# Spring Fundamentals Chad Darby Course

1. When we get the bean from application context, we actually get the bean which is fully assembled. Spring factory creates all the beans and their dependencies and inject those dependencies in the bean. This way beans are fully ready for use and we use them.

We just looked at the constructor injection.

Below is the sample configuration of applicationContext.xml file for constructor injection.

<!-- Define the dependency -->

<bean id="myFortune"

class="com.luv2code.springdemo.HappyFortuneService">

</bean>

<!-- Define your beans here -->

<bean id="myCoach" class="com.luv2code.springdemo.BaseballCoach">

<!-- Set up constructor injection -->

<constructor-arg ref="myFortune"></constructor-arg>

</bean>

2. Setter Injection - In setter injection Spring will inject dependencies by calling setter methods of your class. So in general any property name you have in your class, Spring will attempt to call setter method of that property. Spring takes property name, makes it first letter capital to build method name and then calls that method. Below is one sample configuration done for setter injection,

<!-- Define the dependency -->

<bean id="myFortuneService"

class="com.luv2code.springdemo.HappyFortuneService">

</bean>

<!-- Define your cricket coach here -->

<bean id="myCricketCoach" class="com.luv2code.springdemo.CricketCoach">

<!-- Set up setter injection here -->

<property name="fortuneService" ref="myFortuneService"></property>

</bean>

3. What is the process of injecting literal values ?   
Ans - Look at the below configuration in the applicationContext.xml file for injecting String literal values.  
<!-- Define your cricket coach here -->

<bean id="myCricketCoach" class="com.luv2code.springdemo.CricketCoach">

<!-- Set up setter injection here -->

<property name="fortuneService" ref="myFortuneService"></property>

<!-- Setting string literal value demo here -->

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

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

</bean>

Apart from the above configuration we also need to provide setter methods for above mentioned properties like "emailAddress" and "team".

Q. How do you create application context of spring application and how do we retrieve bean from the application context once created ?  
Ans - Below are the sample code snippets,

ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

//Coach theCricketCoach = context.getBean("myCricketCoach", Coach.class);

CricketCoach theCricketCoach = context.getBean("myCricketCoach", CricketCoach.class);

it depends on how you retrieve the object and assign it ... that determines the visibility you have to the methods.

Q . How to fetch configuration values from property files ?  
Ans - Below is the configuration for fetching values from the property file.  
<!-- Load property file here -->

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

<!-- Define your cricket coach here -->

<bean id="myCricketCoach" class="com.luv2code.springdemo.CricketCoach">

<!-- Set up setter injection here -->

<property name="fortuneService" ref="myFortuneService"></property>

<!-- Setting string literal value demo here -->

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

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

<!-- Loading string literal values from the property file here -->

<property name="emailAddress1" value ="${foo.email}"></property>

<property name="team1" value ="${foo.team}"></property>

</bean>

Apart from above configuration we also created property file sport.properties . Below are the sample entries in the sport.properties file,  
  
foo.email=silly@luv2code.com

foo.team=Mighty Java Coders

Q. What is default bean scope ?  
Ans - Default bean scope is singleton. Look at the below sample code of testing 2 beans if they are equal i.e same bean,

Coach theCricketCoach = context.getBean("myCoach", Coach.class);  
Coach theCricketCoach2 = context.getBean("myCoach", Coach.class);  
//Below statement is the test of equality of 2 beans  
boolean result = (theCricketCoach == theCricketCoach2);  
System.out.println("Pointing to the same object ::: " + result);  
//Below statement will print address of two beans  
System.out.println(theCricketCoach);  
System.out.println(theCricketCoach2);

Note :- If we include scope="prototype" in the bean definition then each time we get a new bean from the spring container. Below is the sample definition of such bean,  
<!-- Define your beans here -->

<bean id="myCoach" class="com.luv2code.springdemo.BaseballCoach" scope="prototype">  
 <!-- Set up constructor injection -->  
 <constructor-arg ref="myFortuneService"></constructor-arg>  
 </bean>  
This is very important point to note. If we remove scope="prototype" from above bean definition then default bean scope will be singleton.  
Q. Write down high level steps in bean lifecycle in context of Spring?  
Ans - Below are the steps of high level of bean lifecycle .

* Bean instantiated
* Dependencies injected
* Internal spring processing
* Custom init method (Given by developer)
* Bean is ready for use

Note :- Important thing to note is that we can have some custom code added during bean initialization and also at time of bean destruction.

