Maven

1. What Maven Really Is
Maven is a **build automation and dependency management tool** for Java.
At its heart, Maven behaves like a *project orchestrator*, pulling libraries, building artifacts, running plugins, and maintaining a standard project structure.
Key Responsibilities

- