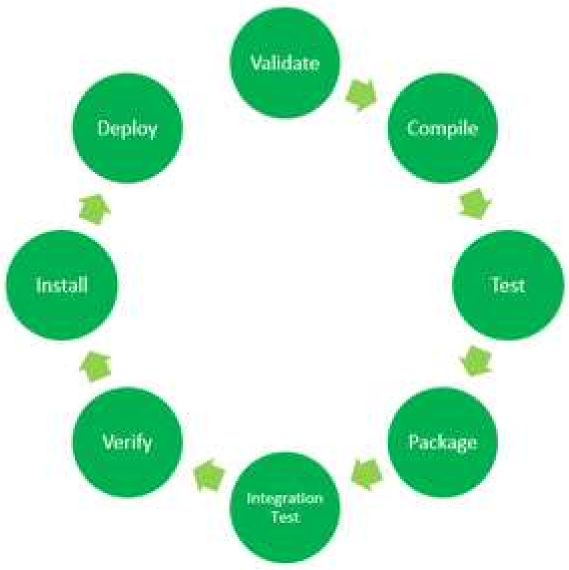
Maven Lifecycle: Below is a representation of the default Maven lifecycle and its 8 steps: Validate, Compile, Test, Package, Integration test, Verify, Install, and Deploy.



8 Phases of the Default Maven Lifecycle

The default Maven lifecycle consists of 8 major steps or phases for compiling, testing, building and installing a given Java project as specified below:

1. **Validate:** This step validates if the project structure is correct. For example – It checks if all the dependencies have been downloaded and are available in the local repository.
2. **Compile:** It compiles the source code, converts the .java files to .class, and stores the classes in the target/classes folder.
3. **Test:** It runs unit tests for the project.
4. **Package:** This step packages the compiled code in a distributable format like JAR or WAR.
5. **Integration test:** It runs the integration tests for the project.
6. **Verify:** This step runs checks to verify that the project is valid and meets the quality standards.
7. **Install:** This step installs the packaged code to the local Maven repository.
8. **Deploy:** It copies the packaged code to the remote repository for sharing it with other developers.

Maven follows a sequential order to execute the commands where if you run step *n*, all steps preceding it (Step 1 to *n-1*) are also executed. For example – if we run the Installation step (Step 7), it will validate, compile, package and verify the project along with running unit and integration tests (Step 1 to 6) before installing the built package to the local repository.

Maven Commands:

- **mvn clean:** Cleans the project and removes all files generated by the previous build.
- **mvn compile:** Compiles source code of the project.
- **mvn test-compile:** Compiles the test source code.
- **mvn test:** Runs tests for the project.

- **mvn package:** Creates JAR or WAR file for the project to convert it into a distributable format.
- **mvn install:** Deploys the packaged JAR/ WAR file to the local repository.
- **mvn site:** generate the project documentation.
- **mvn validate:** validate the project's POM and configuration.
- **mvn idea:idea:** generate project files for IntelliJ IDEA or Eclipse.
- **mvn release:perform:** Performs a release build.
- **mvn deploy:** Copies the packaged JAR/ WAR file to the remote repository after compiling, running tests and building the project.
- **mvn archetype:generate:** This command is used to generate a new project from an archetype, which is a template for a project. This command is typically used to create new projects based on a specific pattern or structure.
- **mvn dependency:tree:** This command is used to display the dependencies of the project in a tree format. This command is typically used to understand the dependencies of the project and troubleshoot any issues.

Generally when we run any of the above commands, we add the **mvn clean** step so that the target folder generated from the previous build is removed before running a newer build. This is how the command would look on integrating the *clean* step with *install* phase:

```
mvn clean install
```

Similarly, if we want to run the step in debug mode for more detailed build information and logs, we will add **-X** to the actual command. Hence, the *install* step with debug mode on will have the following command:

```
mvn -X install
```

Consider a scenario where we do not want to run the tests while packaging or installing the Java project. In this case, we use **-DskipTests** along with the actual command. If we need to run the *install* step by skipping the tests associated with the project, the command would be:

```
mvn install -DskipTests
```