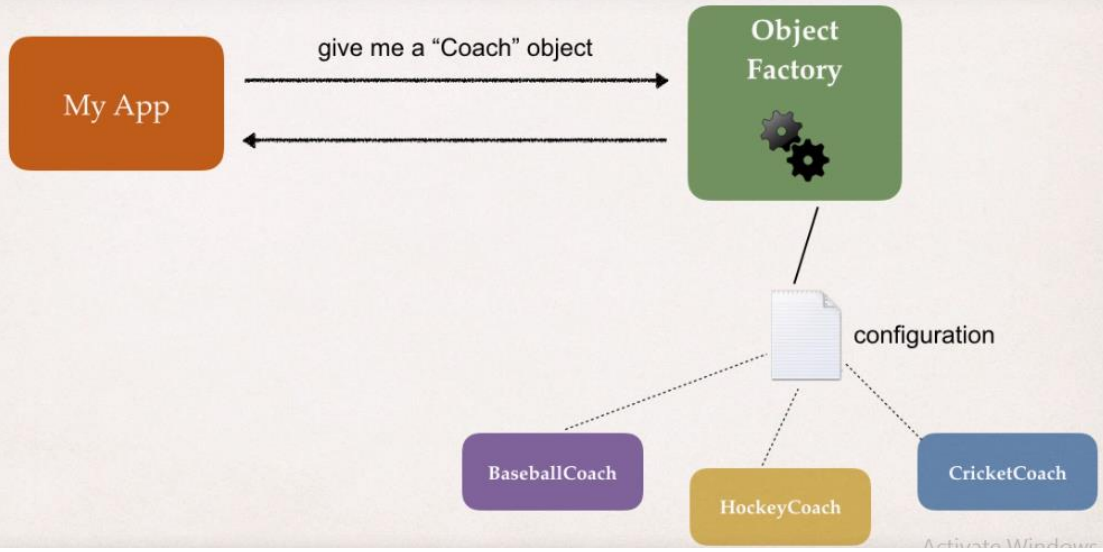


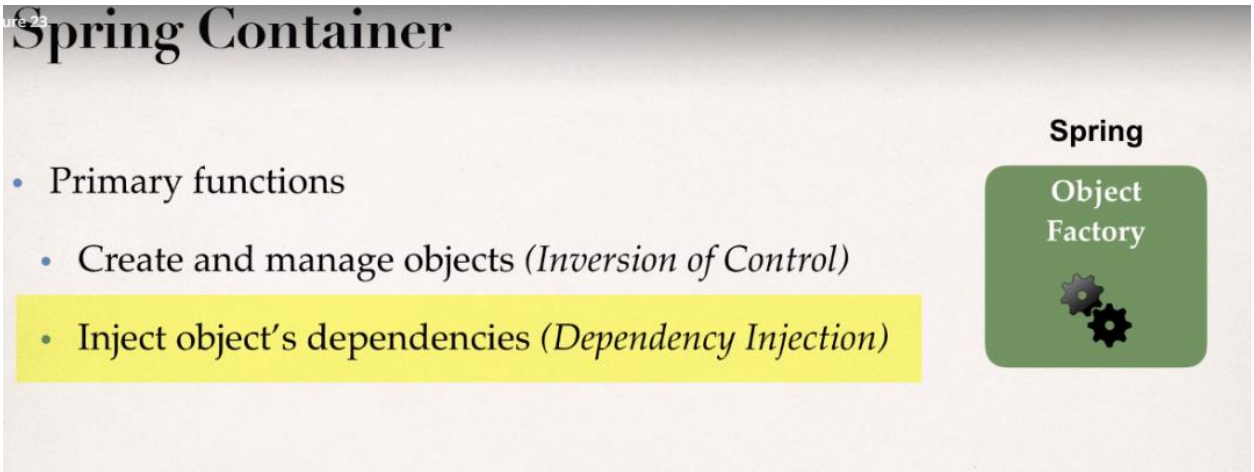
Code Demo

- **MyApp.java**: main method
- **BaseballCoach.java**
- **Coach.java**: interface after refactoring
- **TrackCoach.java**

MyAPP →driver class
baseBallCoach → POJO class
TrackCoach → POJO class
Coach.java →Interface to achieve compatibilty

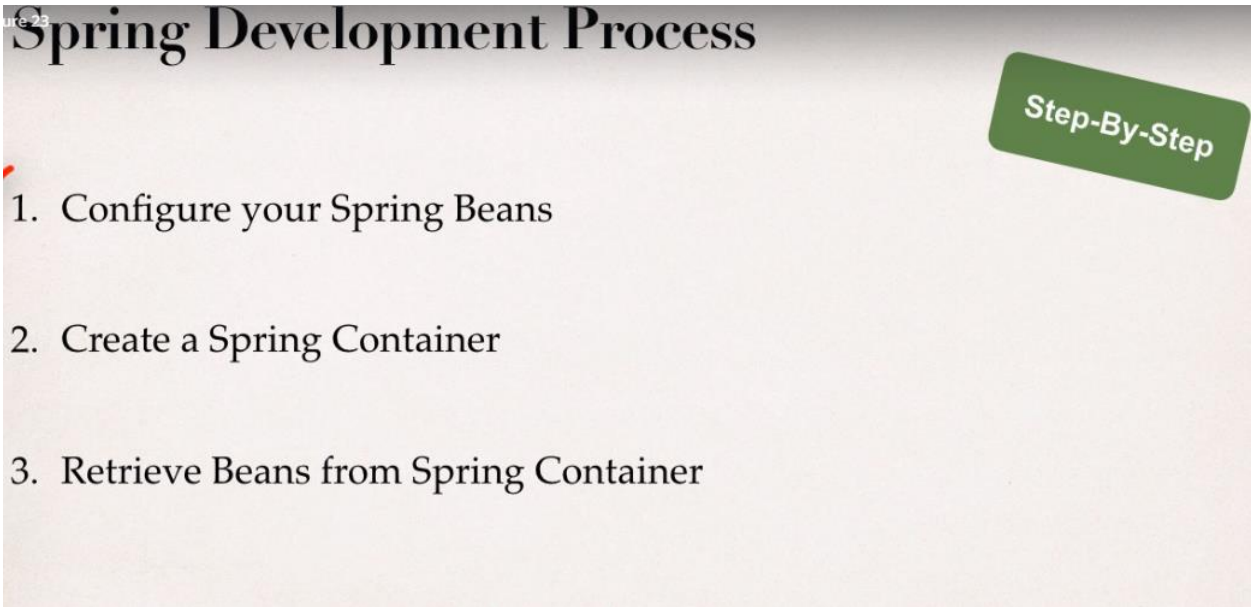
Ideal Solution





Configuring Spring Container:

