Session : MAVEN

EXERCISE

Vishodu Shaozae (EmployeeID: 4174)

1. **Create a non maven project and add use log4j for logging.**

   Ans. Solution given as project: 'mavenAssignment'

2. **Create a maven project and use log4j for logging.**

   Ans. Solution given as project: 'Maven2'

3. **Create a maven project using an archetype(multimodule).**

   Ans. Solution given as project: 'maven-ans3'

4. **Explain different tags (plugins, dependency, parent, profile, properties, modules and project related(i.e. modelVersion, groupId, artifactId, packaging, version, description)) of POM file created using archetype.**

   Ans.

   I. **Plugins:** They are a collection of actions that allow for reuse of common build logic across multiple projects. The behaviour of plugins can be customized through a set of unique parameters that are exposed by plugin goals.

   II. **Dependency:** It is an archive (jar, zip, etc) which our current project needs in order to compile, build, test, and/or to run

   III. **Parent:** The parent tag defines which module is the current module's parent in terms of inheritance. All the configurations defined in the parent tags are the ones inherited from the parent module's POM.

IV. **Profile:** It is a set of configuration values that can be used to set or override default values. Profiles in maven are used when customizing the build for different environments such as Development, Production or Testing.

V. **Properties:** The properties tag allows us to inject custom user defined properties in the pom file. For example, properties tag can be used to specify version of different plugins

VI. **Modules:** A Parent POM that aggregates other modules, specifies those modules in the modules tag.

VII. **ModelVersion:** The modelVersion specifies the POM's version, which in maven 2 and 3 is by default 4.0.0. Therefore modelVersion should be set to 4.0.0 until updated version aren't released.

VIII. **groupID:** Specifies the ID of the project group. This is relevant when multiple projects interact with one another

IX. **artifactID:** artifactID is the name assigned to the project and the artifacts (jar, reports, etc) created in the project.

X. **Packaging:** Packaging specifies the type of artifacts the projects roduces. A projects packaging type specifies the plugin goals that are executed during each Maven build phase. 'Pom' and 'bundle' are two commonly used packaging types for aggregation and bundle creation.

XI. **Version:** Specifies the version of the artifact in the group

XII. **Description:** Like the name suggests, description provides a short and brief description about the artifact/project

5. **Demonstrate the inheritance and aggregation of POM.**

Ans. Solution given as project: 'maven-ans3'. Please keep an eye for comments in the poms of different modules depicting whether they utilised inheritance or aggregation or both.

6. **Point out the differences between Gradle and Maven.**

Ans. Though gradle and maven both have their well defined use cases, yet certain difference do exist:

I. Gradle is based on a graph of task dependencies wherein tasks do all the work. Maven however, iss based on a fixed linear model of phases.

II. Gradle build files are written in groovy whereas Maven pom files are written in xml

III. Whereas gradle has tasks that perform specific actions, Maven has goals that are phase oriented and associated with their corresponding plugins.

IV. Both gradle and maven allow for multi module builds to run in parallel, however, gradle allows for incremental builds wherein it checks which tasks were updated. The tasks which weren't updated aren't executed thereby making gradle a winner when it comes to performance or build time

7. **What is the purpose of mvn clean install and its usage in the project.**

Ans. The 'mvn clean install' command can be explained in three steps:

I. The **mvn** executable is called which requires for maven to be installed

II. The **clean** command has its own lifecycle, which removes all the previously generated resources and files in the target directory. The clean command is applied to all the modules in the project.

III. Like the clean command, the **install** command advocates for the compilation, testing and packaging of the java project. The .jar/.war artifacts thus generated are then installed into the local repository.

End