UNIT II COMPILE AND BUILD USING MAVEN & GRADLE 6

Introduction, Installation of Maven, POM files, Maven Build lifecycle, Build phases(compile build, test, package) Maven Profiles, Maven repositories(local, central, global),Maven plugins, Maven create and build Artifacts, Dependency management, Installation of Gradle, Understand build using Gradle

INTRODUCTION TO MAVEN

Maven - 'to understand, to comprehend'. Apache Maven is a powerful project management and build automation tool used mainly for Java-based projects. It is built on the Project Object Model (POM), it handles tasks like compiling code, managing dependencies, and generating documentation. Compared to older tools like Ant, Maven offers a more advanced, convention-based approach that reduces the need for manual configuration. It streamlines the build process and helps developers better understand and manage complex Java applications.

Reasons for using Maven:

Dependency Management: Maven manages all the project dependencies manually. It includes the complex tasks such as downloading library files, ensuring their compatibility and managing the version control.

Build Automation: Struts, Hibernates, Spring framework create jar and war files manually. But maven creates it automatically.

Standardized Project Structure: Maven creates standardized project structure which can be easily used by developers.

IDE Integration: IDE such as Eclipse, IntelliJ, Netbeans provide integration with Maven and allows importing, building and managing Maven projects.

Plugins and Community support: Maven support plugins that simplify various tasks such as deployment and documentation generation

Features of Maven:

1. Maven adds 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 the central repository of JARs and other dependencies.
4. Maven provides superior dependency management including automatic updating, dependency closures.
5. With the help of Maven, we can build any number of projects into output types like the JAR, WAR, etc without doing any scripting.
6. Maven provides a standard project structure, making it easy for developers to understand the layout of the project and locate specific files.
7. Maven simplifies the process of managing project dependencies, ensuring that the correct versions of libraries and frameworks are used throughout the project.
8. Maven plugins can be used to add additional functionality to the build process, such as code coverage analysis, static code analysis, and more.

Installation process of Maven

The installation of Maven includes the following Steps:

1. Verify that your system has java installed.
2. Check java Environmental variable is set or not.
3. Download maven
4. Unpack your maven zip at any place in your system.
5. Add the bin directory of the created directory apache-maven-3.5.3(it depends upon your installation version) to the PATH environment variable and system variable.
6. open cmd and run mvn -v command. If it print following lines ofcode then installation completed.

```
              C:\Users\JIT>mvn -v
"Apache Maven 3.9.4
(dfbb324ad4a7c8fb0bf182e6d91b0ae20e3d2dd9)

Maven home: C:\Program Files\apache-maven-3.9.4-bin\apache-maven-3.9.4
```

Java version: 1.8.0_381, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-1.8\jre
Default locale: en_GB, platform encoding: Cp1252OS name: "windows 10", version: "10.0", arch: "amd64", family:"windows"”

POM FILES

A **Project Object Model or POM** is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects.

Example for this is the **build directory**, which is target; the **source directory**, which is src/main/java; the **test source directory**, which is src/test/java; and so on. When executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal.
Some of the configurations that can be specified in the POM are **the project dependencies, the plugins or goals that can be executed, the build profiles.**
Other information such as the project version, description, developers, mailing lists and such can also be specified.

Super POM

The Super POM is Maven's default POM. All POMs extend the Super POM unless explicitly set, the configuration specified in the Super POM is inherited by the POMs we create for our projects.

i.e) Maven will inherit settings from the Super POM by default, unless we specifically override them in the project's pom.xml. Maven automatically follows the Super POM settings (like where to find code, how to build, etc.). But if we add our own settings in the pom.xml file, Maven will use our settings instead of the default ones. Maven uses the effective POM (configuration from super pom plus project configuration) to execute relevant goals. It helps developers to specify minimum configuration details 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**

Create a pom.xml in any directory on your computer. Use the content of above mentioned example pom. In the example below, we've created a pom.xml in C:\MVN\project folder. Now open command console, go the folder containing pom.xml and execute the following mvn command. C:\MVN\project>mvn help:effective-pom
Maven will start processing and display the effective-pom.

```
C:\MVN>mvn help:effective-pom
[INFO] Scanning for projects...
[INFO]
[INFO] --------------< com.companyname.project-group:project >----------------
[INFO] Building project 1.0
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-help-plugin:3.2.0:effective-pom (default-cli) @ project ---
[INFO]
Effective POMs, after inheritance, interpolation, and profiles are applied:
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.261 s
[INFO] Finished at: 2021-12-10T19:54:53+05:30
[INFO] ------------------------------------------------------------------------
```

Minimal POM

The minimum requirement (Elements) for a POM are the following:

- **project -** It is the root element of the pom.xml file
- **modelVersion -** modelversion means what version of the POM model used. It should be set to 4.0.0 for maven and maven2

- **groupId** - the id of the project's group. It is unique and most often we use a group ID which is similar to the root Java package name of the project like we used the groupId com.mycompany.app
- **artifactId** - the id of the artifact (project).
- **version** - the version of the artifact under the specified group. It contains the version number of the project. If the project has been released in different versions then it is useful to give a version of the project.

Example:

```
<project xmlns="http://maven.apache.org/POM/4.0.0">
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.mycompany.app</groupId>
 <artifactId>my-app</artifactId>
 <version>1</version>
</project>
```

A POM requires that its **groupId, artifactId, and version** be configured. These three values form the project's fully qualified artifact name. This is in the form of <groupId>:<artifactId>:<version>. As for the example above, its fully qualified artifact name is "com.mycompany.app:my-app:1".

If the configuration details are not specified, Maven will use their defaults. One of these default values is the packaging type. Every Maven project has a packaging type. If it is not specified in the POM, then the default value "jar" would be used.

Furthermore, we can see that in the minimal POM the repositories were not specified. If we build the project using the minimal POM, it would inherit the repositories configuration in the Super POM. Therefore when Maven sees the dependencies in the minimal POM, it would know that these dependencies will be downloaded from **https://repo.maven.apache.org/maven2** which was specified in the Super POM.

Other Elements of Pom.xml file

1. **dependencies** - dependencies element is used to define a list of dependency of a project.

2. **dependency** - dependency element is a crucial section where a project's external library requirements are declared. It defines a single dependency, It is used inside the dependencies tag. Each dependency is described by its groupId, artifactId and version.
3. **name -** this element is used to gives a human-readable name for the project.
4. **scope** - this element used to define the visibility and availability of that dependency throughout the different phases of the build lifecycle (compile, test) and at runtime. .
5. **packaging** - packaging element is used to packaging our project to output types like JAR, WAR etc.
6. **description:** A brief description of the project.
7. **parent:** Used to inherit configuration from a parent POM, enabling multi-module projects and consistent configurations.
8. **properties:** Contains custom properties that can be referenced throughout the POM.
9. **build**: Configures the build process, including plugins, directories, and resources.
10. **plugins:** Within the build element, defines the plugins used in the build process.
11. **plugin:** Defines a single plugin, including its groupId, artifactId, and version.
12. **profiles:** Defines sets of configuration that can be activated under specific conditions.
13. **repositories:** Defines remote repositories where dependencies and plugins can be downloaded.