1495 UNIVERSITY OF ABERDEEN

JC2002 Java Programming

Lecture 15: Introduction to Maven

(Further Reading – won't be included in Coursework/Exam)

What is Apache Maven?

- Project management software making developer's life *much* easier with the following objectives:
  - Making the build process easy
  - Providing a uniform build system
  - Providing quality project information
  - Encouraging better development practices
- Website for more information: https://maven.apache.org/
- Other projects similar to Maven:
  - Gradle (https://gradle.org/)
  - Ant (https://ant.apache.org/)

What will Maven help you with?

- Builds
- Documentation
- Reporting
- Dependencies
- Source code management (SCM)
- Releases
- Distribution

Set up Maven

- In your Linux System, open terminal and check if Maven is installed:

```
mvn –version
```

- If it is not installed (i.e., you got an error “mvn not found”), install it in your terminal:

```
apt update
apt install maven
```

Building a default project in Maven

- Maven *archetypes* are template projects that you can base your own projects on
- Type `mvn archetype:generate` to generate a “HelloWorld” project based on a "quickstart" archetype
  - Enter `2082` (or whatever is the default)
  - Pick archetype version (options given, you can use the default)
  - Enter `groupId` (usually identifier for the organisation)
  - Enter `artifactId` (*jar* file name without version, e.g., *myapp*)
  - Enter the snapshot version (e.g. 1.0)
  - Enter the package name (by default the same as `groupId`)

Dynamic binding

- Maven's configuration file defines the *Project Object Model* (POM)
- Key elements (source: https://maven.apache.org/guides/getting-started/index.html):
  - **project:** This is the top-level element in all Maven pom.xml files.
  - **modelVersion:** This element indicates what version of the object model this POM is using. The version of the model itself changes very infrequently but it is mandatory in order to ensure stability of use if and when the Maven developers deem it necessary to change the model.
  - **groupId:** This element indicates the unique identifier of the organization or group that created the project. The groupId is one of the key identifiers of a project and is typically based on the fully qualified domain name of your organization. For example, org.apache.maven.plugins is the designated groupId for all Maven plugins.

Elements in pom.xml (continues)

- **artifactId:** This element indicates the unique base name of the primary artifact being generated by this project. The primary artifact for a project is typically a JAR file. Secondary artifacts like source bundles also use the artifactId as part of their final name. A typical artifact produced by Maven would have the form <artifactId>-<version>.<extension> (for example, myapp-1.0.jar).
- **version:** This element indicates the version of the artifact generated by the project.
- **name:** This element indicates the display name used for the project. This is often used in Maven's generated documentation.
- **url:** This element indicates where the project's site can be found. This is often used in Maven's generated documentation.
- **properties:** This element contains value placeholders accessible anywhere within a POM.
- **dependencies:** This element's children list dependencies. The cornerstone of the POM.
- **build:** This element handles things like declaring your project's directory structure and managing plugins.

Building a default project in Maven

- No need to compile files individually, just execute:

```
mvn compile
```

- Generate `.jar` file by executing:

```
mvn package
```

  and find the JAR file in the *target* directory. Then, run JAR file:

```
java –jar <file name>.jar
```

- ***Note:*** To make JAR files executable, you must edit the manifest with the information about the class that contains the main method which you want to run (see the next slide)

Adding manifest information in pom.xml

- The manifest is a special block that can contain information about the files packaged in a JAR file (e.g., class to run first)

```
<plugin>
  <artifactId>maven-jar-plugin</artifactId>
  <version>3.0.2</version>
  <configuration>
    <archive>
      <manifest>
        <addDefaultImplementationEntries>true</addDefaultImplementationEntries>
        <addDefaultSpecificationEntries>true</addDefaultSpecificationEntries>
        <addClasspath>true</addClasspath>
        <mainClass>org.test.App</mainClass>
      </manifest>
    </archive>
  </configuration>
</plugin>
```

Look for this placeholder in the `pom.xml` file

The main class of your application

Add this part in the file

Dependency management using pom.xml

- Add references to dependencies in `pom.xml`
  - Add as many dependencies as you need using `<dependency> … </dependency>` tags
- To see the dependency tree, use:

  `mvn dependency:tree`

- Repository of Java open source libraries managed with Maven: https://mvnrepository.com/

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    ...
  </dependency>
</dependencies>
```

Testing your projects with JUnit and Maven

- Maven helps you to automate testing of your code
  - Maven promotes best practices by keeping tests separate from the source code
  - Maven streamlines the execution of tests and reporting of test results
- JUnit is a testing framework for JVM, providing API for writing test cases in Java code
  - Maven is then used to execute the JUnit tests

Default JUnit test

- The ***src/test/java*** directory contains the source code of our unit tests
- If we use Maven quickstart archetype, an example test will be created automatically (file ***AppTest.java***)

AppTest.java