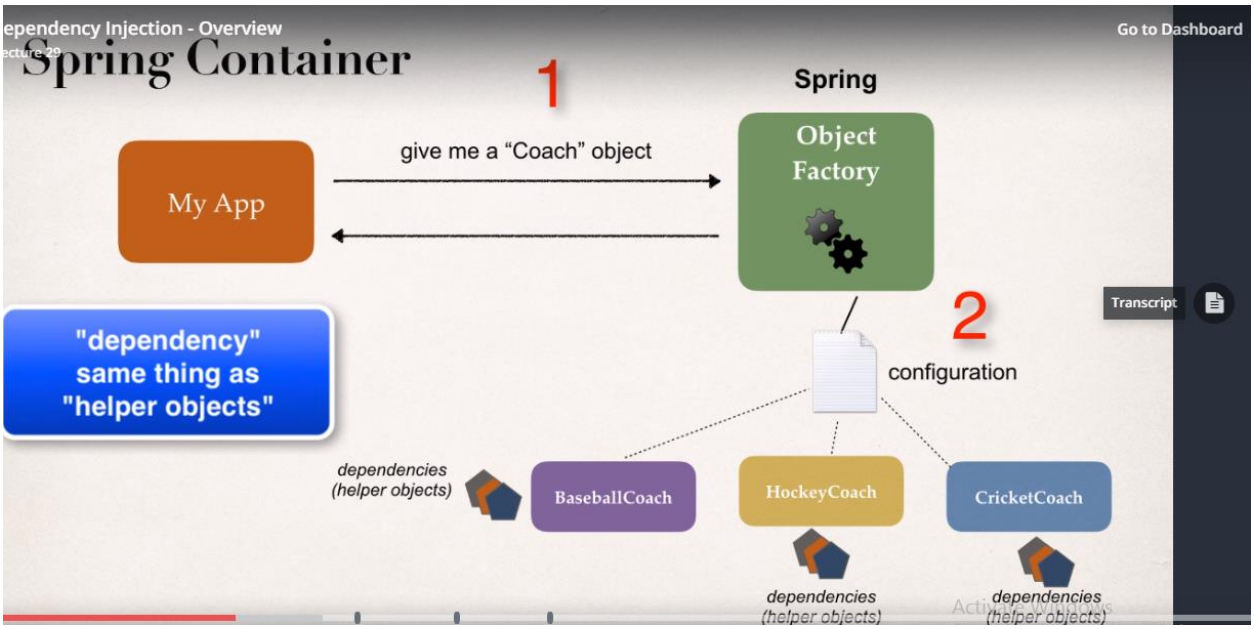
- XML Configuration file
- Java Annotations
- Java source code

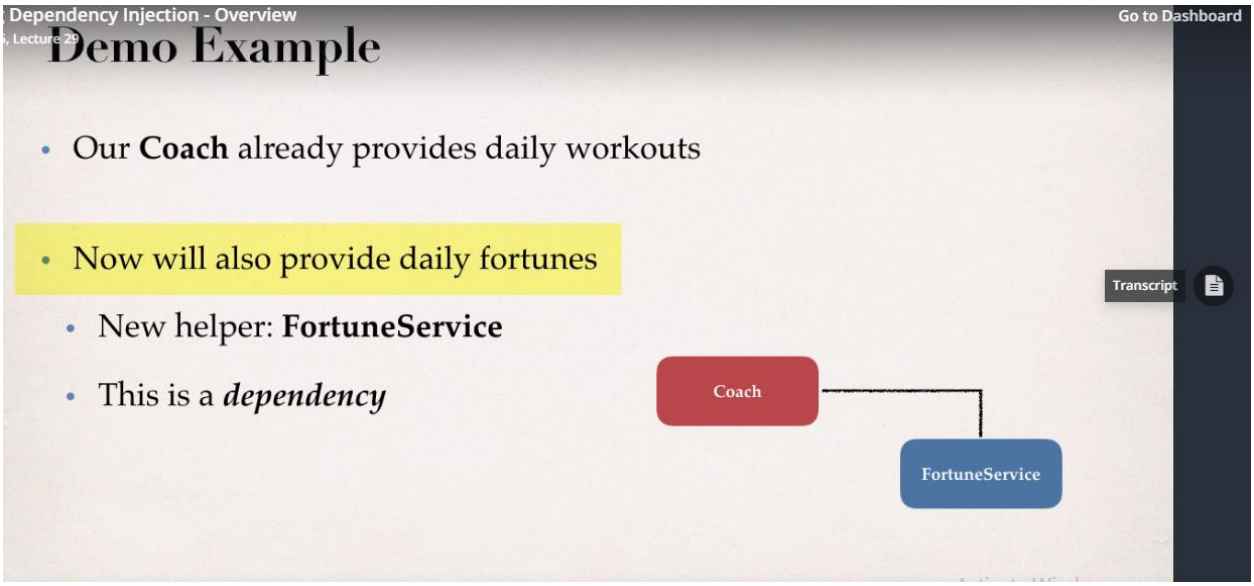


HelloSpringApp

Dependency Injection:

Outsourcing the construction and Injection of Object to External entity, eg car factory.
Dependency is same thing as helper objects

