

## TP5

# Spring Framework

---

### Environment setup

#### Step 1 - Setup Java Development Kit (JDK)

#### Step 2 - Install Apache Common Logging API

- You can download the latest version of Apache Commons Logging API from <https://commons.apache.org/logging/>
- Once you download the installation, unpack the binary distribution into a convenient location.
- For example, in D:\Software\commons-logging-1.2 on Windows, or /usr/local/commons-logging-1.2 on Linux/Unix.
- This directory will have the following jar files and other supporting documents, etc.

 apidocs	7/5/2014 20:11	File folder	
 commons-logging-1.2.jar	7/5/2014 20:11	Executable Jar File	61 KB
 commons-logging-1.2-javadoc.jar	7/5/2014 20:12	Executable Jar File	155 KB
 LICENSE.txt	7/5/2014 20:11	Text Document	12 KB
 NOTICE.txt	7/5/2014 20:11	Text Document	1 KB
 RELEASE-NOTES.txt	7/5/2014 20:11	Text Document	2 KB

#### Step 3 - Setup Eclipse IDE

#### Step 4 - Setup Spring Framework Libraries

Now if everything is fine, then you can proceed to set up your Spring framework. Following are the simple steps to download and install the framework on your machine.

- Make a choice whether you want to install Spring on Windows or Unix, and then proceed to the next step to download .zip file for Windows and .tar.gz file for Unix.
- Download the latest version of Spring framework binaries from <https://repo.spring.io/release/org/springframework/spring>.

## Index of release/org/springframework/spring

Name	Last Modified
<a href="#">../</a>	
<a href="#">1.0/</a>	06-03-22 04:45:51 +0100
<a href="#">1.0-m4/</a>	06-03-22 04:20:57 +0100
<a href="#">1.0-rc1/</a>	06-03-22 13:18:24 +0100
<a href="#">1.0.1/</a>	06-03-22 13:18:26 +0100
<a href="#">1.1/</a>	06-03-22 08:11:32 +0100
<a href="#">1.1-rc1/</a>	06-03-22 13:18:30 +0100
<a href="#">1.1-rc2/</a>	06-03-22 10:20:31 +0100
<a href="#">1.1.1/</a>	06-03-22 13:18:34 +0100
<a href="#">1.1.2/</a>	06-03-22 13:18:36 +0100
<a href="#">1.1.3/</a>	06-03-22 13:18:38 +0100
<a href="#">1.1.4/</a>	06-03-22 13:18:40 +0100
<a href="#">1.1.5/</a>	06-03-22 13:18:42 +0100
<a href="#">1.2/</a>	06-03-22 13:18:45 +0100
<a href="#">1.2-rc1/</a>	06-03-22 13:18:47 +0100
<a href="#">1.2-rc2/</a>	06-03-22 13:18:49 +0100
...	
<a href="#">5.3.9/</a>	14-07-21 09:18:11 +0200

- After the downloaded file was unzipped, it gives for example the following directory structure inside D:\Software\spring-framework-5.3.9

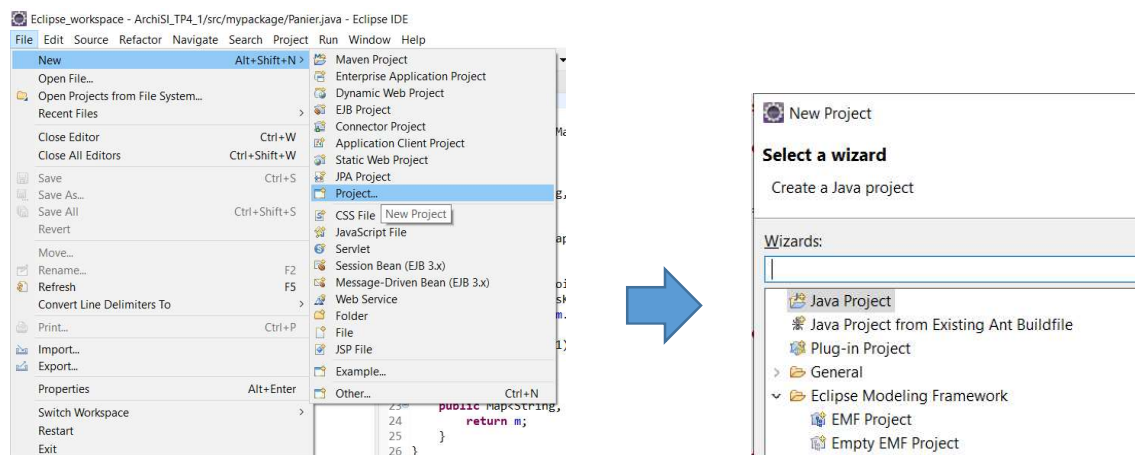
docs	7/14/2021 6:47	File folder	
libs	7/14/2021 6:48	File folder	
schema	7/14/2021 6:48	File folder	
license.txt	7/14/2021 6:36	Text Document	16 KB
notice.txt	7/14/2021 6:36	Text Document	1 KB
readme.txt	7/14/2021 6:36	Text Document	1 KB