//Look at the configuration when you want to supply your initialzation and shutdown / cleanup method also

<!-- Define your beans here -->

<bean id=*"myCoach"* class=*"com.luv2code.springdemo.BaseballCoach"* init-method=*"doInit"* destroy-method=*"doShutdown"*>

<!-- Set up constructor injection -->

<constructor-arg ref=*"myFortuneService"*></constructor-arg>

</bean>

Note :-There is an important point to note that Spring does not call the destroy method for "prototype" scoped beans.

Q. What are Java annotations ?  
Ans - Java annotation are nothing but labels/markers added to Java class. They provide meta-data about the class. They are processed at compile time or run time for special processing.

Q. Why should we do Spring configuration with annotation ?  
Ans - Annotation minimizes the XML configuration. Spring will do lot of background work, do scanning of project and look of special annotation (Like @Component) and if they are found then Spring will automatically register them in the Spring container.So Spring is going to do lot of background work to make things simple for us. Below is the sample class definition which uses @Component annotation for the sanning of Spring beans in the source code.

@Component("thatSillyCoach")

**public** **class** TennisCoach **implements** Coach {

@Override

**public** String getDailyWorkout() {

// **TODO** Auto-generated method stub

**return** "Practice your backend volley";

}

}

Note :- In the above class, bean name is given as "thatSillyCoach". If you don't give bean name then by default bean name will be same as name of class, only its first letter will be small, i.e lowercase.

Q. What do you understand by Autowiring and what are different types of autowiring ?  
Ans - Autowiring is used for dependency injection. Below are the different types of autowiring,

* Constructor injection
* Setter injection
* Field injection

Q. When using Autowiring to inject dependencies, what will happen if there are multiple FortuneService implementations?  
Ans - Use @Qualifier annotation for such scenario.

IMPPP : - Constructor injection autowired is complete

Q. Show the important connfig and code snippet for constructor based autowiring.  
Ans - Below is the important config in the context file (applicationContext.xml file),  
<!-- Add entry to enable component scanning -->

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

Then other important configs in the Java file for annotation based Spring configration are,  
@Component

**public** **class** HappyFortuneService **implements** FortuneService {

.  
.  
}

//  
@Component("thatSillyCoach")

**public** **class** TennisCoach **implements** Coach {

**private** FortuneService fortuneService;

@Autowired

**public** TennisCoach(FortuneService fortuneService) {

**this**.fortuneService = fortuneService;

}  
.  
.  
}  
Q. What are setter injection ?  
Ans - In the setter injection "Autowired" annotation will be put above setter method. Remember that in the constructor injection "Autowired" annotation was applied just above constructor. Below is the sample example of setter injection (i.e code snippet of setter injection),  
  
@Component

**public** **class** CricketCoach **implements** Coach {

**private** FortuneService fortuneService;

@Override

**public** String getDailyWorkout() {

// **TODO** Auto-generated method stub

**return** "Do 5 Hours of Batting Practice";

}

@Override

**public** String getDailyFortune() {

// **TODO** Auto-generated method stub

**return** fortuneService.getFortune();

}

@Autowired

**public** **void** setFortuneService(FortuneService fortuneService) {

**this**.fortuneService = fortuneService;

}

}   
We can see that in the above example @Autowired keyword is placed above setter method which sets "fortuneService" property of the class. This is an example of setter injection.

Q. What is an "Field Injection"?  
Ans - In this method @Autowired annotation is used directly used above class member. Here @Autowired annotation is not used above constructor or setter. Below is the example of such class,  
@Component

**public** **class** FootballCoach **implements** Coach {

@Autowired

@Qualifier("happyFortuneService")

**private** FortuneService fortuneService;

@Override

**public** String getDailyWorkout() {

// **TODO** Auto-generated method stub

**return** "Do 5 Hrs of daily football practice";

}

@Override

**public** String getDailyFortune() {

// **TODO** Auto-generated method stub

**return** fortuneService.getFortune();

}

}  
  
Q. When there are more than one beans of same type what will happen ?

Ans - In that scenario we have to use Qualifier annotation or else Spring will raise error at time of building container. Example below,  
@Component

**public** **class** FootballCoach **implements** Coach {

@Autowired

@Qualifier("happyFortuneService")

**private** FortuneService fortuneService;

@Override

**public** String getDailyWorkout() {

// **TODO** Auto-generated method stub

**return** "Do 5 Hrs of daily football practice";

}

@Override

**public** String getDailyFortune() {

// **TODO** Auto-generated method stub

**return** fortuneService.getFortune();

}

}

Q. How do you enable component scannning and reading property value from a properties file?  
Ans - To do this we need to do below entry in the application context of Spring application file,  
<!-- Add entry to enable component scanning -->

