Aug 31st Lecture and Project spring-mvc-demo

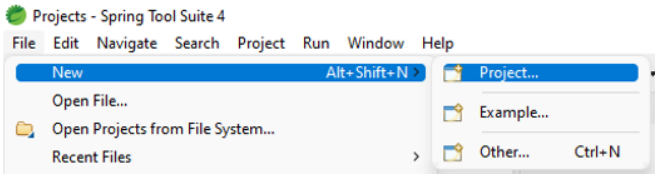
Create a Maven MVC project and deploy on an IDE Tomcat server. This document describes creating a MVC project in the SpringTool IDE. Other documents describes Tomcat server deployments within the IDE, on local machine, and on AWS.

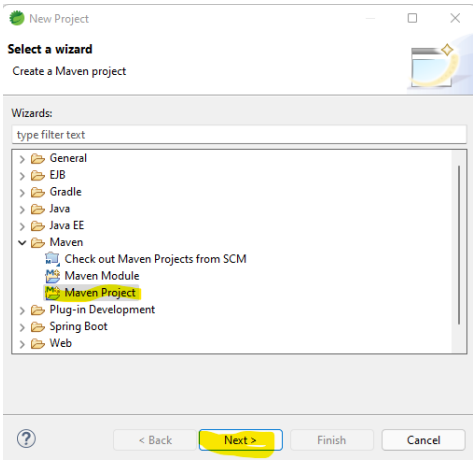
The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE.

Table of Contents

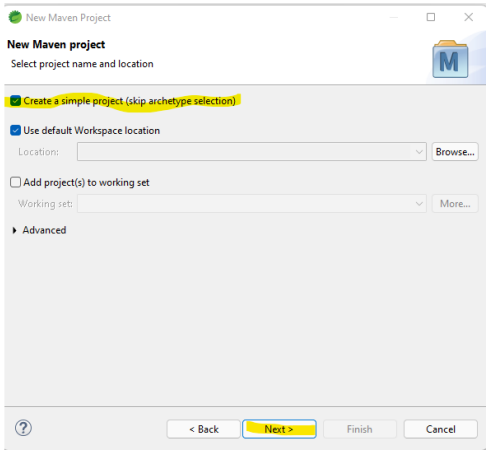
Create a Maven MVC project and deploy on an IDE Tomcat server

Create New Maven Project

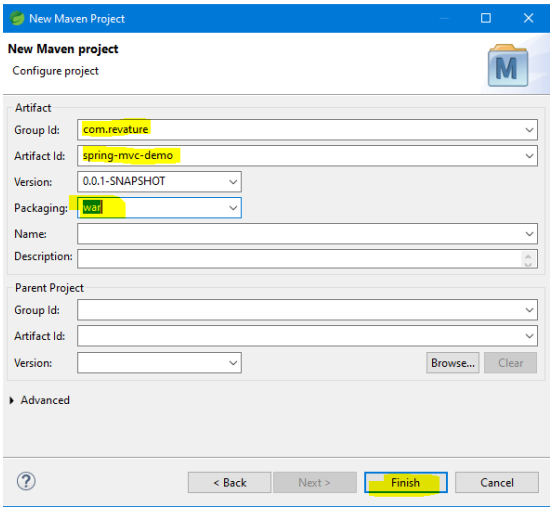




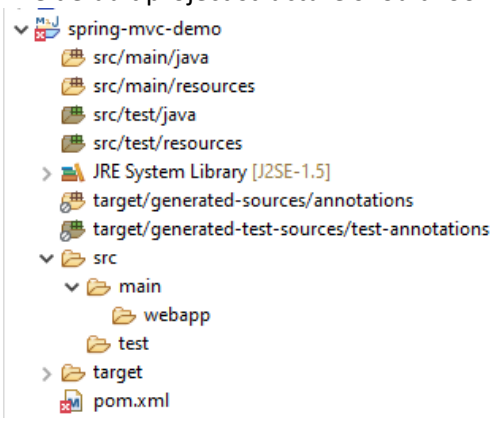
Select “Create a simple project …”
Click “Next >”



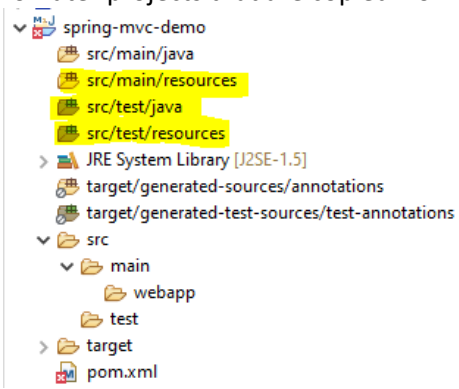
Enter the Group Id: com.revature
Enter the Artifact Id: spring-mvc-demo
Select Packaging: **war**
Note: packaging must be war to deploy on a Tomcat server.
Click Finish



The default project structure should look as follows:



It is important that the project structure does contain the highlighted folders. These will be important for later projects that are copied from this project.



The error indicated will be resolved in future steps.

Update the pom.xml

Update Java Version

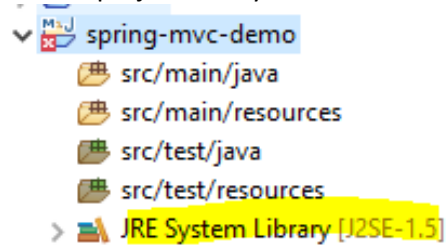
BEFORE PROCEEDING MAKE SURE THE PROJECT IS ON Java 1.8 in the pom.xml and has been rebuilt.

Default pom.xml:

spring-mvc-demo/pom.xml

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0"
2    <modelVersion>4.0.0</modelVersion>
3    <groupId>com.revature</groupId>
4    <artifactId>spring-mvc-demo</artifactId>
5    <version>0.0.1-SNAPSHOT</version>
6    <packaging>war</packaging>
7  </project>
```

Default project library:



Add the Java 1.8 in the pom.xml

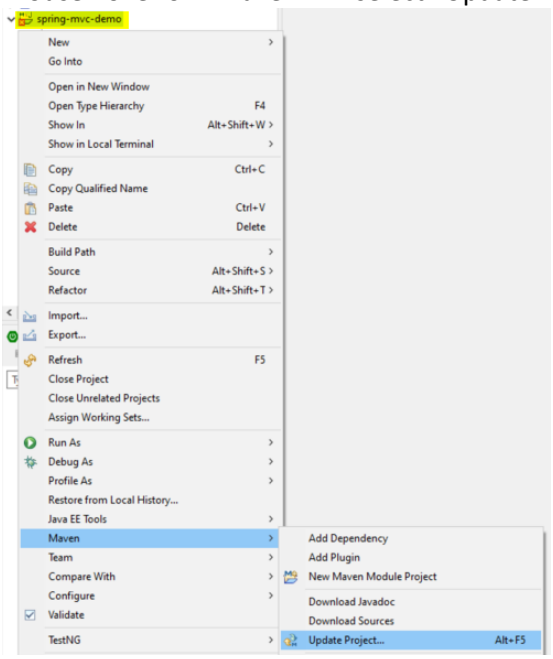
```
<properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

