Program 2

Working with Maven: Creating a Maven Project, Understanding the POM File, Dependency Management and Plugins.

Install the Java JDK

- If you haven't installed the **Java JDK** yet, you can follow the link below to download and install it. Download Java JDK from Oracle

Working with Maven is a key skill for managing Java-based projects, particularly in the areas of build automation, dependency management, and project configuration. Below is a guide on creating a Maven project, understanding the POM file, and using dependency management and plugins:

Overview of the Project

Creating a Maven Project

There are a few ways to create a Maven project, such as using the command line, IDEs like IntelliJ IDEA or Eclipse, or generating it via an archetype.

Using IDEs

Most modern IDEs (like IntelliJ IDEA or Eclipse) provide wizards to generate Maven projects. For example, in IntelliJ IDEA:

1. Go to **File > New Project**.
2. Choose **Maven** from the list of project types.
3. Provide the **groupId** and **artifactId** for your project.

3: Understanding the POM File

- The **POM (Project Object Model)** file is the heart of a Maven project. It is an XML file that contains all the configuration used by Maven to build and manage the project.

- The pom.xml file defines dependencies, plugins, goals, and other settings related to the build process.

A basic **pom.xml** structure looks like this:

<!-- Project Information -->

<groupId>com.example</groupId>

<artifactId>my-app</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<name>My Java Application</name>

<description>A simple Java project using Maven</description>

<url>http://www.example.com</url>

- In a Maven project, dependencies and build are essential sections in the pom.xml file that manage how the project is built and which libraries or frameworks it uses.
- The dependencies section in a Maven pom.xml file is used to define external libraries (JAR files) that your project needs in order to compile, test, or run.

- **Dependencies**: Specifies external libraries your project needs to work correctly. Maven will download them automatically and include them in the project.

<!-- Dependencies -->

<dependencies>

<!-- Example of adding a dependency (JUnit for testing) -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.1</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

- In this example, JUnit is specified as a test dependency, meaning Maven will download it and make it available for compiling and running unit tests, but it won't include it in the final product (like a JAR or WAR file).

Build:

The build section in the pom.xml file controls the actual build process of your project. It defines how Maven should compile, package, and deploy the project, and also allows you to configure plugins that perform these tasks.

<!-- Example of adding a plugin (Maven Compiler Plugin) -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration> <source>1.8</source>

<target>1.8</target> </configuration>

</plugin>

- The maven-compiler-plugin is configured to use Java 1.8 for compiling the project.

Building and Testing Your Maven Project

Once your project is set up and your pom.xml is defined, you can use Maven commands to build and test your application.

Common Maven Commands

• Compile the Project:

mvn compile

Run Unit Tests:

mvn test

Package the Application:

mvn package

This command compiles, tests, and packages your code into a JAR file located in the target directory.

• Clean the Project:

mvn clean

This removes any files generated by previous builds.