Maven Workshop
Lab 1

1 LAB SETUP 1
2 COMPILER / JRE SETTINGS IN ECLIPSE 1
3 GENERATING AND USING A JAR 2
4 USING JARS FROM EXTERNAL LIBRARIES (LOGBACK AND JUNIT) 4

1 Lab setup

Make sure you have the following items installed

- Latest version of JDK 11 (note: labs are tested with JDK 11 but should work with higher versions with no or minimal changes)
- Eclipse Enterprise Edition for Java (or a suitable alternative IDE for Enterprise Java)
- Latest version of Maven
- A suitable text editor (Notepad ++)
- A utility to extract zip files (7-zip)

In each of the main lab folders, there are two subfolders: `changes` and `final`. The `changes` subfolder holds the source code and other related 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 Compiler / JRE settings in Eclipse

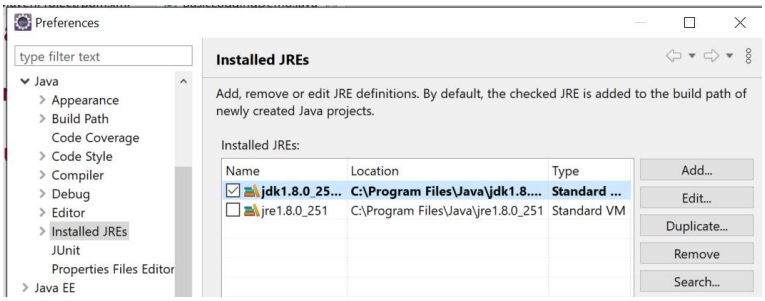
For a new Eclipse installation and also when you change to a new workspace, you may need to double check to ensure the compiler / JRE is set properly to ensure that your programs compile properly and Maven can execute correctly.

Newer versions of Eclipse (4.18 onwards) already include a JRE as part of the installer process and usually do not require any further modification. The JRE version is typically 16 or higher.

If you are planning to compile your applications to run off the JDK that you have installed locally and that JDK is of a lower version (for e.g. 11), you may need to change the Java compliance level in Eclipse.

Go to Windows -> Preferences, and in the Dialog Box, select Java -> Compiler and make sure you select a compiler compliance level that is aligned with the JDK installed locally.

For older versions of Eclipse which do not have a JRE included, it is also advisable to use the installed JDK on your machine as the installed JRE for Eclipse to use. To do this, 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\jdk-11.0.13

If you do not do this on older versions of Eclipse, you might 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
```

3 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 (File -> New -> Java Project).
For the project name, type: `SimpleFirstProject`
Ensure that the execution environment JRE is set to the correct version of Java that you have either installed locally or set as the JRE for Eclipse.
Then click Finish.

A New module-info java dialog box pops up to prompt for a module name. Modules are a new feature introduced in Java 9 that provides a new level of abstraction above packages. We will not be using it here, so click Don't Create.

Create a new package (Right click on the project name, select New -> Package): `com.workshop.operations`

In this package, place these files from `changes`:

`StringOperations.java`
`MainProgram.java`

Run `MainProgram` as normal to verify that it works. With MainProgram as the active editor tab, click anywhere in the editor and from the context menu, select Run As -> Java Application. Verify that it produces the expected result. The output should appear in the Console view among the stack of views below the main editor area.

We will now create an executable, standalone JAR for this project.

Right click on the project name. From the context menu, select Export. In the Export dialog box, select Java -> Runnable JAR file.
Select a suitable export destination folder and name the JAR as: `StringOperations.jar`
For the Launch configuration, select `MainProgram – SimpleFirstProject`.
Click Finish.

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

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

Verify that the execution produces the expected result.

Use 7-z (or any other suitable archive app) to view the contents of 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 using the same approach as before: `SimpleSecondProject`
Create a new package within this project: `com.workshop.user`

In this package, place this file from `changes`:

`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 named `lib` (right click on the project name and select New -> Folder). Copy and paste `StringOperations.jar` into this folder.

We now need to add this JAR to the build path for this project.
Right click on the project, select Properties -> Java Build Path
Click on the Libraries Tab and select the Classpath entry.
Next click on Add JARs. Select the JAR file that you have just copied into the `lib` folder. 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. Verify that you can execute it successfully (right click, select Run as -> Java Application

In this simple example, `StringOperations.jar` is a dependency of `SimpleSecondProject` The dependency must be placed on the build path as well as the runtime path of `SimpleSecondProject` in order for it to be executed successfully.

We now repeat the process of generating a JAR from this project.
Select a suitable export destination folder and name the JAR as: `User.jar`
For the Launch configuration, select `UserProgram – SimpleSecondProject`
Click Finish.

A dialog box pops up warning about the repacking of referenced libraries. This indicates that Eclipse will now include the contents of the previous JAR (`StringOperations.jar`) that is placed on the build path of the current project by extracting the package structure of that JAR and placing it into the newly generated JAR. This ensures that the classes of that JAR remain available to this project so that it can be run as a standalone executable JAR.

Use 7-z (or any other suitable archive app) to view the contents of the JAR, verify that it contains the compiled classes from this project as and from `StringOperations.jar`, as well as the standard manifest (MANIFEST.MF) which indicates the main class to run. You can also unzip the JAR to view its contents.

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