spring-mvc-demo/pom.xml

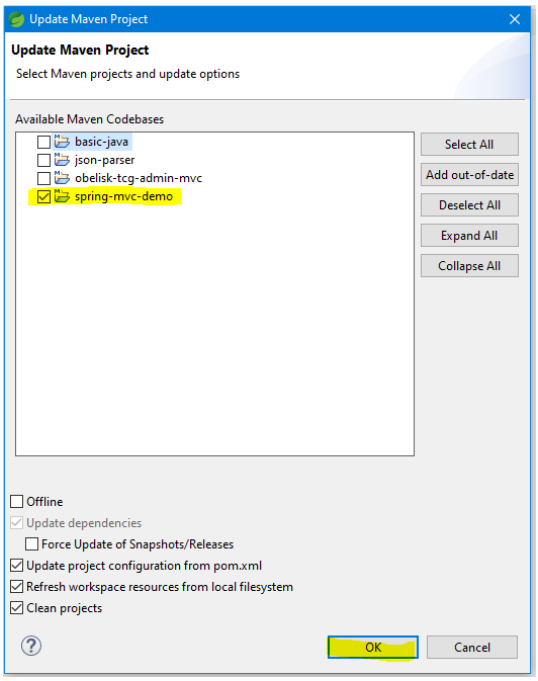
```
1  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="
2    <modelVersion>4.0.0</modelVersion>
   <groupId>com.revature</groupId>
5    <artifactId>spring-mvc-demo</artifactId>
6    <version>0.0.1-SNAPSHOT</version>
7    <packaging>war</packaging>
8
9      <properties>
10         <maven.compiler.source>1.8</maven.compiler.source>
11         <maven.compiler.target>1.8</maven.compiler.target>
12     </properties>
13
14 </project>
```

Rebuild the Maven project

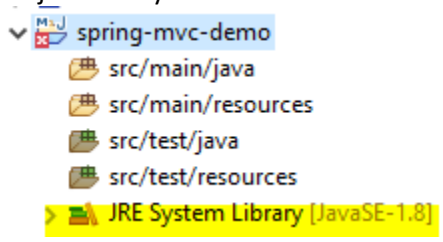
Right click on the Project
Mouse hover on “Maven” ➔ select “Update Project…”



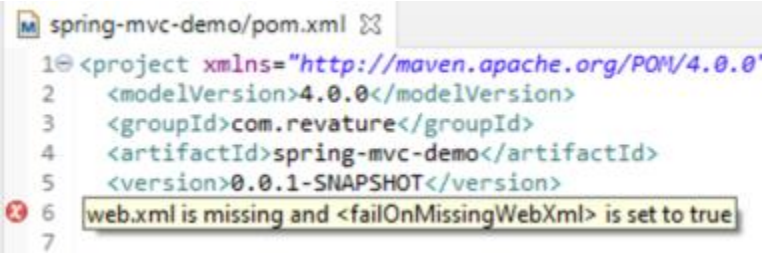
The project should already be selected.
Click “OK”



Project library should now be at the new Java version:

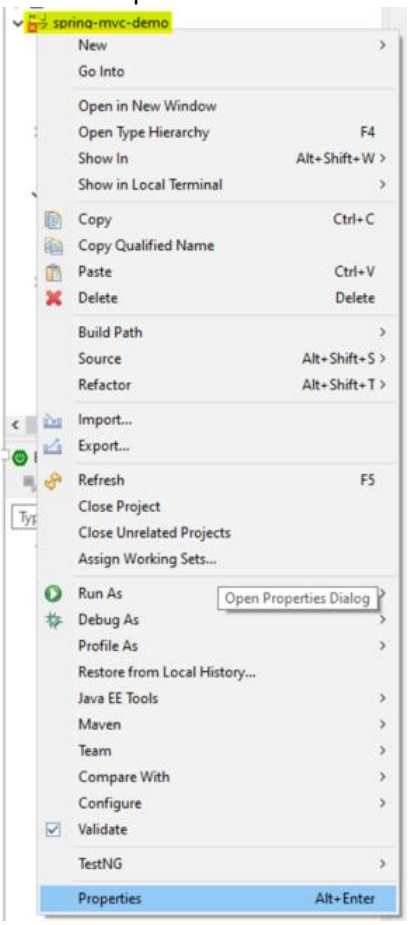


Correct Error: Web.xml is missing

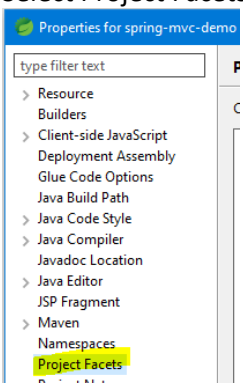


Change Project Properties

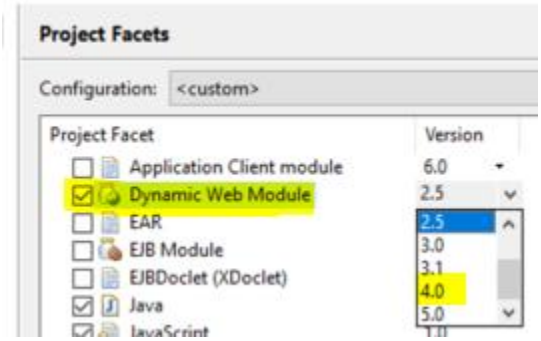
Right click on project
Select “Properties”



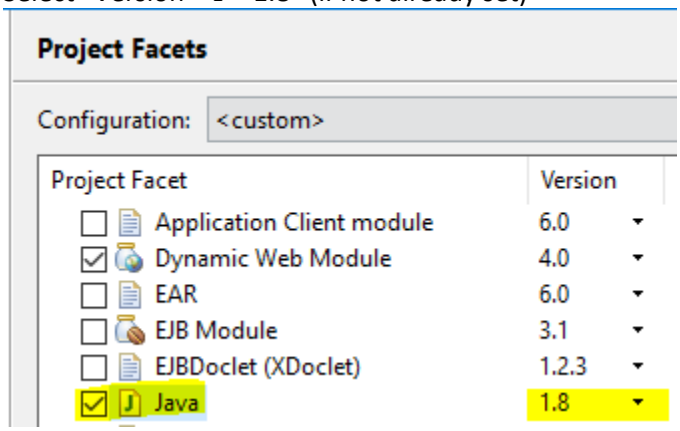
Select Project Facets



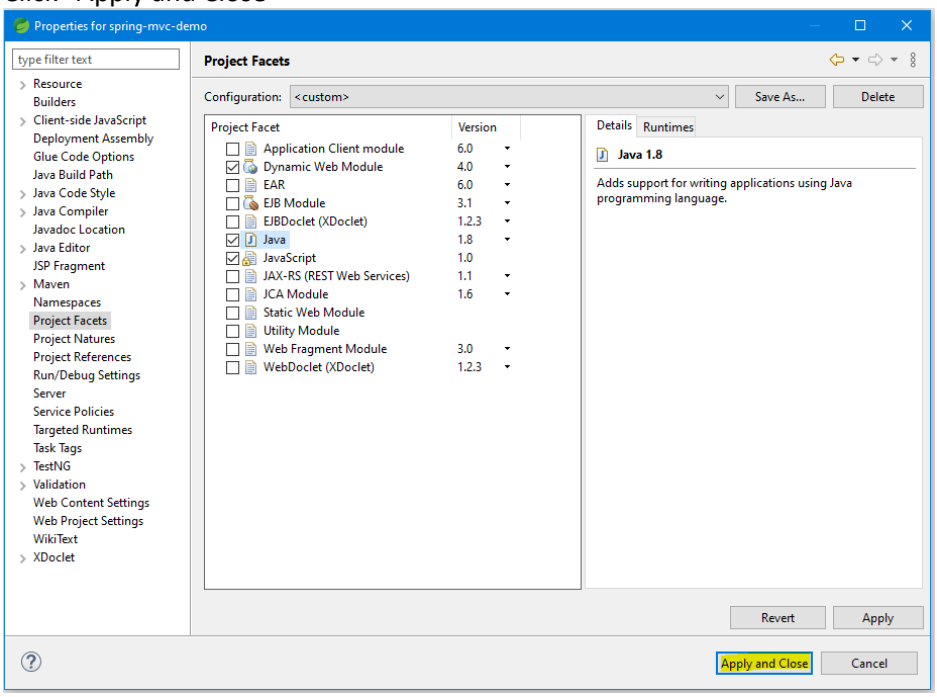
Click on Project Facet: “Dynamic Web Module”, check if unchecked
Select “Version” ➔ “4.0” from the dropdown



