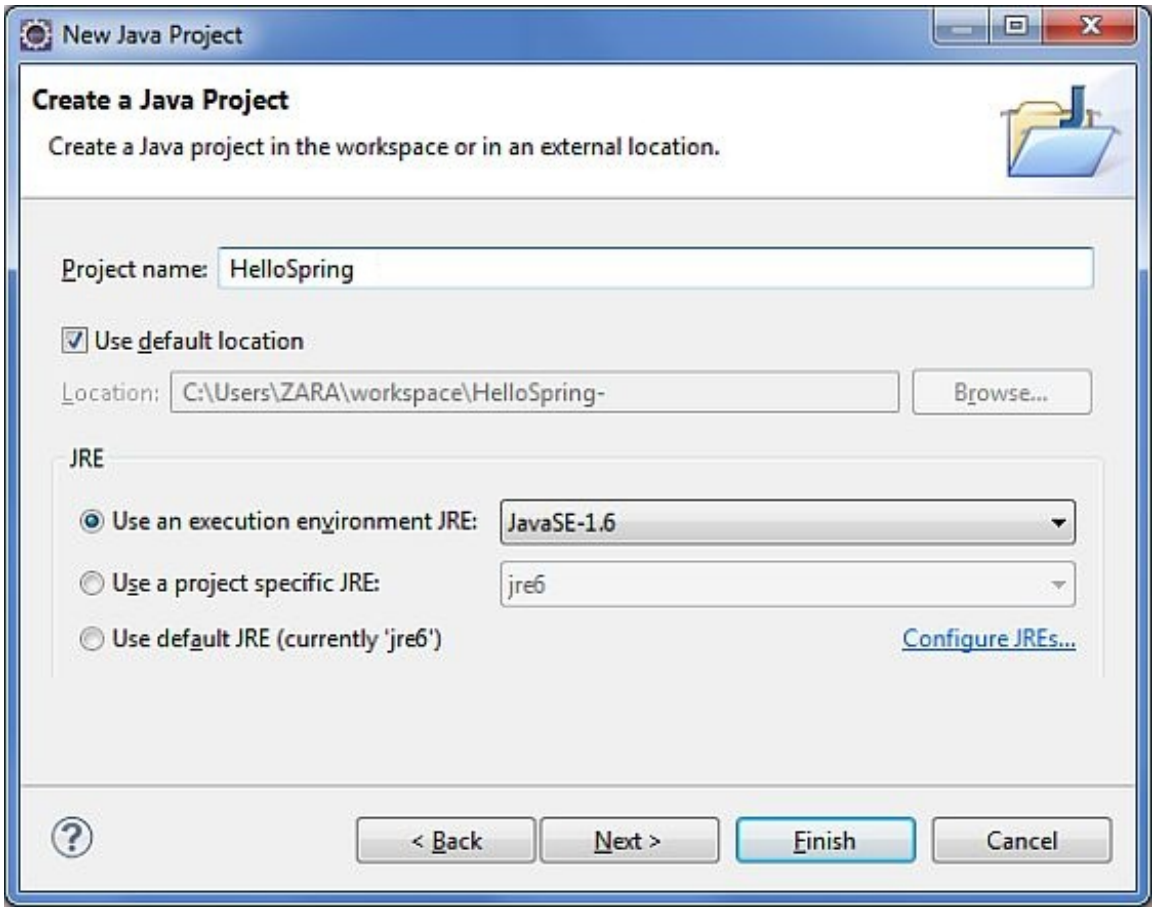
SPRING HELLO WORLD EXAMPLE

Let us start actual programming with Spring Framework. Before you start writing your first example using Spring framework, you have to make sure that you have setup your Spring environment properly as explained in Spring - Environment Setup tutorial. I also assume that you have a little bit working knowledge with Eclipse IDE.