```
1  package myorg;
2
3  import static org.junit.Assert.assertTrue;
4
5  import org.junit.Test;
6
7  /**
8   * Unit test for simple App.
9   */
10 public class AppTest
11 {
12     /**
13      * Rigorous Test :-)
14      */
15     @Test
16     public void shouldAnswerWithTrue()
17     {
18         assertTrue( true );
19     }
20 }
```

Writing JUnit tests

- Automated testing helps you ensure that the code is running as expected even after you make changes later
- *Unit tests* focus on the individual code components (units), not on the external environment, end to end system testing, etc.
- Use assert statements to check if methods return expected outcomes (http://junit.sourceforge.net/javadoc/org/junit/Assert.html)
- Testing can get complex and multiple tests may be needed to test a single method
- *Lack of test writing skills is one of the most frequent complaints from organisations in relation to new graduates entering the job market!*

Example test: App.java

- We want to test two methods: **`calculateSum()`** and **`printSum()`**

App.java

```java
package myorg;

// Hello world!
public class App
{
    public int calculateSum(int a, int b) {
        return a + b;
    }
    public void printSum(int a) {
        System.out.println("The sum is: " + a);
    }
    public static void main( String[] args )
    {
        App app = new App();
        app.printSum(app.calculateSum(1,2));
    }
}
```

Example test: AppTest.java

AppTest.java

```java
package myorg;

import static org.junit.Assert.*;
import org.junit.Test;

import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

// Unit test for simple App.
public class AppTest
{
  private final ByteArrayOutputStream
    outContent = new ByteArrayOutputStream();
  private final PrintStream
    originalOut = System.out;

  // Test printing
  @Test
  public void testPrinting()
  {
    System.setOut(new PrintStream(outContent));
    App obj = new App ();
    obj.printSum(3);
    assertEquals("The sum is: 3",
      outContent.toString().trim());
    System.setOut(originalOut);
  }

  // Test addition
  @Test
  public void addNumbers()
  {
    App obj = new App ();
    assertEquals( 2, obj.calculateSum(1,1));
    assertEquals( -2, obj.calculateSum(-1,-1));
    assertEquals( 0, obj.calculateSum(0,0));
  }
}
```

Example test: AppTest.java

AppTest.java

```
1   package myorg;
2
3   import static org.junit.Assert.*;
4   import org.junit.Test;
5
6   import java.io.ByteArrayOutputStream;
7   import java.io.PrintStream;
8
9   // Unit test for simple App.
10  public class AppTest
11  {
12    private final ByteArrayOutputStream
13      outContent = new ByteArrayOutputStream();
14    private final PrintStream
15      originalOut = System.out;
16
17    // Test printing
18    @Test
19    public void testPrinting()
20    {
21      System.setOut(new PrintStream(outContent));
22      App obj = new App ();
23      obj.printSum(3);
24      assertEquals("The sum is: 3",
25        outContent.toString().trim());
26      System.setOut(originalOut);
27    }
28
29    // Test addition
30    @Test
31    public void addNumbers()
32    {
33      App obj = new App ();
34      assertEquals( 2, obj.calculateSum(1,1));
35      assertEquals( -2, obj.calculateSum(-1,-1));
36      assertEquals( 0, obj.calculateSum(0,0));
37    }
38  }
```

Test for the edge cases, ensure your test has good coverage of the domain

Maven output of passed test

- Run `mvn clean test`

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running myorg.AppTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.023 s - in myorg.AppTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.182 s
[INFO] Finished at: 2022-09-13T09:58:19Z
[INFO] ------------------------------------------------------------------------
```

Failed test example

- What happens if you hange the test case in ***AppTest.java*** to produce a failed test case?

```
17   // Test printing
18   @Test
19   public void testPrinting()
20   {
21     System.setOut(new PrintStream(outContent));
22     App obj = new App ();
23     obj.printSum(3);
24     assertEquals("The sum is: 4",
25       outContent.toString().trim());
26     System.setOut(originalOut);
27   }
```

Maven output of failed test

- Run **`mvn clean test`** again

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running myorg.AppTest
[ERROR] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.031 s <<< FAILURE! - in myorg.AppTest
[ERROR] testPrinting(myorg.AppTest)  Time elapsed: 0.006 s  <<< FAILURE!
org.junit.ComparisonFailure: expected:<The sum is: [4]> but was:<The sum is: [3]>
        at myorg.AppTest.testPrinting(AppTest.java:23)

[INFO]
[INFO] Results:
[INFO]
[ERROR] Failures:
[ERROR]   AppTest.testPrinting:23 expected:<The sum is: [4]> but was:<The sum is: [3]>
[INFO]
[ERROR] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.159 s
[INFO] Finished at: 2022-09-13T10:03:52Z
[INFO] ------------------------------------------------------------------------
```

Summary

- Testing and dependency management are essential parts of software development cycle
  - Proper testing helps to catch the bugs at the early phase and decreases the software maintenance cost
  - Dependency management is important to avoid broken dependencies in implementations using third-party libraries
- In Java development, Maven is a commonly used tool for software project management
  - Brief introduction was given for using Maven and JUnit for dependency management and testing Java programs

Questions, comments?