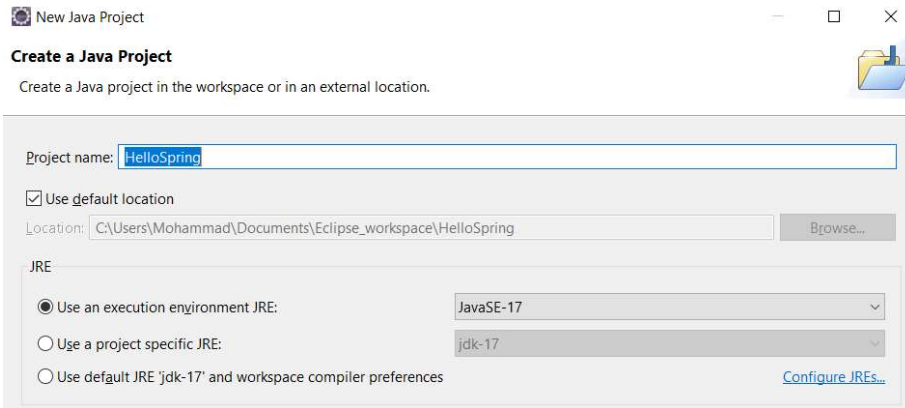
If you are using Eclipse, then it is not required to set CLASSPATH because all the setting will be done through Eclipse.

## Spring - Hello World Example

### 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 –

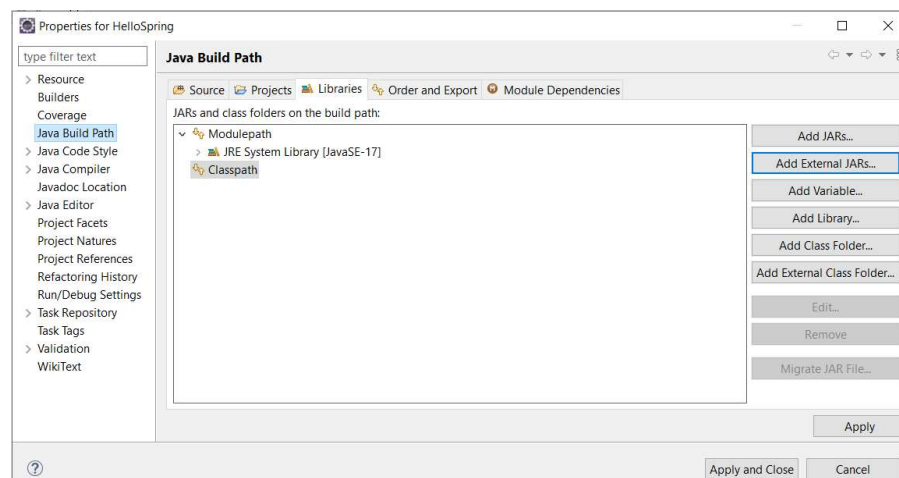
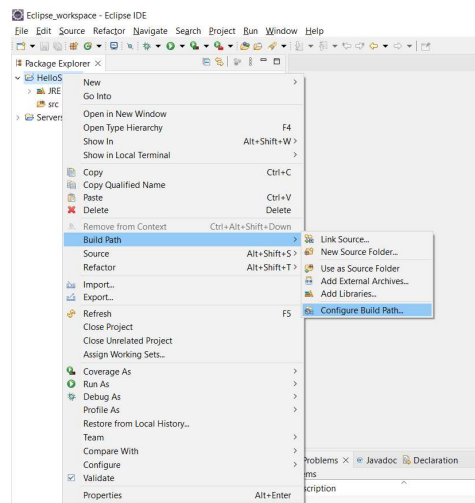




