Task 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.

Maven is based around the central concept of a build lifecycle. What this means is that the process for building and distributing a particular artifact (project) is clearly defined.

For the person building a project, this means that it is only necessary to learn a small set of commands to build any Maven project, and the POM will ensure they get the results they desired.

There are three built-in build lifecycles: default, clean and site. The `default` lifecycle handles your project deployment, the `clean` lifecycle handles project cleaning, while the `site` lifecycle handles the creation of your project's web site.

A Build Lifecycle is Made Up of Phases

Each of these build lifecycles is defined by a different list of build phases, wherein a build phase represents a stage in the lifecycle.

For example, the default lifecycle comprises of the following phases (for a complete list of the lifecycle phases, refer to the Lifecycle Reference):

- `validate` - validate the project is correct and all necessary information is available
- `compile` - compile the source code of the project
- `test` - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- `package` - take the compiled code and package it in its distributable format, such as a JAR.
- `verify` - run any checks on results of integration tests to ensure quality criteria are met
- `install` - install the package into the local repository, for use as a dependency in other projects locally
- `deploy` - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

These lifecycle phases (plus the other lifecycle phases not shown here) are executed sequentially to complete the `default` lifecycle. Given the lifecycle phases above, this means that when the default lifecycle is used, Maven will first validate the project, then will try to compile the sources, run those against the tests, package the binaries (e.g. jar), run integration tests against that package, verify the integration tests, install the verified package to the local repository, then deploy the installed package to a remote repository.**Prerequisites:**

1. Java Development Kit (JDK) installed.
2. Maven installed.

3. An Integrated Development Environment (IDE) like IntelliJ IDEA or Eclipse (optional but recommended).

Step 1: Install Maven

If Maven is not already installed, download it from the Maven official website and follow the installation instructions.

Step 2: Create a Maven Project

You can create a new Maven project from the command line or using an IDE.Step 3: Project Structure

Step 3: Project Structure

Step 4: Understanding pom.xml:

The pom.xml (Project Object Model) file is the core of a project's configuration in Maven. It contains information about the project and config

Step 5: Building the Project

To compile the project, navigate to the project directory in the terminal and run:mvn compile

Step 6: Running Testsmvn test

To run tests (using JUnit in this case), run:mvn test

Step 7: Packaging the Project

To package the compiled code into a JAR file, run:mvn package

Step 8: Running the Application

Sample Project Setup:mvn archetype:generate -DgroupId=com.example -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMcd my-app Mode=falsecd

Cd my-app

1. Clean Phase

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

2. Compile Phase

The `compile` phase compiles the source code of the projec t:mvn compile

What Happens:

- Compiles the Java source files located in `src/main/java`
- .The compiled `.class` files are placed in the `target/classes` directory.

3. Test Phase

The `test` phase runs the unit tests using a suitable testing framework (e.g., JUnit) :mvn test

What Happens:

- Compiles the test source files located in `src/test/java`.
- Executes the compiled test classes.
- The results of the tests are displayed in the console and can be found in the `target/surefire-reports` directory.

4. Package Phase

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

What Happens:

- Takes the compiled code from `target/classes` and packages it into a JAR file.
- The JAR file is placed in the `target` directory, named `my-app-1.0-SNAPSHOT.jar`.

5. Install Phase

The `install` phase installs the package into the local Maven repository, which is typically located at `~/.m2/repository` :mvn install

What Happens:

- Copies the packaged JAR file into the local repository.
- This makes the project available for other projects locally that use this project as a dependency.

6. Deploy Phase

The `deploy` phase copies the final package to a remote repository for sharing with other developers and projects. Note: This step typically requires access to a configured remote repository :mvn deploy

What Happens:

- Deploys the packaged JAR file to the specified remote repository.

- This makes the package available to other developers and projects with access to that repository.