

Enterprise application development with Spring 5

Spring Core + Spring MVC

1	SECTION 1: SETTING UP THE DEVELOPMENT ENVIRONMENT	3
1.1	DEV ENVIRONMENT OVERVIEW	3
1.2	INSTALLING UTILITY APPS	3
1.3	INSTALLING JAVA	4
1.4	INSTALLING MAVEN	4
1.5	INSTALLING ECLIPSE FOR ENTERPRISE JAVA.....	5
1.6	SETTING UP A SIMPLE MAVEN PROJECT IN ECLIPSE	5
1.7	INSTALLING TOMCAT	6
1.8	INTEGRATING TOMCAT WITH ECLIPSE	7
1.9	CONFIGURING TOMCAT	7
1.10	DOWNLOADING SPRING 5 JARS.....	8
2	SECTION 2: WORKING WITH MAVEN.....	8
2.1	MAVEN OVERVIEW	8
2.2	LAB: GENERATING AND USING A JAR.....	8
2.3	LAB: USING JARS FROM EXTERNAL LIBRARIES (LOG4J AND JUNIT)	9
2.4	LAB: CREATING A MAVEN PROJECT USING AN ARCHETYPE	11
2.5	LAB: MAVEN FEATURES.....	14
2.5.1	<i>Lab practice.....</i>	<i>14</i>
2.5.2	<i>Documentation on Maven features.....</i>	<i>19</i>
3	SECTION 3: SPRING CORE	21
3.1	SPRING FRAMEWORK OVERVIEW	21
3.2	LAB: SPRING CONCEPTS: IOC AND DI	21
3.2.1	<i>Lab Practice.....</i>	<i>21</i>
3.2.2	<i>IoC and DI Theory</i>	<i>22</i>
3.3	LAB: XML-BASED CONFIGURATION BASICS	23
3.3.1	<i>Lab Practice.....</i>	<i>23</i>
3.3.2	<i>Theory.....</i>	<i>26</i>
3.4	LAB: XML-BASED CONSTRUCTOR DI.....	27
3.4.1	<i>Approach 1: Nest child bean approach</i>	<i>28</i>
3.4.2	<i>Approach 2: Reference child bean separately</i>	<i>28</i>
3.5	LAB: XML-BASED SETTER DI	28
3.5.1	<i>Approach 1: Nest child bean approach</i>	<i>29</i>
3.5.2	<i>Approach 2: Reference child bean separately</i>	<i>29</i>
3.6	LAB: XML-BASED SETTER DI USING LITERAL VALUES	29
3.6.1	<i>Approach 1: Specify literal values directly in XML configuration file</i>	<i>30</i>
3.6.2	<i>Approach 2: Load values from an external property file.....</i>	<i>30</i>
3.7	LAB: XML-BASED SETTER DI - INJECTING COLLECTIONS	30
3.7.1	<i>Injecting collections of basic values.....</i>	<i>31</i>
3.7.2	<i>Injecting collections of beans</i>	<i>31</i>
3.8	LAB: XML-BASED AUTOWIRING	32
3.8.1	<i>Approach 1: Autowiring using byName</i>	<i>32</i>
3.8.2	<i>Approach 2: Autowiring using byType</i>	<i>32</i>

3.8.3	Approach 3: Autowiring using constructor.....	33
3.9	LAB: ANNOTATION-BASED CONFIGURATION BASICS	33
3.9.1	Using explicit component name	33
3.9.2	Using default component name	34
3.10	LAB: ANNOTATION-BASED DI	34
3.10.1	Constructor injection	34
3.10.2	Setter / method injection	35
3.10.3	Field injection	35
3.10.4	Problems with multiple candidate beans.....	35
3.10.5	Using autowiring by name to select between multiple candidate beans.....	36
3.10.6	Using @Qualifier to select between multiple candidate beans.....	36
3.10.7	Using @Primary to give higher preference to a bean.....	36
3.10.8	Injecting values from a properties file with @Value.....	36
3.10.9	Marking optional dependencies with @Autowired(required = false)	37
3.11	LAB: JAVA-BASED CONFIGURATION	37
3.11.1	Basic configuration setup with @Configuration and @ComponentScan.....	38
3.11.2	Defining beans with @Bean and retrieving them.....	38
3.11.3	Using @Primary to give higher preference to a @Bean factory method.....	38
3.11.4	Injecting values using @PropertySource and @Value	38
3.11.5	Injecting collections.....	39
3.11.6	Using @Component and @Autowired with @Bean.....	39
4	SECTION 4: SERVLET AND JAVA SERVER PAGES (JSP) BASICS	39
4.1	HTTP BASICS	39
4.2	INTRO TO SERVLET AND JSP	40
4.3	LAB: WORKING WITH SERVLETS IN TOMCAT	40
4.3.1	Creating a Dynamic Web Project.....	40
4.3.2	Working with <welcome-file-list> in web.xml	40
4.3.3	THEORY: Schema for web.xml	43
4.3.4	Defining and mapping a servlet in web.xml.....	44
4.3.5	Mapping a servlet with @WebServlet	45
4.3.6	Reading HTML form data and using initialization parameters.....	45
4.3.7	Configuring the integrated Eclipse Tomcat instance	46
4.3.8	Deploying as a WAR to the standalone Tomcat instance.....	47
4.4	LAB: WORKING WITH JSP IN TOMCAT	48
4.4.1	Creating a Dynamic Web Project.....	48
4.4.2	Declaration, expression and scriptlet tags	50
4.4.3	Using JSP implicit objects	50
4.4.4	Servlet forwarding to a JSP	50
4.4.5	Using Expression Language (EL)	51
4.4.6	JSP page and include directive	51
4.4.7	JSTL Core and Function Tags	52
4.4.8	Creating a servlet/JSP webapp as a Maven project.....	53
4.4.9	Add logging functionality to the web app.....	56
4.4.10	Deploying as a WAR to the standalone Tomcat instance	57
5	SECTION 5: SPRING MVC	58
5.1	SPRING MVC OVERVIEW.....	59
5.2	LAB: CREATING A SPRING MVC APPLICATION	59
5.2.1	Using a XML configuration.....	60
5.2.2	Using a Java-based configuration.....	61
5.3	LAB: USING @REQUESTMAPPING, @GETMAPPING, @POSTMAPPING ANNOTATIONS FOR MAPPING.....	61
5.4	LAB: WORKING WITH MODEL	63

5.4.1	Adding attributes to and displaying attributes from Model.....	63
5.4.2	Using @RequestParam to bind form parameters.....	64
5.5	LAB: FORM TAG LIBRARY	65
5.5.1	Text field tag <form:input>	66
5.5.2	Text area tag <form:textarea>.....	66
5.5.3	Drop down list <form:select> with options in form.....	66
5.5.4	Drop down list <form:select> with options from POJO	67
5.5.5	Drop down list <form:select> with options from @Controller class.....	67
5.5.6	Drop down list <form:select> with options from properties file	68
5.5.7	Radio buttons <form:radiobuttons> with options from @Controller class	68
5.5.8	Check boxes <form:checkboxes> with options from properties file	68
5.5.9	Using ModelAndView vs Model.....	69
5.5.10	Other form tags: <form:password> <form:hidden> <form:errors>	69
5.6	LAB: FORM VALIDATION	69
5.6.1	Required field validation	70
5.6.2	Number range validation and custom error messages.....	71
5.6.3	Regular expression validation	71
5.6.4	Using ResourceBundle to provide customized messages in properties file	72
5.6.5	Using @InitBinder with property editors and data binding	73
5.6.6	Creating a custom validator annotation to perform custom validation	73

1 Section 1: Setting up the Development Environment

1.1 Dev Environment Overview

Video link: <https://youtu.be/oMRsTya62cM>

Please make sure you have Chrome installed and at least one other browser (Edge, Firefox)

<https://www.google.com/chrome/>

<https://www.microsoft.com/en-us/edge>

<https://www.mozilla.org/en-US/firefox/new/>

1.2 Installing utility apps

Video link: https://youtu.be/y_dszsAPwKY

Notepad++

<https://notepad-plus-plus.org/downloads/>

7z

<https://www.7-zip.org/download.html>

1.3 Installing Java

Video link: <https://youtu.be/M283rhK1WxE>

We will be using Java 8

<https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>

Windows

<https://devwithus.com/install-java-windows-10/>

<https://www.guru99.com/install-java.html>

Linux

<https://docs.datastax.com/en/jdk-install/doc/jdk-install/installOpenJdkDeb.html>

<https://www.javahelps.com/2015/03/install-oracle-jdk-in-ubuntu.html>

Mac

<https://mkyong.com/java/how-to-install-java-on-mac-osx/>

<https://stackoverflow.com/questions/24342886/how-to-install-java-8-on-mac>

https://docs.oracle.com/javase/8/docs/technotes/guides/install/mac_jdk.html

1.4 Installing Maven

Video link: <https://youtu.be/wv29SLmQLa0>

<https://maven.apache.org/download.cgi>

Windows

<https://maven.apache.org/install.html>

<https://mkyong.com/maven/how-to-install-maven-in-windows/>

<https://howtodoinjava.com/maven/how-to-install-maven-on-windows/>

<https://stackoverflow.com/questions/17136324/what-is-the-difference-between-m2-home-and-maven-home>

Linux

<https://maven.apache.org/install.html>

<https://www.journaldev.com/33588/install-maven-linux-ubuntu>

<https://www.baeldung.com/install-maven-on-windows-linux-mac>

<https://linuxize.com/post/how-to-install-apache-maven-on-ubuntu-20-04/>

Mac

<https://www.journaldev.com/2348/install-maven-mac-os>

<https://www.baeldung.com/install-maven-on-windows-linux-mac>

<https://stackoverflow.com/questions/8826881/maven-install-on-mac-os-x>

1.5 Installing Eclipse for Enterprise Java

Video link: <https://youtu.be/LRqUfMkVfsc>

<https://www.eclipse.org/downloads/packages/release>

Select the appropriate release version and choose the R packages. Release 2020-09 and later are only compatible with Java 11, so make sure you select one of the earlier releases. Also ensure that you download Eclipse IDE for Enterprise Java Developers

Windows

https://www3.ntu.edu.sg/home/ehchua/programming/howto/eclipsejava_howto.html

Linux

<https://www.tecmint.com/install-eclipse-ide-in-ubuntu-debian/>

Mac

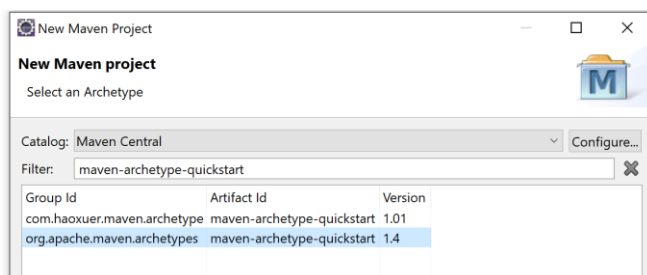
<https://java.tutorials24x7.com/blog/how-to-install-eclipse-for-java-on-mac>

1.6 Setting up a simple Maven project in Eclipse

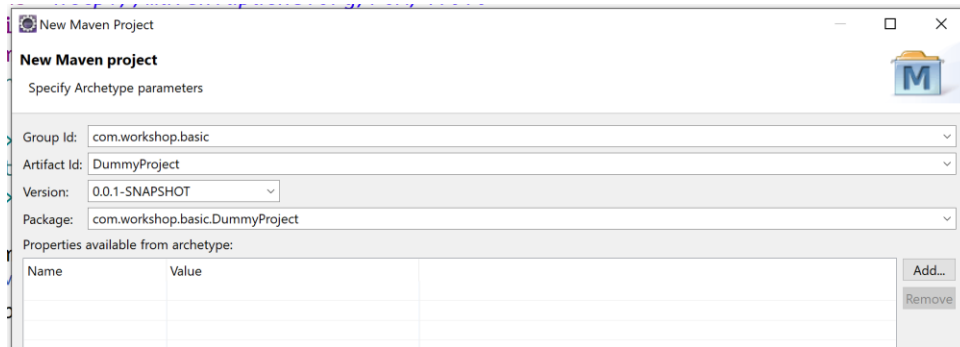
Video link: <https://youtu.be/a2RrlqIHtcQ>

Switch to Java EE perspective

Start with File -> New -> Maven Project. Select Next and type `maven-archetype-quickstart` in the filter. Eclipse may freeze temporarily while it attempts to filter through all the archetypes available at Maven Central. Select the entry with group id: `org.apache.maven.archetypes` and click Next



Enter in the following details and click Finish.



1.7 Installing Tomcat

Video link: <https://youtu.be/6ySwyNeokIE>

<https://tomcat.apache.org/download-90.cgi>

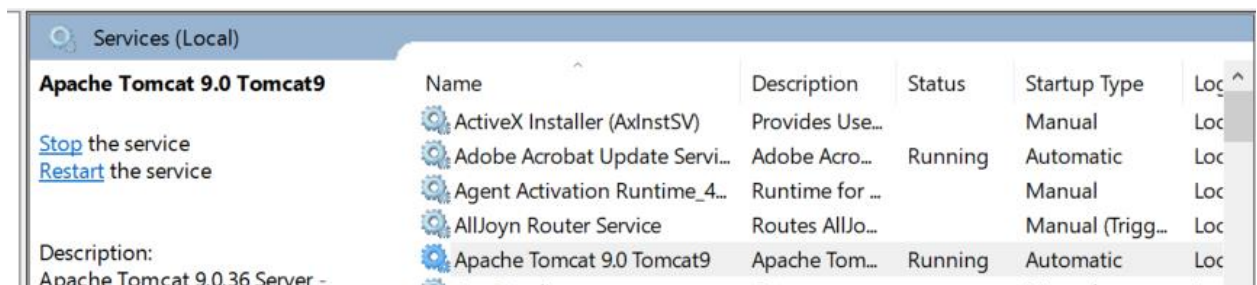
The two main ways to install Tomcat on Windows is either

- through the 32-bit/64-bit Windows installer (apache-tomcat-[version].exe) - which installs Tomcat as a Windows service (this is the approach we will use)
- unzipping the base distribution (apache-tomcat-[version]-windows-x64.zip) into a suitable directory, and starting / stopping Tomcat via the various executables in the `bin` folder

During the installation process using a Windows installer mode:

- In Choose components, select Full. Make sure the Service start up is checked enable Tomcat to be installed as a service during the installation process.
- Keep the default ports in the Configuration window (for the newer version of Tomcat, the default shutdown port is listed as -1 which will cause a problem later when configuring Eclipse to start Tomcat; accept this for now but you will need to change it later as explained in Topic 4..6.1).

Verify the installation is successful by typing `localhost:8080` in a browser tab. You should see the main landing page for Tomcat.



You can bring up the configuration window for Tomcat by right clicking on the Tomcat icon in the bottom right. You can also perform administration through the Services window. Just like any other standard service you can right click to stop it, pause it and restart it.

Windows

<http://tomcat.apache.org/tomcat-9.0-doc/setup.html#Windows>
<https://www.liquidweb.com/kb/installing-tomcat-9-on-windows/>

Linux

<https://www.journaldev.com/39819/install-tomcat-on-linux>
<https://www.digitalocean.com/community/tutorials/how-to-install-apache-tomcat-8-on-ubuntu-16-04>

Mac

<https://wolfpaulus.com/tomcat/>
<https://tonnygaric.com/blog/how-to-install-apache-tomcat-on-macos-10-14-mojave>

1.8 Integrating Tomcat with Eclipse

Video link: <https://youtu.be/N9KpQpkP728>

<https://crunchify.com/step-by-step-guide-to-setup-and-install-apache-tomcat-server-in-eclipse-development-environment-ide/>

1.9 Configuring Tomcat

Video link: <https://youtu.be/-LHZfybJs2g>

Setting up access to the Tomcat manager

<https://clients.javapipe.com/knowledgebase/129/How-to-access-Tomcat-Manager.html>
<https://tomcat.apache.org/tomcat-9.0-doc/manager-howto.html>

Changing default HTTP port on stand-alone Tomcat instance

<https://mkyong.com/tomcat/how-to-change-tomcat-default-port/>
<https://stackoverflow.com/questions/18415578/how-to-change-tomcat-port-number>

Tomcat has a number of default ports:

- Tomcat admin port: 8005
- HTTP/1.1: 8080
- AJP/1.3: 8009

It will not be able to start properly if any other process is bound or listening on those ports. To check for processes that are bound to any of these ports and terminate them, run the following commands in a command prompt in Windows:

```
netstat -aon | findstr port-number
```

TCP	0.0.0.0: <i>port-number</i>	0.0.0.0:0	LISTENING	<i>PID</i>
TCP	[::]: <i>port-number</i>	[::]:0	LISTENING	<i>PID</i>

```
tasklist | findstr PID
```

```
someprogram.exe      PID Console          11      219,532 K
```

```
taskkill /PID PID /F
```

SUCCESS: The process with PID *PID* has been terminated.

