1.7 Maven Build Tool

What is Maven?

- *Maven* is a *build automation* and *project management tool* primarily used for *Java projects*. It plays a crucial role in the *development*, *build*, and *dependency management* of *Spring applications*, including those using Spring Framework's *component scanning* and *annotation-driven* configuration.

Project and dependency management:

- *Maven* provides a standardized way to *manage Java projects* by defining *project structure*, *dependencies*, and *build configurations* using a declarative *XML-based* format (pom.xml).
- Developers use Maven to specify *project's metadata*, *dependencies*, *plugins*, *repositories*, and other *project-related configurations*.
- *Spring Framework* and its various modules (e.g., *Spring Core*, *Spring MVC*, *Spring Boot*) are managed as dependencies in *Maven projects*. Developers specify the Spring dependencies in the *pom.xml*, and *Maven* handles the rest.

Build Automation:

- *Maven* automates the *build process* including *compilation*, *testing*, *packaging*, and *deployment* using predefined *build lifecycle phases* (e.g., clean, compile, test, package, install).
- *Maven* facilitates the *building* and *packaging* of Spring applications into *deployable artifacts* (e.g., JAR files, WAR files) for deployment in production environments.

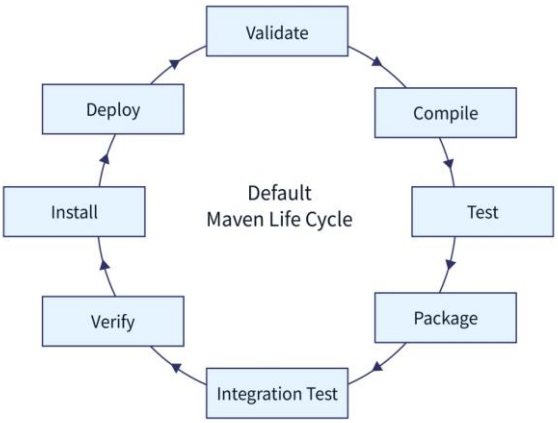
Maven Life Cycle:

1. Validate

- Checks if the project structure and configuration (like pom.xml) are correct.
- Ensures all required information is available before starting the build.

2. Compile

- Compiles the Java source code in *src/main/java* into *.class* files.
- If there are any *syntax errors*, the build will stop here.



3. Test

- Runs *unit tests* (e.g., JUnit) in *src/test/java*.
- These are fast, isolated tests to check individual units of code.

4. Package

- Packages the compiled code into a deployable format:
  - *.jar* for *Java* applications
  - *.war* for *web* applications

5. Integration Test

- Runs *integration tests* that might require a *database* or *server*.
- Ensures *components work together* (not just individually like unit tests).

6. Verify

- Verifies the *integrity* of the *package* and *runs additional checks*.
- Ensures everything is as expected (e.g., tests passed, package exists, no corruption).

7. Install

- Installs the .jar or .war into the *local Maven repository* (usually ~/.m2/repository).
- Allows *other projects on the same system* to *use* this *artifact* as a *dependency*.

8. Deploy

- *Uploads* the *final package* to a *remote repository* (e.g., Nexus, Artifactory).
- Makes it available for use in production or by other developers/teams.

Maven Commands:

Maven Commands	Description
mvn compile	We *compile* the *project's source code* Using the mvn compile command.
mvn clean	Using the mvn clean command, All previous-build files are *removed* from the project.
mvn test	We *execute project testing* steps with the mvn test command.
mvn install	The mvn install command aids in *deploying packaged* WAR or JAR files by *storing* them in the *local repository* as classes.
mvn package	The mvn package command *generates* a *WAR* or *JAR* file for the project so that it can be distributed.
mvn deploy	The mvn deploy command is used *after compilation*, *project testing*, and *project building*. The packaged *WAR* or *JAR* files are *copied* to the *remote repository* so other developers can use them.
mvn spring-boot:run	*Runs* a *Spring Boot application* directly from the source code *without packaging* it into a *JAR* or *WAR* file.
mvn spring-boot:build-image	Builds *a Docker image* of the *Spring Boot application* using the *Spring Boot Maven plugin*.