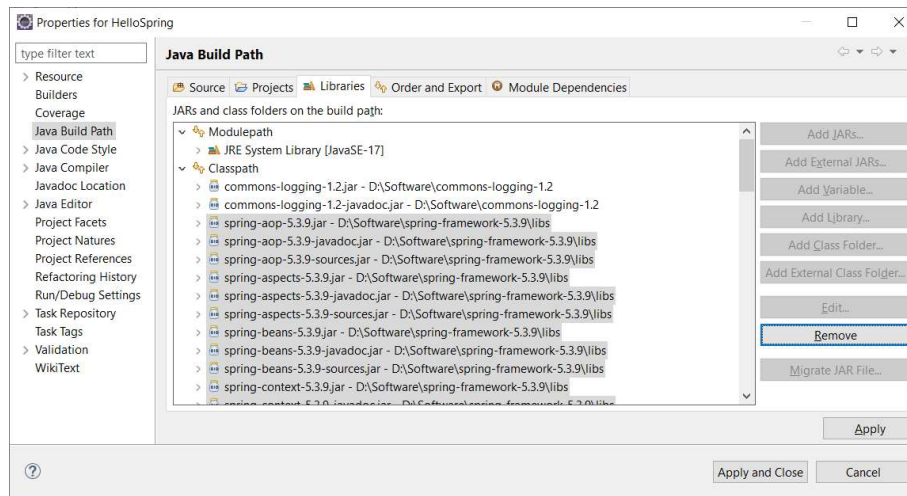
## Step 2 - Add Required Libraries

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 the context menu – **Build Path** → **Configure Build Path** to display the Java Build Path window as follows



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



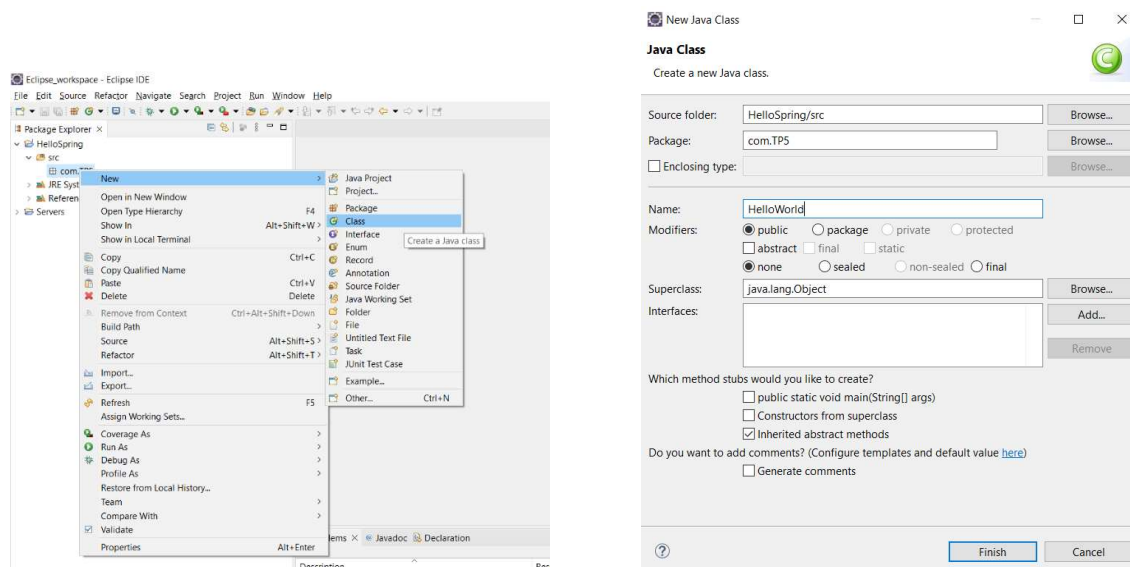
### Step 3 - Create Source Files

First we need to create a package called **com.TP5**. To do this, right click on **src** in package explorer section and follow the option – **New** → **Package**.

