**Name –** Debajyoti Majumdar

**Employee ID -** 8159

**Maven Notes**

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information. Using maven we can build and manage any Java based project.

POM stands for Project Object Model. It is fundamental unit of work in Maven. It is an XML file that resides in the base directory of the project as pom.xml.

The POM contains information about the project and various configuration detail used by Maven to build the project(s).

POM also contains the goals and plugins. While executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, and then executes the goal. Some of the configuration that can be specified in the POM are following −

* project dependencies
* plugins
* goals
* build profiles
* project version
* developers
* mailing list

Before creating a POM, we should first decide the project group (groupId), its name (artifactId) and its version as these attributes help in uniquely identifying the project in repository.

|  |  |
| --- | --- |
|  |  |
| **Sr.No.** | **Node & Description** |
| 1 | Project root  This is project root tag. You need to specify the basic schema settings such as apache schema and w3.org specification. |
| 2 | Model version  Model version should be 4.0.0. |
| 3 | groupId  This is an Id of project's group. This is generally unique amongst an organization or a project. For example, a banking group com.company.bank has all bank related projects. |
| 4 | artifactId  This is an Id of the project. This is generally name of the project. For example, consumer-banking. Along with the groupId, the artifactId defines the artifact's location within the repository. |
| 5 | version  This is the version of the project. Along with the groupId, It is used within an artifact's repository to separate versions from each other. For example −  com.company.bank:consumer-banking:1.0  com.company.bank:consumer-banking:1.1. |

**Super POM**

The Super POM is Maven’s default POM. All POMs inherit from a parent or default (despite explicitly defined or not). This base POM is known as the Super POM, and contains values inherited by default.

Maven use the effective POM (configuration from super pom plus project configuration) to execute relevant goal. It helps developers to specify minimum configuration detail in his/her pom.xml. Although configurations can be overridden easily.

An easy way to look at the default configurations of the super POM is by running the following command: mvn help:effective-pom

**What is Build Lifecycle?**

A Build Lifecycle is a well-defined sequence of phases, which define the order in which the goals are to be executed. Here phase represents a stage in life cycle. As an example, a typical Maven Build Lifecycle consists of the following sequence of phases.

|  |  |  |
| --- | --- | --- |
| **Phase** | **Handles** | **Description** |
| prepare-resources | resource copying | Resource copying can be customized in this phase. |
| validate | Validating the information | Validates if the project is correct and if all necessary information is available. |
| compile | compilation | Source code compilation is done in this phase. |
| Test | Testing | Tests the compiled source code suitable for testing framework. |
| package | packaging | This phase creates the JAR/WAR package as mentioned in the packaging in POM.xml. |
| install | installation | This phase installs the package in local/remote maven repository. |
| Deploy | Deploying | Copies the final package to the remote repository. |

There are always pre and post phases to register goals, which must run prior to, or after a particular phase.

When Maven starts building a project, it steps through a defined sequence of phases and executes goals, which are registered with each phase.

**Maven has the following three standard lifecycles −**

* clean
* default(or build)
* site

**What is Build Profile?**

A Build profile is a set of configuration values, which can be used to set or override default values of Maven build. Using a build profile, you can customize build for different environments such as Production v/s Development environments.

Profiles are specified in pom.xml file using its activeProfiles/profiles elements and are triggered in variety of ways. Profiles modify the POM at build time, and are used to give parameters different target environments (for example, the path of the database server in the development, testing, and production environments).

**Types of Build Profile**

Build profiles are majorly of three types.

|  |  |
| --- | --- |
| **Type** | **Where it is defined** |
| Per Project | Defined in the project POM file, pom.xml |
| Per User | Defined in Maven settings xml file (%USER\_HOME%/.m2/settings.xml) |
| Global | Defined in Maven global settings xml file (%M2\_HOME%/conf/settings.xml) |

**Profile Activation**

A Maven Build Profile can be activated in various ways.

* Explicitly using command console input.
* Through maven settings.
* Based on environment variables (User/System variables).
* OS Settings (for example, Windows family).
* Present/missing files.

**What is a Maven Repository?**

In Maven terminology, a repository is a directory where all the project jars, library jar, plugins or any other project specific artifacts are stored and can be used by Maven easily.

Maven repository are of three types. The following illustration will give an idea regarding these three types.

* local
* central
* remote

**What are Maven Plugins?**

Maven is actually a plugin execution framework where every task is actually done by plugins. Maven Plugins are generally used to −

* create jar file
* create war file
* compile code files
* unit testing of code
* create project documentation
* create project reports

A plugin generally provides a set of goals, which can be executed using the following syntax −

mvn [plugin-name]:[goal-name]

For example, a Java project can be compiled with the maven-compiler-plugin's compile-goal by running the following command.

mvn compiler:compile

**Plugin Types**

Maven provided the following two types of Plugins −

|  |  |
| --- | --- |
| **Sr.No.** | **Type & Description** |
| 1 | Build plugins  They execute during the build process and should be configured in the <build/> element of pom.xml. |
| 2 | Reporting plugins  They execute during the site generation process and they should be configured in the <reporting/> element of the pom.xml. |

**Following is the list of few common plugins −**

|  |  |
| --- | --- |
| **Sr.No.** | **Plugin & Description** |
| 1 | clean  Cleans up target after the build. Deletes the target directory. |
| 2 | compiler  Compiles Java source files. |
| 3 | surefire  Runs the JUnit unit tests. Creates test reports. |
| 4 | jar  Builds a JAR file from the current project. |
| 5 | war  Builds a WAR file from the current project. |
| 6 | javadoc  Generates Javadoc for the project. |
| 7 | antrun  Runs a set of ant tasks from any phase mentioned of the build. |

Maven uses archetype plugins to create projects. To create a simple java application, we'll use maven-archetype-quickstart plugin. In example below, we'll create a maven based java application project in C:\MVN folder.

Let's open the command console, go to the C:\MVN directory and execute the following mvn command. Make sure that C:\MVN directory is empty before running the command.

C:\MVN>mvn archetype:generate

-DgroupId = com.companyname.bank

-DartifactId = consumerBanking

-DarchetypeArtifactId = maven-archetype-quickstart

-DinteractiveMode = false