Servlet JSP Workshop
Lab 2

1 Lab setup

Make sure you have the following items installed

- Latest version of JDK 8 / 11 (note: labs are tested with JDK 8 but should work on JDK 11 with no or minimal changes)
- Eclipse Enterprise Edition for Java (or a suitable alternative IDE for Enterprise Java)
- Tomcat integrated into Eclipse IDE
- Latest version of Maven
- A suitable text editor (Notepad ++)
- A utility to extract zip files

In each of the main lab folders, there are two subfolders: `changes` and `final`. The `changes` subfolder just holds the source code files for the lab, while the `final` subfolder holds the complete Eclipse project starting from its project root folder. We will use the code from the `changes` subfolder to build up our applications from scratch and you can always fall back on the complete Eclipse project if you encounter any errors while building up the application.

2 Working with JSP in Tomcat

2.1 Creating a Dynamic Web Project

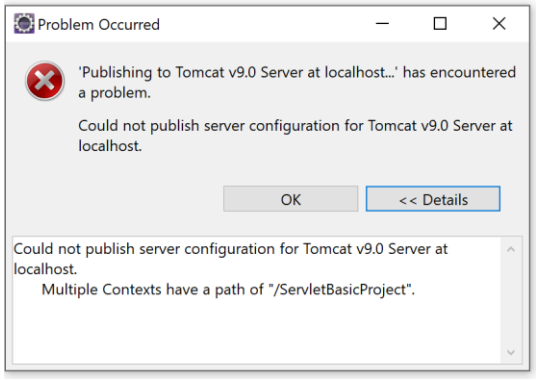
Make sure you have completed the installation of Tomcat and integration into Eclipse as described in the Setting up Development Environment.

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

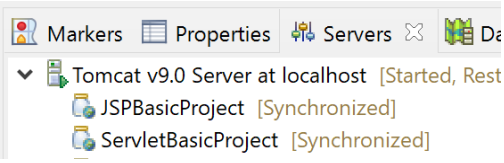
Right click on the project entry, select Properties - > Resource. Click on arrow next to the Location field. Go into `JSPBasicProject/.settings` and modify the settings in this file: `org.eclipse.wst.common.component` to reflect the new project name.

```
    <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-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.

2.2 Declaration, expression and scriptlet tags

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

2.3 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:

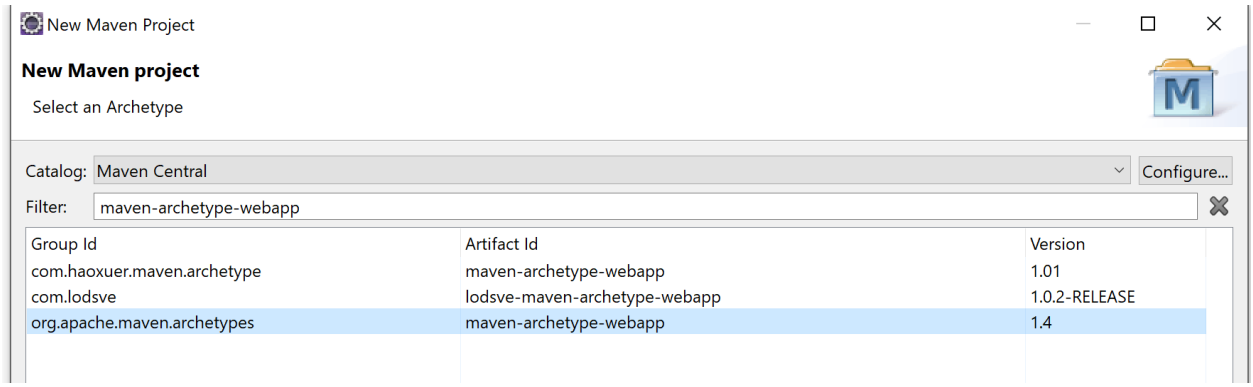
http://localhost:8080/JSPBasicProject/first-form.html

Provide the form data and notice that the response returned differs on the language selected

2.4 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>
```

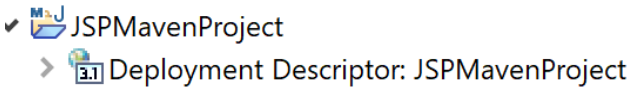
Currently, Web Module (or Servlet) version is set to 2.3. We need to update it to a slightly more recent version (3.1 or 4.0). We will use version 3.1.
Replace the contents of Deployed Resources -> `webapp/WEB-INF/web.xml` in the project with `web.xml` from `changes`. This is based on the schema for 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



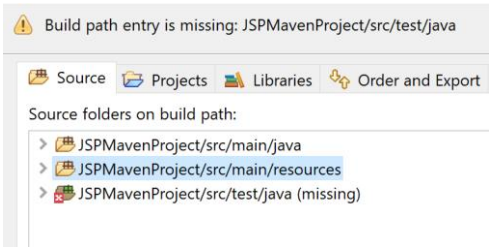
We now need to update the folder structure to match that expected of a typical Maven web app project:

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

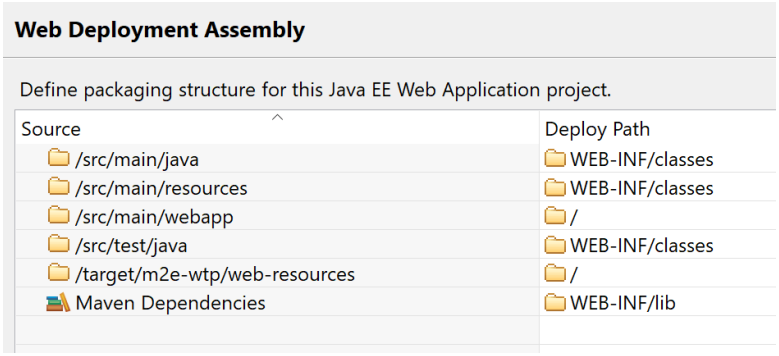
In `src/main`, create two new folders:

- `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.



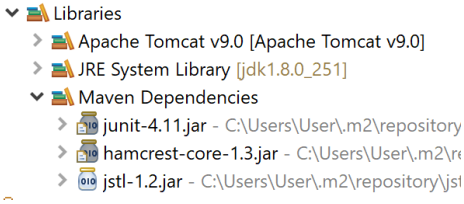
Right click on the project, Properties -> Deployment Assembly. This shows how the folders in your current Maven project maps to the document root of the project when it is deployed in the Tomcat server. It should look like something below:



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`

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.

2.5 Add logging functionality to the web app

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.

2.6 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 a different port from 8080 (for e.g. port 8181).
https://mkyong.com/tomcat/how-to-change-tomcat-default-port/

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`