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

MAVEN BUILD LIFE CYCLE

Maven is based around the central concept of a build lifecycle. It is the process for building and distributing a particular artifact (project). It is only necessary to learn a small set of commands to build any Maven project, and the POM will ensure they get the results they desire.

There are three built-in build lifecycles: **default, clean and site.** The default lifecycle handles your project deployment, the clean lifecycle handles project cleaning, while the site lifecycle handles the creation of your project's web site.

Build Lifecycle Phases

Each of these build lifecycles is defined by a different list of build phases, wherein a build phase represents a stage in the lifecycle. The following lists all build phases of the default, clean and site lifecycles, which are executed in the order given up to the point of the one specified.

Clean Lifecycle

Phase	Description
pre-clean	execute processes needed prior to the actual project cleaning
clean	remove all files generated by the previous build
post-clean	execute processes needed to finalize the project cleaning

Default Lifecycle

Phase	**Description**
validate	validate the project is correct and all necessary information is available.
initialize	initialize build state, e.g. set properties or create directories.
generate-sources	generate any source code for inclusion in compilation.
process-sources	process the source code, for example to filter any values.
generate-resources	generate resources for inclusion in the package.
process-resources	copy and process the resources into the destination directory, ready for packaging.
compile	compile the source code of the project.
process-classes	post-process the generated files from compilation, for example to do bytecode enhancement on Java classes.
generate-test-sources	generate any test source code for inclusion in compilation.
process-test-sources	process the test source code, for example to filter any values.
generate-test-resources	create resources for testing.
process-test-resources	copy and process the resources into the test destination directory.
test-compile	compile the test source code into the test destination directory

process-test-classes	post-process the generated files from test compilation, for example to do bytecode enhancement on Java classes.
test	run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed.
prepare-package	perform any operations necessary to prepare a package before the actual packaging. This often results in an unpacked, processed version of the package.
package	take the compiled code and package it in its distributable format, such as a JAR.
pre-integration-test	perform actions required before integration tests are executed. This may involve things such as setting up the required environment.
integration-test	process and deploy the package if necessary into an environment where integration tests can be run.
post-integration-test	perform actions required after integration tests have been executed. This may including cleaning up the environment.
verify	run any checks to verify the package is valid and meets quality criteria.
install	install the package into the local repository, for use as a dependency in other projects locally.
deploy	done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

Site Lifecycle

Phase	Description
pre-site	execute processes needed prior to the actual project site generation
site	generate the project's site documentation
post-site	execute processes needed to finalize the site generation, and to prepare for site deployment
site-deploy	deploy the generated site documentation to the specified web server

These lifecycle phases are executed sequentially to complete the default lifecycle. Given the lifecycle phases above, this means that when the default lifecycle is used, Maven will first validate the project, then will try to compile the sources, run those against the tests, package the binaries (e.g. jar), run integration tests against that package, verify the integration tests, install the verified package to the local repository, then deploy the installed package to a remote repository.