Next create **HelloWorld.java** and **MainApp.java** files under the **com.TP5** package as follows;

**HelloWorld.java:** Right click on **com.TP5** package – **New** → **Class** → Name: HelloWorld → Finish

**MainApp.java:** Right click on **com.TP5** package – **New** → **Class** → Name: MainApp → check public static void main (String[] args) → Finish



## HelloWorld.java

```

1 package com.TP5;
2
3 public class HelloWorld {
4     private String message;
5     public void setMessage(String message){
6         this.message = message;
7     }
8     public void getMessage(){
9         System.out.println("Your Message : " + message);
10    }
11 }
12

```

## MainApp.java

```

1 package com.TP5;
2
3 import org.springframework.context.ApplicationContext;
4 import org.springframework.context.support.ClassPathXmlApplicationContext;
5
6 public class MainApp {
7     public static void main(String[] args) {
8         ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");
9         HelloWorld obj = (HelloWorld) context.getBean("helloWorld");
10        obj.getMessage();
11    }
12 }

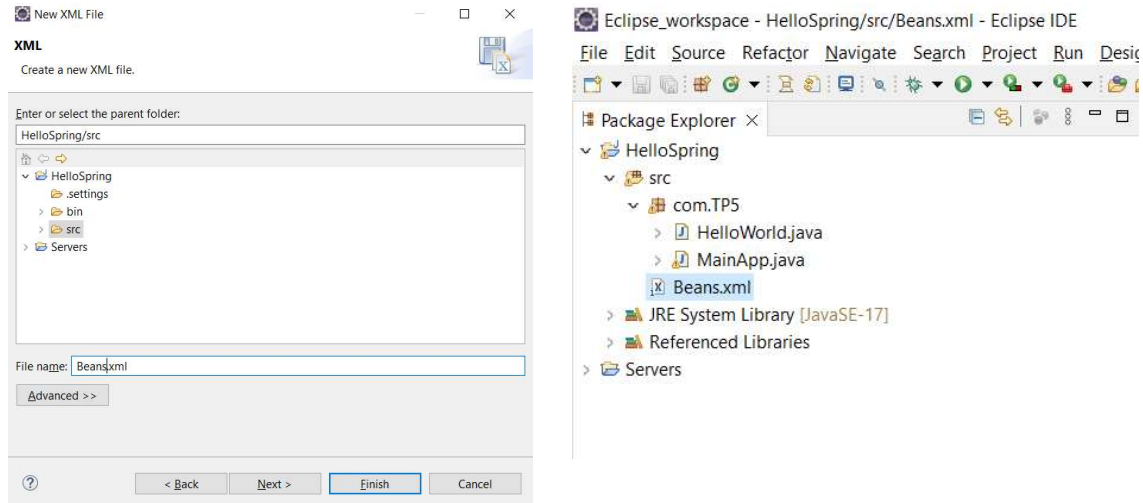
```

Following two important points are to be noted about the main program –

- I. The first step is to create an 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, i.e. beans mentioned in the configuration file.
- II. The second step is used to get the 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 the actual object. Once you have an 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 a cement that glues the beans, i.e. the classes together. This file needs to be created under the **src** directory as shown in the following (src → New → Other → XML 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.

<https://docs.spring.io/spring-framework/docs/4.2.x/spring-framework-reference/html/xsd-configuration.html>

The equivalent file in the XML Schema-style would be...

```
<?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 -->

</beans>
```

For example, using the following file you can pass any value for "message" variable and you can print different values of message without impacting **HelloWorld.java** and **MainApp.java** files.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns = "http://www.springframework.org/schema/beans"
3   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation = "http://www.springframework.org/schema/beans
5     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
6
7   <bean id = "helloWorld" class = "com.TP5.HelloWorld">
8     <property name = "message" value = "Hello World!"/>
9   </bean>
10
11 </beans>
12
```

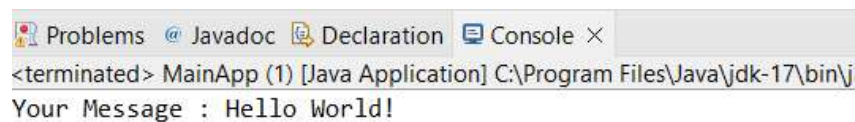


When Spring application gets loaded into the memory, Framework makes use of the above configuration file to create all the beans defined and assigns them a unique ID as defined in **<bean>** tag. You can use **<property>** tag to pass the values of different variables used at the time of object creation.

### Step 5 - Running the Program

Once you are done with creating the 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 –