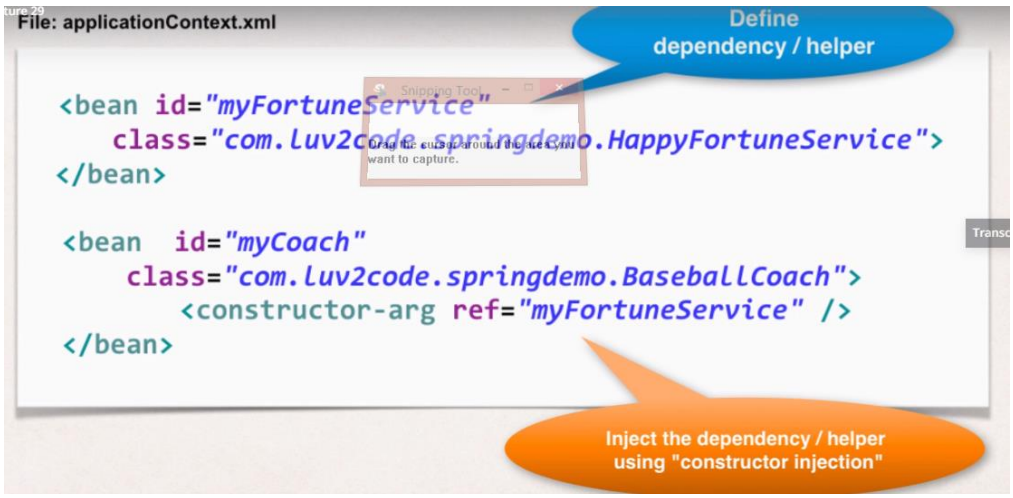




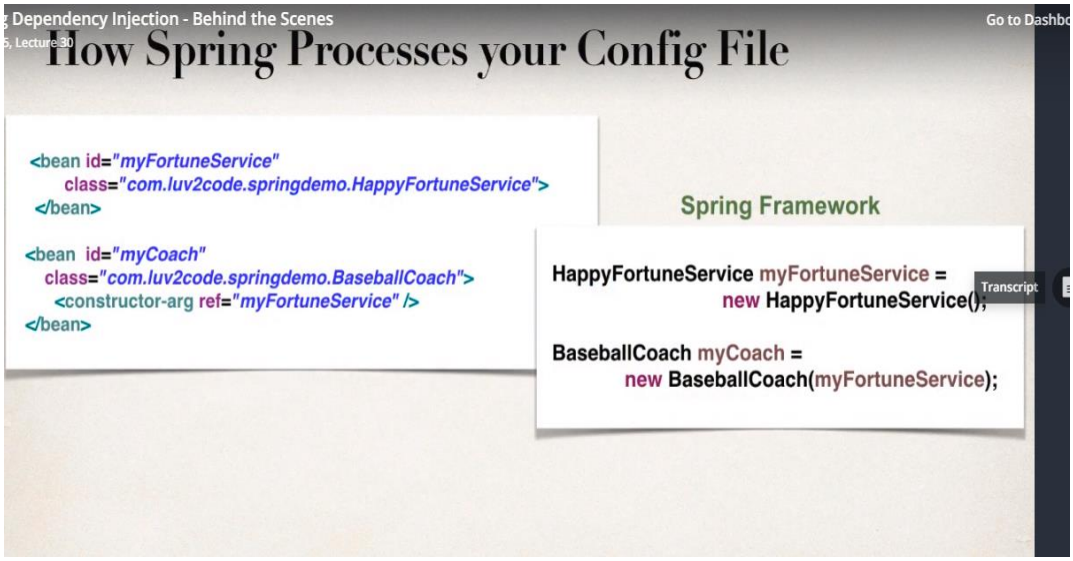
<u>Injection types:</u>

- Constructor Injection
  - Define dependency interface and class
  - Create Constructor in your class to inject dependency
  - Configure dependency injection in Spring config file--<constructor-arg ref="name"/>

-

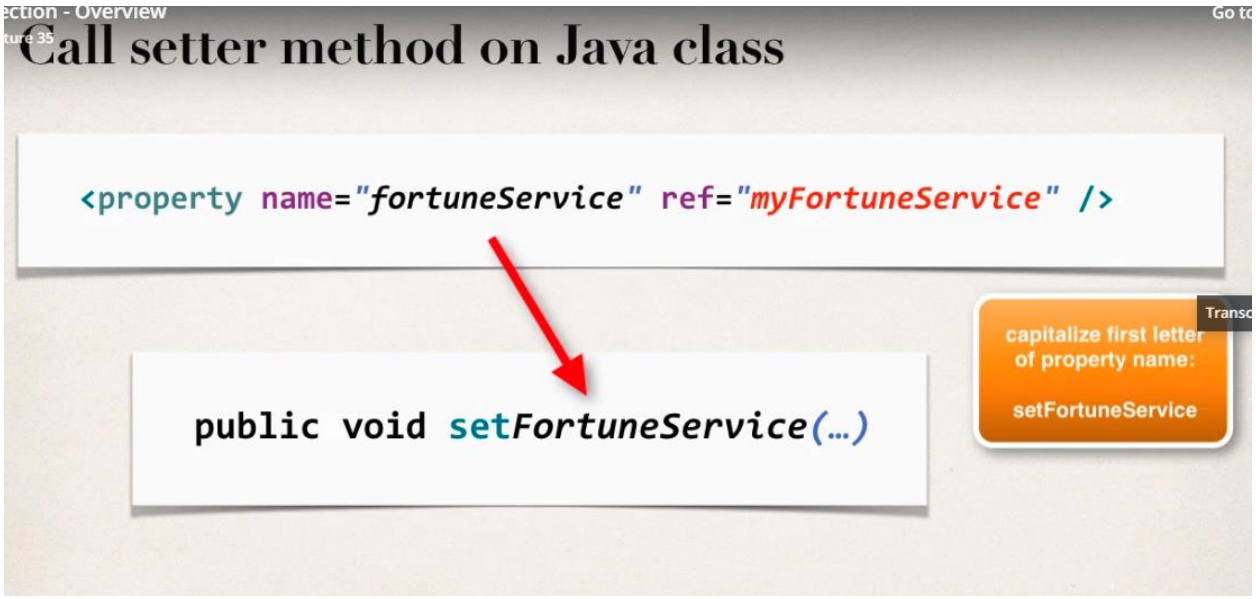


Behind the scene



- <u>Setter Injection</u>
  - Create setter methods in class.
  - Configure dependency injection in Spring config file.
    <property name=”fortuneService” ref=”myFortune”/>

    Property name should match with setter method name with all small letters
    Eg: set*FortuneService*



Injecting Literal values:

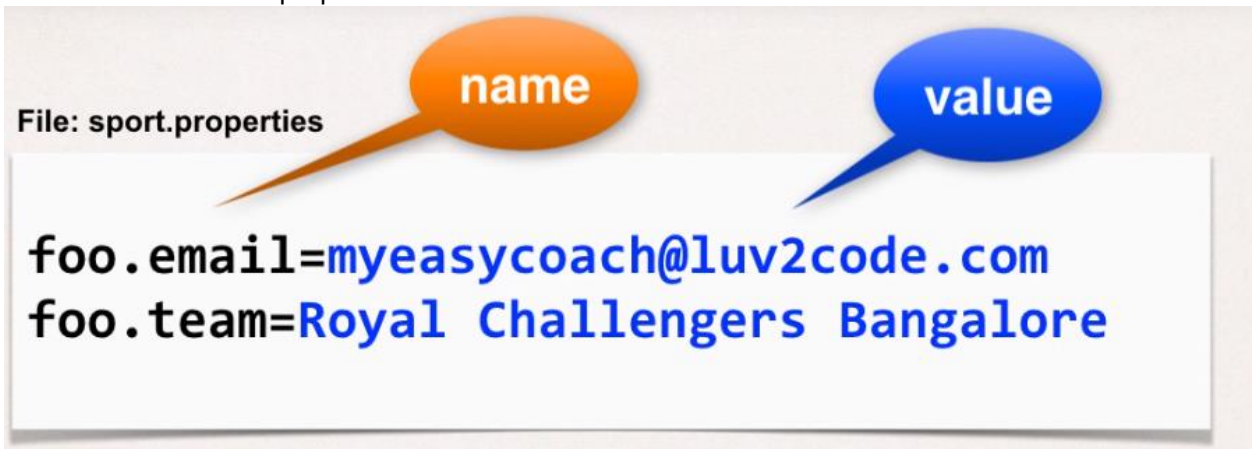
- Create Setter methods for Injection in Class
- Configure Injection is spring config file.

```
<property name="emailAddress" value="thebestcoach@luv2code.com" />

<property name="team" value="Sunrisers Hyderabad" />

</bean>
```

-

We can also set these values from properties file instead of Hardcoding the value.
In this case we need to use the below tag

- Create Properties file
- Load properties file in Spring config file
- Reference values from properties file



