**SpringHelloWorld Applcation**

In spring we use separate java classes and create the beans of those classes and then we can inject in our application means can use those classes in our application without modify and also can remove that class easily. So these are independent to each other.

Like we have java class

**package** com.doj.spring;

**public** **class** Message {

**public** **void** showMessage(){

System.*out*.println("Hello world welcome to DOJ classes!!!");

}

}

I want to use this class in spring. For this I will create a bean for this class and then I can use this class in spring like below:

Now we have to create Bean Configuration file which is an **XML** file which maintain all classes object as bean. It is also called as **Bean Factory Configuration**. Keep this file(spring.xml) on the **src** directory of the spring project.

The **spring.xml** is used to assign unique IDs to different beans and to control the creation of objects with different values without impacting any of the Spring source files. For example, using below file you can pass any value for "message" variable and so you can print different values of message without impacting **HelloWorld.java** and **SpringTestApp.java** files.

<?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 definitions here -->**

**<bean class=*"com.doj.spring.Message"* id=*"message"*/>**

</beans>

There are following two important points to note about the main program:

1. First step is to create application context where we used framework API **ClassPathXmlApplicationContext()**. This API loads beans configuration file and eventually based on the provided API, it takes care of creating and initializing all the objects ie. beans mentioned in the configuration file.
2. Second step is used to get required bean using **getBean()** method of the created **context**. This method uses **bean ID** to return a generic object which finally can be casted to actual object. Once you have object, you can use this object to call any class method.

**package** com.doj.spring;

**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** HelloWorld {

**public** **static** **void** main(String[] args) {

//Message message = new Message();

Resource resource = **new** ClassPathResource("spring.xml");

BeanFactory factory = **new** XmlBeanFactory(resource);

Message message = (Message) factory.getBean("message");

message.showMessage();

}

}

Or

1. package com.doj.spring;
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
6. public class HelloWorld
7. {
8. public static void main(String[] args)
9. {
10. //ClassPathXmlApplicationContext is load all beans in the application
11. ApplicationContext context = new ClassPathXmlApplicationContext("spring.xml");
13. //this step is used to get required bean using getBean() method of the created context
14. Message message = (Message) factory.getBean("message");
15. message.showMessage ();
16. }
17. }

After creation of the bean **configuration file(Spring.xml)**, now we put this file into **"src"** directory of the application(i.e. which is present in classpath). Then **configuration file(Spring.xml)** is loaded in the ApplicationContext as follows.

1. ApplicationContext context = new ClassPathXmlApplicationContext("spring.xml");

Now suppose we put **configuration file(Spring.xml)** into the different directory package other than **"src"** like **"com.dineshonjava.sdnext.springConfig"** then we have to give classpath of **configuration file(Spring.xml)** to ApplicationContext as follows:

1. ApplicationContext context = new ClassPathXmlApplicationContext("classpath:com/dineshonjava/sdnext/springConfig/spring.xml");

We can use wildcard for this as follows:

1. ApplicationContext context = new ClassPathXmlApplicationContext("classpath\*:com/dineshonjava/\*\*/springConfig/spring.xml");
2. ApplicationContext context = new ClassPathXmlApplicationContext("classpath\*:com/\*/\*\*/springConfig/\*-spring.xml");

When Spring application gets loaded into the memory, Framework makes use of the above configuration file to create all the beans defined and assign them a unique ID as defined in tag. You can use tag to pass the values of different variables used at the time of object creation.