Click on Project Facet: “Java”, check if unchecked
Select “Version” ➔ “1.8” (if not already set)



Click “Apply and Close”

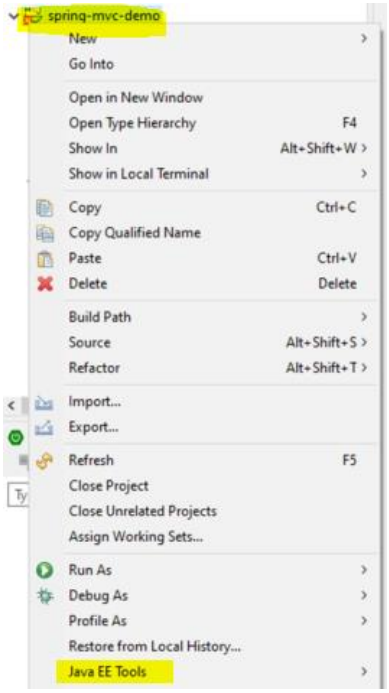


Install Java EE Tool if needed

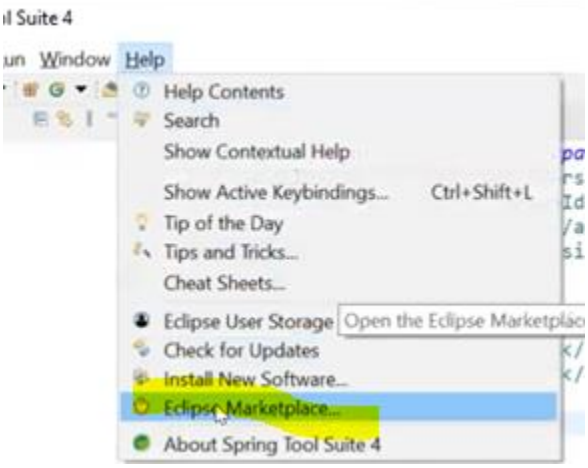
Right click on project

If “Java EE Tools” is in the menu list skip to “**Generate Deployment Descriptor Stub**”

If “Java EE Tools” is not in the menu list follow the next two steps



Step 1: Install via Eclipse Market Place under the Help menu



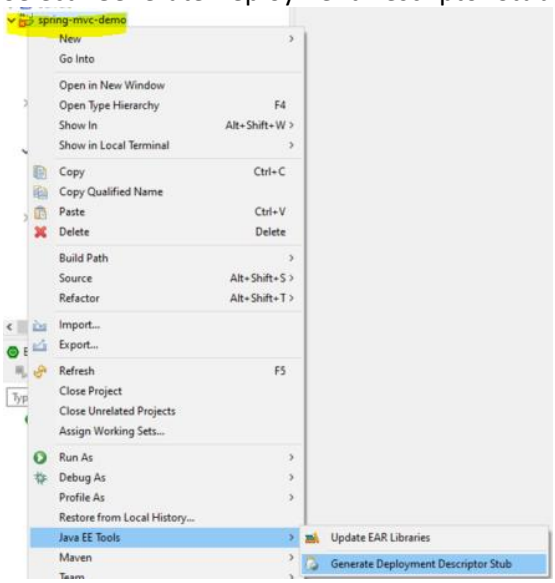
Step 2: Search tab: enter enterprise

If not already installed, install the component

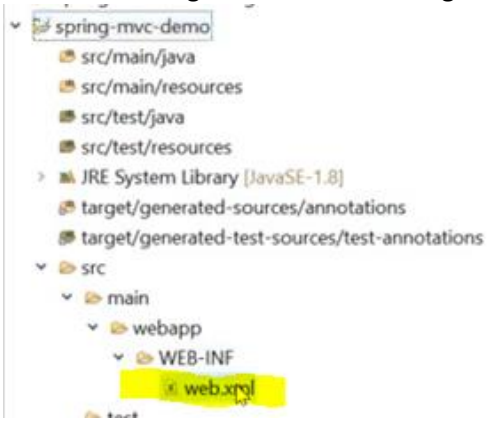
Once the installation completes **restart** the IDE for the changes to take affect

Generate Deployment Descriptor Stub

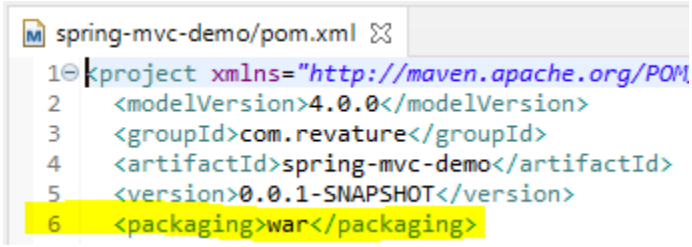
Right click on the project
Highlight “Java EE Tools”
Select “Generate Deployment Descriptor Stub”



The web.xml is generated in following location.



The pom.xml error should now be resolved.

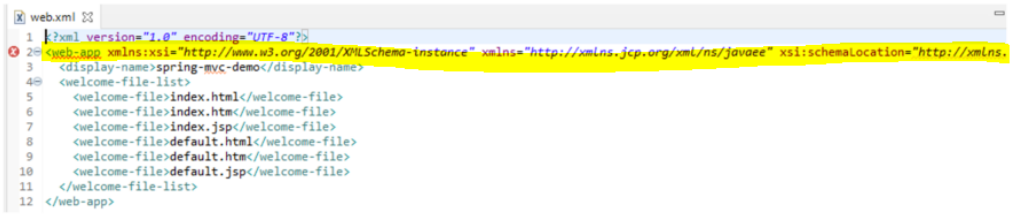


Update web.xml

Any time the web.xml file it can all of sudden have an error. This is some kind of bug with the IDE. To resolve the false error just add a space or add a line and save the file.