Linux

<https://www.tecmint.com/find-out-which-process-listening-on-a-particular-port/>

<https://www.cyberciti.biz/faq/kill-process-in-linux-or-terminate-a-process-in-unix-or-linux-systems/>

1.10 Downloading Spring 5 Jars

Video link: <https://youtu.be/BqEBs3rlsRg>

Select the latest Spring 5 release distribution folder from the link below. This should one of the last folders at the bottom:

<https://repo.spring.io/release/org/springframework/spring/>

Download the zip from here and unzip into a suitable folder. We will be using the JAR files from the `lib` folder in our subsequent labs

2 Section 2: Working with Maven

2.1 Maven overview

<https://maven.apache.org/what-is-maven.html>

<https://maven.apache.org/maven-features.html>

<https://maven.apache.org/glossary.html>

2.2 LAB: Generating and using a JAR

The source code for this lab is found in `demo-jar/changes` folder.

Switch to Java SE perspective.

Create a new Java project: `SimpleFirstProject`

Create a new package: `com.workshop.operations`

In this package, create two classes:

`StringOperations.java`

MainProgram.java

Run MainProgram as normal to verify that it works (Run As -> Java Application).

Select project and Export as a Runnable JAR file.

Select a suitable destination folder and name the JAR as: StringOperations.jar

For the Launch configuration, select MainProgram - SimpleFirstProject.

Click Finish.

In a command prompt or shell terminal, navigate to the destination folder and run the JAR with:

```
java -jar StringOperations.jar
```

Using 7-z to open the JAR, verify that it contains the compiled classes from the project as well as a manifest (MANIFEST.MF) which indicates the main class to run. You can also unzip the JAR to view its contents.

Create another Java project: SimpleSecondProject

Create a new package: com.workshop.user

In this package, create a class: UserProgram.java

Notice that it has a syntax error as the StringOperations class is not on its project build path.

To solve this problem, we need to add the JAR that we generated earlier to its build path.

In SimpleSecondProject, create a new folder lib. Copy and paste StringOperations.jar into this folder.

On the project, select Properties -> Java Build Path

Right click Libraries Tab, select Add JARs. Select the JAR file that you just pasted into lib. Click Apply and Close.

Notice that you can expand the JAR file to see its contents in the Referenced Libraries entry in the Package Explorer.

The syntax error in UserProgram should disappear and you should be able to run it successfully.

In this simple example, StringOperations.jar is a dependency of SimpleSecondProject

2.3 LAB: Using JARs from external libraries (Log4J and JUnit)

The source code for this lab is found in demo-log4j-junit/changes folder.

Switch to Java SE perspective.

Create a new Java project: SimpleLoggingProject

Create a new package: com.workshop.operations

In this package, create a class: BasicLoggingDemo

In the project, create a folder `properties` and create the file `log4j.properties` in it.

Create a new package: `com.workshop.test`

In this package, create two classes:

`TestJUnit`

`TestRunner`

Notice that there are syntax errors again in both classes as the required classes from the Log4J and JUnit library are not on the build class path yet.

In the project, create a new folder `lib`.

Download the binary distribution for Log4j (either zip for Windows or tar.gz for Linux) from this link and unzip into a folder. Use 7-Zip to unzip as this is faster than the built-in Windows zip extractor <https://logging.apache.org/log4j/1.2/download.html>

Copy and paste `log4j-1.2.17.jar` into the `lib` folder. There is a copy of this in the `changes` folder as well if the download is too slow.

Download the JAR files for JUnit from this link below:

<https://github.com/junit-team/junit4/wiki/Download-and-Install>

Click on the links for `junit.jar` and `hamcrest-core.jar` to redirect to the Maven Central Repository search to download them. Copy and paste these two JAR files into the `lib` folder.

There is a copy of these two files in the `changes` folder as well if the download is too slow.

On the project, select Properties -> Java Build Path

Right click Libraries Tab, select Add JARs. Select all the 3 JAR files that you just pasted into `lib`. Click Apply and Close.

Notice that you can expand any of these JAR files to see its contents in the Referenced Libraries entry in the Package Explorer.

To run `BasicLoggingDemo`, we need to place the `log4.properties` file onto the runtime classpath during execution.

Right-click on the project and select Run As -> Run Configurations.

Double click on the Java application entry

In the Name field, type: `Run with properties`

In the Main tab, click on Search for the Main class: entry and select `BasicLoggingDemo`

Go to the Classpath tab and select `SimpleLogging Project`. Click on Advanced button.

Select the Add Folders option and click on OK button.

Browse for the folder `properties` which contains the `log4j.properties` file

Select that folder and click on OK, Apply then Run button.

You should now be able to see the log output from `BasicLoggingDemo`

```
From BasicLoggingDemo
[main] DEBUG com.workshop.operations.BasicLoggingDemo - Hello this
is a debug message
```

```
[main] INFO    com.workshop.operations.BasicLoggingDemo    - Hello  
this is an info message
```

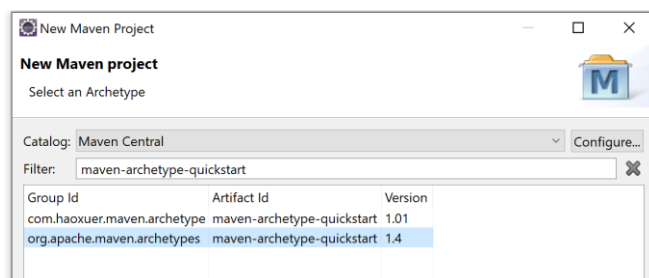
To see the JUnit test in action, go to `TestRunner` and do `Run As -> Java Application`
Try changing the strings being compared in `TestJUnit` to see how an error is flagged.

2.4 LAB: Creating a Maven project using an archetype

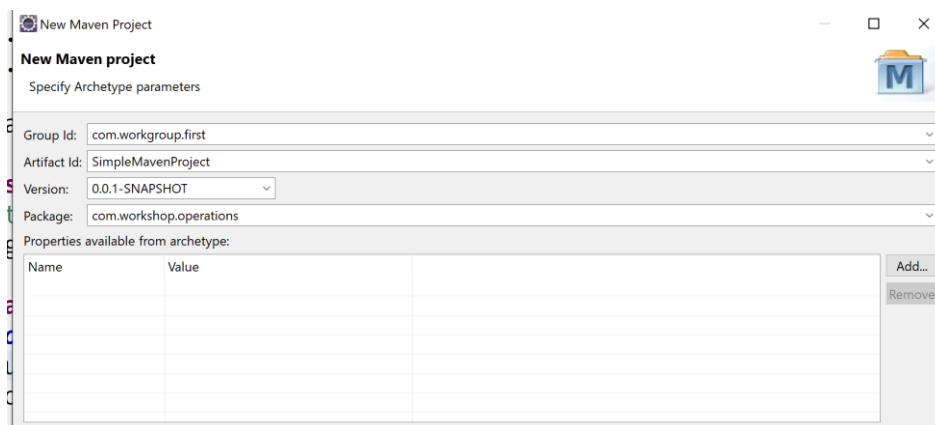
The source code for this lab is found in `demo-maven-basic/changes` folder.

Switch to Java EE perspective

Start with `File -> New -> Maven Project`. Select `Next` and type `maven-archetype-quickstart` in the filter. Eclipse may freeze temporarily while it attempts to filter through all the archetypes available at Maven Central. Select the entry with group id: `org.apache.maven.archetypes` and click `Next`



Enter in the following details and click `Finish`.

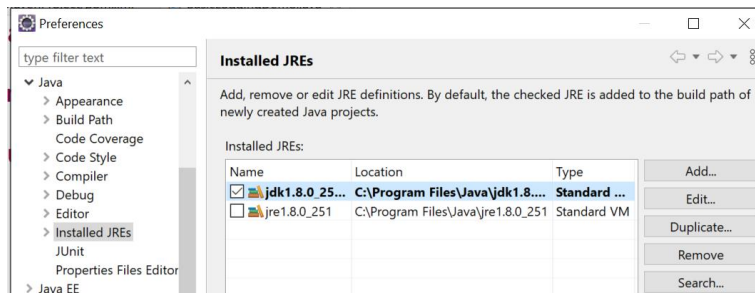


Replace the contents of the `pom.xml` in the project with `pom.xml` from `demo-maven-basic/changes`. Right click on the project, select `Maven -> Update Project`, and click `OK`. You should see the JRE system library entry in the project list update to `JavaSE-1.8`.

> JRE System Library [JavaSE-1.8]

In order for a Maven build to run properly in Eclipse, you will need to use the JDK entry for the installed JRE. To check whether this is already done, go to: Window → Preferences → Java → Installed JREs

The JDK entry should be present and should be selected. If it is present and not selected, select it and click Apply and Close.



If it is not present, click Add. In the JRE Type dialog box, select Standard VM. Click Next.

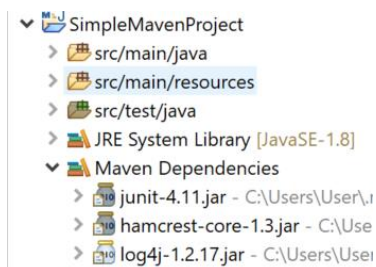
In the JRE Home entry, click the Directory button and navigate and select the installation directory of the JDK, for e.g: C:\Program Files\Java\jdk1.8.0_251

Click Finish, then select the JDK entry and click Apply and Close.

If you do not do this, you will get this particular error message during the Maven build:

```
[ERROR] No compiler is provided in this environment. Perhaps you are  
running on a JRE      rather than a JDK?  
[INFO] 1 error
```

Expand on the Maven dependencies entry in the project. You should be able to see the 3 JAR files that we used in the previous lab:



Right click on project and select New -> Folder. Create a new folder named `resources` in `src/main`

Right click on project and select Properties. Select Java Build Path and click on the Source tab. Select Add Folder. Tick on the checkbox for `src/main/resources` and click ok. Click Apply and Close. You should now see this folder being registered in the entries below the project name (all these entries indicate folders which are on the application build path).

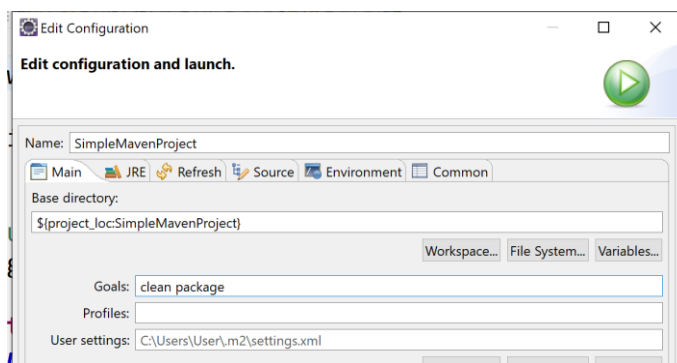
- ▼ SimpleMavenProject
 - > src/main/java
 - > src/main/resources
 - > src/test/java
 - > JRE System Library [JavaSE-1.8]
 - > Maven Dependencies

Copy `BasicLoggingDemo.java` from the previous `SimpleLoggingProject` and paste it into the package `com.workshop.operations` in `src/main/java`

Copy `log4j.properties` from the previous `SimpleLoggingProject` and paste it into `src/main/resources`.

Open `BasicLoggingDemo` in the editor and do Run As -> Java application. Verify that everything works the same as before.

Right click on the project, select Run As -> 3. Maven Build. In the Edit Configuration, type the following as goals: `clean package` and click Run.



Check in the console output that the tests were run (these are JUnit tests by default) and that a JAR has been generated holding the classes from our app.

```
[INFO] -----
[INFO]  T E S T S
[INFO] -----
[INFO] Running com.workshop.operations.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.013 s - in
com.workshop.operations.AppTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-jar-plugin:3.0.2:jar (default-jar) @ SimpleMavenProject ---
[INFO] Building jar: G:\code\ee-eclipse\SimpleMavenProject\target\SimpleMavenProject-0.0.1-
SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.182 s
[INFO] Finished at: 2020-11-24T10:10:22+08:00
[INFO] -----
```

If you run this project again as a Maven build, you will be prompted with an existing launch configuration to use. If you wish to run the Maven build with completely new goals, do Run As -> 4. Maven Build instead.

Right click on the project and select Refresh. Expand the `target` folder. Notice that it contains a JAR file in it (`SimpleMavenProject-0.0.1-SNAPSHOT.jar`).

In a separate File Explorer window, navigate to the project directory within your Eclipse workspace. Notice that the `target` subfolder actually contains two additional folders (`classes` and `test-classes`) that are not shown in the package explorer view in Eclipse. These two folders contain the compiled code from `src/main/java` and `src/test/java`

Copy the single JAR file out to another folder and check its contents with 7-Zip. Notice that

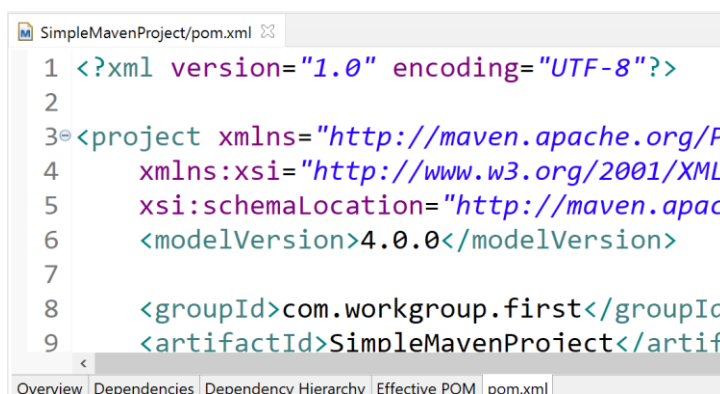
- The class files for the source code in the `src/main/java` directory are included here
- The class files for the source code in the `src/test/java` directory are NOT included here
- The `log4j.properties` file which was originally in `src/main/resources` is now in the root folder of the JAR file (i.e. corresponding to the runtime class path)

Note that this is NOT a standalone, executable JAR that you can directly run with `java -jar`. Instead, it is meant to be used as a dependency JAR for another application that needs it.

