What is Maven

Introduction to Maven

Maven began its life in Apache's `Jakarta Alexandria Project` in 2001. After 5 months of development, it was implemented in the project, Jakarta Turbine. Maven's contribution to the project was to `simplify the building processes`.

At that time, there were several layers for the project like the persistence layer, web layer, or services layer. These layers needed similar building processes but each layer contained different ANT build files and complicated templates of ANT. So, this complicated ANT environment demanded a change and the developers decided to develop and use Maven as a solution.

As a project management tool, Apache Maven helps to `build multiple projects easily, publish documentation for the projects, accomplish an easy deployment, share JARs across several other projects and help in collaboration with development teams.`

Maven can **manage a software project's builds with various versions, compile source code into binary, download dependencies, put additional JAR (Java Archive) files on a classpath, add documentation, run tests, package compiled code into deployable artifacts such as `JAR, WAR, EAR and ZIP files`, and deploy these artifacts to an application server or a repository**.

Also, Maven can automate these tasks and eliminate the risks of manual compiling. If we made all these tasks mentioned manually, we could have ended up with many problems. Consider a project that grows day by day. Some of your struggles might have included the followings:

- Dependency management and adding so many JAR files into your project which is maybe the biggest problem and cumbersome task. For example, if we need some of the big and commonly used dependencies or frameworks, we should add sets of or hundreds of JAR files in each project.
- Building the right structure for your project is also an issue. You must put the right files into the right folders. Otherwise, you can't make your project work seamlessly. Also, the developers new to the project would need much more time to get used to the project structure.
- On the way of releasing the project, also building and deploying would cause you a lot of trouble.

Convention over Configuration

Maven adopts `Convention over Configuration.` That means that the developers `should not create building environment and should not deal with build processes manually.` They must not struggle with the smallest configuration item.

Maven handles the cumbersome and detailed processes for developers. It starts with a clean and well-defined project structure. It helps to define life-cycle goals within plugins (will be discussed later) and dependencies. Plugins, as part of the `POM (Project Object Model)` file, control the stages like compiling, building, testing, or packaging.

However, it's not necessary for a developer to fully understand how the plugins work because you can start a correctly structured and configured project with just a few clicks. There is only one critical thing that a developer should do. That is to use the directory structure and files in a proper way. Especially **the POM file is the most important one** for the overall health of the project.

Features of Maven

- It's **easy to start** with Maven.
- You can start with a **variety of options** according to your needs.
- It has the **same structure** across a variety of different projects.
- It's **easy to integrate** into a developing team when they are working on Maven.
- It has a **powerful dependency management tool**.
- There is a **large repository of libraries**. You can find what you want among them.
- You have the chance to reach **extra features with plugins** in Java or scripting languages. You can also write plugins.
- Maven can give **different outputs** like a jar, ear, war, or metadata for the same project.
- Maven can **generate a website and a PDF** with the documentation in the project.
- Maven can **integrate with your source control system** such as CVS and manages the release of a project.
- Maven can **support the older versions**.

Directory Structure

Having a common project directory layout would bring the developers to `"Oh, I'm at home."` feeling in separate Maven projects.

The directory layout expected by Maven and the directory layout created by Maven is as in the picture below. So a maven project's directory structure should conform to this structure. But it's also possible to modify the structure.

```
└──maven-project
    ├──pom.xml
    ├──README.txt
    ├──NOTICE.txt
    ├──LICENSE.txt
    └──src
        ├──main
        |   ├──java
        |   ├──resources
        |   ├──filters
        |   └──webapp
        ├──test
        |   ├──java
        |   ├──resources
        |   └──filters
        ├──it
        ├──site
        └──assembly
```

Directory	Explanation
src/main/java	Application/Library source code
src/main/resources	Application/Library resources
src/main/filters	Resource filter files
src/main/webapp	Web application sources
src/test/java	Test source code
src/test/resources	Test resources
src/test/filters	Test resource filter files
src/it	Integration Tests (primarily for plugins)
src/assembly	Assembly descriptors
src/site	Site
LICENSE.txt	Project's license
NOTICE.txt	Notices and attributions required by libraries that the project depends on
README.txt	Project's readme

In this structure, the most important file is the pom.xml file. POM stands for Project Object Model and It defines the project's `configuration details` which are consumed by Maven for building the project. In addition to pom.xml, there are also text-based documents like README.txt and LICENCE.txt. These are also under the root directory for the sake of their importance.

Here there are two main directories which are `"src"` and `"target"` directories. The target directory is a destination point for `any kind of product of a build`. On the other hand, src is the directory where all the `source code` and `related files` for building the project are stored. Src directory has 2 directories which are main and test. The main folder is used for the application source code, application resources, and application resource filter files. Whereas the test directory is used for housing the unit test code and test resources and filter files.