Open Web.xml File

Update outdated link



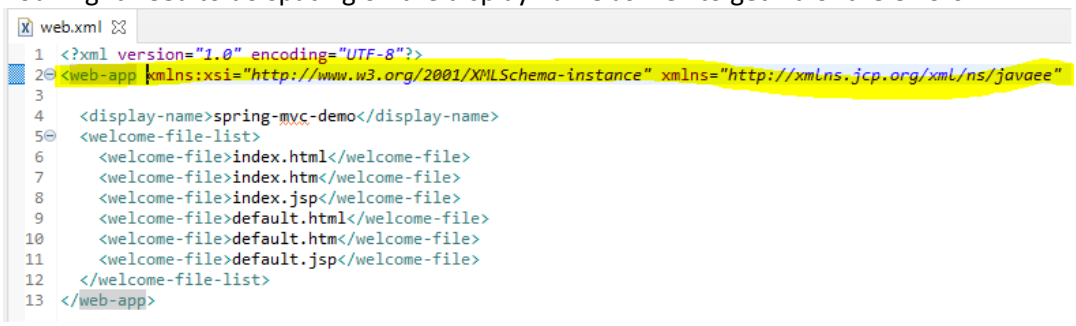
From:

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
version="2.5">
```

To:

```
<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" version="4.0">
```

You might need to do spacing on the display name as well to get rid of the errors.



Add DispatcherServlet to web.xml

The web.xml file is used for Spring Framework and Tomcat server configuration.
Add the following to the web.xml file

```
  <!-- This is where we configure our DispatcherServlet (which comes from Spring Web) -->
  <!-- The DispatcherServlet is the sole Servlet that receives HTTP requests through our Tomcat
server -->
  <!-- It then routes the HTTP requests to the appropriate controller -->
  <servlet>
        <servlet-name>DispatcherServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

        <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>>/WEB-INF/applicationContext.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
        <servlet-name>DispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
  </servlet-mapping>
```

web.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3      xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4      xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/ja
5      version="4.0">
6
7      <display-name>spring-mvc-demo</display-name>
8      <welcome-file-list>
9          <welcome-file>index.html</welcome-file>
10         <welcome-file>index.htm</welcome-file>
11         <welcome-file>index.jsp</welcome-file>
12         <welcome-file>default.html</welcome-file>
13         <welcome-file>default.htm</welcome-file>
14         <welcome-file>default.jsp</welcome-file>
15     </welcome-file-list>
16
17     <!-- This is where we configure our DispatcherServlet (which comes from
18         Spring Web) -->
19     <!-- The DispatcherServlet is the sole Servlet that receives HTTP requests
20         through our Tomcat server -->
21     <!-- It then routes the HTTP requests to the appropriate controller -->
22     <servlet>
23         <servlet-name>DispatcherServlet</servlet-name>
24         <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
25
26         <init-param>
27             <param-name>contextConfigLocation</param-name>
28             <param-value>/WEB-INF/applicationContext.xml</param-value>
29         </init-param>
30         <load-on-startup>1</load-on-startup>
31     </servlet>
32
33     <servlet-mapping>
34         <servlet-name>DispatcherServlet</servlet-name>
35         <url-pattern>/</url-pattern>
36     </servlet-mapping>
37
38 </web-app>
```

All http request goes to the DispatcherServlet which in turn sends to the controller(s) endpoints.
“Front Controller Design Pattern”

```
23          <servlet-name>DispatcherServlet</servlet-name>
24          <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
```

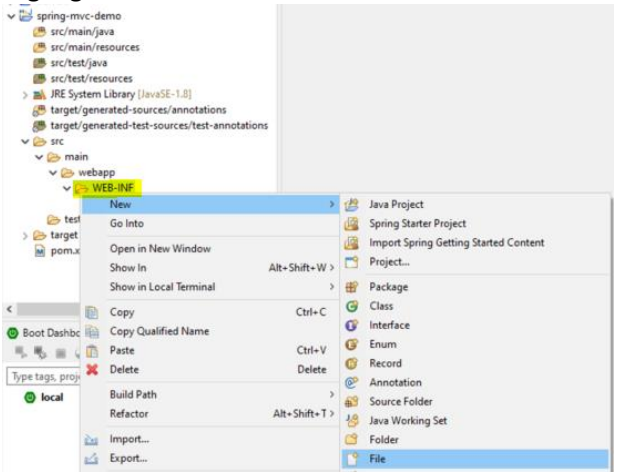
Create an applicationContext.xml File

The web.xml file makes reference to an applicationContext.xml file:

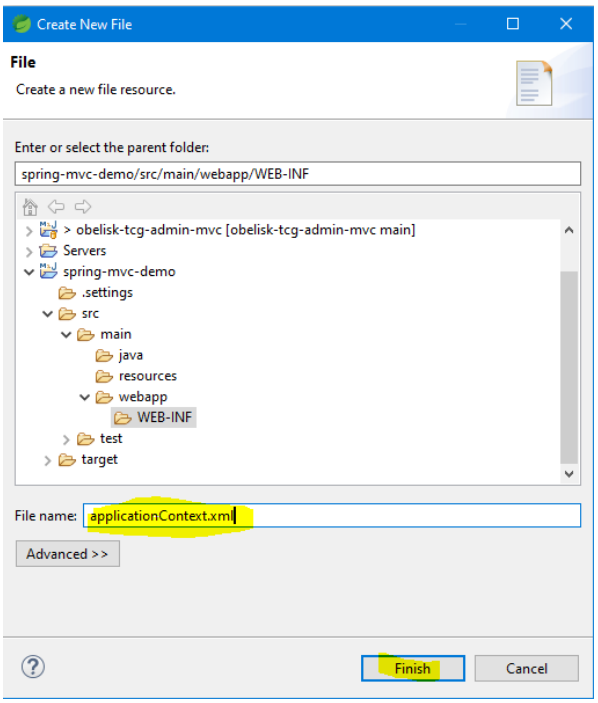
```
27              <param-name>contextConfigLocation</param-name>
28              <param-value>/WEB-INF/applicationContext.xml</param-value>
```

Create the applicationContext.xml file in the directory:
spring-mvc-demo\src\main\webapp\WEB-INF\

Right click on WEB-INF folder
Highlight “New” ➔ select “File”



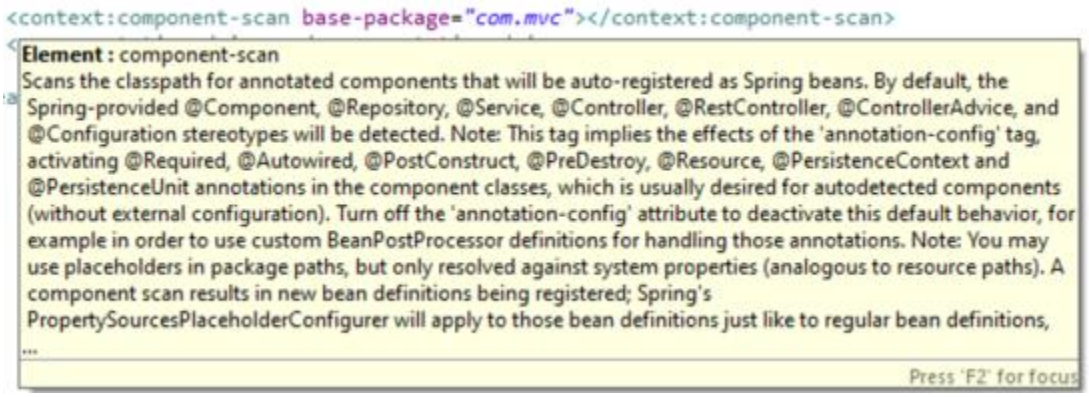
Enter the “File name:” applicationContext.xml
Click “Finish”



Update the applicationContext.xml File

Add the following to the xml file:

