In the last chapter, we have studied about Dependency Injection.

Dependency Injection can be done using Setter Injection or Constructor Injection.

To do so, we were using a <property> and <constructor-arg> tag.

But Spring is more powerful than this. Spring can prevent us to even mention those <property> or <constructor-arg> tag by using the technique of **Autowiring.**

**Note : Autowiring is not possible for primitive type and String.**

Suppose we have class **Person** having dependency of **Heart** class object.

**Heart.java :**

public class Heart {

public void pump() {

System.out.println("The heart is pumping.");

}

}

**Person.java :**

public class Person {

private Heart heart;

public Person(Heart heart) {

super();

this.heart = heart;

}

public void setHeart(Heart heart) {

this.heart = heart;

}

public void startPumping() {

if(heart!=null) {

heart.pump();

}

else {

System.out.println("Heart is not pumping!");

}

}

}

**Person.java :**

public class Body {

public static void main(String[] args) {

String p = "com/xml/autowiring/config.xml";

ApplicationContext context = new ClassPathXmlApplicationContext(p);

Person person = context.getBean("person", Person.class);

person.startPumping();

}

}

**How were we injecting dependencies ?**

<bean id="heart" class="com.xml.autowiring.Heart" />

<bean id="person" class="com.xml.autowiring.Person" >

<property name="heart" ref="heart" />

<bean />

**OR**

<bean id="person" class="com.xml.autowiring.Person" >

<property name="heart">

<bean class="com.xml.autowiring.Heart" />

</property>

</bean>

**How spring will inject dependencies automatically using Autowiring ?**

Every <bean> tag has a attribute called **autowire** which has the following value :

* byName
* byType
* constructor

Using byName

<bean id="heart" class="com.xml.autowiring.Heart" />

<bean id="person" class="com.xml.autowiring.Person" autowire="byName" />

If we use the above xml code into <beans> tag.

What spring does is to first find out all the dependencies present in the **Person** class i.e.

private Heart heart;

above dependency has name **heart.**

Now the next task of spring is to find out if there is bean object present in the IOC container with the **id** same as the name of dependency.

As we have mentioned a <bean> tag for **Heart** class with id **heart**. So spring will find one. Now using this object spring will call the appropriate Setter.

Using byType

<bean id="heart" class="com.xml.autowiring.Heart" />

<bean id="person" class="com.xml.autowiring.Person" autowire="byType" />

Now we have changed the value to the attribute **autowire** as **“byType”.** But code will still work without any error. It is because, now spring will check for the Type of dependency not for the name :

private Heart heart;

above dependency is a type of **Heart.**

So spring will then look up into IOC container for if there is a Bean object present of the class **Heart.** And as we have mentioned a <bean> tag for class **Heart (** class="com.xml.autowiring.Heart" ) that’s why it is running.

Using constructor

<bean id="heart" class="com.xml.autowiring.Heart" />

<bean id="person" class="com.xml.autowiring.Person" autowire="constructor" />

While using the **“constructor”** value it is not required the dependencies Setter should be present.

What require is the presence of constructor in the class which is setting the all dependencies.

public Person(Heart heart) {

super();

this.heart = heart;

}

Now the same thing going to happen spring first check what all dependencies are there in the class **Person**. Then check for their object in the IOC container if available then spring will create **Person** object by passing the dependencies object through constructor call.

**Autowiring using @Autowired Annotation**

It is our first annotation we are going to use. So it is important to know what annotation is ?

So annotations basically are used to provide additional information to program. And they can be use as an alternative to XML.

How we going to use **@Autowired** to achieve autowiring using annotation.

Before deep dive first look into the below XML :

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

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

<context:annotation-config />

</beans>

By default annotation is disabled in spring. So to enabled them we need to add a special tag into our <beans> tag i.e.

<context:annotation-config />

Because of this tag comes from **context schema** so we have to add **context** **xml name space (xmlns)** this is what we have done in the above xml. So we going to use this XML when will we use any Annotation.

Now we are all set to Rock!

**Heart.java :**

public class Heart {

public void pump() {

System.out.println("The heart is pumping.");

}

}

The person class is the one in which we are going to inject dependencies. So while using annotation, it is good programming practice to always provide default constructor if you are using any parameterized constructor in it.