-

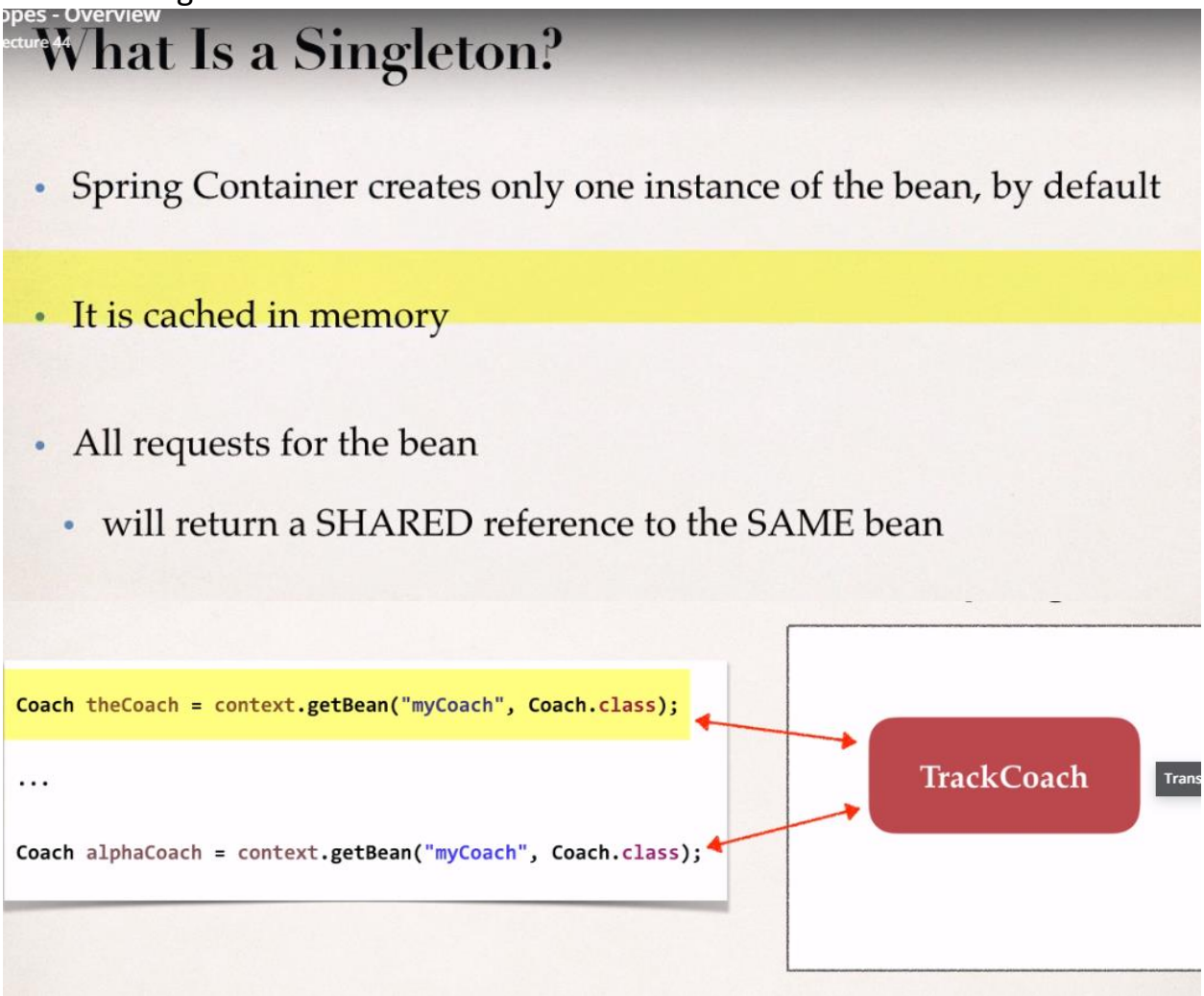
File: applicationContext.xml

```
<context:property-placeholder location="classpath:sport.properties"/>
```

Bean Scopes:

- Scope refers to lifecycle of the beans
- How long does the bean live
- How many instances are created.
- How bean is shared.

Note:- default scope for bean is singleton, when we don't specify any scope it assumes singleton.



We can explicitly specify the bean scope as below

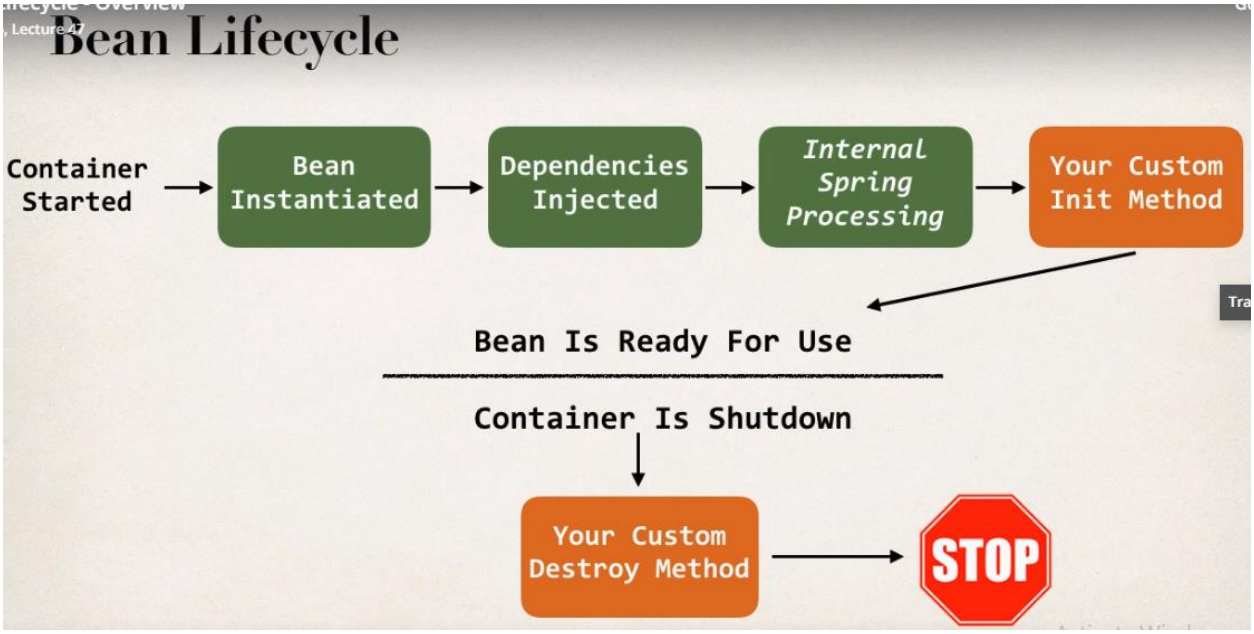
```
<beans … >

    <bean  id="myCoach"
        class="com.luv2code.springdemo.TrackCoach"
        scope="singleton">
      …
    </bean>

</beans>
```

