
🛠️ Maven?

Apache Maven is a **build automation and project management tool** used primarily for **Java projects**.
 It makes it easier for developers to **build, test, package, and deploy applications**.

Key features of Maven:

- 📦 **Dependency Management** → Automatically downloads and manages libraries from central repositories.
- 🔨 **Build Automation** → Standardized build lifecycle (compile → test → package → deploy).
- 📂 **Project Structure** → Uses a standard directory structure (`src/main/java`, `src/test/java`).
- 📜 **POM (Project Object Model)** → Central XML file (`pom.xml`) that defines project info, dependencies, plugins, etc.

⚡ Maven Build Lifecycle

Maven defines **three built-in lifecycles**:

1. **default (build lifecycle)** → handles project build (compiling, testing, packaging, deployment).
2. **clean lifecycle** → cleans old builds.
3. **site lifecycle** → generates project documentation.

Each lifecycle has **phases** (steps). When you run one phase, Maven executes **all phases before it** automatically.

✅ 1. Clean Lifecycle

- **pre-clean** → Do work before cleaning.
- **clean** → Remove old build output (like deleting target/).
- **post-clean** → Do work after cleaning.

✅ 2. Default (Build) Lifecycle

This is the most important one, with many phases:

1. **validate** → Check if the project is correct and has necessary information.
2. **initialize** → Initialize build state (like setting properties).
3. **generate-sources** → Generate source code if required.
4. **process-sources** → Process source code (filtering, etc.).
5. **generate-resources** → Generate resources (like config files).
6. **process-resources** → Copy/modify resources to output directory.
7. **compile** → Compile source code.
8. **process-classes** → Post-processing of compiled classes (bytecode enhancement, etc.).
9. **generate-test-sources** → Generate test source code.

10. **process-test-sources** → Process test sources.
11. **generate-test-resources** → Generate test resources.
12. **process-test-resources** → Copy/modify test resources.
13. **test-compile** → Compile test source code.
14. **process-test-classes** → Post-processing of compiled test classes.
15. **test** → Run unit tests (with frameworks like JUnit/TestNG).
16. **prepare-package** → Prepare things needed for packaging (e.g., code obfuscation).
17. **package** → Package compiled code into a distributable format (JAR/WAR).
18. **pre-integration-test** → Setup before integration tests (e.g., start server/db).
19. **integration-test** → Run integration tests.
20. **post-integration-test** → Cleanup after integration tests.
21. **verify** → Run checks to ensure package is valid and meets criteria.
22. **install** → Install package into the local Maven repository (~/.m2/repository).
23. **deploy** → Deploy package to a remote repository (like Nexus, Artifactory).

✅ 3. Site Lifecycle

- **pre-site** → Work before generating site docs.
- **site** → Generate documentation (reports, javadocs).

- **post-site** → Post-processing after site generation.
- **site-deploy** → Deploy site docs to a server.

🎯 Example

If you run:

mvn package

Maven will run all phases from **validate** → **package**.

If you run:

mvn install

It will run **all phases up to install** (so compile, test, package, verify, install).

🔄 Flow Diagram (Default Lifecycle)

arduino

Copy code

```arduino
validate → initialize → generate-sources → process-sources → generate-resources
   ↓
process-resources → compile → process-classes
   ↓
generate-test-sources → process-test-sources → generate-test-resources → process-test-resources
   ↓
test-compile → process-test-classes → test
   ↓
prepare-package → package
   ↓
pre-integration-test → integration-test → post-integration-test
   ↓
verify → install → deploy
```