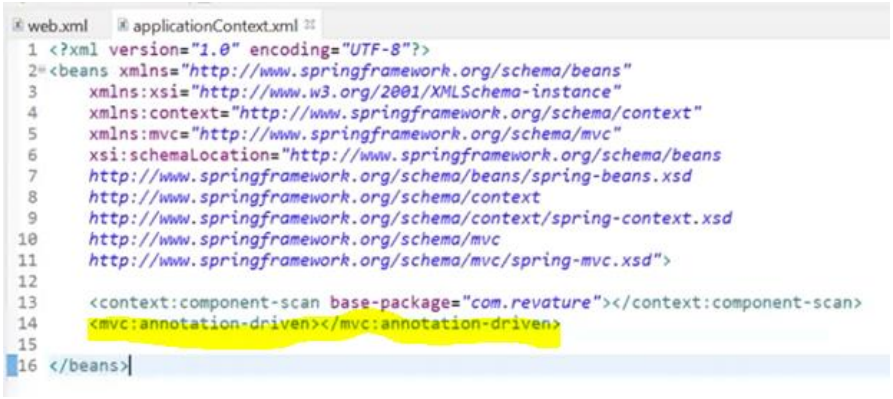
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:mvc="http://www.springframework.org/schema/mvc"
    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

        <context:component-scan base-package="com.revature"></context:component-scan>
        <mvc:annotation-driven></mvc:annotation-driven>

</beans>
```

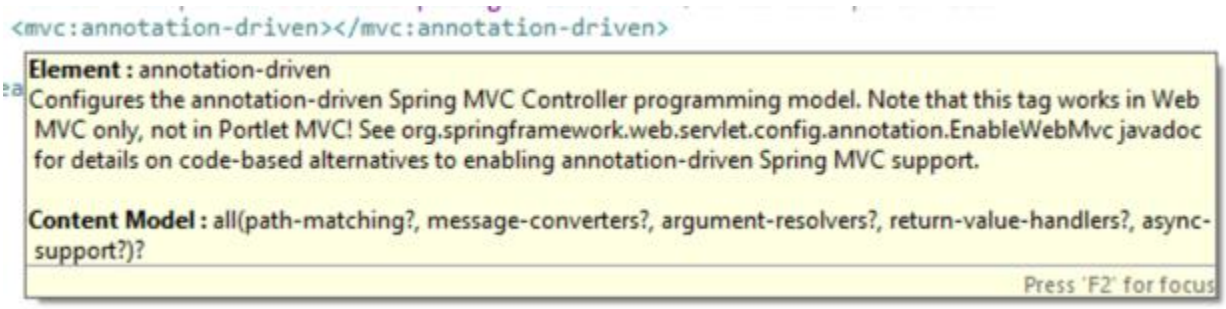


Make sure to update the base package: com.revature to the project's definition.

Make sure to add the <mcv:annotation> tags

mcv:annotation allows for the @Component, @Service and @RestController tags.





Add other project dependencies to pom.xml

The current pom.xml file should look like the following after we Java 1.8 to the default file earlier in this document:

spring-mvc-demo/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="ht
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.revature</groupId>
  <artifactId>spring-mvc-demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

</project>
```

Add dependencies tags

Between the dependencies tags

- Add dependencies for Spring Framework
- Add dependencies for Tomcat
- Add dependencies for Project Lomback
- Other dependencies for other common project items in future projects depending on needs.

The dependencies added are taken from the Maven Repository: https://mvnrepository.com/. Future dependencies needed can also be found on this website.

```
        <dependencies>
                <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
                <dependency>
                        <groupId>javax.servlet</groupId>
                        <artifactId>javax.servlet-api</artifactId>
                        <version>4.0.1</version>
                        <scope>provided</scope>
                </dependency>
                <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-
databind -->
                <dependency>
                        <groupId>com.fasterxml.jackson.core</groupId>
                        <artifactId>jackson-databind</artifactId>
                        <version>2.12.5</version>
                </dependency>
                <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-webmvc</artifactId>
                        <version>5.3.9</version>
                </dependency>
                <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
                <dependency>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                        <version>1.18.20</version>
                        <scope>provided</scope>
                </dependency>
        </dependencies>
```

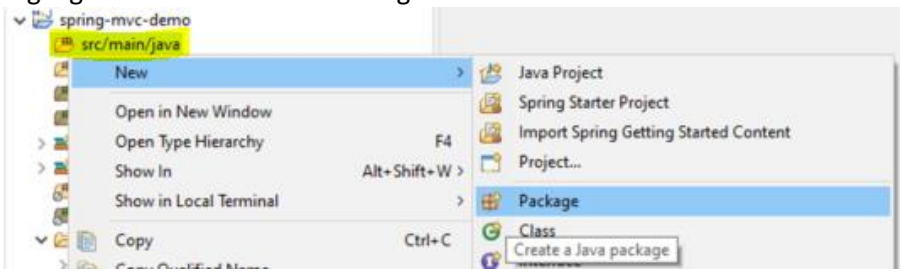
spring-mvc-demo/pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-i
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.revature</groupId>
  <artifactId>spring-mvc-demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>4.0.1</version>
            <scope>provided</scope>
        </dependency>
        <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.12.5</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.3.9</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.20</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