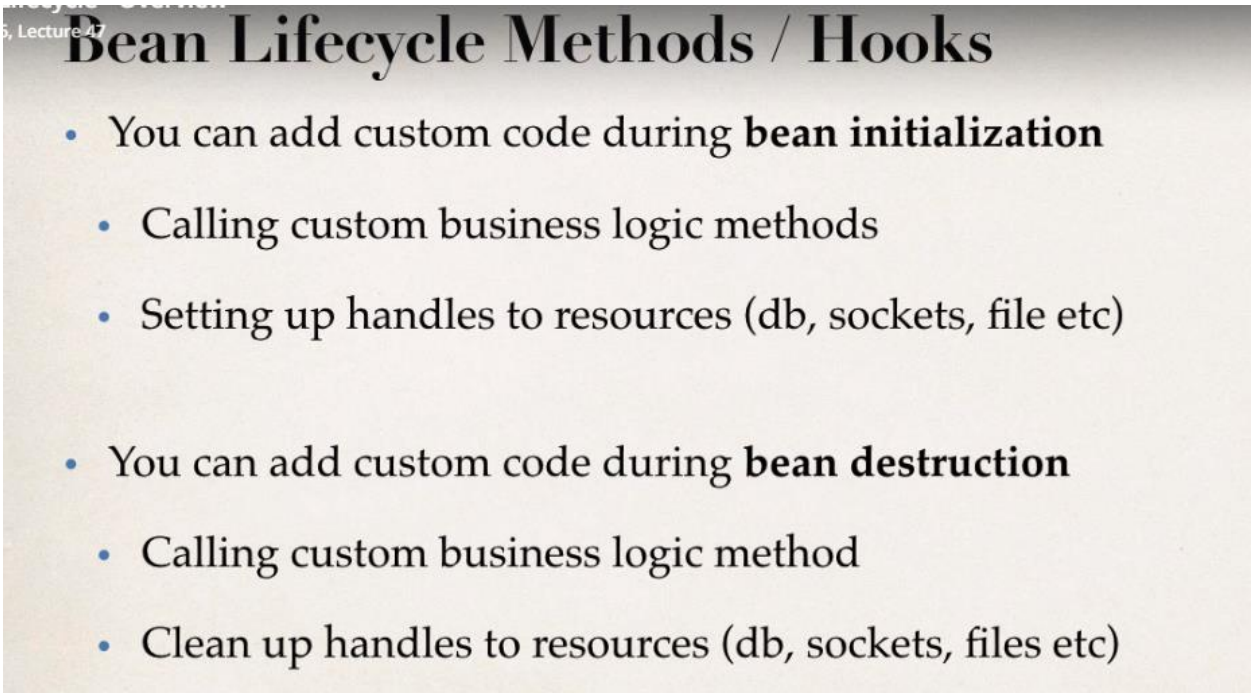
Set bean scope

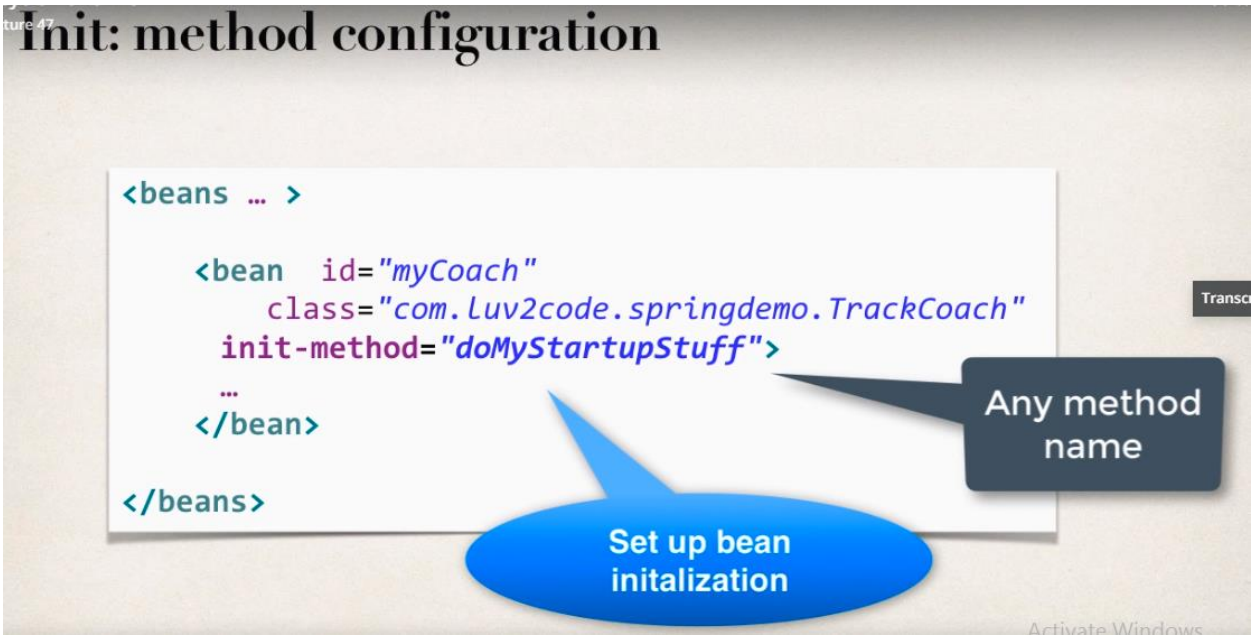
Types of Scope

Scope	Description
singleton	Create a single shared instance of the bean. Default scope.
prototype	Creates a new bean instance for each container request.
request	Scoped to an HTTP web request. Only used for web apps.
session	Scoped to an HTTP web session. Only used for web apps.
global-session	Scoped to a global HTTP web session. Only used for web apps.

Bean lifecycle Methods:







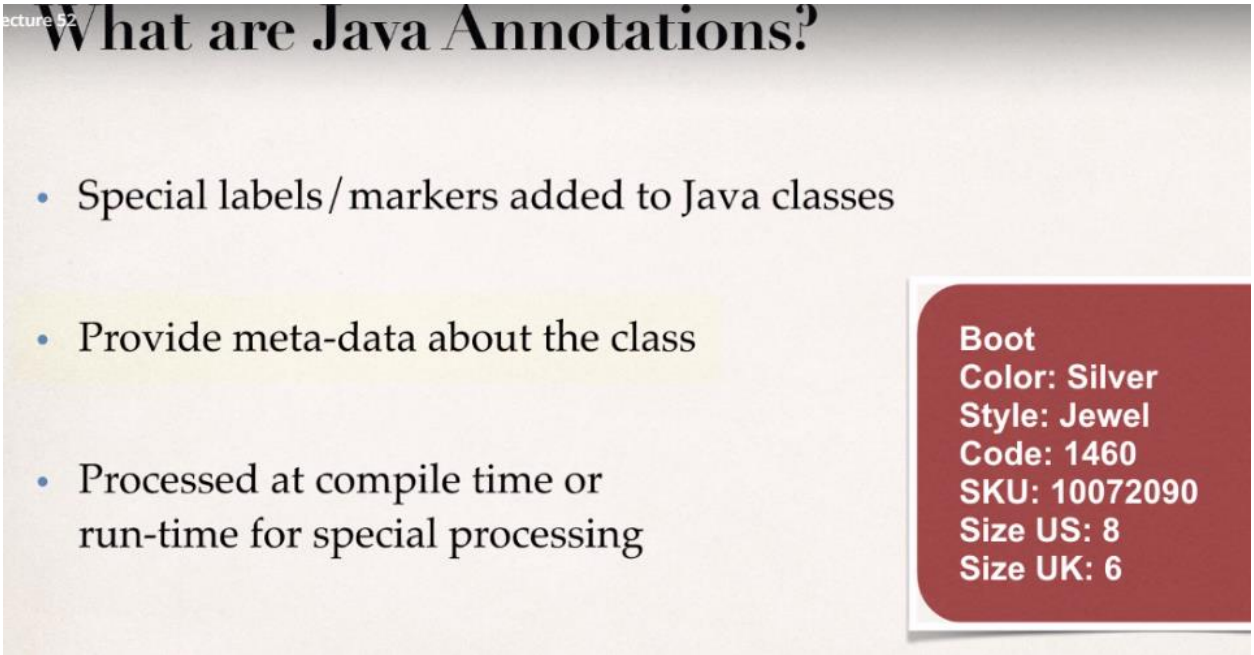
Development Process:

- Define Startup and destroy methods for class
- Configure those methods in Spring Config file
-

Note:- method cannot have any arguments, it should be no arg method.
Method can have any return type but cannot capture return data.
For prototype scope destroy methods are not called.

