SPRING BEANFACTORY CONTAINER

This is the simplest container providing basic support for DI and defined by the org.springframework.beans.factory.BeanFactory interface. The BeanFactory and related interfaces, such as BeanFactoryAware, InitializingBean, DisposableBean, are still present in Spring for the purposes of backward compatibility with the large number of third-party frameworks that integrate with Spring.

There are a number of implementations of the BeanFactory interface that come supplied straight out-of-the-box with Spring. The most commonly used BeanFactory implementation is the **XmlBeanFactory** class. This container reads the configuration metadata from an XML file and uses it to create a fully configured system or application.

The BeanFactory is usually preferred where the resources are limited like mobile devices or applet based applications. So use an ApplicationContext unless you have a good reason for not doing so.

Example:

Let us have working Eclipse IDE in place and follow the following steps to create a Spring application:

Step	Description
1	Create a project with a name *SpringExample* and create a package *com.tutorialspoint* under the **src** folder in the created project.
2	Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter.
3	Create Java classes *HelloWorld* and *MainApp* under the *com.tutorialspoint* package.
4	Create Beans configuration file *Beans.xml* under the **src** folder.
5	The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below.

Here is the content of **HelloWorld.java** file:

```
package com.tutorialspoint;

public class HelloWorld {
   private String message;

   public void setMessage(String message){
      this.message  = message;
   }

   public void getMessage(){
      System.out.println("Your Message : " + message);
   }
}
```

Following is the content of the second file **MainApp.java**:

```
package com.tutorialspoint;

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

public class MainApp {
```

```
   public static void main(String[] args) {
      XmlBeanFactory factory = new XmlBeanFactory
                               (new ClassPathResource("Beans.xml"));

      HelloWorld obj = (HelloWorld) factory.getBean("helloWorld");
      obj.getMessage();
   }
}
```

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

1. First step is to create factory object where we used framework API **XmlBeanFactory** to create the factory bean and **ClassPathResource** API to load the bean configuration file available in CLASSPATH. The **XmlBeanFactory** API 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 bean factory object. 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.

Following is the content of the bean configuration file **Beans.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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

   <bean  >
       <property name="message" value="Hello World!"/>
   </bean>

</beans>
```

Once you are done with creating source and bean configuration files, let us run the application. If everything is fine with your application, this will print the following message:

```
Your Message : Hello World!
```