</project>
```

Complete the MVC demo project

Create controller package

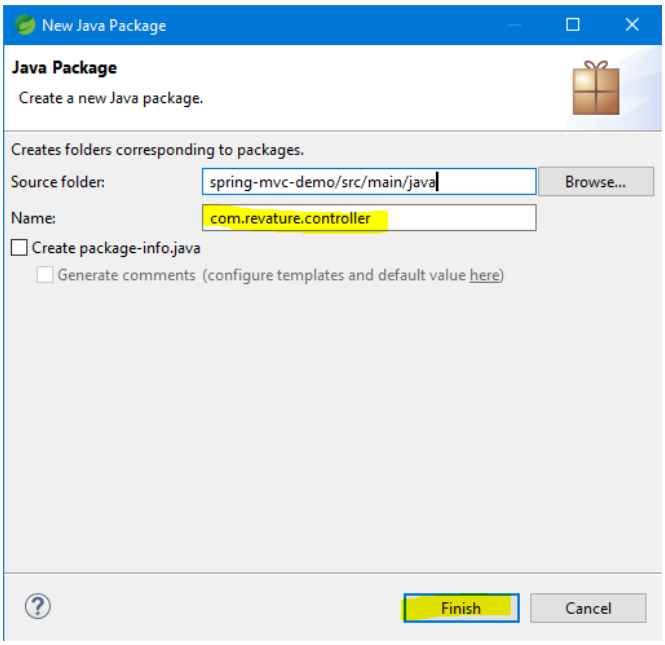
Right click folder: “src/main/java”
Highlight “New” ➔ select “Package”



Enter “Name:”

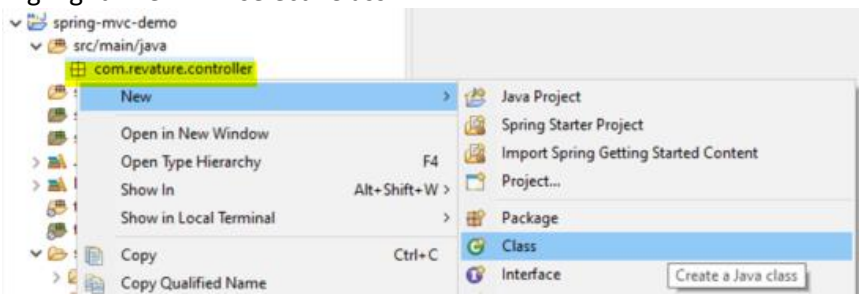
com.revature.controller

Click “Finish”



Create controller class

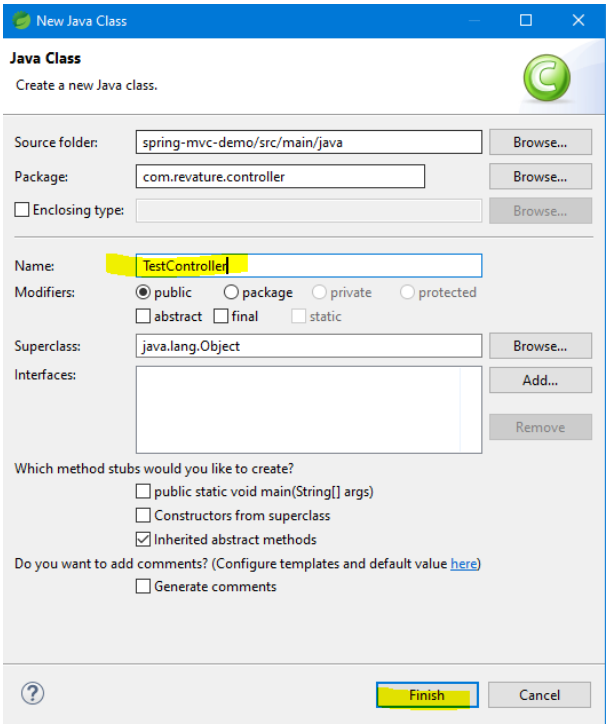
Right click package: “com.revature.controller”
Highlight “New” ➔ select “Class”



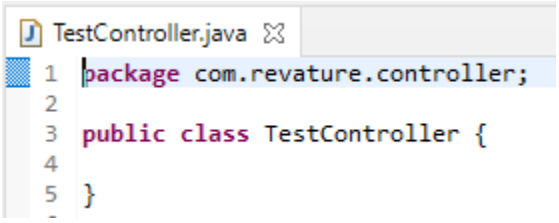
Enter “Name:”

TestController

Click “Finish”



Update the class shell TestController



The final class code will contain:

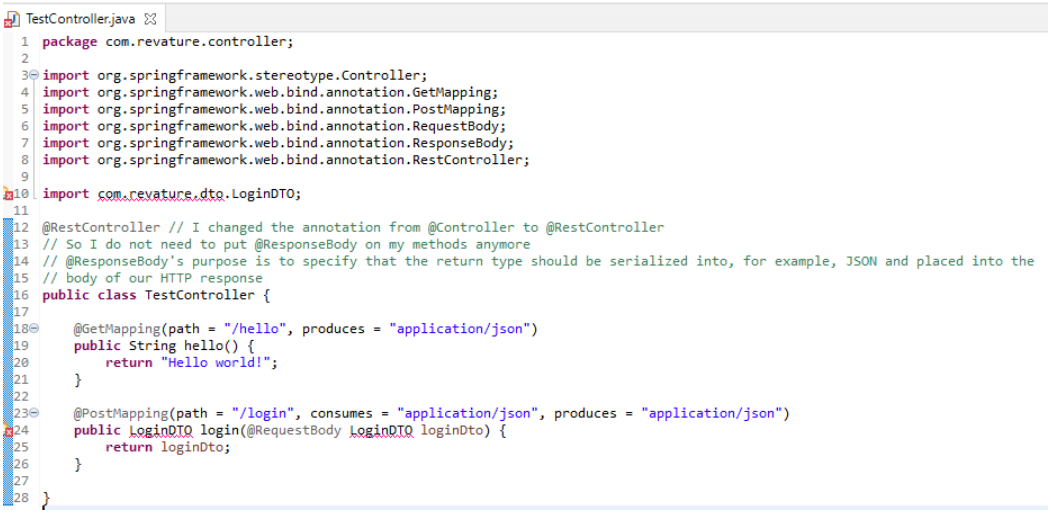
```
package com.revature.controller;

import org.springframework.stereotype.Controller;
```

```java
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

import com.revature.dto.LoginDTO;

@RestController // I changed the annotation from @Controller to @RestController
// So I do not need to put @ResponseBody on my methods anymore
// @ResponseBody's purpose is to specify that the return type should be serialized into, for
example, JSON and placed into the
// body of our HTTP response
public class TestController {

	@GetMapping(path = "/hello", produces = "application/json")
	public String hello() {
		return "Hello world!";
	}