Annotations

Annotations minimizes the xml configuration.



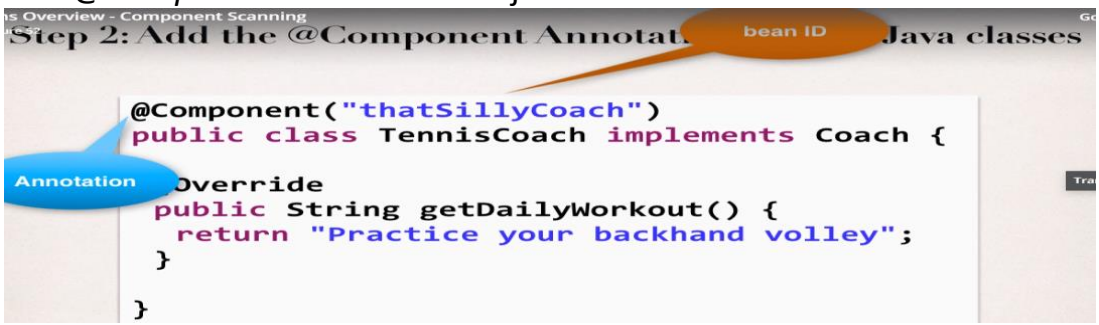
Spring will scan the java classes with special annotation and register them with Spring container

<u>Development process:</u>

- Enable component scanning in Spring config file

```
<beans … >

  <context:component-scan base-package="com.luv2code.springdemo" />

</beans>
```

- Add *@Component* Annotation to java class



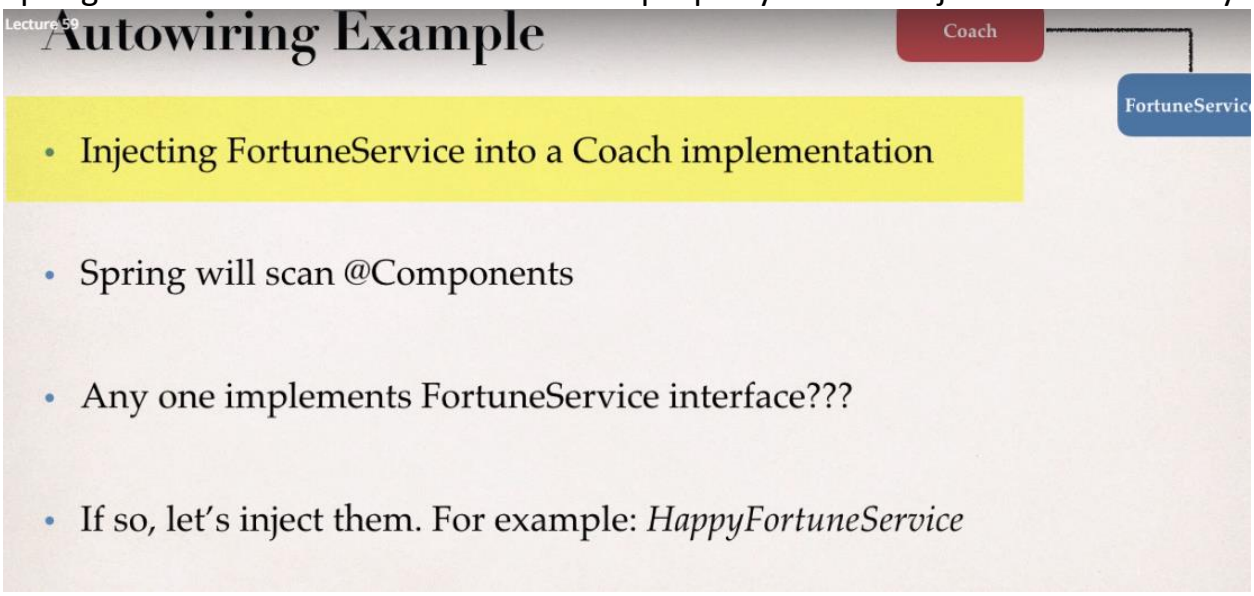
- Note:- providing bean name is not necessary, we can use default class name with first letter as non caps.

Eg: getBean("tennisCoach",Coach.class);

- Retrieve bean from Spring Container: same as previous context.getBean()

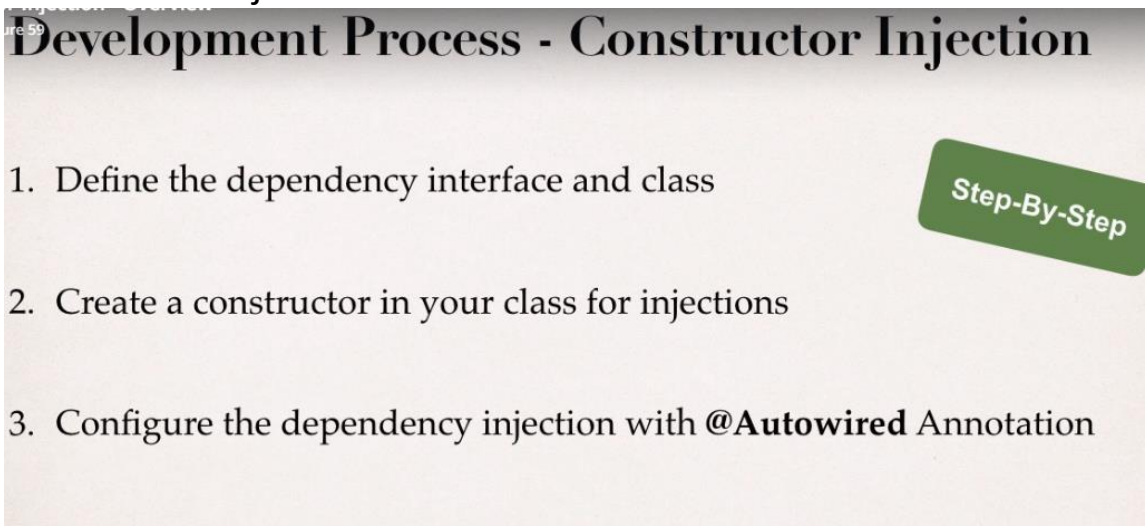
Autowiring:

Spring will look for class that matches the property and will inject it automatically.