```
<terminated> MainApp (1) [Java Application] C:\Program Files\Java\jdk-17\bin\j
Your Message : Hello World!
```

---

## Spring MVC Example

---

### Step 1- Spring MVC Example Eclipse Setup

- Right click on the project explorer window and click on “New -> Dynamic Web Project”.
- Provide name as “spring-mvc-example” in the next popup page, rest of the things should not require to be changed.
- On next page, provide the source folder as “src/main/java”.
- Next is the web module page, provide the context root of application as “spring-mvc-example” and make sure to check “Generate web.xml deployment descriptor” option.
- Click on Finish and you will have a new Dynamic Web Project in your eclipse project explorer.

### Step 2- Converting Dynamic Web Project to Maven Project

- Right click on the project and select “Configure -> Convert to Maven Project”.
- Next provide the **pom.xml** configurations as shown below.
- Our maven web application project skeleton code is ready. Now we can start making changes to it and create our spring mvc hello world example application.

**Create new POM**

**Maven POM**

This wizard creates a new POM (pom.xml) descriptor for Maven.

Project: /spring-mvc-example

Artifact:

Group Id: spring-mvc-example

Artifact Id: spring-mvc-example

Version: 0.0.1-SNAPSHOT

Packaging: war

Name: Spring MVC Example

Description: Spring MVC HelloWorld Example

Finish Cancel

### Step 3- Spring MVC Dependencies to pom.xml

We need to add **spring-web** and **spring-webmvc** dependencies in **pom.xml**, also add **servlet-api**, **jsp-api** and **jstl** dependencies. Our final **pom.xml** file will be like below.

```

7  <name>Spring MVC Example</name>
8  <description>Spring MVC HelloWorld Example</description>
9  <!-- Add Spring Web and MVC dependencies -->
10 <dependencies>
11   <dependency>
12     <groupId>org.springframework</groupId>
13     <artifactId>spring-webmvc</artifactId>
14     <version>4.3.9.RELEASE</version>
15   </dependency>
16   <dependency>
17     <groupId>org.springframework</groupId>
18     <artifactId>spring-web</artifactId>
19     <version>4.3.9.RELEASE</version>
20   </dependency>
21   <!-- Servlet -->
22   <dependency>
23     <groupId>javax.servlet</groupId>
24     <artifactId>servlet-api</artifactId>
25     <version>2.5</version>
26     <scope>provided</scope>
27   </dependency>
28   <dependency>
29     <groupId>javax.servlet.jsp</groupId>
30     <artifactId>jsp-api</artifactId>
31     <version>2.1</version>
32     <scope>provided</scope>
33   </dependency>
34   <dependency>
35     <groupId>javax.servlet</groupId>
36     <artifactId>jstl</artifactId>
37     <version>1.2</version>
38   </dependency>
39 </dependencies>
40 <build>

```



## Step 4- Spring MVC DispatcherServlet as Front Controller

We have to add Spring MVC framework to our web application, for that we need to configure **DispatcherServlet** in **web.xml** as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd" id="WebApp_ID" version="4.0">
  <display-name>spring-mvc-example</display-name>
  <!-- Add Spring MVC DispatcherServlet as front controller -->
    <servlet>
      <servlet-name>spring</servlet-name>
      <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
      <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring-servlet.xml</param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
      <servlet-name>spring</servlet-name>
      <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

**contextConfigLocation** init-param is used to provide the location of spring bean configuration file.

## Step 5- Spring MVC Example Bean Configuration File

Next step is to create spring bean configuration file **spring-servlet.xml** as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="https://www.springframework.org/schema/mvc"
xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xmlns:beans="https://www.springframework.org/schema/beans"
xmlns:context="https://www.springframework.org/schema/context"
xsi:schemaLocation="https://www.springframework.org/schema/mvc
https://www.springframework.org/schema/mvc/spring-mvc.xsd
https://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans.xsd
https://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context.xsd">

  <!-- DispatcherServlet Context: defines this servlet's request-processing
  infrastructure -->

  <!-- Enables the Spring MVC @Controller programming model -->
  <annotation-driven />
  <context:component-scan base-package="com.TP5.spring" />

  <!-- Resolves views selected for rendering by @Controllers to .jsp resources
  in the /WEB-INF/views directory -->
  <beans:bean
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <beans:property name="prefix" value="/WEB-INF/views/" />
    <beans:property name="suffix" value=".jsp" />
  </beans:bean>
</beans:beans>
```

There are three important configurations.