2.5 LAB: Maven features

2.5.1 Lab practice

Click on the tabs to see the 5 different representations of the POM



Check the `<dependencies>` section of the `pom.xml` to see where you specified the dependencies for Log4J and JUnit:

```

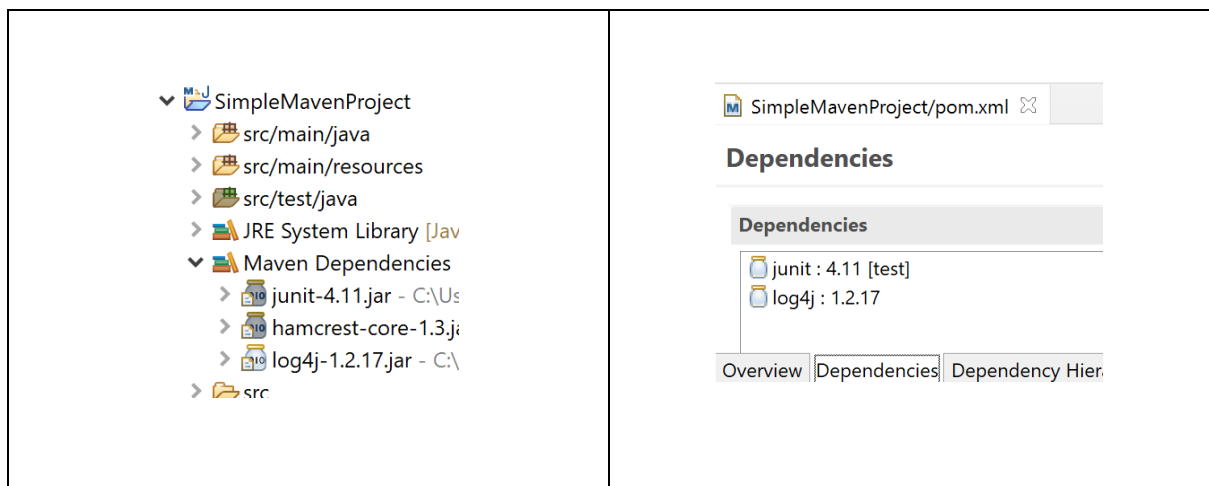
<dependencies>

  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
  </dependency>
</dependencies>

```

Notice that you can view the Maven dependency JARs that you have specified in your POM in the view as well as the package explorer:



Click on the effective POM to see how it is a merging of the Super POM and your project POM
<https://maven.apache.org/ref/3.6.3/maven-model-builder/super-pom.html>

Go to the Maven Central Repository Search site : **search.maven.org**

Try to see whether you can hunt down the GAV coordinates for the two dependencies that you have included in your project: JUnit and Log4J

Type: JUnit in the search box. Notice that there are many groupIDs (representing different organizations or different depts in the same organization) for the same artifactID. The actual artifact that we are looking for is circled in red. Click on the Latest Version link to go there.

junit		Updated 16-Nov-2020
org.robolectric	Artifact ID: junit	
	Latest Version: 4.5-alpha-3	
au.com.dius.pact.provider		Updated 09-Nov-2020
	Artifact ID: junit	
	Latest Version: 4.2.0-beta.1	
au.com.dius.pact.consumer		Updated 09-Nov-2020
	Artifact ID: junit	
	Latest Version: 4.2.0-beta.1	
com.buschmais.jqassistant.plugin		Updated 29-Oct-2020
	Artifact ID: junit	
	Latest Version: 1.9.0-RC1	
junit		Updated 11-Oct-2020
	Artifact ID: junit	
	Latest Version: 4.13.1	

From this area, you can get the GAV coordinates for this artifact to paste in your project POM, download the JARs and also view the project POM.

If you look through the project POM for JUnit, you will see that it has a dependency of its own (Hamcrest).

```
<dependencies>
  <dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest-core</artifactId>
    <version>${hamcrestVersion}</version>
  </dependency>

  <dependency>
    <groupId>org.hamcrest</groupId>
    <artifactId>hamcrest-library</artifactId>
    <version>${hamcrestVersion}</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

However, in your own POM, you don't have to specify this dependency explicitly. Instead Maven is intelligent enough to figure out this additional dependency on its own (called transitive dependency) when it downloads the POM for JUnit. It will then additionally fetch the Hamcrest dependency as well. You can verify this in the Dependency Hierarchy view perspective of the POM.xml

SimpleMavenProject/pom.xml

Dependency Hierarchy [test]

Dependency Hierarchy

- ▼ junit : 4.11 [test]
 - hamcrest-core : 1.3 [test]
 - log4j : 1.2.17 [compile]

Repeat this for log4j. Note that the artifact we are actually looking for is the older version of log4j and not the newer version log4j 2.

log4j		Updated 26-May-2012
Artifact ID	log4j	
Latest Version	1.2.17	

If you want a more detailed categorization of your search items, use the Classic Search menu instead:

<https://search.maven.org/classic/>

Click on Advanced Search.

One of the more useful things to search for is based on classname. So for e.g. if you have a piece of source code that you copied from a website which uses external Java libraries, but you have no idea which library it uses (and therefore cannot search for it properly), you can search on the imported classes used in that source code. You may get a lot of results which you can browse through to attempt to find the correct organization and artifact.

In addition to the Maven Central Repository, most organizations also host publicly accessible Maven repositories which may contain dependencies that are not available in the Maven Central repository. The site below indexes all these additional repositories

<https://mvnrepository.com>

You can also do a search here for the artifact that you are looking for here as well, and the search results are generally more easy to understand than at **search.maven.org**

Also the Popular Categories list on the left hand side allows you to quickly determine the most popular projects for a particular type of usage category. This can be very helpful in determining which particular Java framework/library that you want to use in the event there are several candidates possible.

The list of all top indexed repositories for the different organizations / companies is available at:

<https://mvnrepository.com/repos>

At the top is the Maven Central Repository itself. Notice that the Spring project itself has two public repositories (Spring Plugins and Spring Lib M) which have an extremely high number of indexed JARs. This goes to show how active the Spring framework community is !

Another way to determine the Maven GAV coordinates for a dependency that you need to use in your project is to search through the home website dedicated to that project (most major Java libraries have a dedicated home website)

For e.g. if we want to use Hibernate (a popular JPA ORM provider), we can find its Maven dependency here

<https://hibernate.org/orm/documentation/getting-started/>

If we want to use Logback (another popular Java logging framework), we can find its Maven dependency here

<http://logback.qos.ch/setup.html>

If you can't find this on the project home website, you can try doing a search on:

xxxx maven dependencies

in google search, where xxx is the name of the artifact/project you are looking for. Usually the first 5 – 10 links will either navigate you to <https://mvnrepository.com/>, from which you can then explore further. Or it might redirect you to another website which lists the dependencies that you are looking for.

For e.g. if I type:

spring maven dependencies

I get links to the following sites that can help me locate the Maven dependency coordinates for the specific artifact I am interested in:

<https://mvnrepository.com/artifact/org.springframework/spring-core>

Alternatively this article, gives you information on the dependencies for all the major components of the Spring framework

<https://www.baeldung.com/spring-with-maven>

The central Maven repository where the actual dependency JARs for the various project artifacts that you need to download and use in your project at is located at this link:

<https://repo.maven.apache.org/maven2>

Go here and navigate down through any one of the project links to locate the various JAR files and other project artifacts (source code, documentation, project POM, etc)

Go to the default location for the local Maven repository cache on your machine:

Windows: C:\Users\<User_Name>\.m2\repository

Linux: /home/<User_Name>/.m2/repository

Mac: /Users/<user_name>/.m2/repository

Check that you already have a folder structure corresponding to the GAV coordinates of the two dependencies that you downloaded, for e.g:

C:\Users\<User_Name>\.m2\repository\junit\junit\4.11

C:\Users\<User_Name>\.m2\repository\log4j\log4j\1.2.17

Notice that the folder contains the JAR, source code as well as project POM. The next time your run a build that references these dependencies, Maven will first check in the local repo cache and use these JARs first. It will only check the central Maven repository if it can't find it there.

Try and delete any of these folders and then do a Maven -> Update Project on SimpleMavenProject. You will see Maven download all of the relevant dependency artifacts and create the folder anew to place them within it.

So far we have been running Maven from inside Eclipse via Eclipse's built in Maven plugin. As an alternative, we can also run Maven from the command line if we have Maven installed.

Open a command prompt / bash shell and navigate to the SimpleMavenProject folder in your Eclipse workspace folder and type:

```
mvn clean package
```

You will see the same console output as you did when you ran these goals using Eclipse's configuration dialog box.

We can also generate our own Maven archetype project from the command line using the `archetype:generate` plugin goal.

In your main Eclipse workspace directory, type the following command to generate a new Maven archetype project:

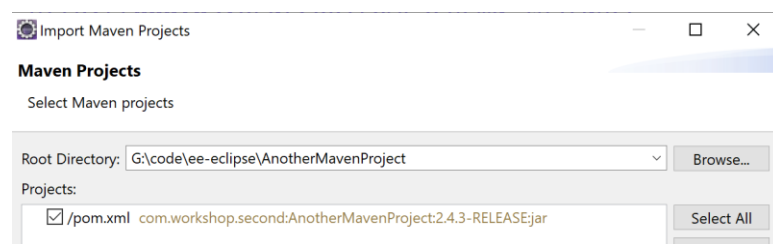
```
mvn archetype:generate -DgroupId=com.workshop.second -
DartifactId=AnotherMavenProject -Dversion=2.4.3-RELEASE -
DarchetypeArtifactId=maven-archetype-quickstart -Dpackage=jar -
DinteractiveMode=false
```

This will create a new Maven project folder with the name `AnotherMavenProject`

You can import this project directly into Eclipse to work on it inside there.

Do File -> Import -> Maven -> Existing Maven Projects

Click browse and look for the `AnotherMavenProject` folder, and select it and click Finish. Eclipse detects that this folder contains a `pom.xml`, and is therefore a Maven project by default.



2.5.2 Documentation on Maven features

Archetypes

<https://maven.apache.org/guides/introduction/introduction-to-archetypes.html>

Directory layout

<https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>

Project POM

<https://maven.apache.org/guides/introduction/introduction-to-the-pom.html#what-is-a-pom>
<https://maven.apache.org/guides/introduction/introduction-to-the-pom.html#minimal-pom>
<https://maven.apache.org/pom.html#what-is-the-pom>
<https://maven.apache.org/pom.html#quick-overview>

Maven / GAV coordinates and packaging

<https://maven.apache.org/pom.html#maven-coordinates>
<https://maven.apache.org/pom.html#packaging>

Dependency list

<https://maven.apache.org/pom.html#dependencies>
<https://maven.apache.org/pom.html#exclusions>

Transitive dependencies

<https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html#transitive-dependencies>

Dependency scopes

<https://www.baeldung.com/maven-dependency-scopes>

Super POM and effective POM

<https://maven.apache.org/pom.html#the-super-pom>
<https://maven.apache.org/ref/3.6.3/maven-model-builder/super-pom.html>

Repositories

<https://maven.apache.org/guides/introduction/introduction-to-repositories.html>
<https://www.baeldung.com/maven-local-repository>

Maven life cycle

<https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#build-lifecycle-basics>
<https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#lifecycle-reference>
<https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#usual-command-line-calls>

Plugins and goals

<https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#plugins>

<https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#a-build-phase-is-made-up-of-plugin-goals>

<https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#setting-up-your-project-to-use-the-build-lifecycle>

<https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#built-in-lifecycle-bindings>

Details for configuring any particular existing plugin can be found by clicking on the link for that specific plugin

<https://maven.apache.org/plugins/index.html>

3 Section 3: Spring Core

3.1 Spring framework overview

https://www.tutorialspoint.com/spring/spring_overview.htm

<https://www.journaldev.com/20714/spring-5>

<https://www.baeldung.com/spring-why-to-choose>

<https://www.journaldev.com/16922/spring-framework>

https://www.tutorialspoint.com/spring/spring_architecture.htm

<https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html>

3.2 LAB: Spring concepts: IoC and DI

3.2.1 Lab Practice

The source code for this lab is found in `Basic-Concepts/changes` folder.

Switch to Java Perspective.

Create a new Java project: `DemoBasicConcepts`

Create a new package `com.workshop.original`

Create these 3 original Java files in this package.

SwimmingExercise.java
JoggingExercise.java
Student.java

In Student.java, right click and select Run as -> Java Application

What happens if SwimmingExercise wants to change its implementation of doSwimming ?

SwimmingExercise-v2.java
Student-v2.java

What happens if Student wants to do Jogging instead?

Student-v3.java

Create a new package com.workshop.useinterface

Create these 4 original Java files in this package.

Exercise.java
JoggingExercise.java
Student.java
SwimmingExercise.java

What happens if SwimmingExercise wants to change its implementation of doSwimming ?

SwimmingExercise-v2.java

What happens if Student wants to do Jogging instead?

Student-v2.java

Demonstrating Inversion of control (IoC)

Student-v3.java

3.2.2 IoC and DI Theory

<https://dzone.com/articles/programming-to-an-interface>

<https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>

<https://dzone.com/articles/ioc-vs-di>

https://www.tutorialspoint.com/spring/spring_bean_definition.htm

https://www.tutorialspoint.com/spring/spring_ioc_containers.htm

<https://howtodoinjava.com/spring-core/different-spring-ioc-containers/>

3.3 LAB: XML-based configuration basics

3.3.1 Lab Practice

The primary issue with creating an initial Spring project is to ensure that the relevant Spring module JARs are on the build path of our application. There are two main ways to ensure this:

- 1) Create a basic Java project. Download and include relevant Spring module JARs on the build path of our project
- 2) Create a basic Maven project. Specify dependencies for relevant Spring modules into POM.xml

The source code for both approaches is found in `XML-Config-Basics/changes` folder.

We will start with the first approach.

Basic Java Project

Switch to Java SE perspective.

Create a new Java project: `XMLConfigWithJARs`

In the `src` folder, create a file `beansDefinition.xml`

Create a new package: `com.workshop.configxml`

In this package, create 5 classes:

```
SwimmingExercise.java
JoggingExercise.java
CyclingExercise.java
Exercise.java
XMLConfigBasicMainApp.java
```

Notice that there is a syntax error registered on `XMLConfigBasicMainApp` as the relevant Spring module classes are not on the build class path yet.

In the project, create a new folder `lib`.

If you haven't already done so from the Section on Setting up the Development Environment, download the latest Spring release distribution from the link below and unzip into a suitable folder.

<https://repo.spring.io/release/org/springframework/spring>

Copy and paste the following JAR files from the `lib` folder into the `lib` folder of your project.

- `spring-aop-x.y.z.jar`
- `spring-beans-x.y.z.jar`

- spring-context-x.y.z.jar
- spring-core-x.y.z.jar
- spring-jcl-x.y.z.jar
- spring-expression-x.y.z.jar

On the project, select Properties -> Java Build Path

Right click Libraries Tab, select Add JARs. Select all the 6 JAR files that you just pasted into lib. Click Apply and Close.

Notice that you can expand any of these JAR files to see its contents in the Referenced Libraries entry in the Package Explorer.

Right click on XMLConfigBasicMainApp and select Run As -> Java Application. Verify that the correct bean is created and its console log displayed in the Console view.

Change the contents of beansDefinition.xml to reflect different classes for favoriteExercise, for e.g.

```
<bean id="favouriteExercise"
      class="com.workshop.configxml.CyclingExercise">
</bean>
```

```
<bean id="favouriteExercise"
      class="com.workshop.configxml.JoggingExercise">
</bean>
```

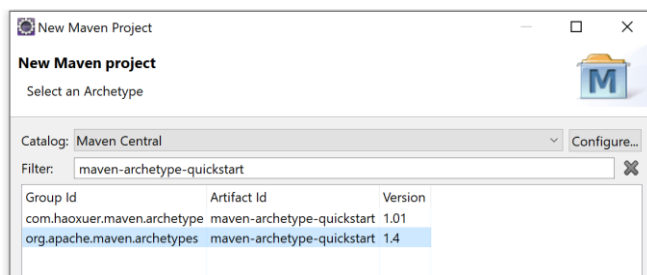
And again run XMLConfigBasicMainApp to verify that the correct bean is instantiated based on the console output to the screen.

We now continue with the second approach

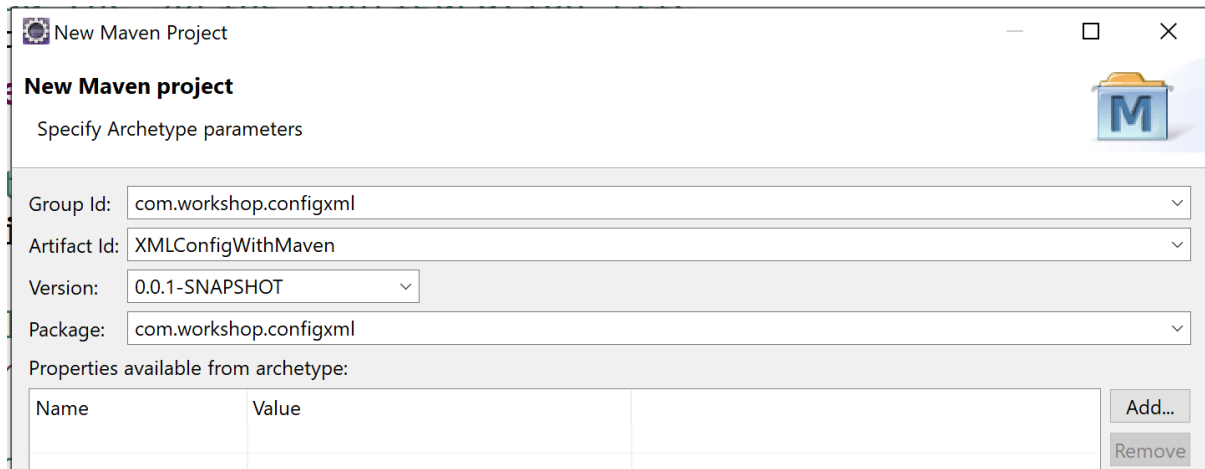
Basic Maven Project

Switch to Java EE perspective.

Start with File -> New -> Maven Project. Select Next and type maven-archetype-quickstart in the filter. Eclipse may freeze temporarily while it attempts to filter through all the archetypes available at Maven Central. Select the entry with group id: org.apache.maven.archetypes and click Next












Enter in the following details and click Finish.



Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select Maven -> Update Project, and click OK. You should see the JRE system library entry in the project list update to JavaSE-1.8.






>  JRE System Library [JavaSE-1.8]

You should also the following JARs automatically added as Maven dependencies

- ▼  Maven Dependencies
 - >  junit-4.11.jar - C:\Users\User\.m2\repository\junit\junit\4.11
 - >  hamcrest-core-1.3.jar - C:\Users\User\.m2\repository\org\hamcrest\hamcrest-core\1.3
 - >  spring-context-5.3.1.jar - C:\Users\User\.m2\repository\org\springframework\spring-context\5.3.1
 - >  spring-aop-5.3.1.jar - C:\Users\User\.m2\repository\org\springframework\spring-aop\5.3.1
 - >  spring-beans-5.3.1.jar - C:\Users\User\.m2\repository\org\springframework\spring-beans\5.3.1
 - >  spring-core-5.3.1.jar - C:\Users\User\.m2\repository\org\springframework\spring-core\5.3.1
 - >  spring-jcl-5.3.1.jar - C:\Users\User\.m2\repository\org\springframework\spring-jcl\5.3.1
 - >  spring-expression-5.3.1.jar - C:\Users\User\.m2\repository\org\springframework\spring-expression\5.3.1

Right click on project and select New -> Folder. Create a new folder named `resources` in `src/main`

Right click on project and select Properties. Select Java Build Path and click on the Source tab. Select Add Folder. Tick on the checkbox for `src/main/resources` and click ok. Click Apply and Close. You should now see this folder being registered in the entries below the project name (all these entries indicate folders which are on the application build path).

- ▼  XMLConfigWithMaven
 - >  src/main/java
 - >  src/main/resources
 - >  src/test/java
 - >  JRE System Library [JavaSE-1.8]

In `src/main/java`, there should already be a package: `com.workshop.configxml`

Copy all the previous 5 classes from the previous project `XMLConfigWithJars/src` and place them in the same package `com.workshop.configxml` in this Maven project.