Autowiring Injection types:

- Constructor Injection

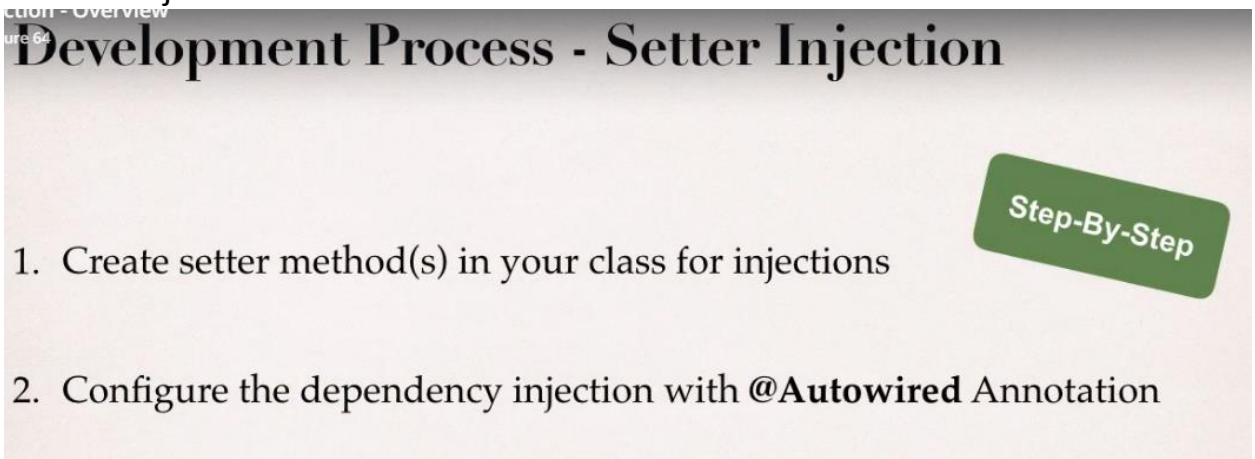


Question
I have finished the video "Constructor Injection - Writing Code part2".
I have commented the Autowired annotation. But still it worked. How did it work?

```
   //@Autowired
   public TennisCoach(FortuneService theFortuneService) {
      System.out.println(" theFortuneService " + theFortuneService);
      fortuneService = theFortuneService;
   }
```

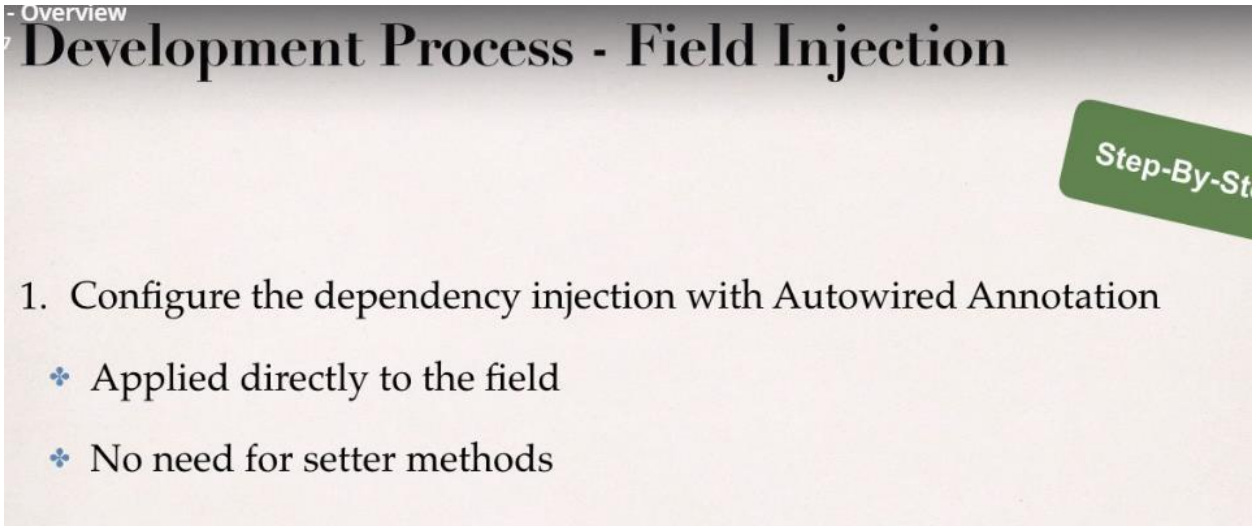
===
Answer
This is a new feature of Spring 4.3.
Here is the snippet from the Spring Docs.

- Method Injection



Note:- Method name can be any name, not necessary setter.

- Field Injection



Question
What if there are multiple implementations of FortuneService Interface?

If we use above approach then we will get Error message *"NoUniqueBeanDefinitionException".*

<u>Solution:</u>
With @Autowired we should use another annotation @Qualifier(<beanname>)

Eg:@Qualifier("happyFortuneService")

Note:- when the class name is having 2nd letter CAPS it should be used as it is.

When using with Constructor it should be as below

```
@Autowired
   public TennisCoach(@Qualifier("randomFortuneService")
FortuneService theFortuneService)
```

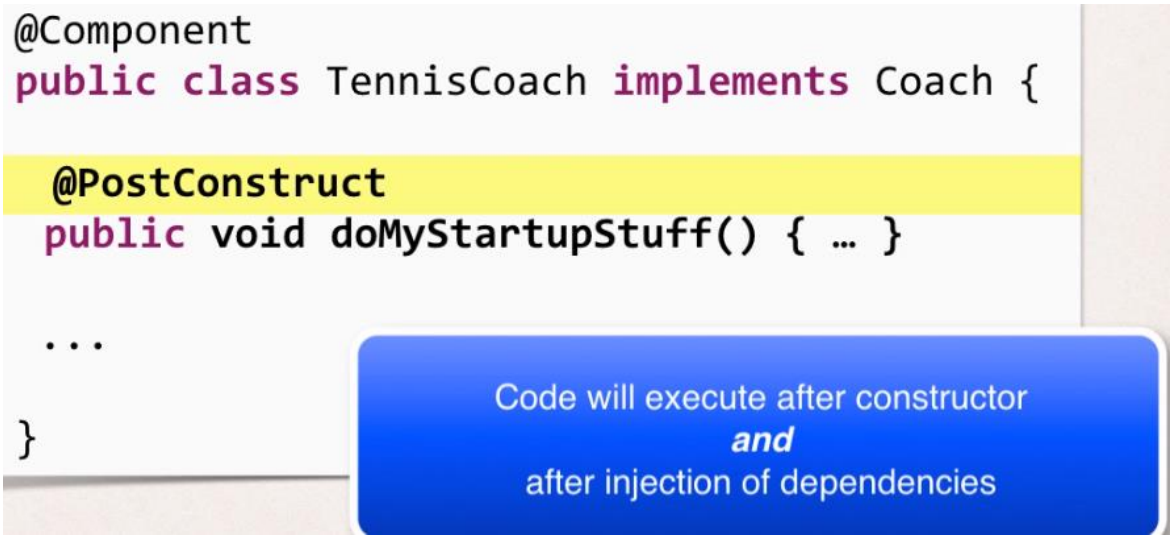
To Inject values from propery file use @Value annotation