- I. **annotation-driven** tells DispatcherServlet to look for Controller classes using **@Controller** annotation.
- II. **context:component-scan** tells DispatcherServlet where to look for controller classes.
- III. **InternalResourceViewResolver** bean configuration to specify location of view pages and suffix used. Controller class methods return name of the view page and then suffix is added to figure out the view page to use for rendering the response.

## Step 6- Spring MVC Controller Class

We have a single controller class to respond for two URIs – “/” for home page and “/user” for user page.

First we need to create a package called **com.TP5.spring.controller** To do this, right click on **src** in package explorer section and follow the option – **New → Package**.

**HomeController.java:** Right click on **com.TP5.spring.controller** package – **New → Class → Name: HomeController → Finish**

```
package com.TP5.spring.controller;

import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.TP5.spring.model.User;

@Controller
public class HomeController {

    /**
     * Simply selects the home view to render by returning its name.
     */
    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String home(Locale locale, Model model) {
        System.out.println("Home Page Requested, locale = " + locale);
        Date date = new Date();
        DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG,
DateFormat.LONG, locale);

        String formattedDate = dateFormat.format(date);

        model.addAttribute("serverTime", formattedDate);

        return "home";
    }

    @RequestMapping(value = "/user", method = RequestMethod.POST)
    public String user(@Validated User user, Model model) {
        System.out.println("User Page Requested");
        model.addAttribute("userName", user.getUserName());
        return "user";
    }
}
```

## Step 7- Spring MVC Model Class

We have a simple model class with a single variable and it's getter-setter methods. It's a simple POJO class.

First we need to create a package called **com.TP5.spring.model** To do this, right click on **src** in package explorer section and follow the option – **New → Package**.

**User.java:** Right click on **com.TP5.spring.model** package – **New → Class → Name: User → Finish**

```
package com.TP5.spring.model;

public class User {
    private String userName;

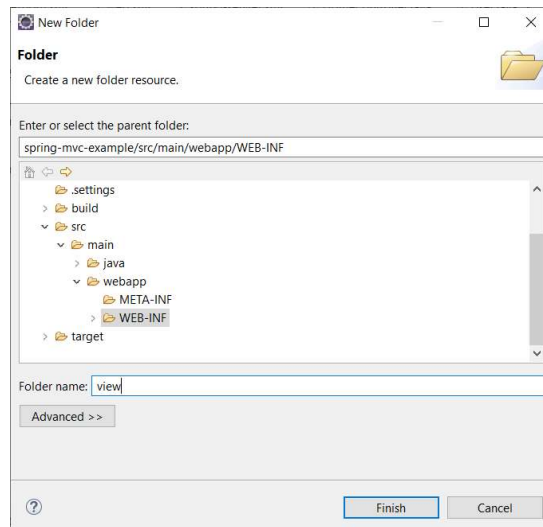
    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }
}
```

## Step 8- Spring MVC View Pages

We have two view pages as defined below.

Right click on the **WEB-INF** → **New → Folder New → Folder name: view**



Create 2 jsp files i.e. **home.jsp** and **user.jsp** inside the view folder as follows

### *home.jsp*

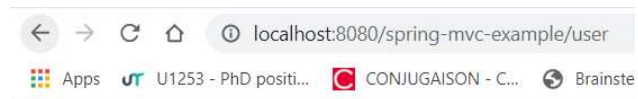


## Hello world!

The time on the server is May 22, 2022 at 10:25:59 PM CEST.

### *user.jsp*



**Hi Mohammad**

### Step 9- Spring MVC Eclipse Project Deployment

- We can use Eclipse export as WAR file option to deploy it directly to any running tomcat server webapps directory.
- However you can also use command line to build the project and then copy it into your favourite servlet container deployment directory.
- Or right click on the project → Run As → Run on Server