	@PostMapping(path = "/login", consumes = "application/json", produces =
"application/json")
	public LoginDTO login(@RequestBody LoginDTO loginDto) {
		return loginDto;
	}

}
```

You can type each line of code starting with `@RestController`. If you take this approach then required imports are generally added automatically and you can take care of other dependencies or errors as you type.

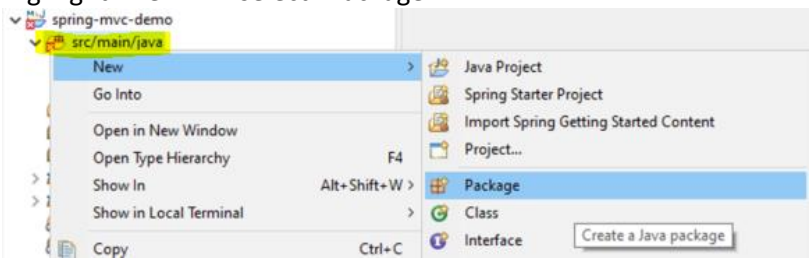
Alternatively just copy and paste code above into the TestController class. Then continue following instruction in this document and by the end all error will be resolved.

TestController class after copy and paste showing errors for a dependent package and class.



Create Data Transfer Object (DTO) package

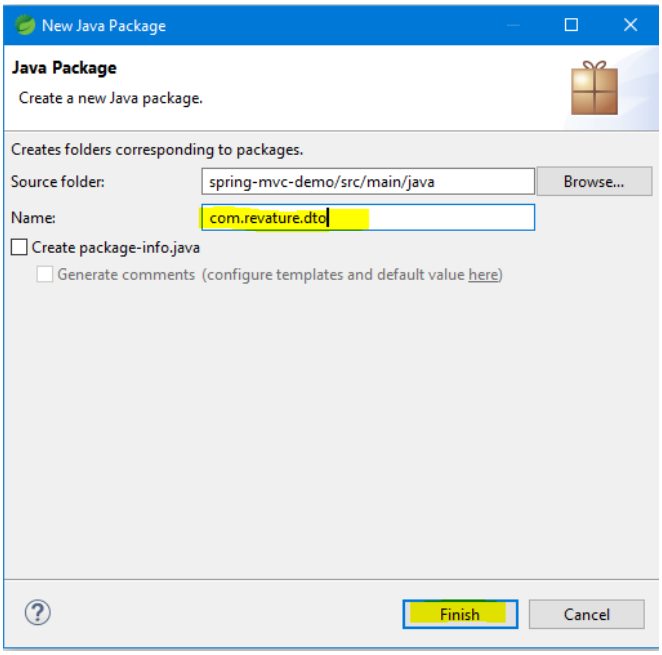
Right click folder: “src/main/java”
Highlight “New” ➔ select “Package”



Enter “Name:”

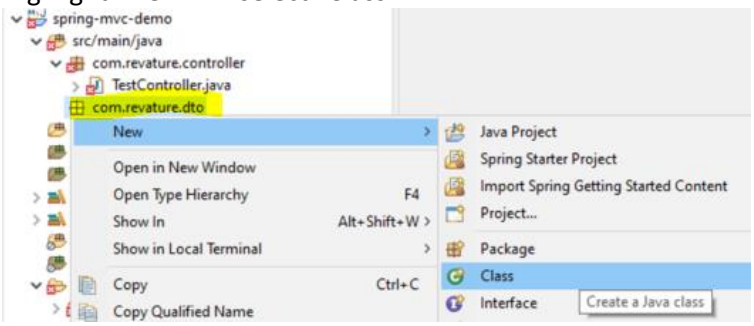
com.revature.dto

Click “Finish”



Create DTO class

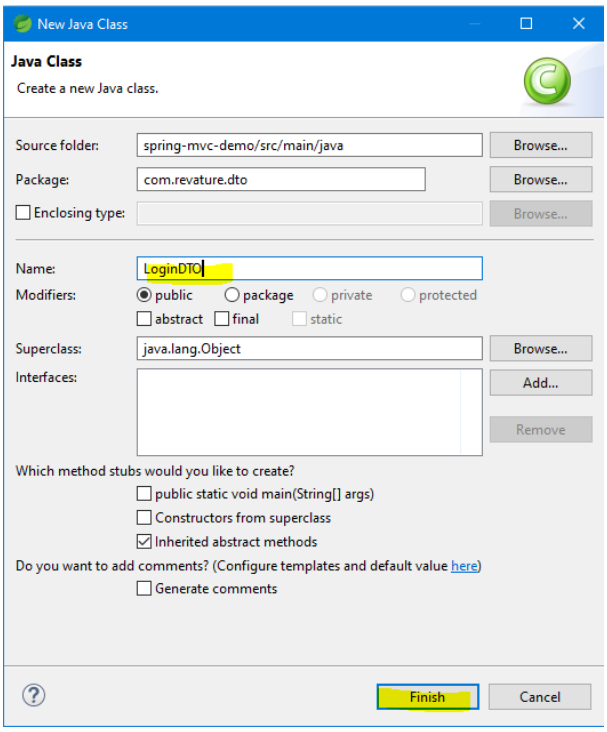
Right click package: “com.revature.dto”
Highlight “New” ➔ select “Class”



Enter “Name:”

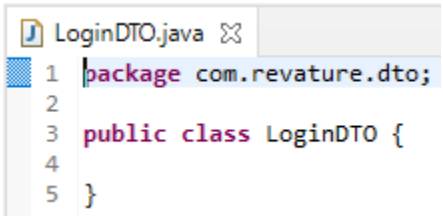
LoginDTO

Click “Finish”



Once the LoginDTO class is created the errors in TestController class are resolved.

Update the class shell LoginDTO



The final class code will contain:

