Step 1: Create a Maven Project

1. Open your terminal or command prompt.
2. Use the following command to create a new Maven project:

```
mvn archetype:generate -DgroupId=com.example -DartifactId=Task1
-DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false
```

This will generate a basic Maven project structure.

Step 2: Navigate to the Project Directory

```
cd Task1
```

Step 3: Execute Maven Lifecycle Phases

Clean Phase

The `clean` phase removes all files generated by the previous build.

```
mvn clean
```

Documentation:

- The `target` directory, which contains the compiled classes and other generated files, is deleted.

Compile Phase

The `compile` phase compiles the source code of the project.

```
mvn compile
```

Documentation:

- The source code located in the `src/main/java` directory is compiled into the `target/classes` directory.

Test Phase

The `test` phase runs the unit tests of the project.

```
mvn test
```

Documentation:

- The test source code located in the `src/test/java` directory is compiled.
- The compiled tests are executed, and the results are displayed in the console.
- Test reports are generated in the `target/surefire-reports` directory.

Package Phase

The `package` phase packages the compiled code into a distributable format, such as a JAR or WAR file.

```
mvn package
```

Documentation:

- The compiled code is packaged into a JAR file located in the `target` directory (e.g., `Task1-1.0-SNAPSHOT.jar`).

Install Phase

The `install` phase installs the package into the local repository, which can be used as a dependency in other projects.

```
mvn install
```

Documentation:

- The JAR file is installed into the local Maven repository (usually located in `~/.m2/repository`).

Deploy Phase

The `deploy` phase copies the final package to the remote repository for sharing with other developers and projects. This requires configuring a remote repository in the `pom.xml` file.

```
mvn deploy
```

Documentation:

- The packaged JAR is deployed to a remote repository. This step usually requires proper repository configuration and credentials.

Sample Project Structure

css

```
Task1
├── pom.xml
└── src
    ├── main
    │   └── java
    │       └── com
    │           └── example
    │               └── App.java
    └── test
        └── java
            └── com
                └── example
                    └── AppTest.java
```

Sample `pom.xml`

Here is a basic `pom.xml` for the project:

xml

```
<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>Task1</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>Task1</name>
    <url>http://maven.apache.org</url>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</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>
```

Sample Java Files

App.java

```
package com.example;

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

AppTest.java

```
package com.example;

import static org.junit.Assert.assertTrue;

import org.junit.Test;

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

This setup demonstrates the use of Maven lifecycle phases. You can run each phase and observe the results to understand what happens during each phase.