SWEN301 Tutorial - Getting Started with Maven

Objective

Maven is a tool to automate standard workflows that have to be performed in each iteration, such as compiling, testing and creating deployable components. Projects created with Maven have a highly standardised layout (structure of folders and files) that can be easily imported into most IDEs (IntelliJ, Eclipse, etc). Furthermore, Maven can manage dependencies on libraries by connecting to a central repository, and support additional tools as plugins. We will use Maven as a "project backbone" in this course.

Instructions

1. Check whether Maven is installed on your computer by running **mvn -version** , if it is not installed (on your own device), install it following instructions from https://maven.apache.org/ .
2. Create a Maven project using the command line command **mvn archetype:generate** using instructions from the web:

   https://www.mkyong.com/maven/how-to-create-a-java-project-with-maven/

   Set your own artefactId and groupId, inspect jars (aka artefacts) in the Maven repository for the kind of names used: https://mvnrepository.com/ .

   *The archetype id should be **maven-archetype-quickstart** . Note that if you copy and paste a command with line breaks into the terminal, only the first line will be executed. Solution: copy and paste this into a text editor, and remove the new line characters !*

2. Inspect the file structure created -- this is the standardised structure of a Maven project.
3. Inspect the **pom.xml** file created (you can use a standard text editor, web browser or a CLI command like **more** for this purpose) - this is the file that defines project properties.
4. In **pom.xml**, locate the xml element that defines the dependency of the project on junit. The project will depend on junit version 3.8.1. Change this to version 4 or 5. To do so, search for junit 4 or 5 in https://mvnrepository.com/ , and replace the dependency element (the XML snippet starting with **<dependency>**) with the spec copied from this website.
5. Load the project into an IDE , for instance, open the project in IntelliJ. You can also edit the project using a text editor, such as Sublime or Atom. *Hint: in Eclipse, use the*

function "Import > General > Projects from Folders or Archive", import the folder that has the pom.xml file.

6. Create a simple Java package **nz.ac.vuw.ecs.swen301.tut1** and a class **Calculator** in this package with a public method **int add(int i1,int i2)** in the source code folder **src/main/java**
7. Compile the project by running **mvn compile** from the command line. Hint: if you use IntelliJ, you can use the built-in terminal. If you get error messages referring to an unsupported version of Java, configure Java 8 in pom.xml following these instructions: https://www.baeldung.com/maven-java-version (I recommend to use the properties option)
8. Create some simple JUnit tests in **src/test/java ,** name the test class **TestCalculator** (the naming convention is important for Maven to recognise tests). Prefer using JUnit 4 features (**@Test** annotation on tests).
9. Run tests in the IDE, and from the command line: **mvn test**
10. Build the project from the command line: **mvn package** , inspect the results in the **/target** folder.

What to try next (not part of the tutorial)

1. Add a command-line interface to the calculator, and create an executable jar.
2. Add a main method to **Calculator**, it expects two int parameters, and prints the result of adding the numbers to the console (**System.out**).
3. Use a Maven plugin described here: https://www.baeldung.com/executable-jar-with-maven to create an executable jar. In an executable jar, the manifest contains a main class, and the application can be used using the following command: **java -jar calculator.jar 2 2**
4. For an easier way to write a full-featured CLI application, consider using the Apache Commons CLI library. A tutorial how to get started can be found here: https://www.tutorialspoint.com/commons_cli/index.htm , the library is in the Maven repository and can be added as dependency, similar to how junit was added.