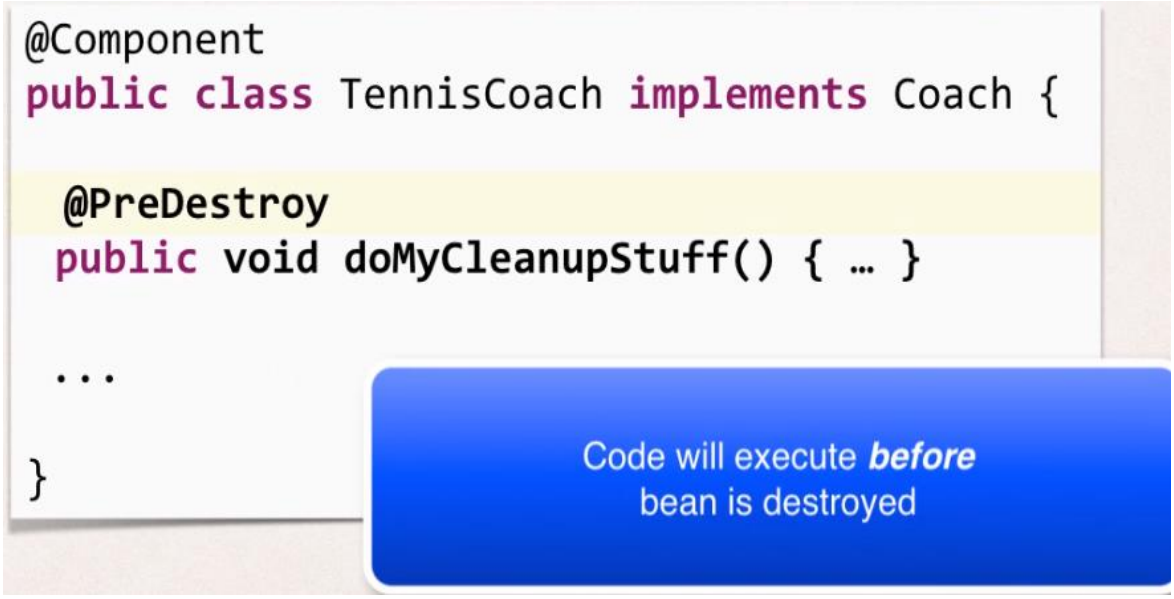
Scope:

@Scope("singleton") or

BeanLifeCycleMehods:

Just like Init and destroy we have below





Special Note about @PostConstruct and @PreDestroy Method Signatures

Section 9, Lecture 80

Special Note about @PostConstruct and @PreDestroy Method Signatures

I want to provide additional details regarding the method signatures of @PostContruct and @PreDestroy methods.

Access modifier

The method can have any access modifier (public, protected, private)

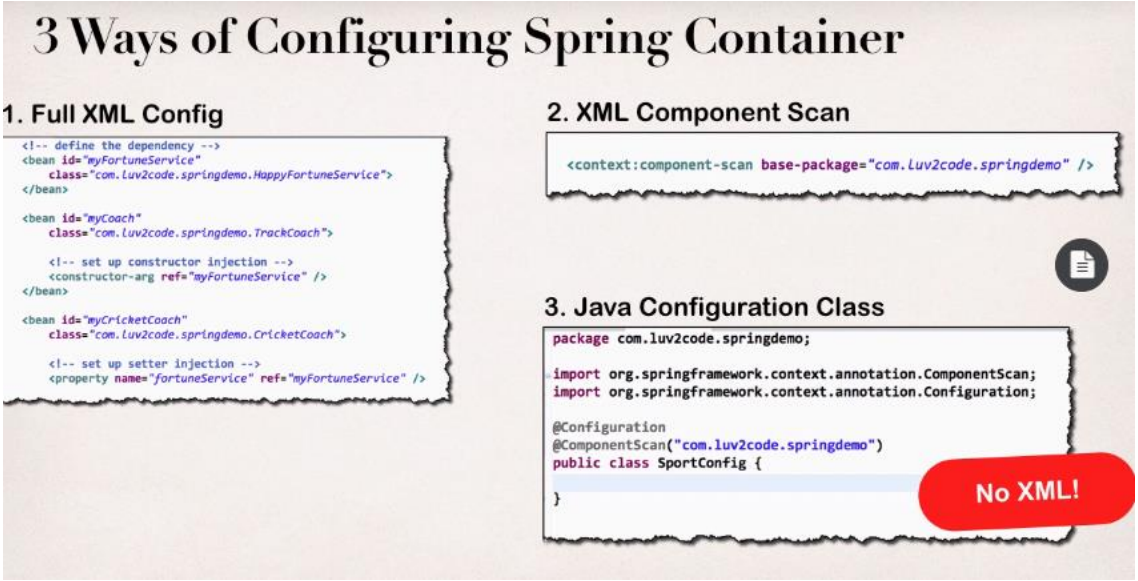
Return type
The method can have any return type. However, "void' is most commonly used. If you give a return type just note that you will not be able to capture the return value. As a result, "void" is commonly used.

Method name
The method can have any method name.

Arguments
The method can not accept any arguments. The method should be no-arg.

For "prototype" scoped beans, Spring does not call the @PreDestroy method.

Configuring Spring Container using JavaCode



Development Process

Step-By

1. Create a Java class and annotate as **@Configuration**
2. Add component scanning support: **@ComponentScan** (optional)
3. Read Spring Java configuration class
4. Retrieve bean from Spring container

1.
```
@Configuration
public class SportConfig {

}
```

2.
```
@Configuration
@ComponentScan("com.luv2code.springdemo")
public class SportConfig {

}
```

3.
```
AnnotationConfigApplicationContext context =
      new AnnotationConfigApplicationContext(SportConfig.class);
```

4. Retrieve beans

<u>Defining beans:</u>

Development Process

1. Define method to expose bean
2. Inject bean dependencies
3. Read Spring Java configuration class
4. Retrieve bean from Spring container

Step 1: Define method to expose bean

This method name will be the "bean id"

No component scan

```
onfig {

  @Bean
  public Coach swimCoach() {
    SwimCoach mySwimCoach = new SwimCoach();

    return mySwimCoach;
  }

}
```

Step 2: Inject bean dependencies

```
@Configuration
public class SportConfig {

  @Bean
  public FortuneService happyFortuneService() {
    return new HappyFortuneService();
  }

  @Bean
  public Coach swimCoach(FortuneService fortuneService) {
    SwimCoach mySwimCoach = new SwimCoach( happyFortuneService() );

    return mySwimCoach;
  }

}
```

Method name is bean ID

To read from properties file- @PropertySource(classpath:sport.properties) and @Value annotation, This @Value should be used in SwimCoach.java

Before Spring 4.3 It had used below for property

```
@Bean
   public static PropertySourcesPlaceholderConfigurer
                 propertySourcesPlaceHolderConfigurer() {

      return new PropertySourcesPlaceholderConfigurer();
   }
```