Assignment 1: Build Lifecycle Demonstrate the use of Maven lifecycle phases (clean, compile, test, package, install, deploy) by executing them on a sample project and documenting what happens in each phase. write this whole in java.

Answer:

1. Create a Maven Project

First, generate a new Maven project using the Maven archetype**:**

mvn archetype:generate -DgroupId=com.example -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false cd my-app

2. Project Structure

The generated project structure looks like this:

```
my-app
|  pom.xml
└──src
   └──main
   |  └──java
   |     └──com
   |        └──example
   |           |  App.java
   └──test
      └──java
         └──com
            └──example
               |  AppTest.java
```

3. pom.xml File

Here's the content of the **pom.xml** file for the project:

```
<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-app</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <!-- Dependency for JUnit (for testing) -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <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>
</project>
```

4. Java Classes

App.java:

```
package com.example;

public class App {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

AppTest.java:

```
package com.example;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AppTest {
    @Test
    public void testApp() {
        assertTrue(true);
    }
}
```

5. Maven Lifecycle Phases

Let's execute each lifecycle phase and document what happens.

Clean Phase

Command:

mvn clean

What happens:

- The clean phase deletes the target directory, which contains all the files generated by the previous build, including compiled classes and packaged JAR files.

Compile Phase

Command:

mvn compile

What happens:

- The compile phase compiles the source code in the src/main/java directory. The compiled .class files are placed in the target/classes directory.

Test Phase

Command:

mvn test

What happens:

- The test phase compiles and runs the test code located in the src/test/java directory. The test results are displayed in the console. Maven uses the JUnit framework to run the tests.

Package Phase

Command:

mvn package

What happens:

- The package phase creates a JAR file containing the compiled classes and resources. The JAR file is placed in the target directory. The file is named my-app-1.0-SNAPSHOT.jar.

Install Phase

Command:

mvn install

What happens:

- The install phase installs the JAR file into the local Maven repository, typically located in the ~/.m2/repository directory. This allows other Maven projects on the same machine to use this JAR as a dependency.

Deploy Phase

Command:

mvn deploy

What happens:

- The deploy phase copies the packaged JAR file to a remote repository for sharing with other developers and projects. This requires configuration of a remote repository in your pom.xml or settings.xml.

Example: Full Build Lifecycle Execution

You can execute all the phases in sequence by running:

mvn clean compile test package install deploy

This command will:

1. **Clean**: Delete the target directory.
2. **Compile**: Compile the source code.
3. **Test**: Compile and run the tests.
4. **Package**: Create a JAR file.
5. **Install**: Install the JAR file to the local repository.
6. **Deploy**: Deploy the JAR file to a remote repository (if configured).

Summary

Here is a summary of what happens in each phase:

1. **Clean**: Deletes the target directory.
2. **Compile**: Compiles the source code and generates .class files.
3. **Test**: Compiles and runs the test code, reports the results.
4. **Package**: Packages the compiled code into a JAR file.
5. **Install**: Installs the JAR file into the local Maven repository.
6. **Deploy**: Copies the JAR file to a remote repository (requires configuration).

This demonstrates the use of Maven's build lifecycle phases on a sample Java project. Maven provides a powerful and flexible build system that can handle all aspects of project management, from dependency resolution to deployment.