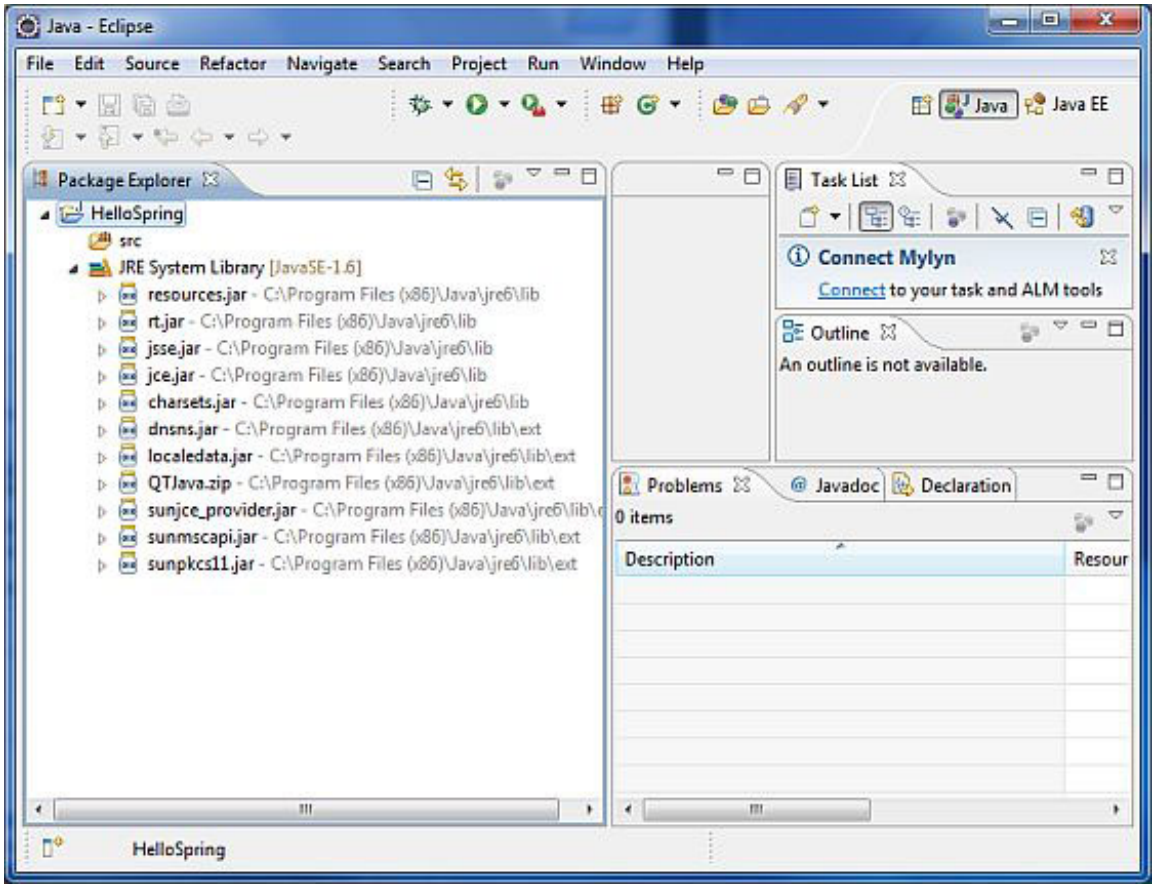
So let us proceed to write a simple Spring Application which will print "Hello World!" or any other message based on the configuration done in Spring Beans Configuration file.