```
java -jar User.jar
```

4 Using JARs from external libraries (Logback and JUnit)

Logback is a popular logging framework that is a successor to the older log4J project. Logging is a common activity used in many production grade applications.
JUnit is the de-facto library for unit testing in Java.

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

Switch to Java SE perspective.

Create a new Java project: `SimpleLoggingProject`

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

In this package, place this file from `changes`:

`BasicLoggingDemo`

Notice that there are a variety of syntax errors flagged in the editor as the necessary classes in the import statements are not present on the build path.

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

Copy the following JARs from the `jars-to-use` folder into `lib`:

```
logback-classic-x.y.z.jar
logback-core-x.y.z.jar
slf4j-api-x.y.z.jar
```

We will again add these JARs to the build path for this project.
Right click on the project, select Properties -> Java Build Path
Click on the Libraries Tab and select the Classpath entry.
Next click on Add JARs. Select the JAR files that you have just copied into the `lib` folder. Click Apply and Close.

Notice that you can expand the JAR files to see their contents in the Referenced Libraries entry in the Package Explorer. The two classes that you just imported in `BasicLoggingDemo` can be located inside `slf4j-api-x.y.z.jar`

The syntax errors in `BasicLoggingDemo` should disappear. Verify that you can execute it successfully (right click, select Run as -> Java Application. The output you see in the Console view is the standard output that a logging framework typically produces. We will examine logging frameworks in more detail in a subsequent lab.

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

In this package, place these files from `changes`:

```
TestJunit
TestRunner
```

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

Copy the following JARs from the `jars-to-use` folder into `lib`:

```
junit-x.y.z.jar
hamcrest-core-x.y.z.jar
```

We will again add these JARs to the build path for this project.
Right click on the project, select Properties -> Java Build Path
Click on the Libraries Tab and select the Classpath entry.
Next click on Add JARs. Select the JAR files that you have just copied into the `lib` folder. Click Apply and Close.

Notice that you can expand the JAR files to see their contents in the Referenced Libraries entry in the Package Explorer. The classes that you have imported in both `TestJunit` and `TestRunner` can be found in `junit-x.y.z.jar`

To see the JUnit test in action, go to `TestRunner` and do Run As -> Java Application

The JARs that were provided in this project could also have been downloaded from the respective official project websites:

http://logback.qos.ch/download.html
https://github.com/junit-team/junit4/wiki/Download-and-Install

Notice that the Logback project does not directly provide the JARs for you to download, instead it redirects you to the Maven central repository to download these JARs - which we will examine shortly in a coming lab.