```
SwimmingExercise.java
JoggingExercise.java
CyclingExercise.java
Exercise.java
XMLConfigBasicMainApp.java
```

Copy `beansDefinition.xml` the previous project `XMLConfigWithJars/src` and paste it into `src/main/resources` in this Maven project

Open and right click on `XMLConfigBasicMainApp` and select `Run As -> Java Application`. Verify that the correct bean is created and its console log displayed in the Console view.

Change the contents of `beansDefinition.xml` to reflect different classes for `favoriteExercise`, for e.g.

```
<bean id="favouriteExercise"
      class="com.workshop.configxml.CyclingExercise">
</bean>
```

```
<bean id="favouriteExercise"
      class="com.workshop.configxml.JoggingExercise">
</bean>
```

And again run `XMLConfigBasicMainApp` to verify that the correct bean is instantiated based on the console output to the screen.

We can configure the IoC container `ClassPathXmlApplicationContext` to read from more than one XML configuration file at once in order to obtain bean definitions. Let's demonstrate this:

Make the following changes:

- Add `backupDefinition.xml` to `src/main/resources`
- Make the change `XMLConfigBasicMainApp-v2.java`

3.3.2 Theory

<https://howtodoinjava.com/spring5/core/spring-bean-xml-config/>

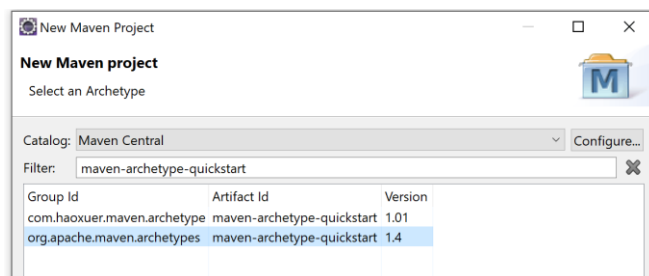
<https://www.baeldung.com/spring-classpathxmlapplicationcontext>

3.4 LAB: XML-based constructor DI

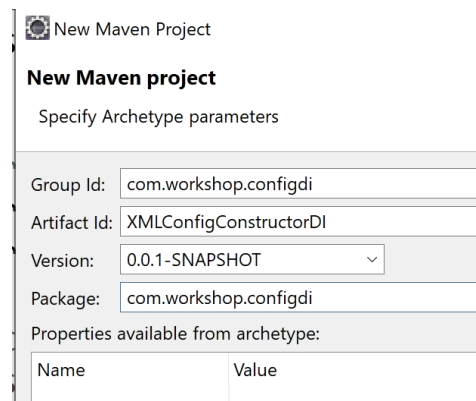
The source code for this lab is found in `XML-Config-Constructor-DI/changes` folder.

Switch to Java EE perspective.

Start with `File -> New -> Maven Project`. Select `Next` and type `maven-archetype-quickstart` in the filter. Eclipse may freeze temporarily while it attempts to filter through all the archetypes available at Maven Central. Select the entry with group id: `org.apache.maven.archetypes` and click `Next`



Enter in the following details and click `Finish`.



Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select `Maven -> Update Project`, and click `OK`.

Right click on project and select `New -> Folder`. Create a new folder named `resources` in `src/main`

Right click on project and select `Properties`. Select `Java Build Path` and click on the `Source` tab. Select `Add Folder`. Tick on the checkbox for `src/main/resources` and click `ok`. Click `Apply` and `Close`.

There are 2 approaches to specify constructor injection

- Nest the child bean within the parent bean as a child element of the of the parent bean using the `<constructor-arg>` element
- Define the child bean separately and then reference it via the `ref` attribute of the `<constructor-arg>` element

3.4.1 Approach 1: Nest child bean approach

Place `beansDefinition.xml` into `src/main/resources`

Create the following classes in `com.workshop.configdi`

Create the following classes:

```
SwimmingExercise.java
JoggingExercise.java
CyclingExercise.java
Exercise.java
Student.java
HighSchoolStudent.java
CollegeStudent.java
XMLConfigConstructorDIMainApp.java
```

Open and right click on `XMLConfigConstructorDIMainApp` and select `Run As -> Java Application`. Verify that the correct parent and child bean have been created and their output logged to the console correctly.

3.4.2 Approach 2: Reference child bean separately

Make the following changes:

```
beansDefinition-v2.xml
XMLConfigConstructorDIMainApp-v2.java
```

Right click on `XMLConfigConstructorDIMainApp` and select `Run As -> Java Application`. Verify that the correct parent and child bean have been created and their output logged to the console correctly.

3.5 LAB: XML-based setter DI

The source code for this lab is found in `XML-Config-Setter-DI/changes` folder.

We can create a Maven project from scratch, or we can make a copy from an existing one. We will make a copy of the previous lab project: `XMLConfigConstructorDI`

In the Project Explorer, right click on `XMLConfigConstructorDI`, select `Copy` and then right click in any empty space in the Explorer and select `Paste`.

For the new project name, type: `XMLConfigSetterDI`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select `Maven -> Update Project`, and click `OK`.

Delete all the packages and files in `src/main/java` and `src/main/resources`. We will start populating the project from scratch.

There are 2 approaches to specify setter injection

- Nest the child bean within the parent bean as a child element of the of the parent bean using the **<property>** element
- Define the child bean separately and then reference it via the **ref** attribute of the **<property>** element

3.5.1 Approach 1: Nest child bean approach

Place `beansDefinition.xml` into `src/main/resources`

Create the package `com.workshop.setterdi` and place these classes in it:

```
SwimmingExercise.java
JoggingExercise.java
CyclingExercise.java
Exercise.java
Student.java
HighSchoolStudent.java
CollegeStudent.java
XMLConfigSetterDIMainApp.java
```

Open and right click on `XMLConfigSetterDIMainApp` and select `Run As -> Java Application`. Verify that the correct parent and child bean have been created and their output logged to the console correctly.

3.5.2 Approach 2: Reference child bean separately

Make the following changes:

```
beansDefinition-v2.xml
XMLConfigSetterDIMainApp-v2.java
```

Right click on `XMLConfigSetterDIMainApp` and select `Run As -> Java Application`. Verify that the correct parent and child bean have been created and their output logged to the console correctly.

3.6 LAB: XML-based setter DI using literal values

The source code for this lab is found in `XML-Config-Setter-LiteralValues-DI/changes` folder.

We can create a Maven project from scratch, or we can make a copy from any of the existing Maven projects.

Choose any previous Maven lab project to make a copy from, for e.g.: `XMLConfigConstructorDI`

In the Project Explorer, right click on `XMLConfigConstructorDI`, select `Copy` and then right click in any empty space in the Explorer and select `Paste`.

For the new project name, type: `XMLConfigSetterLiteralValues`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select Maven -> Update Project, and click OK.

Delete all the packages and files in `src/main/java` and `src/main/resources`. We will start populating the project from scratch.

There are 2 approaches to specify setter injection with literal values

- Specify literal values directly in XML configuration file
- Load values from an external property file

3.6.1 Approach 1: Specify literal values directly in XML configuration file

Place `beansDefinition.xml` into `src/main/resources`

Create the following classes in `com.workshop.literal`

```
SwimmingExercise.java
JoggingExercise.java
CyclingExercise.java
Exercise.java
Student.java
HighSchoolStudent.java
CollegeStudent.java
XMLConfigSetterLiteralValuesMainApp.java
```

Open and right click on `XMLConfigSetterLiteralValuesMainApp` and select Run As -> Java Application. Verify that the correct parent and child bean have been created and their output logged to the console correctly.

3.6.2 Approach 2: Load values from an external property file

Place `highSchool.properties` into `src/main/resources`

Make the following changes:

```
beansDefinition-v2.xml
XMLConfigSetterLiteralValuesMainApp-v2.java
```

Open and right click on `XMLConfigSetterLiteralValuesMainApp` and select Run As -> Java Application. Verify that the correct parent and child bean have been created and their output logged to the console correctly.

3.7 LAB: XML-based setter DI - injecting collections

The source code for this lab is found in `XML-Config-Collections-DI/changes` folder.

We can create a Maven project from scratch, or we can make a copy from any of the existing Maven projects.

Choose any previous Maven lab project to make a copy from, for e.g.: `XMLConfigConstructorDI`

In the Project Explorer, right click on `XMLConfigConstructorDI`, select **Copy** and then right click in any empty space in the Explorer and select **Paste**.

For the new project name, type: `XMLConfigCollectionsDI`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select **Maven -> Update Project**, and click **OK**.

Delete all the packages and files in `src/main/java` and `src/main/resources`. We will start populating the project from scratch.

We can either inject collections of literal values or collections of beans.

3.7.1 Injecting collections of basic values

Place `beansDefinition.xml` into `src/main/resources`

Create the following classes in `com.workshop.collections`

```
Exercise.java
Student.java
CollegeStudent.java
XMLConfigCollectionsDIMainApp.java
DemoCollectionsUse.java
```

Open and right click on `DemoCollectionsUse` and select **Run As -> Java Application** for a basic example of storing and iterating over a `List` and a `Map`.

Open and right click on `XMLConfigCollectionsDIMainApp` and select **Run As -> Java Application**. Verify that the contents of the `List` and `Map` inside `CollegeStudent` have been initialized correctly from the XML configuration and are output to the console.

3.7.2 Injecting collections of beans

Make the following changes:

```
beansDefinition-v2.xml
XMLConfigCollectionsDIMainApp-v2.java
```

```
HighSchoolStudent.java
SwimmingExercise.java
JoggingExercise.java
CyclingExercise.java
```

Open and right click on `XMLConfigCollectionsDIMainApp` and select `Run As -> Java Application`. Verify that the contents of the List inside `HighSchoolStudent` have been initialized correctly from the XML configuration and are output to the console.

3.8 LAB: XML-based autowiring

The source code for this lab is found in `XML-Config-Autowiring/changes` folder.

We can create a Maven project from scratch, or we can make a copy from any of the existing Maven projects.

Choose any previous Maven lab project to make a copy from, for e.g.: `XMLConfigConstructorDI`

In the Project Explorer, right click on `XMLConfigConstructorDI`, select `Copy` and then right click in any empty space in the Explorer and select `Paste`.

For the new project name, type: `XMLConfigAutowiring`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select `Maven -> Update Project`, and click `OK`.

Delete all the packages and files in `src/main/java` and `src/main/resources`. We will start populating the project from scratch.

There are 3 approaches to performing autowiring by XML configuration

- using `byName`
- using `byType`
- using `constructor`

3.8.1 Approach 1: Autowiring using `byName`

Place `beansDefinition.xml` into `src/main/resources`

Create the following classes in `com.workshop.autowiring`

```
SwimmingExercise.java
Exercise.java
Student.java
CollegeStudent.java
XMLConfigAutowiringMainApp.java
```

Open and right click on `XMLConfigAutowiringMainApp` and select `Run As -> Java Application`. Verify that the correct parent and child bean have been created and their output logged to the console correctly.

3.8.2 Approach 2: Autowiring using `byType`

Create the following changes:

```
HighSchoolStudent.java
XMLConfigAutowiringMainApp-v2.java
beansDefinition-v2.xml
```

Open and right click on `XMLConfigAutowiringMainApp` and select `Run As -> Java Application`. Verify that the correct parent and child bean have been created and their output logged to the console correctly.

3.8.3 Approach 3: Autowiring using constructor

Create the following changes:

```
HighSchoolStudent-v2.java
beansDefinition-v3.xml
```

Open and right click on `XMLConfigAutowiringMainApp` and select `Run As -> Java Application`. Verify that the correct parent and child bean have been created and their output logged to the console correctly.

3.9 LAB: Annotation-based configuration basics

The source code for this lab is found in `Annotation-Config-Basics/changes` folder.

We can create a Maven project from scratch, or we can make a copy from any of the existing Maven projects.

Choose any previous Maven lab project to make a copy from, for e.g.: `XMLConfigConstructorDI`

In the Project Explorer, right click on `XMLConfigConstructorDI`, select `Copy` and then right click in any empty space in the Explorer and select `Paste`.

For the new project name, type: `AnnotationConfigBasics`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select `Maven -> Update Project`, and click `OK`.

Delete all the packages and files in `src/main/java` and `src/main/resources`. We will start populating the project from scratch.

3.9.1 Using explicit component name

Place `beansDefinition.xml` into `src/main/resources`

Create the following classes in `com.workshop.annotation`

```
SwimmingExercise.java
```

```
Exercise.java
AnnotationConfigBasicMainApp.java
```

Open and right click on `AnnotationConfigBasicMainApp` and select `Run As -> Java Application`. Verify that the correct bean has been created and its output logged to the console correctly.

3.9.2 Using default component name

Create the following changes:

```
SwimmingExercise-v2.java
AnnotationConfigBasicMainApp-v2.java
```

Open and right click on `AnnotationConfigBasicMainApp` and select `Run As -> Java Application`. Verify that the correct bean has been created and its output logged to the console correctly.

3.10 LAB: Annotation-based DI

The source code for this lab is found in `Annotation-Config-DI/changes` folder.

We can create a Maven project from scratch, or we can make a copy from any of the existing Maven projects.

Choose any previous Maven lab project to make a copy from, for e.g.: `XMLConfigConstructorDI`

In the Project Explorer, right click on `XMLConfigConstructorDI`, select `Copy` and then right click in any empty space in the Explorer and select `Paste`.

For the new project name, type: `AnnotationConfigDI`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select `Maven -> Update Project`, and click `OK`.

Delete all the packages and files in `src/main/java` and `src/main/resources`. We will start populating the project from scratch.

3.10.1 Constructor injection

Place `beansDefinition.xml` into `src/main/resources`

Create the following classes in `com.workshop.annotation`