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

<!-- Load property file here -->

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

Q. How will you read few properties like for example "foo.email" and "foo.team" from the property file ?  
Ans - Below are the steps which needs to be done in order to read properties from the file,

* Add fields in the class
* Generate getters for the added fields
* Add annotation to read value from the property file

@Value("${foo.email}")

**private** String email;

@Value("${foo.team}")

**private** String team;

Below is the full class,

@Component

**public** **class** FootballCoach **implements** Coach {

@Autowired

@Qualifier("randomFortuneService")

**private** FortuneService fortuneService;

@Value("${foo.email}")

**private** String email;

**public** FortuneService getFortuneService() {

**return** fortuneService;

}

@Value("${foo.team}")

**private** String team;

@Override

**public** String getDailyWorkout() {

// **TODO** Auto-generated method stub

**return** "Do 5 Hrs of daily football practice";

}

@Override

**public** String getDailyFortune() {

// **TODO** Auto-generated method stub

**return** fortuneService.getFortune();

}

**public** String getEmail() {

**return** email;

}

**public** String getTeam() {

**return** team;

}

}

@Component

**public** **class** FootballCoach **implements** Coach {

@Autowired

**private** FortuneService fortuneService;

@Override

**public** String getDailyWorkout() {

// **TODO** Auto-generated method stub

**return** "Do 5 Hrs of daily football practice";

}

@Override

**public** String getDailyFortune() {

// **TODO** Auto-generated method stub

**return** fortuneService.getFortune();

}

}

We are skipping bean scopes by annotation and Spring application just by Java annotation and no XML.

\*\*\* We have started small project of Spring MVC application, as covered in this Chard Darby course now.

**IMP Things to Note :**1. How do you access form data in the Spring controllers ?  
Ans - You have to access data from **HttpServletRequest** object. **HttpServletRequest** object is a parameter of method in the controller handling a request mapping. For example we have just read one of form data using below code,

// read the request parameter from the HTML form

String theName = request.getParameter("studentName");

2. How can you return data / send data to form or let say a JSP page ?  
Ans - We can return data using **Model** class. Model type is initially empty when received in the method.  
Below is the example,  
**String** result =   
**List**<Student> studentList =   
**ShoppingCart** theShoppingCart =

Above defined data's represents **Model** of MVC design pattern. Below are the sample code statements which needs to be done in the controller in order to return these data to form / JSP or any other view technology in order to display in the browser,

model.addAttribute("message",result);  
model.addAttribute("students",studentList);  
model.addAttribute("shoppingCart",theShoppingCart);

\*\*\*IMP Note\*\*\*  
1. There is a question on how to access CSS, JavaScript and images in Spring MVC Webapp. Taking note of this we will refer this later . There is also a github link from where source code can be downloaded for reference.  
\*\*\*Note - Now we are studying @RequestParam annotation. This will simplyfy accessing request parameter in the controller code. Again we can think that Spring is helping in reducing or removing boilerplate code . As you remember that earlier code written in controller to access request parameter was ,  
// read the request parameter from the HTML form

String theName = request.getParameter("studentName");

Q. What is @RequestParam annotation used for ?  
Ans - Below is the example and use of @RequestParam annotation,

@RequestMapping("/processFormVersionThree")

**public** String letsShoutDude2(@RequestParam("studentName") String theName, Model model) {

//Sample code to go here

.

.

.

}

Next Topic :- Adding Request Mappings on the Controller level

We can also add request mapping at the controller level. All other request mappings present on the method level will be relative to the controller level.

\*\*\*Note - We are looking at Spring MVC Form Tags :

Java Bean Validation API  
Q. What is a "Java Bean Validation API" ?  
Ans -

Q. How did you solve problem of JAR issue ?  
Ans - I searched for the type on the google. One of the link took me to the maven repository site. On that site i clicked on the jar(90KB) link and that particular jar got downloaded. Once JAR was donwloaded i copied that JAR in the lib folder of WebContent and that way problem was resolved.

Please note that "ride\_tracker-master\_spring\_jdbc" application is deployed on the Tomcat 8.5. I am removing this .

Note \*\*\* We are working on Spring MVC and Hibernate Database application  
  
We are building CRM (Customer Relationship Management) application. Below will be the steps,

1. Setup database dev environment
2. List Customers
3. Add a customer
4. Update a customer
5. Delete a customer