Maven

Same like maven, so many build management tool are present

1. Grade
2. Ant

Introduction to maven:

Website—> maven.apache.org
If we are using any IDEs, maven is inbuilt available to set up.
Else we need to set up the path of the maven in command line.
Before setting up the maven, first we need to set

- JAVA_HOME(for java)
- M2_HOME(for maven)

1. JAVA_HOME = (path of JDk)
2. M2_HOME = (path of maven bin file)—> should be downloaded before from maven website.
3. We need set path for maven in global variables settings
    1. This command is temporary path,
    2. Set PATH= %PATH%; %M2_HOME%/bin;
4. mvn -v —> for maven version, will get the details when the environment is set perfectly.
5. Bin, conf, lib, boot —> folders present in the maven folder which is extracted from zip file.

What is maven

It is a project management tool and comprehension tool and as such provides a way to help with managing.
-> open source tool to support java projects or some other program projects.
-> automated tool used to manage the complex and ambiguous processes involved in the project.
->Used to create or manage:

- Builds
- Documentation
- Reporting
- Dependencies
- SCMs
- Releases
- Distribution

-> Maven is based on the concept of a project object model(POM), Maven can manage a project's build, reporting and documentation from a central pieces of information.
-> The advantage of automating the build process is that you minimize the risk of human making errors while building the software manually.
-> Additionally, an automated build tool is typically faster than a human performing the same steps manually.

Maven Objectives

1. Making the build process easy
2. Providing a uniform build system
3. Providing a quality project information
4. Encouraging better development practices
5. One artifact

Maven Features:

1. Refer online

Convention over Configuration

-> refer online

Maven- How it works?

Maven——————————>	POM file	
	<--Build Life Cycles--> Phases, Goals	
	Dependencies(JARS)----->	Maven Local Repository
	Build Plugins	
	Build Profiles	

1. Reads pom.xml
2. Downloads dependencies into local repository
3. Executes life cycles, build phases and/or goals.
4. Executes plugins

—> All executed according to selected build profile

Project Object Model(POM)

—> In our project, we need to create a pom.xml file to maintain the builds.
—> Below code is for basic configuration of maven pom.xml file

<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.virtusa.employee</groupId>
        <artifactId>employee</artifactId>
        <version>1.0.0</version>
        <packaging>jar</packaging>
    </project>
```

—> The above configuration code will be changed for time being. But, structure will be almost similar.
—> modelVersion is the version of Pom which we are using in the maven project.
—> groupId, artifactId and version are all called as maven co-ordinates.
—> above groupId is given as "com.virtusa.employee". We can give any name as per our convenience. groupId like an packagename of project. And artifactId is like folderName.
—> For any maven projects we create, we need three coordinates mentioned like above code.
—> <packaging> is exceptional. We can provide or not, no issue.
——————————————————————————32:00——————————————————
—> after setting up the maven, open command line and go the directory where this pom.xml is saved.
—>Give the below command

```
mvn compile
```

—> the above command will do something at the backend and at last we will get build success.

```
mvn validate
```

—> It will give a status —>"building {artifact id name}"
—> after mvn command whatever we give, it will be considered as lifecycle phases.
—> We should give valid lifecycle phases.
—> some valid lifeCycle phases are validate, initialize, generate-sources, process-sources, process-resources, compile, process-classes, generate-test-sources, process-test-sources e.t.c,.

Maven Build Life Cycles, Phases, and Goals

The build process in a maven is split u[lto build life cycles, phases and goals.
—> A build life cycle may consists of a sequence of build phases, and each build phase consists of a sequence of plugin goals.

—> When we run a Maven, you pass a command to maven. This command is the name of a build life cycle, phase or goal.
—> If a life cycle is requested to be executed, all the build phases in that life cycle are executed.
—> If a build phase is requested to be executed, all build phases before it in the pre-defined sequence of build phases are executed too.

Maven Build Life Cycles

Maven has 3 build life cycles. These are:

1. Default
2. Clean
3. Site

—> Maven will follow a standard structure of maintaining the project. Only when we follow that structure, then the source files will be compiled.
—> if our project folder is “Employee”
—> In this folder, the structure will be as below
—> **src-> main -> java -> source files**
—>pom.xml
—> Only The source files will be compiled by maven when we give the command man compile.
—> After successfully compiling the source files, the maven will create a target folder and saves the .class files in that folder.

————————————————————-01:00:45————————————

—> Clean life cycle is used to clean the entire project to get it to the fresh start again.
—> All compiled files, reports everything will be cleared and only src and pom files will be present.
—> **mvn clean** ——> will clear all logs of the project and make it new again.
—> **mvn clean compile** ——> will clear all logs and freshly start the build again.
Mvn help:effective-pom —> is used to get the manual documentation and conventions of the maven. Will get all the configurations of the maven
—> we can see the conventions of source directories, testSourcedirectories, and all directories.
—> maven uses **plugins** to compile, test, and to create the reports and to maintain the project
—-> we can see the plugins part also in the mvn help:effective-pom
—-> A phase and goals are the part of the plugin.

————————————————————01:40:00————————————————————

—> **mvn package** ——> will run all the previous phases and at last **create a .jar file** to make it as a build.
—> package is the phase in default life cycle.

We can run individual phase without running the previous phases in the life cycle by using the command as follows:

—> **mvn {pluginname}:{goal}**

—> for example, if you want to run the test phase without running the previous phases in its life cycle. then,

—> **mvn surefire:test**

---> here surefire is the plugin associated with test phase and the test is the goal name.

—> try this commands to know about the packages well

Mvn help:help

Mvn compiler:help

Mvn surefire:help

Dependencies

—-> Main things in the dependencies are:

1. GroupId
2. ArtifactId
3. Version

When we add a dependency in the maven. Maven will first check the groupId(treated like a package name) in the local repo if it can find, then it will check for the related Artifact id(treated like module) and so on

—> If it couldn't find the groupID in the local repo, then it will search it from central repo in the sequence of GroupId—> ArtifactID —> Version. Then download that respective required extension file (like .jar, .sha1 or others).

—-> **Maven will maintain a local repository with a name m2** in our system. When it download any .jar file from the central repo, it will save them in this local repo.

—-> so each .jar file is downloaded only once in the maven. The particular .jar file is used in every project if needed. But not downloaded again for another project.

—> The location of the local repo is **c/user/{userName}/.m2/**

—-> IF our downloaded dependency is internally using another dependency then, that dependency also got downloaded along with our required dependency which is called **transitive dependencies.**

—> Maven can manage all these dependencies downloading. It won't download any dependency twice.