```
SwimmingExercise.java
Exercise.java
CollegeStudent.java
Student.java
AnnotationConfigDIMainApp.java
```

Open and right click on `AnnotationConfigDIMainApp` and select `Run As -> Java Application`. Verify that the correct bean has been created and its output logged to the console correctly.

3.10.2 Setter / method injection

Make the following changes:

`CollegeStudent-v2.java`

Open and right click on `AnnotationConfigDIMainApp` and select `Run As -> Java Application`. Verify that the correct bean has been created and its output logged to the console correctly.

To demonstrate method injection, make the following changes:

`CollegeStudent-v3.java`

Open and right click on `AnnotationConfigDIMainApp` and select `Run As -> Java Application`. There are 3 methods (including the setter) which have been marked with `@Autowired`. All these methods are called in a random sequence after the default constructor is executed. Try running the application multiple times to confirm this.

3.10.3 Field injection

Make the following changes:

`CollegeStudent-v4.java`

Open and right click on `AnnotationConfigDIMainApp` and select `Run As -> Java Application`. Verify that the correct bean has been created and its output logged to the console correctly.

3.10.4 Problems with multiple candidate beans

Make the following changes:

`CyclingExercise.java`
`JoggingExercise.java`

Open and right click on `AnnotationConfigDIMainApp` and select `Run As -> Java Application`. Notice the exception regarding multiple matching beans (double click on the Console view to enlarge it for a better view)

```
org.springframework.context.support.AbstractApplicationContext refresh
WARNING: Exception encountered during context initialization - cancelling refresh attempt:
org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name
'collegeStudent': Unsatisfied dependency expressed through field 'myExercise'; nested exception is
org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type
'com.workshop.annotation.Exercise' available: expected single matching bean but found 3:
cyclingExercise, joggingExercise, swimmingExercise
```

3.10.5 Using autowiring by name to select between multiple candidate beans

Make the following changes:

CollegeStudent-v5.java

Open and right click on AnnotationConfigDIMainApp and select Run As -> Java Application. Verify that the correct bean has been created and its output logged to the console correctly.

3.10.6 Using @Qualifier to select between multiple candidate beans

Make the following changes:

CollegeStudent-v6.java

Open and right click on AnnotationConfigDIMainApp and select Run As -> Java Application. Notice that the bean is now instantiated with the correct dependency due to the use of the @Qualifier annotation.

When working with constructor injection (as opposed to field injection), @Qualifier must be applied to the specific arguments in the constructor method signature.

Make the following changes:

CollegeStudent-v7.java

Open and right click on AnnotationConfigDIMainApp and select Run As -> Java Application. Notice that the bean is now instantiated with the correct dependency due to the use of the @Qualifier annotation.

3.10.7 Using @Primary to give higher preference to a bean

Make the following changes:

CollegeStudent-v8.java
CyclingExercise-v2.java

Open and right click on AnnotationConfigDIMainApp and select Run As -> Java Application. Notice that the bean is now instantiated with the correct dependency due to the use of the @Primary annotation.

3.10.8 Injecting values from a properties file with @Value

Place highSchool.properties into src/main/resources

Make the following changes:

HighSchoolStudent.java
AnnotationConfigDIMainApp-v2.java
beansDefinition-v2.xml

Open and right click on `AnnotationConfigDIMainApp` and select `Run As -> Java Application`. Notice that the correct values are read for `HighSchoolStudent`'s fields.

3.10.9 Marking optional dependencies with `@Autowired(required = false)`

Make the following changes:

Study.java
HighSchoolStudent-v2.java

Open and right click on `AnnotationConfigDIMainApp` and select `Run As -> Java Application`. The application runs fine even if there is no implementation available for the `myStudy` dependency in `HighSchoolStudent`, as this is marked as optional. In `HighSchoolStudent`, change

```
@Autowired(required = false)
private Study myStudy;
```

to

```
@Autowired
private Study myStudy;
```

and this time attempting to run the application results in an error, as the IoC container will now throw an exception if it is not able to find an implementation for the given dependency.

3.11 LAB: Java-based configuration

The source code for this lab is found in `Java-Config-DI/changes` folder.

We can create a Maven project from scratch, or we can make a copy from any of the existing Maven projects.

Choose any previous Maven lab project to make a copy from, for e.g.: `XMLConfigConstructorDI`

In the Project Explorer, right click on `XMLConfigConstructorDI`, select `Copy` and then right click in any empty space in the Explorer and select `Paste`.

For the new project name, type: `JavaConfigDI`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select `Maven -> Update Project`, and click `OK`.

Delete all the packages and files in `src/main/java` and `src/main/resources`. We will start populating the project from scratch.

3.11.1 Basic configuration setup with @Configuration and @ComponentScan

Create the following classes in `com.workshop.javaconfig`

```
SwimmingExercise.java
Exercise.java
MainConfig.java
JavaConfigDIMainApp.java
```

Open and right click on `JavaConfigDIMainApp` and select `Run As -> Java Application`. Verify that the correct bean has been created and its output logged to the console correctly.

3.11.2 Defining beans with @Bean and retrieving them

Make the following changes:

```
JavaConfigDIMainApp-v2.java
MainConfig-v2.java
SwimmingExercise-v2.java

JoggingExercise.java
Student.java
CollegeStudent.java
```

Open and right click on `JavaConfigDIMainApp` and select `Run As -> Java Application`. Verify that the correct bean has been created and its output logged to the console correctly.

3.11.3 Using @Primary to give higher preference to a @Bean factory method

Make the following changes:

```
CyclingExercise.java
JavaConfigDIMainApp-v3.java
MainConfig-v3.java
```

Open and right click on `JavaConfigDIMainApp` and select `Run As -> Java Application`. Verify that the correct bean has been created and its output logged to the console correctly.

Remove the `@Primary` on `public Exercise cyclingExercise()` in `MainConfig` and run the application again. Notice this time an exception is thrown as Spring is unable to determine which implementation for the dependency to use.

3.11.4 Injecting values using @PropertySource and @Value

Make the following changes:

Place `highschool.properties` in `src/main/resources`

`HighSchoolStudent.java`
`MainConfig-v4.java`
`JavaConfigDIMainApp-v4.java`

Open and right click on `JavaConfigDIMainApp` and select `Run As -> Java Application`. Notice that the correct values are read for `HighSchoolStudent`'s fields.

3.11.5 Injecting collections

Make the following changes:

`CollegeStudent-v2.java`
`MainConfig-v5.java`
`JavaConfigDIMainApp-v5.java`

Open and right click on `JavaConfigDIMainApp` and select `Run As -> Java Application`. Notice that the correct values are read for `CollegeStudent`'s collection fields.

3.11.6 Using `@Component` and `@Autowired` with `@Bean`

Make the following changes:

`MainConfig-v6.java`
`JavaConfigDIMainApp-v6.java`
`HighSchoolStudent-v2.java`
`CollegeStudent-v3.java`

Open and right click on `JavaConfigDIMainApp` and select `Run As -> Java Application`.

4 Section 4: Servlet and Java Server Pages (JSP) Basics

4.1 HTTP Basics

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>
https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

<https://www.geeksforgeeks.org/difference-between-url-and-uri/>
<https://zvelo.com/anatomy-of-full-path-url-hostname-protocol-path-more/>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

<https://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177>

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages>

https://www.tutorialspoint.com/http/http_requests.htm
https://www.tutorialspoint.com/http/http_responses.htm
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Session>
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>

4.2 Intro to Servlet and JSP

<https://beginnersbook.com/2013/05/servlet-tutorial/>
<https://www.geeksforgeeks.org/introduction-java-servlets/>
<https://beginnersbook.com/2013/05/jsp-tutorial-introduction/>
<https://www.tutorialspoint.com/jsp/index.htm>

4.3 LAB: Working with Servlets in Tomcat

4.3.1 Creating a Dynamic Web Project

Make sure you have completed the installation of Tomcat and integration into Eclipse as described in the Setting up Development Environment.

The source code for this lab is found in `Servlet-Basic-Project/changes` folder.

Switch to Java EE perspective.

Go to File -> New -> Dynamic Web Project. Type in the project name: `ServletBasicProject`

Make sure that the Target runtime is set to the correct version of Apache Tomcat

Change the dynamic web module version to 3.1

Click Next to go to the Source Folders build path dialog box.

Click Next again to go to the WebModule dialog box. Check the Generate web.xml deployment descriptor box and click Finish.

Eclipse will take a while to complete generating the project. When it is done, you should see a `web.xml` (the deployment descriptor) in the `WebContent/WEB-INF` directory. Note that if you did not initially generate a deployment descriptor `web.xml` for your project during the creation phase, you can do so later through Eclipse by right clicking on the project, Java EE Tools -> Generate Deployment Descriptor. Create it this way (rather than manually creating a `web.xml` in the `WEB-INF` folder) to ensure that Eclipse takes into account its settings when deploying the app.

4.3.2 Working with <welcome-file-list> in web.xml

<https://beginnersbook.com/2014/04/welcome-file-list-in-web-xml/>

Create the following 5 new files in WebContent (NOTE: Be very careful to place these files in WebContent and not any of its subfolders such as META-INF or WEB-INF).


```
first-welcome.jsp
second-welcome.html
third-welcome.jsp
index.html
index.jsp
```

Replace the contents of the existing `WEB-INF/web.xml` with `changes/web.xml`

To run the web application, right click on the project entry and select `Run As -> Run on Server`. Select Choose an existing server and select the Tomcat entry in the server list. Check the: Always use this server when running the project box. Click Finish.

Eclipse will open up a browser tab in the Editor view at the deployment URL of this app in Tomcat, which in this case is:

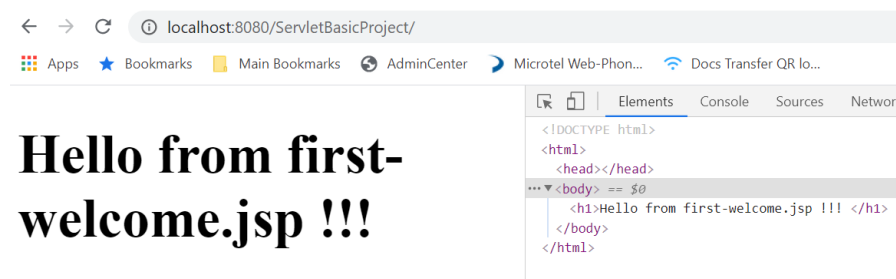
<http://localhost:8080/ServletBasicProject/>

and you should see the contents from `first-welcome.jsp` being displayed.

Try typing this URL into a browser tab in a standalone browser such as Chrome or Firefox. You should also be able to view the contents from `first-welcome.jsp`

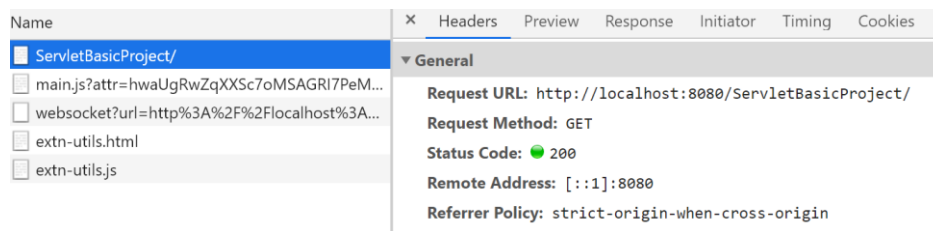
All major browsers provide facility to inspect and interact with the HTML content rendered in the browser. In Chrome, select: More tools -> Developer tools.

Select the Elements view, and you should be able to see the HTML for the content currently rendered in the browser view.

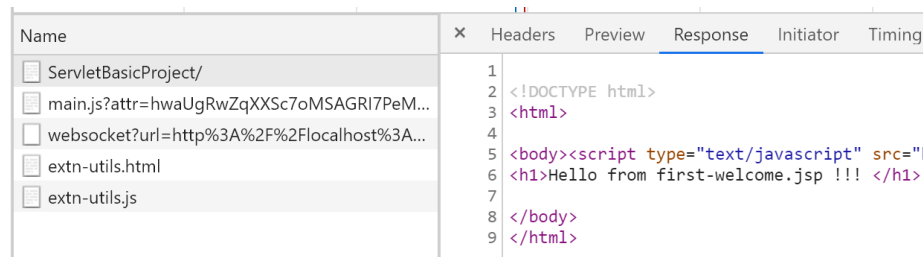


Select the Network view and refresh the page (F5). You should be able to see list of outgoing HTTP requests from the browser. Some of these requests may be related to plugins that you already have in your browser (for e.g. for your antivirus software), however you should see at least one request for the URL: `/ServletBasicProject`.

Click on this and in the Headers view you should be able to see information about the outgoing HTTP request (URL, Request Method, remote address, request headers) as well as the HTTP response (status code, response headers).



In the Response view, you can see the actual content of the HTTP response body (in the event there is content).



A web app is deployed into Tomcat at a context root, which is usually the same as the project name in Eclipse (in this case, ServletBasicProject). You can view the current context root by right clicking on the project -> Properties -> Web Project Settings.

The first part of the URL path after the domain name / port number (localhost:8080) is always the context-root, i.e.:

<http://localhost:8080/ServletBasicProject/>

Change the order of listing of the three `<welcome-file>` elements in `web.xml`. To check which file is now loaded by the server, you can reload the web app to the integrated Eclipse server by:

- Right click on the Tomcat server entry in the Servers tab, and selecting Restart
- Right click on the project, Run As -> Run on Server

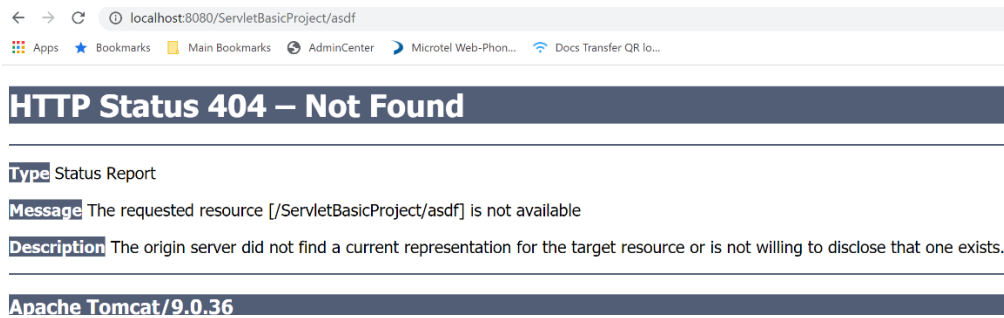
By default, Eclipse will periodically reload the web app to the integrated Eclipse server after you make changes to any of the relevant files in the web project. However, if these changes do not appear, then do an explicit reload by using any of the two steps above.

Refresh the browser tab to reload the web app. Verify this change in your standalone browser.

Notice that you can always retrieve a specific file by appending it to the end of the basic deployment URL, for e.g.:

<http://localhost:8080/ServletBasicProject/second-welcome.html>
<http://localhost:8080/ServletBasicProject/third-welcome.jsp>

Notice also that if you type a random second part to the initial deployment URL path(for e.g. <http://localhost:8080/ServletBasicProject/asdf>), the server returns a 404 Status error message. This is because the server attempts to find a mapping for a resource (such as a servlet) for the `asdf` portion of the path and fails to do so. You can check the Network view in the Chrome Developer tools to verify this in more detail



Comment out the entire `<welcome-file-list>` block of elements (highlight the statements, right click -> Source -> Toggle Comment). Save and reload.

Notice now that `index.html` is loaded.

If you are having issues obtaining the correct results in the embedded browser in Eclipse, do this in a standalone browser (Chrome, Firefox, etc) and make sure you rerun the project or restart the server.

Rename `index.html` to `temp.html`. Reload the app.

Notice now that `index.jsp` is loaded.

Rename `index.jsp` to `temp.jsp`. Reload the app.

Notice now that the server returns a 404 Status error message as it is no longer able to find any of the default files that it is set up to look for.

Uncomment the entire `<welcome-file-list>` block of elements. Save and reload. Confirm that the relevant files are displayed accordingly as usual.

4.3.3 THEORY: Schema for `web.xml`

It is important to double check that you are using the right schema version and layout for `web.xml`, especially when you are working with examples from the Internet that use older versions.

The official schema page:

<http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/index.html>

All Java EE 7 and newer Deployment Descriptor Schemas share the namespace <http://xmlns.jcp.org/xml/ns/javaee/>. This corresponds to Servlet 3.1 (Java EE 7) and 4.0 (Java EE 8) specifications.

Java EE 6 and older Deployment Descriptor Schemas share the namespace: <http://java.sun.com/xml/ns/javaee>. This corresponds to Servlet 3.0 (Java EE6) and Servlet 2.5 (Java EE5).

Schemas with a DTD with a DOCTYPE DTD are even older.

Example of schemas for `web.xml` for Servlet 3.1 and 4.0. Replace `x.y` in the snippet below with 3.1 or 4.0

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_x.y.xsd"
  version="x.y">
.....
.....
.....
.....

</web-app>
```

Example of schemas for `web.xml` for Servlet 3.0 and 2.5. Replace `x.y` in the snippet below with 3.0 or 2.5

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_x.y.xsd"
  version="x.y">
.....
.....
.....
.....