```
package com.revature.dto;

import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@Getter @Setter @EqualsAndHashCode @NoArgsConstructor @ToString
public class LoginDTO {

	private String username;
	private String password;

	public LoginDTO(String username, String password) {
		this.username = username;
		this.password = password;
	}
```

```
}
```

As before uou can type each line of code starting with `@Getter`. If you take this approach then required imports are generally added automatically and you can take care of other dependencies or errors as you type.

Alternatively just copy and paste code above into the LoginDTO class. After completing this class there should be no other errors in the project.

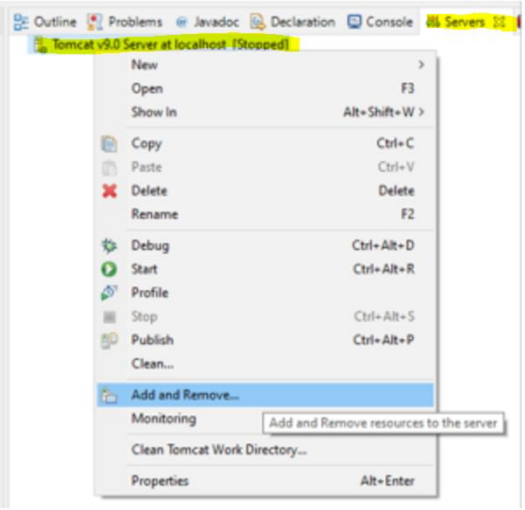
If you do not have a Tomcat server installed in the SpringToolSuite IDE proceed to section: Spring MVC Project Next Steps for Tomcat Server(s).

If you do have a Tomcat server installed in the SpringToolSuite IDE proceed to the next section: Test the MVC Demo project on the IDE Tomcat Server.

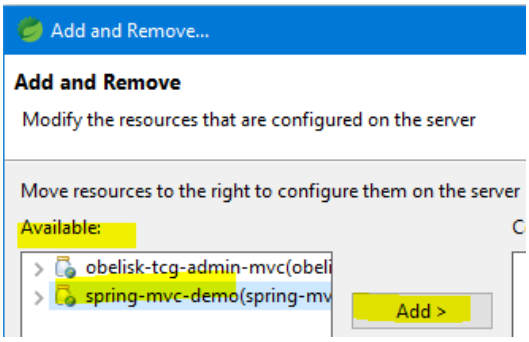
Test the MVC Demo project on the IDE Tomcat Server

Your location of the Servers tab and window may vary depending on your IDE layout.

In the “Servers” tab
Right click the “Tomcat …” server ➔ select “Add and Remove…”

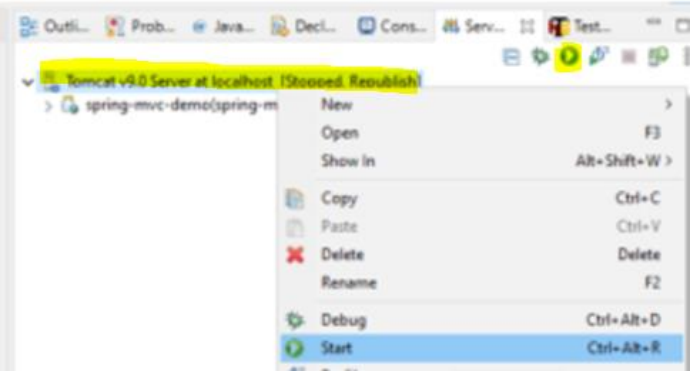


Select our project from “Available:” window
Click “Add >”



Start the Tomcat Server

Right click the server
Select “Start”

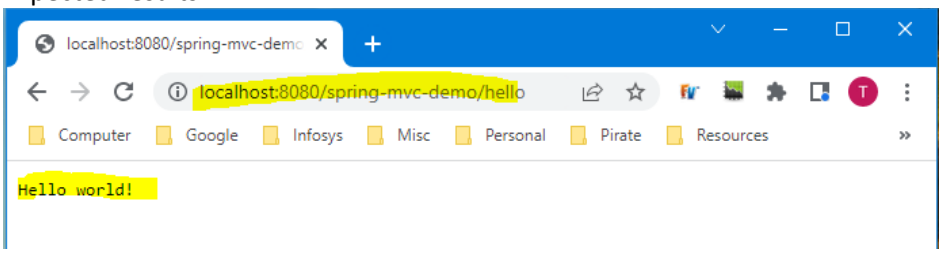


Hello World Test

Enter the following URL in a web browser:

localhost:8080/spring-mvc-demo/hello

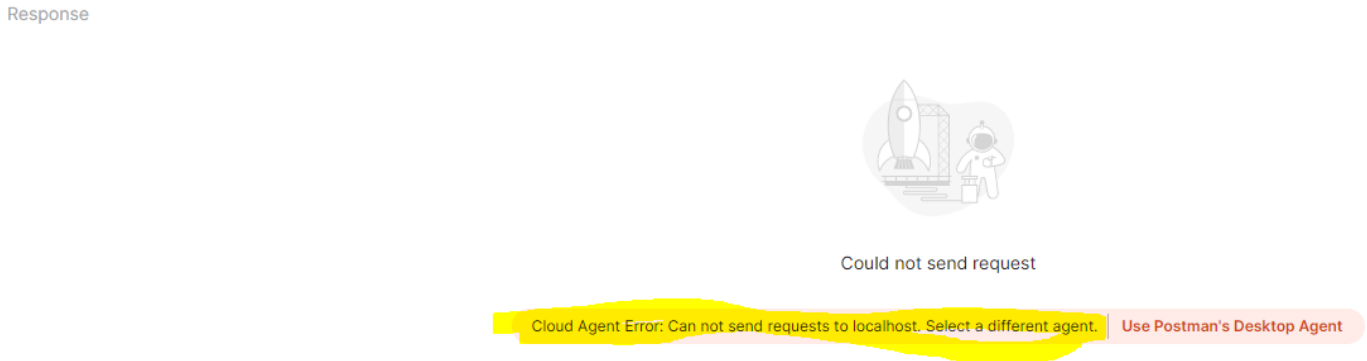
Expected results:



Login Test

Postman web (cloud) based issue

At some point the web based version of Postman no longer allows sending a request to localhost. You will need to use the Desktop Agent (**install** it if not already done).



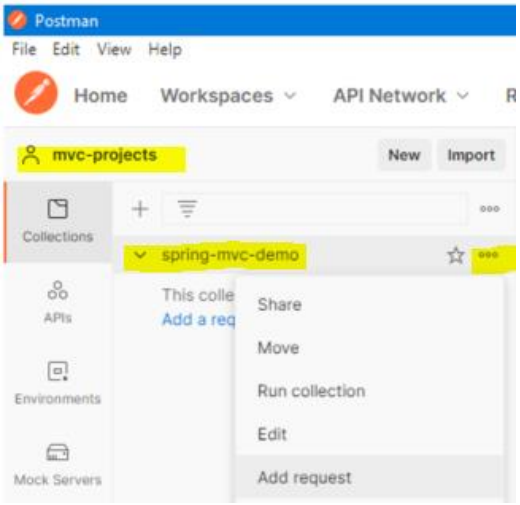
After installing and logging into the desktop version of Postman, you should have access to any Collections in your Workspace created using the web based version.

Open Postman Desktop Agent

In your work space you can create a new collection if desired.

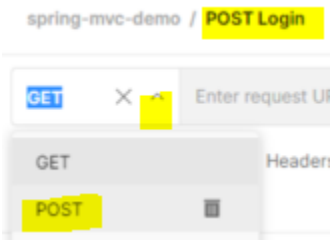
Add a Request

In a collection select the menu dots
Select “Add request”



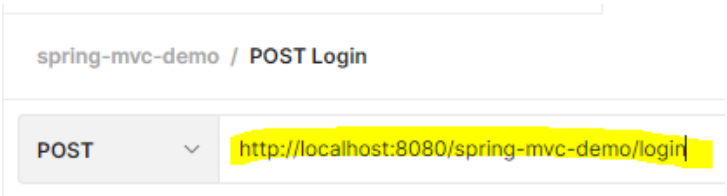
Modify the request

Give the request a name: POST Login
Select “POST” from request type dropdown

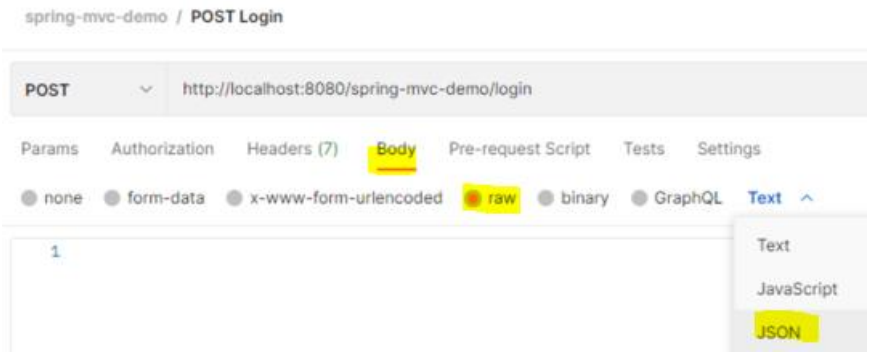


Enter the POST URL:

http://localhost:8080/spring-mvc-demo/login



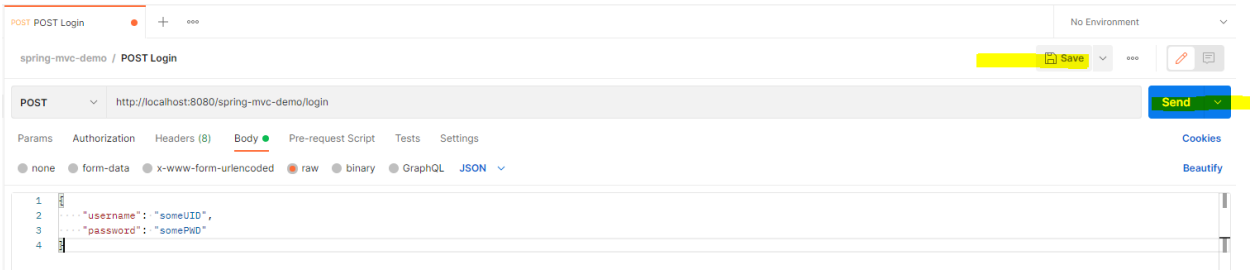
Add login request body
Select tab: “Body”
Click radio button: “raw”
Select “JSON” from type dropdown



Add body definition:

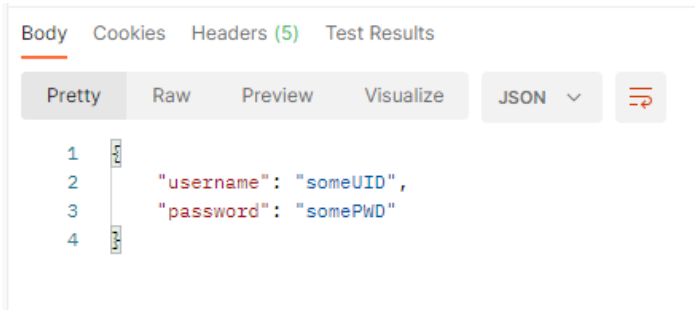
```
{
    "username": "someUID",
    "password": "somePWD"
}
```

Save request
Send request



Expected Response

The spring-mvc-demo is setup to echo the same information it received in JSON format back to the caller.



Spring MVC Project Next Steps for Tomcat Server(s)

After completing the code in this document, follow the steps in the “Tomcat 01 IDE Setup Aug 31st” setup document which walks through the steps to create a Tomcat server in the Spring Tool IDE and run this MVC based applications.

Another Tomcat document “Tomcat 02 localhost Deploy war Sep 8th” detail Tomcat setup on a local computer, deploy and run an MVC application war file and connect to the application through localhost.

Another Tomcat document “Tomcat 03 AWS EC2 Sep 8th” detail Tomcat installation and setup on an AWS EC2 instance, deploy and run an MVC application war file and connect to the application through an AWS URL.