Session-26

Setter injection with primitive and object type

Setter Injection with primitive data types:

Folder Structure:

1. Create a new Java Project “**Spring_Dependency_Injection”** and create a package for our src files **“com.ep“**
2. Add the required libraries to the build path. **Java Build Path ->Libraries ->Add External JARs** and add the below jars.

   commons-logging-1.1.1.jar
   spring-beans-3.2.9.RELEASE.jar
   spring-core-3.2.9.RELEASE.jar
   spring-context-3.2.9.RELEASE.jar
   spring-expression-3.2.9.RELEASE.jar

3. Create the Java classes **Employee.java** and **ClientLogic.java** under com.ep folder.
4. Place our configuration file **SpringConfig.xml** in the **src** directory

Employee.java

It is a simple java class containing the getters and setters of the employee details such as **id**, **name** and **city**. Whose values will be set through the configuration file and **getEmployeeDetails()** method prints the employee details which is set through the **setter injection**

```
package com.ep;

public class Employee
{
   private int id;
   private String name;
   private String city;

   public int getId() {
       return id;
   }
   public void setId(int id) {
       this.id = id;
   }
   public String getName() {
       return name;
   }
   public void setName(String name) {
       this.name = name;
   }
   public String getCity() {
       return city;
   }
   public void setCity(String city) {
       this.city = city;
   }
   public void getEmployeeDetails()
   {
       System.out.println("**Employee Details**");
       System.out.println("ID : "+id);
       System.out.println("Name : "+name);
       System.out.println("City : "+city);
   }
}
```

SpringConfig.xml

The **SpringConfig.xml** has the bean definition

- We have set bean id as **“employee”** for our Employee class which will act as the reference for calling our Employee class.
- Using **<property>** tag we have set the values to the properties in the Employee class**(Setter Injection)**

```
 <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-3.0.xsd">

     <bean id="employee" class="com.ep.Employee">
         <property name="id" value="123"></property>
         <property name="name" value="Enterprise Programming"></property>
         <property name="city" value="Chennai"></property>
    </bean>
</beans>
```

<u>ClientLogic.java</u>

- In our ClientLogic class we will Read the Configuration file**(SpringConfig.xml)** and get all the bean definition through **BeanFactory**
- Get the Employee Class instance by calling the **getBean()** method over the bean factory.
- The String passed to **getBean()** method should be equivalent to the **id** defined in the **SpringConfig.xml**
- Call the **getEmployeeDetails()** method to display the values which we injected through the setter.

```
package com.ep;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class ClientLogic
{
   public static void main(String args[])
   {
      //Read the configuration file
      Resource resource = new ClassPathResource("SpringConfig.xml");
      //Read all the bean definition
      BeanFactory bf = new XmlBeanFactory(resource);
      //Get the Student instance
      Employee employee = (Employee)bf.getBean("employee");
      employee.getEmployeeDetails();
   }
}
```

<u>Output</u>

Once we run our **ClientLogic.java** we will get the below output

```
**Employee Details**
ID : 123
Name : Enterprise Programming
City : Chennai
```

Setter Injection with Dependent Object Example

Like Constructor Injection, we can inject the dependency of another bean using setters. In such case, we use **property** element. Here, our scenario is **Employee HAS-A Address**. The Address class object will be termed as the dependent object. Let's see the Address class first:

Address.java

This class contains four properties, setters and getters and toString() method.

```
package com.javatpoint;


public class Address {
private String addressLine1,city,state,country;


//getters and setters


public String toString(){
    return addressLine1+" "+city+" "+state+" "+country;
}
```

Employee.java

It contains three properties id, name and address(dependent object) , setters and getters with displayInfo() method.

```
package com.javatpoint;


public class Employee {
private int id;
private String name;
private Address address;


//setters and getters


void displayInfo(){
    System.out.println(id+" "+name);
    System.out.println(address);
}
}
```

applicationContext.xml

The **ref** attribute of **property** elements is used to define the reference of another bean.

```
<?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:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="address1" class="com.javatpoint.Address">
<property name="addressLine1" value="51,Lohianagar"></property>
<property name="city" value="Ghaziabad"></property>
<property name="state" value="UP"></property>
<property name="country" value="India"></property>
</bean>

<bean id="obj" class="com.javatpoint.Employee">
<property name="id" value="1"></property>
<property name="name" value="Sachin Yadav"></property>
<property name="address" ref="address1"></property>
</bean>

</beans>
```

Test.java

This class gets the bean from the applicationContext.xml file and calls the displayInfo() method.

```
package com.javatpoint;


import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class Test {
public static void main(String[] args) {
    Resource r=new ClassPathResource("applicationContext.xml");
    BeanFactory factory=new XmlBeanFactory(r);

    Employee e=(Employee)factory.getBean("obj");
    e.displayInfo();

}
}
```