Step 1 - Create Java Project:

The first step is to create a simple Java Project using Eclipse IDE. Follow the option **File -> New -> Project** and finally select **Java Project** wizard from the wizard list. Now name your project as **HelloSpring** using the wizard window as follows:

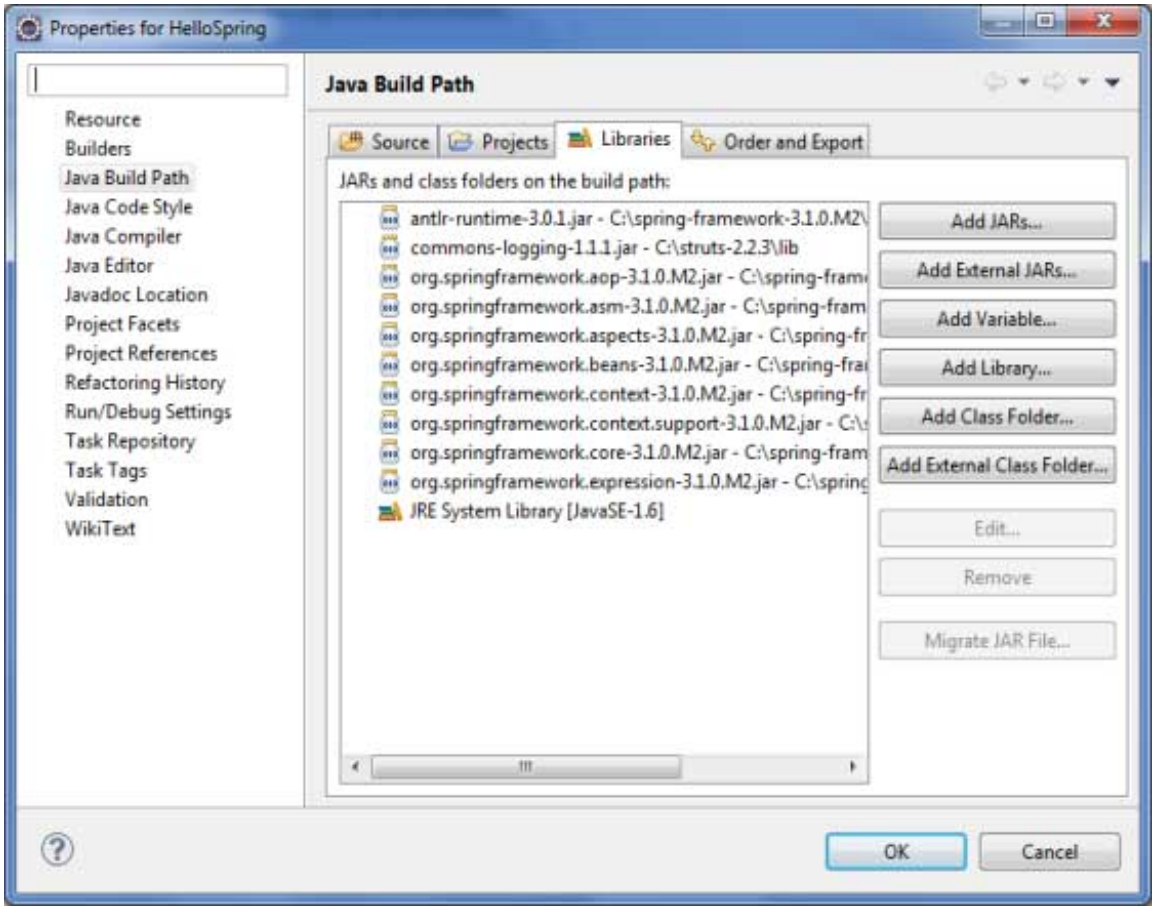


Once your project is created successfully, you will have following content in your **Project Explorer:**



Step 2 - Add Required Libraries:

As a second step let us add Spring Framework and common logging API libraries in our project. To do this, right click on your project name **HelloSpring** and then follow the following option available in context menu: **Build Path -> Configure Build Path** to display the Java Build Path window as follows:



