sourcemind

Advanced Java and Spring course

Maven

Problem: Library Management

- How to add libraries to projects?
- How to upgrade them?
- How to share large projects with many libraries?
- How to manage dependency trees?

Maven's Objectives

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

- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Encouraging better development practices

Problems without Maven

- Collecting dependencies and building artifacts
  - Artifact: Something that is produced by the project
- Transitive dependencies: A brings B, B brings C, ...
- Projects sharing common libraries and dependencies
- Upgrading used libraries

Working with Maven

- Download & install Maven
  - Download, unzip
  - Define Maven home $M2_HOME
  - Update PATH=$PATH:$M2_HOME/bin
  - Try mvn --version
- Try mvn in an empty folder

Maven Configs

- Add in .bashrc or .zshrc

```
export M2_HOME=/path/to/maven/apache-maven-3.8.2
export M2=${M2_HOME}/bin
export PATH=$M2:$PATH
export MAVEN_OPTS='-Xmx2048m -Xms2048m'
```

sourcemind

A simple POM.XML

- POM: Project Object Model

```
<project>
    <groupId>com.sourcemind.demo</groupId>
    <artifactId>simple-app</artifactId>
    <version>1.0</version>
    <modelVersion>4.0.0</modelVersion>
    <packaging>jar</packaging>
    <build>
      <pluginManagement>
         <plugins>
           <plugin>
             <groupId>org.apache.maven.plugins</groupId>
             <artifactId>maven-compiler-plugin</artifactId>
             <version>3.11.0</version>
             <configuration>
               <!-- put your configurations here -->
             </configuration>
           </plugin>
         </plugins>
         </pluginManagement>
       </build>
   </project>
```

sourcemind

Creating a Maven project

- Create a folder and pom.xml inside
- Create folder structure: src > main > java
- Define pom.xml and implement a class
- Try mvn compile, run

POM <properties> tag

- A <properties> tag is used to keep version of related stuff in the project

```
<properties>
    <java.version>17</java.version>
    <compiler.version>3.10.1</compiler.version>
</properties>
```

- Value of properties is used by ${java.version} syntax

```
<version>${compiler.version}</version>
```

Specifying Java version

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>${compiler.version}</version>
      <configuration>
          <source>${java.version}</source>
          <target>${java.version}</target>
      </configuration>
      </plugin>
  </plugins>
</build>
```

sourcemind

Generating Maven projects

- Generating archetypes
    - Quickstart
    - Web App
- Exploring pom.xml file

Building with Maven

- Build and install with Maven
- Running tests
- Maven lifecycle and phases
- Maven artifacts
- Maven goals

Maven local repository

- Build and install a JAR
- Find the artifacts in ~/.m2/repository
- Maven settings.xml

sourcemind

Executable JARs with Maven

- Copy dependencies in a single JAR file
- Describe the main class to run
- There are multiple options:
    - Apache Maven Assembly Plugin
    - Apache Maven Shade Plugin
    - Spring Boot Maven Plugin
    - etc.

Apache Maven Assembly Plugin

```
<build><plugins><plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <executions>
        <execution>
            <phase>package</phase>
            <goals>
                <goal>single</goal>
            </goals>
            <configuration><!-- plugin configuration --></configuration>
        </execution>
    </executions>
</plugin></plugins></build>>
```

sourcemind

Apache Maven Assembly Plugin: Configuration

```
<plugin>
<!-- ... -->
<configuration>
<archive>
    <manifest>
        <mainClass>
            com.vahep.demo.MavenDemo
        </mainClass>
    </manifest>
</archive>
<descriptorRefs>
    <descriptorRef>jar-with-dependencies</descriptorRef>
</descriptorRefs>
</configuration>
<!-- ... -->
</plugin>
```

sourcemind

Adding project dependencies

- Use **Javalin** in order to create a simple web server
- Compile, package, and build a single executable JAR
- Run `java -jar my-application-1.0-SNAPSHOT.jar`

Testing with Maven

- Add JUnit dependency
- Write unit tests
- Use mvn test to test.
- Package contains the test phase.
- The option -DskipTests=true bypasses the tests.

Homework 1

- Read about multi-module Maven projects in this article.

In this homework you will work with a project that contains two modules. In the first module (name it core), you will implement a library that performs text analysis, such as counting number of lines and number of words. In the seconds module (name it service), you will implement a command line application that uses the artifact from the first module in order to process text files. Notice that it is not the task of the main application to perform the text analysis. Instead, it should only import the first project as a dependency, and only accept input files from the user.

Publish the first Maven module as a JAR library. It does not need to be executable.

Publish the second Maven module as a single executable JAR file. The usage of the application should be similar to the following:

```
java -jar textapp-1.0.jar input-file.txt

Lines: 10
Words: 50
```