**What is Maven and Usage?**

Maven is a powerful project management tool that is based on POM (project object model). It is used for projects build, dependency and documentation. It simplifies the build process like ANT. But it is too much advanced than ANT.  
In short terms we can tell maven is a tool that can be used for building and managing any Java-based project. maven make the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project.

We can easily build a project using maven-

1. We can add jars and other dependencies of the project easily using the help of maven.
2. Maven provides project information (log document, dependency list, unit test reports etc.)
3. Maven is very helpful for a project while updating central repository of JARs and other dependencies.
4. With the help of Maven we can build any number of projects into output types like the JAR, WAR etc without doing any scripting.
5. Using maven we can easily integrate our project with source control system (such as Subversion or Git).

**How to create a Maven Project ?**

After opening Eclipse, choose the workspace you want to use.

The Eclipse window opens on the screen. Since there aren’t any projects yet, complete the following steps:

1. Go to the File option In the drop-down menu, select New Select the Project option

If you want to create a Java project, you can select the “Java Project” option. Since we are not creating a Java project specifically,

we have chosen the “Project” option.

2. The dialog box that appears on the screen will display different types of projects.

Select the Maven Project option,Click on Next,new\_project

A dialog box will appear. Select the default workspace.

Click on “Next”,Several Group IDs, Artifact IDs, and Versions will then appear.

3. Select a plugin there and click on “Next”

new-maven-project

In the next dialog box that appears, you’ll complete the following steps:

Enter the Group ID

“com.mehul.project”

Enter the Artifact ID

“mavenproject”

4. The version will appear on the screen

These items can all be modified at a later time if needed.

Click on “Finish”

maven-project

5. The project is now created.

Open the pom.xml file

You can see all the basic information that you have entered on the screen, such as the Artifact ID, Group ID, etc.

You can see the junit dependencies have been added.

This process takes place by default in Eclipse. There will also be some by default test cases.

default-test

6. There you can find AppTest.java to be a default test case.

When you click on that, you can see the test cases written in JUnit on your Eclipse screen.

package-exploler

When it comes to adding more test cases, it will depend on the user, but these test cases and commands can easily be added in the workspace.

If we try to remove certain dependencies from our file, we will receive error messages.

7. To troubleshoot this, complete the following steps:

Go to another tab: mavenproject/pom.xml

Delete any dependencies

8. Save the file

Immediately, there will be several error messages in the AppTest.java.

Return to the previous screen and undo the deletion. The errors that occurred will disappear.

The demo shows the relationship between the dependencies and the Eclipse. When a Maven project is selected,all such dependencies are automatically downloaded. If any dependencies are not present, Eclipse will show errors.

**Command used in maven?**

**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 deploy:** Copies the packaged JAR/ WAR file to the remote repository after compiling, running tests and building the project.

Maven Commands

**mvn clean:** Cleans the project and removes all files generated by the previous build.

**mvn –version:** Prints out the version of Maven you are running.

**mvn clean package**

mvn clean package -Dmaven.test.skip=true

mvn verify

mvn clean verify

mvn install

mvn install -Dmaven.test.skip=true

mvn clean install

mvn clean install -Dmaven.test.skip=true

mvn dependency:copy-dependencies

mvn clean dependency:copy-dependencies

mvn dependency:tree

mvn dependency:tree -Dverbose

mvn dependency:tree -Dincludes=com.fasterxml.jackson.core

mvn dependency:tree -Dverbose -Dincludes=com.fasterxml.jackson.core

mvn dependency:build-classpath

$ mvn clean

$ mvn compiler:compile

$ mvn compiler:testCompile

$ mvn package

$ mvn install

**Maven Life cycle?**

The default Maven life cycle 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 target/classes folder.
3. **Test:** It runs unit tests for the project.
4. **Package:** This step packages the compiled code in attributable 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.