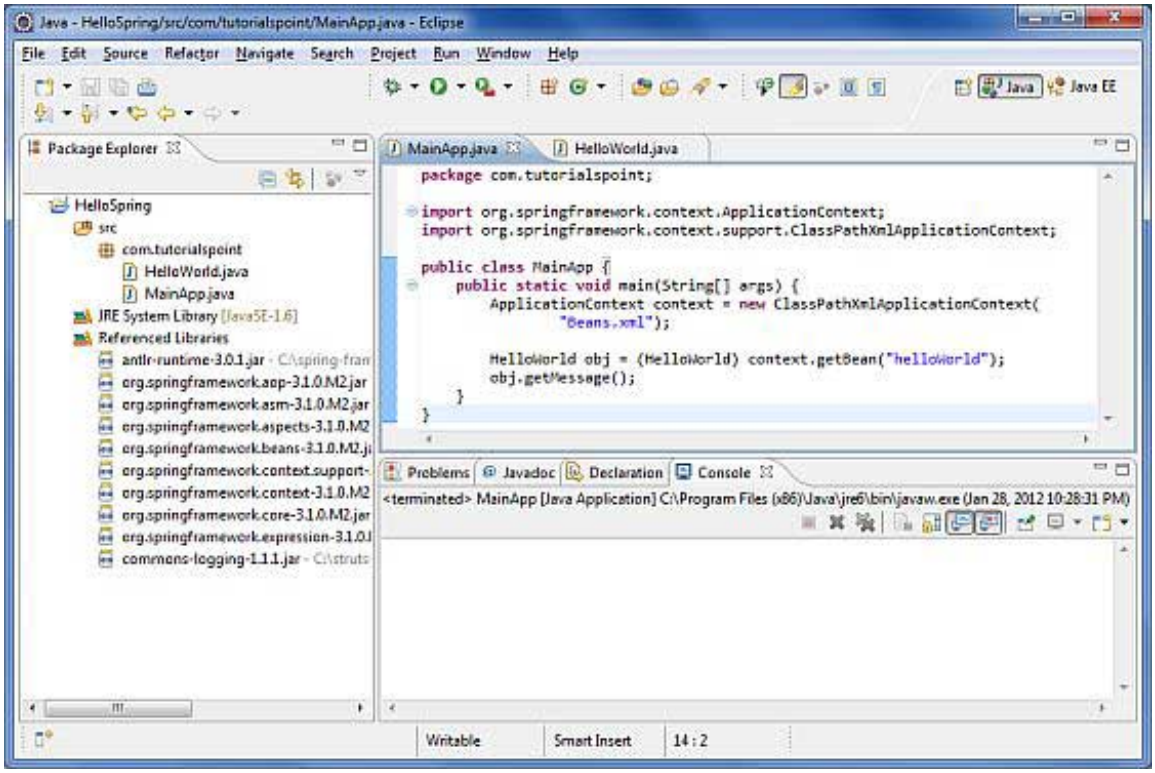
Now use **Add External JARs** button available under **Libraries** tab to add the following core JARs from Spring Framework and Common Logging installation directories:

- antlr-runtime-3.0.1
- org.springframework.aop-3.1.0.M2
- org.springframework.asm-3.1.0.M2
- org.springframework.aspects-3.1.0.M2
- org.springframework.beans-3.1.0.M2
- org.springframework.context.support-3.1.0.M2
- org.springframework.context-3.1.0.M2
- org.springframework.core-3.1.0.M2
- org.springframework.expression-3.1.0.M2
- commons-logging-1.1.1

Step 3 - Create Source Files:

Now let us create actual source files under the **HelloSpring** project. First we need to create a package called **com.tutorialspoint**. To do this, right click on **src** in package explorer section and follow the option: **New -> Package**.

Next we will create **HelloWorld.java** and **MainApp.java** files under the com.tutorialspoint package.