</web-app>
```

4.3.4 Defining and mapping a servlet in web.xml

<http://tutorials.jenkov.com/java-servlets/web-xml.html#configuration>

Create a new package: `com.workshop.servlets` in `src`

Create a new Servlet in this package: `FirstServlet`. Click Finish.

Repeat for another new Servlet: `SecondServlet`

Make the following changes:

`web-v2.xml`

Deploy the web app as usual. Go to:

<http://localhost:8080/ServletBasicProject/awesome>

and verify that the HTML content from `FirstServlet` is returned to the browser. You should also be able to see the console output from the `System.out.println` in the `doGet` method

Check that the following URLs map to `SecondServlet` by typing them manually into the address bar and verifying that the HTML content from `SecondServlet` is returned to the browser

<http://localhost:8080/ServletBasicProject/cool>

<http://localhost:8080/ServletBasicProject/first/second>

<http://localhost:8080/ServletBasicProject/third>

<http://localhost:8080/ServletBasicProject/third/fourth>

<http://localhost:8080/ServletBasicProject/random.html>

<http://localhost:8080/ServletBasicProject/second-welcome.html>

Notice that the last URL still maps to `SecondServlet`, even though there is actually a `second-welcome.html` file in `WebContent` that could be returned. The mapping in `web.xml` thus overrides the default mapping of the web server to resources in the web project.

4.3.5 Mapping a servlet with `@WebServlet`

Create two more new Servlets in the same package: `ThirdServlet`, `FourthServlet`

Deploy the web app as usual. Go to:

<http://localhost:8080/ServletBasicProject/superman>

and verify that the HTML content from `ThirdServlet` is returned to the browser. You should also be able to see the console output from the `System.out.println` in the `doGet` method

Check that the following URLs map to `FourthServlet` by typing them manually into the address bar and verifying that the HTML content from `FourthServlet` is returned to the browser

<http://localhost:8080/ServletBasicProject/spiderman>

<http://localhost:8080/ServletBasicProject/marvel>

<http://localhost:8080/ServletBasicProject/marvel/heroes>

4.3.6 Reading HTML form data and using initialization parameters

Create a new servlet in the same package: `FormServlet`

Create a new HTML file in `WebContent`: `welcome-form.html`

Make the following changes:

`web-v3.xml`

Deploy the web app as usual. Go to:

<http://localhost:8080/ServletBasicProject/>

and complete the form. Check the Headers sub view in the Network view of the Developer Tools console to view the Query String Parameters.

Notice the HTML response returns differs depending on the language of choice.

Note that if no method is explicitly specified for a HTTP form submission, the default method is specified is GET.

We can now explicitly state the method as POST in the HTML form.

Make the following changes:

```
welcome-form-v2.html
```

Deploy the web app as usual. Go to:

<http://localhost:8080/ServletBasicProject/>

and complete the form. Check the Headers sub view in the Network view of the Developer Tools console. Notice that the query string parameters are no longer visible in the Request URL but are now instead placed in the body of the request message.

4.3.7 Configuring the integrated Eclipse Tomcat instance

Double click on the Tomcat server entry in the Servers view. This should open up the Overview pane where you can configure a variety of options.

Click on the Module tab next to the Overview tab at the bottom of the view. This opens up the Modules view where you can add Web Modules (i.e. web app projects in Eclipse that are deployable to Tomcat).

In the Overview pane, click on Open Launch Configuration. In the Edit Configuration dialog box, click on Arguments. In the VM arguments box you will see a variety of arguments in the form of `-Dkey=value`. These are passed to the JVM process of the Tomcat server when it is started.

The environment variable `CATALINA_BASE` holds the root folder name for a runtime configuration of a specific Tomcat instance, which needs to be set if you want to have multiple Tomcat instances on the same machine. In our case, we have one instance that runs as an integrated server with Eclipse and the other runs as a standalone Windows service.

You should see one of the arguments with this variable specified:

```
-Dcatalina.base = "your workspace directory\.metadata\.plugins\org.eclipse.wst.server.core\tmp0"
```

Navigate to this directory in File Explorer. In the `wtppwebapps` subfolder, you should see one or more folders corresponding to the various web apps that you have deployed. Navigate into `ServletBasicProject`. Check that the contents of this directory matches the content of `WebContent` (the root of the web project) in the same project in Eclipse. If you browse to `WEB-`

`INF\classes`, you will see that it contains the compiled classes for all the servlets in the `src` folder in the Eclipse project. This directory layout matches that which is expected by Tomcat for a proper web-app deployment.

<https://tomcat.apache.org/tomcat-9.0-doc/appdev/deployment.html>

Topic: Standard Directory layout

Close File Explorer. Back in Eclipse, right click on the project entry in the drop down list below the Tomcat server entry in the Servers view and select Remove. This undeploys the web app from the integrated Tomcat server instance.

Open File Explorer and navigate back to the `wtppwebapps` subfolder and you should see that the project folder has been deleted. Close File Explorer, redeploy the app again from Eclipse, navigate back to the `wtppwebapps` subfolder and you should see that the project folder has been restored.

4.3.8 Deploying as a WAR to the standalone Tomcat instance

Right click on the project, Export -> WAR file. In the WAR Export dialog box, click the Browse button and select a suitable folder (for e.g. Desktop) to save the `ServletBasicProject.war` file in. Then click Finish.

Open the WAR file with 7-Zip and browse its contents. Again notice that it has the same contents and directory layout as `WebContent` in the Eclipse project which matches that which is expected by Tomcat for a proper web-app deployment.

Make sure that the Tomcat instance that is running as a Windows service has been set up for HTTP on port 8181 (this should have been done in the initial development environment setup).

Start up the service from the Services dialog box.

Navigate to: <http://localhost:8181/>

There are several ways to deploy a web app into a standalone Tomcat server instance:

<https://tomcat.apache.org/tomcat-9.0-doc/appdev/deployment.html>

Topic: Deployment With Tomcat

Click on Manager App and complete the username / password prompt. You will be navigated to the Tomcat Web Application Manager main page.

In the WAR file to deploy section, select Choose File button. Then select `ServletBasicProject.war` and click Deploy. You should be able to see `/ServletBasicProject` among the list of deployed applications; click on this link to be redirected to the app.

<http://localhost:8181/ServletBasicProject/>

Verify that the functionality is exactly the same as when it was running in the integrated Tomcat server in Eclipse.

Go to the deployment folder for this Tomcat installation. If you installed it as a Windows Service, then it should likely be at:

`C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps`

Just as in the case of the Eclipse instance, notice that the `ServletBasicProject` folder is in here with its contents matching that of `WebContent` in the same project in Eclipse

Click on the Undeploy button for the `/ServletBasicProject` entry. The application disappears from the list and the project folder is also removed from `webapps`

Another way to deploy is to simply copy and paste `ServletBasicProject.war` into the `webapps` folder. After a short while, Tomcat will automatically unzip this WAR into a project folder and the app is now deployed (you should be able to see it listed in the Tomcat Web Application Manager).

4.4 LAB: Working with JSP in Tomcat

4.4.1 Creating a Dynamic Web Project

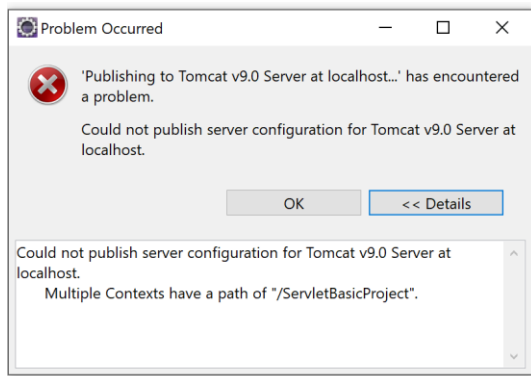
Make sure you have completed the installation of Tomcat and integration into Eclipse as described in the Setting up Development Environment.

We can create a new Dynamic Web project from scratch following the steps that we did for the earlier lab. Alternatively, we can create a new copy based on the previous lab project and modify it appropriately so that it can run properly.

In the Project Explorer, right click on `ServletBasicProject`, select Copy and then right click in any empty space in the Explorer and select Paste.

For the new project name, type: `JSPBasicProject`

The main issue with creating a copy of an existing project is that the new copy will retain the context root of the previous project. This will result in a naming conflict on the Tomcat server if you attempt to deploy this project. Do this now: right click on `JSPBasicProject` and select Run As -> Run on Server. If `ServletBasicProject` is still running on Tomcat, then this error message will occur:



You need to give the new project a new, distinct context root of its own, which will typically be the same as the project name.

Right click on the project entry, select Properties -> Web Project Settings and enter: JSPBasicProject. Click Apply, Ok and then click Apply and Close

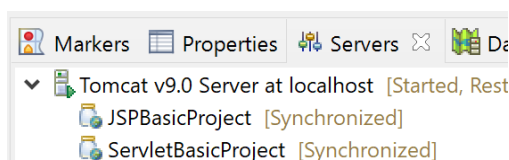
Right click on the project entry, select Properties - > Resource. Click on arrow next to the Location field. Go into JSPBasicProject/.settings and modify the settings in this file: org.eclipse.wst.common.component to reflect the new project name.

```
<wb-module deploy-name="JSPBasicProject">
  <wb-resource      deploy-path="/"      source-path="/WebContent"
tag="defaultRootSource"/>
  <wb-resource deploy-path="/WEB-INF/classes" source-path="/src"/>
    <property                                name="java-output-path"
value="/JSPBasicProject/build/classes"/>
    <property name="context-root" value="JSPBasicProject"/>
  </wb-resource>
</wb-module>
```

Save the file and refresh the project.

Go to the Servers view and remove the existing conflicting JSPBasicProject instance.

Run the project again in the server. It should now be running with its own name (no conflicts) reported in the list of modules below the Tomcat entry in the Servers view:



This time, the project deploys at a new URL with exactly the same view as the previous project

<http://localhost:8080/JSPBasicProject/>

Delete all the existing JSP files as well as web.xml

Delete all the Java source code in com.workshop.servlets

We will populate the project from scratch with new content.

4.4.2 Declaration, expression and scriptlet tags

<https://beginnersbook.com/2013/11/jsp-declaration-tag/>
<https://beginnersbook.com/2013/11/jsp-expression-tag/>
<https://examples.javacodegeeks.com/enterprise-java/jsp/jsp-scriptlet-example/>

Create the following files in WebContent (NOTE: Be very careful to place these files in WebContent and not any of its subfolders such as META-INF or WEB-INF).

`FirstExample.jsp`

To run the web application, right click on the project entry and select Run As -> Run on Server. Select Choose an existing server and select the Tomcat entry in the server list. Check the: Always use this server when running the project box. Click Finish.

Access the webapp at this URL (either in the built-in browser in Eclipse or in a separate Chrome browser)

<http://localhost:8080/JSPBasicProject/FirstExample.jsp>

Note: As a shortcut to immediately open the Eclipse browser tab at this URL, right click on the specific JSP file you wish to execute (instead of on the project entry itself) and select Run As -> Run on Server

4.4.3 Using JSP implicit objects

Create the following files in WebContent.

`first-form.html`
`complete-first-form.jsp`

Create this file in WEB-INF

`web.xml`

Deploy the web app and access this URL:

<http://localhost:8080/JSPBasicProject/first-form.html>

Provide the form data and notice that the response returned differs on the language selected (this is a pure JSP implementation of the example that we did earlier using a Servlet).

4.4.4 Servlet forwarding to a JSP

Create the following file in WebContent:

`basic-first-form.jsp`

Create the following package in `src: com.workshop.servlets` and create this servlet in this package: `FirstFormServlet.java`

Make the following changes:

`first-form-v2.html`

Deploy the web app and access this URL:

<http://localhost:8080/JSPBasicProject/first-form.html>

Provide the form data and check that the response remains the same.

4.4.5 Using Expression Language (EL)

<https://beginnersbook.com/2013/11/jsp-expression-language-el/>
<https://www.journaldev.com/2064/jsp-expression-language-el-example-tutorial>

Create the following file in WebContent:

`el-first-form.jsp`

Make the following changes:

`FirstFormServlet-v2.java`

Deploy the web app and access this URL:

<http://localhost:8080/JSPBasicProject/first-form.html>

Provide the form data and check that the response remains the same.

4.4.6 JSP **page** and **include** directive

<https://examples.javacodegeeks.com/enterprise-java/jsp/jsp-directives-example/>
<https://beginnersbook.com/2013/11/jsp-include-directive/>

Create the following files in WebContent:

`page-directive-demo.jsp`
`demo-error.jsp`

Create this file in `com.workshop.servlets: Student.java`

Deploy the web app and access this URL:

<http://localhost:8080/JSPBasicProject/page-directive-demo.jsp>

Then access this URL:

<http://localhost:8080/JSPBasicProject/demo-error.jsp>

Notice the status 500 error returned due to the arithmetic exception in the JSP.

Make the following changes:

```
demo-error-v2.jsp  
handle-error.jsp
```

Access the URL again:

<http://localhost:8080/JSPBasicProject/demo-error.jsp>

Notice this time that a different JSP with a customized error message is loaded instead.

Create the following files in WebContent:

```
header.html  
footer.html  
main-content.jsp
```

Access this URL:

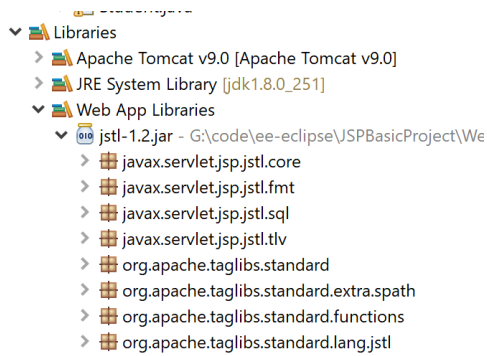
<http://localhost:8080/JSPBasicProject/main-content.jsp>

Notice that the styling rule for `<h1>` in `header.html` is also applied to the content in `footer.html`

4.4.7 JSTL Core and Function Tags

<https://www.codejava.net/java-ee/jstl/introduction-to-jstl>
<https://beginnersbook.com/jsp-jstl-tutorial-jstl-functions-and-core-tags/>
<https://www.guru99.com/jsp-tag-library.html>
<https://www.javatpoint.com/jstl-function-tags>

JSTL is not part of the runtime libraries of the Tomcat servlet container, so you will need to provide a JAR containing its classes to your web project in order to use it. Copy `jstl-1.2.jar` in changes to `WEB-INF/lib` of the project. This places the JSTL classes on your build path. You can view this in the Project Explorer view by expanding on the Libraries -> Web App Libraries.



Create the following file in WebContent:

`jstl-core-function-demo.jsp`

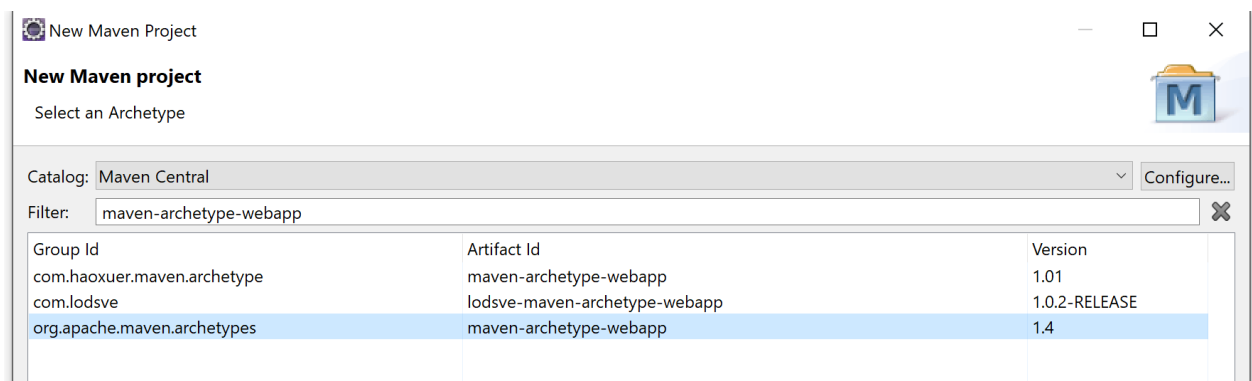
Access this URL:

<http://localhost:8080/JSPBasicProject/jstl-core-function-demo.jsp>

4.4.8 Creating a servlet/JSP webapp as a Maven project

Switch to Java EE perspective.

Start with File -> New -> Maven Project. Select Next and type `maven-archetype-webapp` in the filter. Select the entry with group id: `org.apache.maven.archetypes` and click Next



Enter in the following details in the New Maven Project dialog box and click Finish

Group Id: `com.workshop.servlets`

Artifact Id: `JSPMavenProject`

Version: `0.0.1-SNAPSHOT`

Package: `com.workshop.servlets`

You will initially have an error flagged in project entry as there is an `index.jsp` generated automatically and the necessary Servlet classes to compile it are not yet present on the build path. To correct this, right click on the project entry -> Properties -> Targeted runtimes. In the dialog box, select the Apache Tomcat Server (or any other targeted application server of choice). Click Apply and Close.

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select Maven -> Update Project, and click OK. Notice that it now includes a dependency for JSTL:

```
<dependency>
  <groupId>jstl</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
```

Currently, Web Module (or Servlet) version is set to 2.3. We need to update it to a slightly more recent version (3.1 or 4.0). We will use version 3.1.



Replace the contents of Deployed Resources -> `webapp/WEB-INF/web.xml` in the project with `web.xml` from `changes`. This is based on the schema for Servlet 3.1

Right click on the project entry Properties -> Resource. Click on the drop down box next to the Location entry in the Resource dialog box. This opens a File Explorer at the project folder in your current Eclipse workspace. Navigate into the `settings` subfolder and edit the file: `org.eclipse.wst.common.project.facet.core.xml`

Put in the new Web Module version (3.1) in this element below and save the file.

