Apache Maven – A Beginner-Friendly, Practical Guide (Java)

A simple guide to Maven structure, POM, and adding dependencies -->

What is Maven?

Apache Maven is a **build automation and dependency management tool** mainly used in Java projects.

In simple words:

- Maven **builds your project** (compile, test, package)
- Maven **downloads and manages libraries (JARs)** for you
- Maven **maintains a standard project structure** so every developer understands your project easily

👉 **Without Maven:** You manually download JAR files and manage versions. 👉 With Maven: You just declare dependencies once, Maven handles everything.

What Problem Maven Solves which developers face:

Before Maven:

- Download JAR files manually
- Add them to build path
- Manage versions yourself
- Different setup on different machines

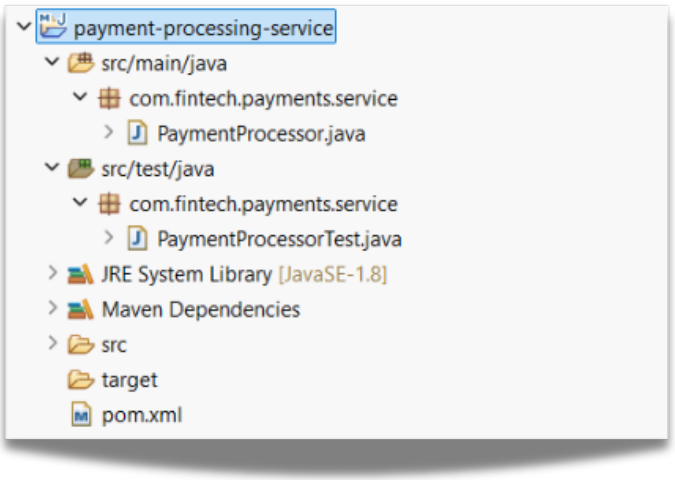
With Maven:

- One pom.xml file controls everything
- Dependencies download automatically
- Same build on every system
- Industry-standard project structure

Key point:
[Maven brings **consistency and automation**.

Maven Standard Project Structure

Maven defines a standard project layout, making Java projects consistent and easy to understand.



Explanation:

- src/**main**/java → your main application code
- src/**test**/java → your test cases
- pom.xml → the heart of Maven

What is pom.xml?

pom.xml (Project Object Model) is the **heart of a Maven project**.

It contains:

- Project details (name, version)
- Dependencies (libraries)
- Plugins (how to build)
- Java version

Example (basic structure):

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>demo-app</artifactId>
  <version>1.0</version>

</project>
```

📌 **Beginner tip**: Maven identifies your project using **groupId** + **artifactId** + version.

How to Add Dependencies in pom.xml

Dependency = external library (like JUnit, Spring, Hibernate)

Example: Adding **JUnit** dependency

A simple POM looks like:

```
16
17    <dependencies>
18      <!-- https://mvnrepository.com/artifact/junit/junit -->
19  <dependency>
20      <groupId>junit</groupId>
21      <artifactId>junit</artifactId>
22      <version>4.13.2</version>
23      <scope>test</scope>
24  </dependency>
25    </dependencies>
26  </project>
27
```

What happens internally?

- Maven downloads JUnit from **Maven Central Repository**
- Stores it in local .m2 folder
- Automatically links it to your project

What is a Maven Plugin?

A plugin tells Maven HOW to perform a task.

Examples:

- Compile Java code
- Run tests
- Package JAR file

Example: `maven-compiler-plugin`

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.10</version>
  <configuration>
    <source>17</source>
    <target>17</target>
  </configuration>
</plugin>
```

📌 Important note:

Every Maven build action is executed by a plugin. Maven itself only **coordinates plugins**.

Maven Build Lifecycle (Very Important)

Maven build runs in **phases**:

- clean → removes old build (target folder)
- compile → compiles Java code
- test → runs JUnit tests
- package → creates JAR/WAR
- install → installs JAR to local repo

Command example:

```
mvn clean install
```

What is the target Folder?

target is a **temporary build folder** created by Maven.

It contains:

- .class files
- Final JAR/WAR
- Test reports
- Temporary build files

⚠ Important rule:

Never manually edit anything inside target. Maven recreates it every build.

What Happens in the test Phase?

When Maven reaches test phase:

- It executes all test classes under src/test/java
- Uses JUnit (or other testing frameworks)
- Generates test reports

```less
src/test/java
(Test Classes)
        ↓
      JUnit
 (Runs @Test methods)
        ↓
   Test Reports
 (Pass / Fail)
```

What is JUnit?

JUnit is a **Java testing framework**.

It provides:

- @Test annotation
- Assertion methods (assertEquals, assertTrue)
- Automatic test execution

⬅ END Final Summary (Beginner Friendly)

- **Maven standardizes project structure**
- pom.xml controls everything
- Dependencies are auto-managed
- Plugins do the real build work
- target folder is temporary
- JUnit tests run automatically

💡 **If you understand Maven, you already think like an industry-level Java developer.**