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.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context =
                new ClassPathXmlApplicationContext("Beans.xml");

        HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

        obj.getMessage();
    }
}
```

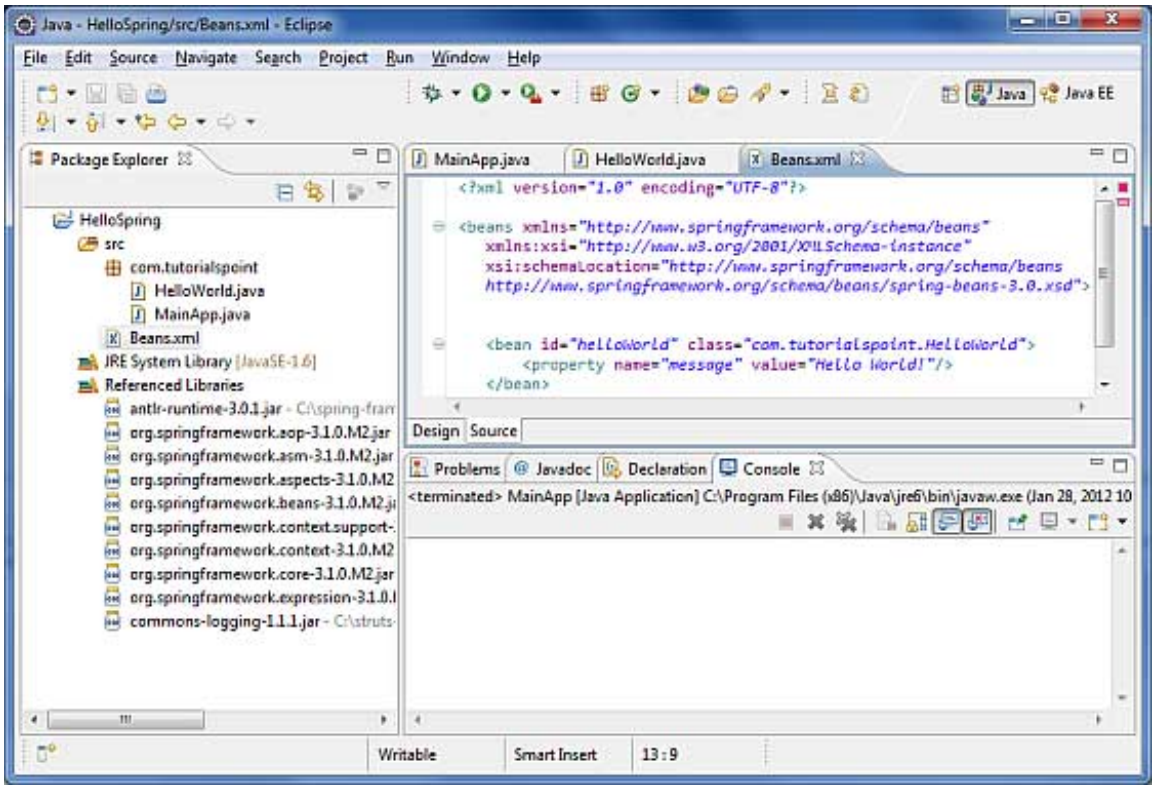
There are following two important points to note about the main program:

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.

Step 4 - Create Bean Configuration File:

You need to create a Bean Configuration file which is an XML file and acts as cement that glues the beans ie. classes together. This file needs to be created under the **src** directory as shown below:



Usually developers keep this file name as **Beans.xml**, but you are independent to choose any name you like. You have to make sure that this file is available in CLASSPATH and use the same name in main application while creating application context as shown in MainApp.java file.

The Beans.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 MainApp.java files. Let us see how it works:

```
<?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>
```

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 **<bean>** tag. You can use **<property>** tag to pass the values of different variables used at the time of object creation.

Step 5 - Running the Program:

Once you are done with creating source and beans configuration files, you are ready for this step which is compiling and running your program. To do this, Keep MainApp.Java file tab active and use either **Run** option available in the Eclipse IDE or use **Ctrl + F11** to compile and run your **MainApp** application. If everything is fine with your application, this will print the following message in Eclipse IDE's console:

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

Congratulations, you have created your first Spring Application successfully. You can see the flexibility of above Spring application by changing the value of "message" property and keeping both the source files unchanged. Further, let us start doing something more interesting in next few chapters.