```
<installed facet="jst.web" version="3.1"/>
```

Right click on the project name -> Refresh. The deployment descriptor should now show version 3.1

✓  JSPMavenProject
 >  Deployment Descriptor: JSPMavenProject

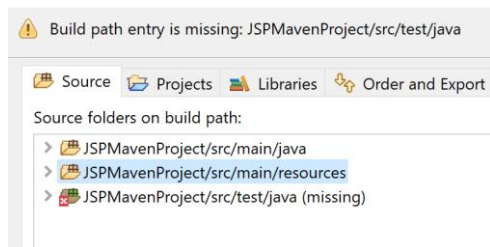
We now need to update the folder structure to match that expected of a typical Maven web app project:

<https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>

In `src/main`, create two new folders:

- `src/main/java`
- `src/main/resources`

Right click on the project, Properties -> Java Build Path. In the Source tab view, add the `src/main/resources` folder to the build path and click Apply and Close. Eclipse complains that we haven't created `src/test/java` yet, but we don't need to in this example as we are not doing any testing.



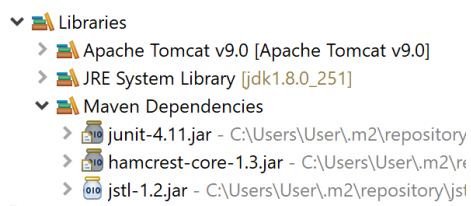
Right click on the project, Properties -> Deployment Assembly. This shows how the folders in your current Maven project maps to the document root of the project when it is deployed in the Tomcat server. It should look like something below:

Web Deployment Assembly	
Define packaging structure for this Java EE Web Application project.	
Source	Deploy Path
/src/main/java	WEB-INF/classes
/src/main/resources	WEB-INF/classes
/src/main/webapp	/
/src/test/java	WEB-INF/classes
/target/m2e-wtp/web-resources	/
Maven Dependencies	WEB-INF/lib

Key equivalence between folder structure in a Dynamic Web project and a Maven web app project

Dynamic Web Project	Maven web app
src	src/main/java
WebContent	src/main/webapp

Notice that all the JARs included in the Maven dependencies will be placed in WEB-INF/lib, which is the default directory in Tomcat for external JAR classes. Recall that we placed `jstl-1.2.jar` into here when we were working on a JSTL feature. If you go to the Libraries entry, you should see the same JAR in the Maven dependencies entry:



We are now ready to populate the Maven web app project with JSP, HTML and Servlet resources:

In `src/main/java`, create the following package: `com.workshop.servlets`

In the package, create:

`FirstFormServlet.java`

`Student.java`

Copy `first-form-v2.html` into Deployed Resources -> webapp and rename it to `first-form.html`

In the same folder (Deployed Resources -> webapp), create

```
header.html
basic-first-form.jsp
jstl-core-function-demo.jsp
```

Right click on the Project, select Run As -> Run on Server. Select the Tomcat server entry and tick the checkbox: Always use this server when running this project. Click finish.

Access this URL:

<http://localhost:8080/JSPMavenProject/first-form.html>

Enter the form details as usual, and check that the response is the same as was the case when we constructed it using a Dynamic Web Project.

Access this URL:

<http://localhost:8080/JSPMavenProject/jstl-core-function-demo.jsp>

Verify that the results for the JSTL functionality are the same as was the case when we constructed it using a Dynamic Web Project.

4.4.9 Add logging functionality to the web app

<https://stackify.com/compare-java-logging-frameworks/>
<http://www.java-logging.com/concepts/>

We will continue working with the Maven web app from the previous section. We will start off with providing a very simple log example based on Log4J2.

Make the following changes:

```
FirstFormServlet-v3.java
pom-v2.xml
```

Notice that we have included the dependency for Log4J2 as well as Servlet 3.1 (which we need for the Maven build process):

```
<!-- Dependency for log4j2 -->
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.13.2</version>
</dependency>

<!-- Dependency for Servlet 3.1 -->
<!-- So that Maven can run a build properly -->
<dependency>
    <groupId>javax.servlet</groupId>
```



```
<artifactId>javax.servlet-api</artifactId>
<version>3.1.0</version>
<scope>provided</scope>
</dependency>
```

Add the logging configuration file to `src/main/resources`:

`log4j2.properties`

Access this URL:

<http://localhost:8080/JSPMavenProject/first-form.html>

Enter the form details as usual, and check for the log output in the Console view.

Change the logging level (debug, info, trace, warn, error) in the configuration file `log4j2.properties`

```
rootLogger.level = Logging-Level
```

and run again to see which log messages are now filtered out from being displayed in the Console view.

4.4.10 Deploying as a WAR to the standalone Tomcat instance

Right click on the project, Run as -> 4 Maven Build.

In the Goals text box, type: `clean package`

Click Run

You should see messages indicating a successful build in the Console:

```
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.workshop.servlets:JSPMavenProject >-----
[INFO] Building JSPMavenProject Maven Webapp 0.0.1-SNAPSHOT
[INFO] -----[ war ]-----
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ JSPMavenProject ---
[INFO] Deleting G:\code\ee-eclipse\JSPMavenProject\target
...
...
...
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.385 s
[INFO] Finished at: 2020-12-21T10:59:41+08:00
[INFO] -----
```

Right click on the project and select Refresh.

Expand the target directory, you should be able to see: `JSPMavenProject.war`

Copy this file to any folder (e.g. Desktop), open it with 7-Zip and browse its contents. Notice that it has the same contents and directory layout as Deployed Resources - webapp in the Eclipse project which matches that which is expected by Tomcat for a proper web-app deployment. Notice as well that `WEB-INF\lib\` contains the JAR files for the JSTL and Log4j2 dependencies that you added in earlier, but not for Servlet 3.1; as this is specified with the scope of `<provided>`.

Make sure that the Tomcat instance that is running as a Windows service has been set up for HTTP on port 8181 (this should have been done in the initial development environment setup).

Start up the service from the Services dialog box.

Navigate to: <http://localhost:8181/>

There are several ways to deploy a web app into a standalone Tomcat server instance:

<https://tomcat.apache.org/tomcat-9.0-doc/appdev/deployment.html>

Topic: Deployment With Tomcat

Click on Manager App and complete the username / password prompt. You will be navigated to the Tomcat Web Application Manager main page.

In the WAR file to deploy section, select Choose File button. Then select `JSPMavenProject.war` and click Deploy. You should be able to see `/JSPMavenProject` among the list of deployed applications; click on this link to be redirected to the app.

Navigate to:

<http://localhost:8181/JSPMavenProject/first-form.html>

Verify that the functionality is the same as it was when running in the integrated Tomcat server in Eclipse.

Go to the deployment folder for this Tomcat installation. If you installed it as a Windows Service, then it should likely be at:

`C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps`

Just as in the case of the Eclipse instance, notice that the `JSPMavenProject` folder is in here with its contents matching that of Deployed Resources - webapp in the same project in Eclipse

Remember that you can also deploy this app by simply copying and pasting the WAR file into:

`C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps`

5 Section 5: Spring MVC

5.1 Spring MVC overview

<https://howtodoinjava.com/spring-mvc/spring-mvc-hello-world-example/>
https://www.tutorialspoint.com/springmvc/springmvc_overview.htm

5.2 LAB: Creating a Spring MVC application

The source code for this lab is found in `SpringMVC-Basic-Project/changes` folder.

We can create a new Maven web app project from scratch following the steps that we did for the earlier lab. Alternatively, we can create a new copy based on the previous lab project and modify it appropriately so that it can run properly.

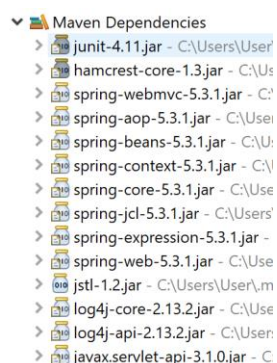
In the Project Explorer, right click on `JSPMavenProject`, select `Copy` and then right click in any empty space in the Explorer and select `Paste`.

For the new project name, type: `BasicSpringMVC`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select `Maven -> Update Project`, and click `OK`. Notice that it includes dependencies for Spring MVC, JSTL, Log4J2 as well as Servlet 3.1 API.

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring.framework.version}</version>
</dependency>
```

If you go to the Libraries entry, you should see the various JARs for Spring core, context, web and webmvc – they are all transitive dependencies that are added in via the `spring-webmvc` dependency.



Right click on the project entry, select `Properties -> Resource`. Click on arrow next to the `Location` field. Go into `BasicSpringMVC/.settings` and modify the settings in this file: `org.eclipse.wst.common.component` to reflect the new project name.

```
<?xml version="1.0" encoding="UTF-8"?><project-modules id="moduleCoreId" project-
version="1.5.0">
  <wb-module deploy-name="BasicSpringMVC">
    <wb-resource          deploy-path="/"          source-path="/target/m2e-wtp/web-
resources"/>
    <wb-resource          deploy-path="/"          source-path="/src/main/webapp"
tag="defaultRootSource"/>
    <wb-resource          deploy-path="/WEB-INF/classes"          source-
path="/src/main/java"/>
    <wb-resource          deploy-path="/WEB-INF/classes"          source-
path="/src/main/resources"/>
    <property              name="java-output-path"
value="/BasicSpringMVC/target/classes"/>
    <property name="context-root" value="BasicSpringMVC"/>
  </wb-module>
</project-modules>
```

Save the file and refresh the project.

Delete the package `com.workshop.servlets`

Delete all the JSP files in webapp

Delete `web.xml`

We are now ready to populate the Maven web app project with the relevant resources to create a proper Spring-MVC project

5.2.1 Using a XML configuration

Create the following files in WEB-INF

`spring-mvc-configuration.xml`

`web.xml`

Create a folder `views` in WEB-INF. Place these files there:

`main-menu.jsp`

`awesome.jsp`

Place `log4j2.properties` in `src/main/resources`

In `src/main/java`, create a package: `com.workshop.mvc`

Create a class here: `FirstController.java`

Delete `index.jsp` in Deployed Resources -> Webapp, if you haven't already done so. The reason for this is that the Tomcat server will automatically look for this file first when we access the root deployment URL: <http://localhost:8080/BasicSpringMVC/>, which will override the access to the mapping in `FirstController.java`

Right click on the Project, select Run As -> Run on Server

Access these URLs:

<http://localhost:8080/BasicSpringMVC/>

<http://localhost:8080/BasicSpringMVC/first>

and verify that the appropriate JSP file is returned in accordance with the URL path mapping. You can also check for the output of DEBUG messages in the Console view that gives some insight into the working of the Spring MVC architecture.

Try typing a random URL that does not map to any `@Controller` `@RequestMapping` method, and check the DEBUG message output in the Console view as well.

5.2.2 Using a Java-based configuration

In the package: `com.workshop.mvc`, create the following classes:

```
MyWebAppInitializer.java  
WebConfig.java
```

Delete these XML files in WEB-INF:

```
spring-mvc-configuration.xml  
web.xml
```

Redeploy the app and verify that you can still access these URLs and they return the correct JSP files:

<http://localhost:8080/BasicSpringMVC/>
<http://localhost:8080/BasicSpringMVC/first>

For the next lab, we will work using an XML configuration. Delete the two new Java classes you created and restore the 2 XML files into WEB-INF again.

5.3 LAB: Using `@RequestMapping`, `@GetMapping`, `@PostMapping` annotations for mapping

We continue with the same project from the previous lab.

Create these files in WEB-INF/views:

```
cool.jsp  
fancy.jsp  
first-form.jsp
```

In the package `com.workshop.mvc`, create these files:

```
SecondController.java  
ThirdController.java
```

Access these URLs below and verify that they map correctly to their respective `@RequestMapping` methods by checking the log output in the Console view:

<http://localhost:8080/BasicSpringMVC/stuff/interesting>
<http://localhost:8080/BasicSpringMVC/stuff/fancy/pants>

<http://localhost:8080/BasicSpringMVC/stuff/cool>
<http://localhost:8080/BasicSpringMVC/stuff/reallycool>

Note: If there are too many DEBUG messages appearing in the Console view, remember that you can filter these out by changing the log level to info in `log4j2.properties` in `src/main/resources`

```
rootLogger.level = info
```

Navigate to:

<http://localhost:8080/BasicSpringMVC/firstform>

and complete the form and submit it. Since the default HTTP method for a form submission is GET, verify that the corresponding `@RequestMapping` method is called.

Make this change:

```
first-form-v2.jsp
```

which now explicitly makes the submit method POST, and verify that a different `@RequestMapping` is called in this instance.

Make another change:

```
ThirdController-v2.java
```

Navigate again to:

<http://localhost:8080/BasicSpringMVC/firstform>

and complete the form and submit it. Switch between the two implementations (`first-form.jsp` – using GET) and `first-form-v2.jsp` – using POST) and verify that the Console output indicates the correct matching method is called again.

Make another change:

```
spring-mvc-configuration-v2.xml
```

Navigate to these URLs:

<http://localhost:8080/BasicSpringMVC/happy>
<http://localhost:8080/BasicSpringMVC/funny>

and confirm the correct views are returned. We use the `<mvc:view-controller>` element when we need to map paths directly to views without the need for any additional interceding business logic that would typically go in the `@RequestMapping` methods.

5.4 LAB: Working with Model

The source code for this lab is found in `SpringMVC-Forms/changes` folder.

We can create a new Maven web app project from scratch following the steps that we did for the earlier lab. Alternatively, we can create a new copy based on the previous lab project and modify it appropriately so that it can run properly.

In the Project Explorer, right click on `BasicSpringMVC`, select `Copy` and then right click in any empty space in the Explorer and select `Paste`.

For the new project name, type: `FormsSpringMVC`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select `Maven -> Update Project`, and click `OK`. Notice that it includes dependencies for Spring MVC, JSTL, Log4J2 as well as Servlet 3.1 API.

