Lab 2: Working with Maven: Creating a Maven Project, Understanding the POM File, Dependency Management and Plugins

I. 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.

1. **Using Command Line**: (unruled side in record)
   - To create a basic Maven project using the command line, you can use the following command:

```
mvn archetype:generate -DgroupId=com.example -DartifactId=myapp -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

- **groupId:** A unique identifier for the group (usually the domain name).
- **artifactId:** A unique name for the project artifact (your project).
- **archetypeArtifactId:** The template you want to use for the project.
- **DinteractiveMode=false:** Disables prompts during project generation.

This will create a basic Maven project with the required directory structure and **pom.xml** file.

2. **Using IDEs**

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

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

II. 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 details about the project. Below is an example of a simple POM file:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>my-project</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <dependencies>
    <!-- Dependencies go here -->
  </dependencies>

  <build>
    <plugins>
      <!-- Plugins go here -->
    </plugins>
  </build>

</project>
```

Key element in pom.xml:

- <**groupId**>**:** The group or organization that the project belongs to.
- <**artifactId**>**:** The name of the project or artifact.
- <**version**>**:** The version of the project (often follows a format like 1.0-SNAPSHOT).
- <**packaging**>**:** Type of artifact, e.g., jar, war, pom, etc.
- <**dependencies**>**:** A list of dependencies the project requires.
- <**build**>**:** Specifies the build settings, such as plugins to use.\

III. Dependency Management

Maven uses the <dependencies> tag in the pom.xml to manage external libraries or dependencies that your project needs. When Maven builds the project, it will automatically download these dependencies from a repository (like Maven Central).

Example of adding a dependency:

```
<dependencies>
  <dependency>
    <groupId>org.apache.commons</groupId>
```

```
      <artifactId>commons-lang3</artifactId>
      <version>3.12.0</version>
    </dependency>
</dependencies>
```

- **Transitive Dependencies**

  Maven automatically resolves transitive dependencies. For example, if you add a library that depends on other libraries, Maven will also download those.

- **Scopes**

  Dependencies can have different scopes that determine when they are available:

  - **compile** (default): Available in all build phases.
  - **provided**: Available during compilation but not at runtime (e.g., a web server container).
  - **runtime**: Needed only at runtime, not during compilation.
  - **test**: Required only for testing.

IV. Using Plugins

Maven plugins are used to perform tasks during the build lifecycle, such as compiling code, running tests, packaging, and deploying. You can specify plugins within the <build> section of your pom.xml.

- **Adding Plugins**

  You can add a plugin to your pom.xml like so:

```
<build>
  <plugins>
    <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>
  </plugins>
</build>
```

In this example, the **maven-compiler-plugin** is used to compile Java code and specify the source and target JDK versions.

1. **Common Plugins**
   - **maven-compiler-plugin**: Compiles Java code.
   - **maven-surefire-plugin**: Runs unit tests.
   - **maven-jar-plugin**: Packages the project as a JAR file.
   - **maven-clean-plugin**: Cleans up the target/ directory.

2. **Plugin Goals** Each plugin consists of goals, which are specific tasks to be executed. For example:
   - **mvn clean install:** This will clean the target directory and then install the package in the local repository.
   - **mvn compile:** This will compile the source code.
   - **mvn test:** This will run unit tests.

Working with Maven Project (*unruled side in record)*

Note: *Always create separate folder to do any program***.**

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. Screenshot Tip: Capture the listing of the target directory showing the JAR file.

-Clean the Project:

```
mvn clean
```

This removes any files generated by previous builds.