**Person.java :**

import org.springframework.beans.factory.annotation.Autowired;

public class Person {

private Heart heart;

public Person() { }

@Autowired

public void setHeart(Heart heart) {

this.heart = heart;

System.out.println("Setter called");

}

public void startPumping() {

if(heart!=null) {

heart.pump();

}

else {

System.out.println("Heart is not pumping!");

}

}

}

**Body.java :**

public class Body {

public static void main(String[] args) {

String p = "com/xml/autowiring/config.xml";

ApplicationContext context = new ClassPathXmlApplicationContext(p);

Person person = context.getBean("person", Person.class);

person.startPumping();

}

}

**config.xml :**

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

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

<context:annotation-config />

<bean id="heart" class="com.autowiring.annotation.Heart" />

<bean id="person" class="com.autowiring.annotation.Person" />

</beans>

While working with **@Autowired** annotation, it also provides two way for dependency injection.

We can annotate our Setters as we did in previous but we can also annotate our constructor like below :

**Person.java :**

import org.springframework.beans.factory.annotation.Autowired;

public class Person {

private Heart heart;

public Person() {

}

@Autowired

public Person(Heart heart) {

super();

this.heart = heart;

System.out.println("Construtor called");

}

public void startPumping() {

if(heart!=null) {

heart.pump();

}

else {

System.out.println("Heart is not pumping!");

}

}

}

**An Important Point :**

What about “byName”, “byType” and “constructor” which we have covered in “Autowiring using XML” ? Is there any similar concept in here ?

Answer :

Yes, spring use “byName” and “byType” while injecting dependency in both Setter and Constructor injection but in a different way.

Let’s understand it in case of Setter Injection,

@Autowired

public void setHeart(Heart heart) {

this.heart = heart;

System.out.println("Setter called");

}

When spring look the above code, it will try to inject dependency using “byType” initially but suppose there is two <bean> tags available for a Same class **Heart**. So in this case spring will unable to decide which object needs to be inject. So spring will turn itself towards “byName” injection.

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

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

<context:annotation-config />

<bean id="heart" class="com.autowiring.annotation.Heart" />

<bean id="brokenHeart" class="com.autowiring.annotation.Heart" />

<bean id="person" class="com.autowiring.annotation.Person" />

</beans>

As we can see, now we have two <bean> of **Heart**. So in this case spring going to use “byName” strategy.

So now, spring will match the dependency name with id “heart” and “brokenHeart”. Whichever, object id match with dependency name will be injected into dependency.

**Point to be noted :**

It is not important the **bean id** should match with **dependency name** if there is only one **bean** available.

@Qualifier Annotation

In **Person** class, we have dependency of **Heart** class.

private Heart heart;

And while injecting dependency spring can resolve the dependency between multiple bean using “byType” then “byName”. But what if in the below situation :

<context:annotation-config />

<bean id="stoneHeart" class="com.autowiring.annotation.Heart" />

<bean id="brokenHeart" class="com.autowiring.annotation.Heart" />

<bean id="person" class="com.autowiring.annotation.Person" />

In this situation, spring can’t resolve as both bean belong to **Heart** class and both of them have different **id** than **name** of dependency.

Now, **@Qualifier** annotation comes into the picture. So using this annotation we can explicitly tell to spring that we want to inject this particular bean.

Syntax : **@Qualifier(“**bean\_id”**)**

**config.xml :**

<context:annotation-config />

<bean id="stoneHeart" class="com.autowiring.annotation.Heart" />

<bean id="brokenHeart" class="com.autowiring.annotation.Heart" />

<bean id="person" class="com.autowiring.annotation.Person" />

**Person.java :**

import org.springframework.beans.factory.annotation.Autowired;

public class Person {

private Heart heart;

public Person() { }

@Autowired

@Qualifier("brokenHeart")

public void setHeart(Heart heart) {

this.heart = heart;

System.out.println("Setter called");

}

public void startPumping() {

if(heart!=null) {

heart.pump();

}

else {

System.out.println("Heart is not pumping!");

}

}

}

**Heart.java :**

public class Heart {

public void pump() {

System.out.println("The heart is pumping.");

}

}

**Body.java :**

public class Body {

public static void main(String[] args) {

String p = "com/xml/autowiring/config.xml";

ApplicationContext context = new ClassPathXmlApplicationContext(p);

Person person = context.getBean("person", Person.class);

person.startPumping();

}

}

**Few Important Point :**

* We can use @Autowired annotation with dependency itself instead of Setter or Constructor. In this case, spring will directly inject the dependencies without even call constructor or setter.
* We cannot use @Autowired annotation with Primitives or String.