Right click on the project entry, select `Properties -> Resource`. Click on arrow next to the `Location` field. Go into `FormsSpringMVC/.settings` and modify the settings in this file: `org.eclipse.wst.common.component` to reflect the new project name.

```
<wb-module deploy-name="FormsSpringMVC">
  <wb-resource      deploy-path="/"      source-path="/target/m2e-wtp/web-
resources"/>
  <wb-resource      deploy-path="/"      source-path="/src/main/webapp"
tag="defaultRootSource"/>
  <wb-resource      deploy-path="/WEB-INF/classes"      source-
path="/src/main/java"/>
  <wb-resource      deploy-path="/WEB-INF/classes"      source-
path="/src/main/resources"/>
  <property          name="java-output-path"
value="/FormsSpringMVC/target/classes"/>
  <property name="context-root" value="FormsSpringMVC"/>
</wb-module>
```

Save the file and refresh the project.

Delete all the classes in `com.workshop.mvc`

Delete all the JSP files in `WEB-INF/views`

In `WEB-INF`, create the following files:

`web.xml`

`spring-mvc-configuration.xml`

We are now ready to populate the Maven web app project with the relevant resources to create a proper Spring-MVC project

5.4.1 Adding attributes to and displaying attributes from Model

In WEB-INF/views, create these files:

main-menu.jsp
employee-details.jsp
employee-form.jsp

Place log4j2.properties in src/main/resources

Create these classes in the package: com.workshop.mvc

HomeController.java
EmployeeController.java

Delete index.jsp in Deployed Resources -> Webapp if it still exists. The reason for this is that the Tomcat server will automatically look for this file first when we access the root deployment URL: <http://localhost:8080/FormsSpringMVC/>, which will override the access to the mapping in HomeController.java

Deploy the app:

<http://localhost:8080/FormsSpringMVC/>

Click on the employee details form link and complete the form that is returned. Check that the form data is correctly returned. Notice that we use JSP EL in employee-details.jsp to obtain the parameter values.

Add the following classes to com.workshop.mvc

JobDetails.java

Make the following changes:

EmployeeController-v2.java
employee-details-v2.jsp
employee-form-v2.jsp

Deploy the app:

<http://localhost:8080/FormsSpringMVC/>

Click on the employee details form link and complete the form that is returned. Check that the form data and model attributes is correctly returned. Notice that we use JSTL Core tags in employee-details.jsp to perform an if-else evaluation (we could have also done this using a scriptlet Java code block as well).

5.4.2 Using @RequestParam to bind form parameters

Make the following changes:

EmployeeController-v3.java

Deploy the app:

<http://localhost:8080/FormsSpringMVC/>

Click on the employee details form link and complete the form that is returned. Check that the form data and model attributes is correctly returned. Notice that we use `@RequestParam` annotation in the method signature of `processFormv2` in order to bind the request parameters to the method parameters.

5.5 LAB: Form tag library

The source code for this lab is found in `SpringMVC-FormTagLibrary/changes` folder.

We can create a new Maven web app project from scratch following the steps that we did for the earlier lab. Alternatively, we can create a new copy based on the previous lab project and modify it appropriately so that it can run properly.

In the Project Explorer, right click on `FormsSpringMVC`, select `Copy` and then right click in any empty space in the Explorer and select `Paste`.

For the new project name, type: `FormTagSpringMVC`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select `Maven -> Update Project`, and click `OK`.

Right click on the project entry, select `Properties -> Resource`. Click on arrow next to the `Location` field. Go into `FormTagSpringMVC/.settings` and modify the settings in this file: `org.eclipse.wst.common.component` to reflect the new project name.

```
<wb-module deploy-name="FormTagSpringMVC">
  <wb-resource      deploy-path="/"      source-path="/target/m2e-wtp/web-
resources"/>
  <wb-resource      deploy-path="/"      source-path="/src/main/webapp"
tag="defaultRootSource"/>
  <wb-resource      deploy-path="/WEB-INF/classes"      source-
path="/src/main/java"/>
  <wb-resource      deploy-path="/WEB-INF/classes"      source-
path="/src/main/resources"/>
  <property          name="java-output-path"
value="/FormTagSpringMVC/target/classes"/>
  <property name="context-root" value="FormTagSpringMVC"/>
</wb-module>
```

Save the file and refresh the project.

Delete all the classes in `com.workshop.mvc`

Delete all the JSP files in `WEB-INF/views`

In `WEB-INF`, create the following files:

`web.xml`

`spring-mvc-configuration.xml`

We are now ready to populate the Maven web app project with the relevant resources to create a proper Spring-MVC project

5.5.1 Text field tag <form:input>

Make the following changes:

Create these files in WEB-INF\views:

employee-details.jsp
employee-form.jsp
main-menu.jsp

Place log4j2.properties in src/main/resources

Create these classes in: com.workshop.mvc

Employee.java
EmployeeController.java
HomeController.java

Deploy the app:

<http://localhost:8080/FormTagSpringMVC/>

Click on the employee details form link and complete the form that is returned. Check that the Employee field values are correctly set and displayed.

5.5.2 Text area tag <form:textarea>

Make the following changes:

EmployeeController-v2.java
Employee-v2.java
employee-form-v2.jsp
employee-details-v2.jsp

Deploy the app:

<http://localhost:8080/FormTagSpringMVC/>

Click on the employee details form link and complete the form that is returned. Check that the Employee field values are correctly set and displayed.

5.5.3 Drop down list <form:select> with options in form

Make the following changes:

EmployeeController-v3.java
Employee-v3.java

```
employee-form-v3.jsp  
employee-details-v3.jsp
```

Deploy the app:

<http://localhost:8080/FormTagSpringMVC/>

Click on the employee details form link and complete the form that is returned. Check that the Employee field values are correctly set and displayed.

5.5.4 Drop down list <form:select> with options from POJO

We can make a slight modification so that the list of country options is provided from the Employee class as a HashMap rather than hardcoded in the employee form itself:

Make the following changes:

```
Employee-v3-2.java  
employee-form-v3-2.jsp
```

Deploy the app:

<http://localhost:8080/FormTagSpringMVC/>

Click on the employee details form link and complete the form that is returned. Check that the Employee field values are correctly set and displayed.

5.5.5 Drop down list <form:select> with options from @Controller class

We can make a slight modification so that the list of country options is provided from the EmployeeController class as a HashMap. Notice that we place this HashMap into our model through the use of the @ModelAttribute annotation applied to a method. All methods with this annotation will get their return values placed into the Model automatically without an explicit addAttribute method call.

Make the following changes:

```
Employee-v3-3.java  
EmployeeController-v3-2.java  
employee-form-v3-3.jsp
```

Deploy the app:

<http://localhost:8080/FormTagSpringMVC/>

Click on the employee details form link and complete the form that is returned. Check that the Employee field values are correctly set and displayed.

5.5.6 Drop down list <form:select> with options from properties file

We can make a slight modification so that the list of country options is read from a properties file, then stored in a field in `EmployeeController` and then added to the Model to be extracted and used in the form itself:

Make the following changes:

```
EmployeeController-v3-3.java  
employee-form-v3-4.jsp  
spring-mvc-configuration-v2.xml
```

Place this file in `src/main/resources`:
`countries.properties`

Deploy the app:

<http://localhost:8080/FormTagSpringMVC/>

Click on the employee details form link and complete the form that is returned. Check that the Employee field values are correctly set and displayed.

5.5.7 Radio buttons <form:radiobuttons> with options from @Controller class

Make the following changes:

```
EmployeeController-v4.java  
Employee-v4.java  
employee-form-v4.jsp  
employee-details-v4.jsp
```

Deploy the app:

<http://localhost:8080/FormTagSpringMVC/>

Click on the employee details form link and complete the form that is returned. Check that the Employee field values are correctly set and displayed.

5.5.8 Check boxes <form:checkboxes> with options from properties file

Make the following changes:

```
EmployeeController-v5.java  
Employee-v5.java  
employee-form-v5.jsp  
employee-details-v5.jsp
```

Place this file in `src/main/resources`:
`frameworks.properties`

Deploy the app:

<http://localhost:8080/FormTagSpringMVC/>

Click on the employee details form link and complete the form that is returned. Check that the Employee field values are correctly set and displayed.

5.5.9 Using ModelAndView vs Model

Make the following changes:

EmployeeController-v6.java

Deploy the app:

<http://localhost:8080/FormTagSpringMVC/>

Click on the employee details form link and complete the form that is returned. Check that the Employee field values are correctly set and displayed.

5.5.10 Other form tags: <form:password> <form:hidden> <form:errors>

https://www.tutorialspoint.com/springmvc/springmvc_password.htm

https://www.tutorialspoint.com/springmvc/springmvc_hidden.htm

5.6 LAB: Form validation

The source code for this lab is found in `SpringMVC-FormValidation/changes` folder.

We can create a new Maven web app project from scratch following the steps that we did for the earlier lab. Alternatively, we can create a new copy based on the previous lab project and modify it appropriately so that it can run properly.

In the Project Explorer, right click on `FormsSpringMVC`, select **Copy** and then right click in any empty space in the Explorer and select **Paste**.

For the new project name, type: `FormValidationSpringMVC`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select **Maven -> Update Project**, and click **OK**.

Notice that it includes dependencies for the Bean Validation API as well as the Hibernate Validator reference implementation for this API.

```

<!-- Reference implementation for Validation API -->
<dependency>
  <groupId>org.hibernate.validator</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>6.1.6.Final</version>
</dependency>

<!-- Dependency for Validation API -->
<dependency>
  <groupId>javax.validation</groupId>
  <artifactId>validation-api</artifactId>
  <version>2.0.1.Final</version>
</dependency>

```

Right click on the project entry, select Properties -> Resource. Click on arrow next to the Location field. Go into FormTagSpringMVC/.settings and modify the settings in this file: org.eclipse.wst.common.component to reflect the new project name.

```

<wb-module deploy-name="FormValidationSpringMVC">
  <wb-resource      deploy-path="/"      source-path="/target/m2e-wtp/web-
resources"/>
  <wb-resource      deploy-path="/"      source-path="/src/main/webapp"
tag="defaultRootSource"/>
  <wb-resource      deploy-path="/WEB-INF/classes"      source-
path="/src/main/java"/>
  <wb-resource      deploy-path="/WEB-INF/classes"      source-
path="/src/main/resources"/>
  <property                                name="java-output-path"
value="/FormValidationSpringMVC/target/classes"/>
  <property name="context-root" value="FormValidationSpringMVC"/>
</wb-module>

```

Save the file and refresh the project.

Delete all the classes in com.workshop.mvc

Delete all the JSP files in WEB-INF/views

In WEB-INF, create the following files:

web.xml

spring-mvc-configuration.xml

We are now ready to populate the Maven web app project with the relevant resources to create a proper Spring-MVC project

5.6.1 Required field validation

Make the following changes:

Create these files in WEB-INF\views:

```
employee-details.jsp  
employee-form.jsp  
main-menu.jsp
```

Place `log4j2.properties` in `src/main/resources`

Create these classes in: `com.workshop.mvc`

```
Employee.java  
EmployeeController.java  
HomeController.java
```

Deploy the app:

<http://localhost:8080/FormValidationSpringMVC/>

Click on the employee details form link and click on submit without entering any value in the name field. Notice the default error message returned (size must be between 1 and 2147483647). Check the log output in the Console to see the contents of `BindingResult` when an error occurs. Enter a valid name, submit and verify that the name is displayed.

Notice that if you enter empty spaces for the name field, this is still accepted as a valid input. We will see how to solve this later using a property editor.

When performing Spring MVC validation, in the method signature, the `BindingResult` parameter must appear immediately after the `@ModelAttribute`, otherwise any validation rules that you apply will be ignored.

5.6.2 Number range validation and custom error messages

Make the following changes:

```
Employee-v2.java  
  
employee-form-v2.jsp  
employee-details-v2.jsp
```

Deploy the app:

<http://localhost:8080/FormValidationSpringMVC/>

Click on the employee details form link and enter incorrect data for the age field: no entry, an entry higher than 100 or lower than 12 and verify that the appropriate customized error messages are returned. Notice that typing in a string in the age field results in a type conversion error as Spring framework unsuccessfully tries to typecast this – we will see how to address this later.

5.6.3 Regular expression validation

Make the following changes:

`Employee-v3.java`

`employee-form-v3.jsp`

`employee-details-v3.jsp`

Deploy the app:

<http://localhost:8080/FormValidationSpringMVC/>

Click on the employee details form link and try entering a few strings for the email address field to check with ones are flagged as being invalid based on the regular expression being used.

Valid email addresses:

`thor@gmail.com`

`1234@gmail`

`P1~!#$%^.&@gmail.com`

Invalid email addresses:

`thor.gmail.com`

`//\@gmail.com`

There are regular expressions of varying length and complexity which can perform increasingly stringent validation of what constitutes a valid email address:

<https://howtodoinjava.com/java/regex/java-regex-validate-email-address/>

For more information on Java regular expressions in general:

https://www.tutorialspoint.com/java/java_regular_expressions.htm

<https://howtodoinjava.com/java-regular-expression-tutorials/>

5.6.4 Using ResourceBundle to provide customized messages in properties file

Make the following changes:

`spring-mvc-configuration-v2.xml`

Create a file `messages.properties` in `src/main/resources`

Deploy the app:

<http://localhost:8080/FormValidationSpringMVC/>

Click on the employee details form link and try typing invalid entries into the input field (including a string into the age field). Confirm that the error messages that appear are from `messages.properties`

Notice that the name of the properties in `messages.properties` corresponds to the error code output from `bindingResult` when an error occurs. The value of the properties are the matching error messages to display for that particular error code. Notice that we now have a matching error now for the type mismatch error that occurred earlier for a string entry in the age field.

5.6.5 Using `@InitBinder` with property editors and data binding

We saw earlier that when you enter empty spaces for the name field, this is still accepted as a valid input. We will solve this problem using a property editor.

Make the following changes:

```
EmployeeController-v2.java  
Employee-v4.java
```

Deploy the app:

<http://localhost:8080/FormValidationSpringMVC/>

Click on the employee details form link and try typing empty spaces into all of the fields and submitting the form. Confirm that the error message that appears corresponds to the custom message for `@NotNull` annotation in `Employee.java`

Notice here that we have custom error messages provided in both `Employee.java` and also `messages.properties`. Spring will use the messages in `messages.properties` first for any matching error codes, and then only check in `Employee.java` if it is not able to find any.

We register the `StringTrimmer` custom editor in a method marked with `@InitBinder` in `EmployeeController.java`. This works on all `String` requestParams in the incoming request to trim empty spaces in the string before it is processed for validation. A string consisting of empty spaces will therefore become null when it is processed by the validator, resulting in a `NotNull` error due to the `@NotNull` annotation on all the fields in `Employee.java`

For the list of all property editors that can be registered for pre-processing of incoming requestParams to the controller:

<https://docs.spring.io/spring-framework/docs/current/javadoc-api/org.springframework.beans.propertyeditors/package-summary.html>

5.6.6 Creating a custom validator annotation to perform custom validation

Make the following changes:

```
ContainsPhrase.java  
ContainsPhraseConstraintValidator.java
```

```
Employee-v5.java  
employee-form-v4.jsp
```

employee-details-v4.jsp

Deploy the app:

<http://localhost:8080/FormValidationSpringMVC/>

Click on the employee details form link and enter valid entries for all fields, and an entry that does not have the phrase `Spring` in it in the Job Description field. Check that an error message is returned for this field based on the default value in `ContainsPhrase`. Then enter an entry with the phrase `Spring` in it and verify that validation succeeds.

Make the following changes:

Employee-v6.java

Deploy the app:

<http://localhost:8080/FormValidationSpringMVC/>

Click on the employee details form link and enter valid entries for all fields, and an entry that does not have the phrase `Java EE` in it in the Job Description field. Check that an error message is returned for this field based on the value provided in `Employee`. Then enter an entry with the phrase `Java EE` in it and verify that validation succeeds.

In the annotation, there are two parameters defined for use in the validation process (we can include more if we wish). The `value` parameter is supplied with the annotation, while `message` parameter holds actual error message that appears when the rule is violated. Default values are provided for both parameters here in the event that the user does not specify any when applying the annotation.

The `@Target` annotation identifies where this custom annotation can be applied to (method, class, field, etc)

The `@Retention` annotation specifies how long we should retain it. `RetentionPolicy.RUNTIME` means keeping this annotation in the compiled Java bytecode so we can use it, and introspect on it, and instrument on it during runtime.

Using `@interface` allows the definition of a custom annotation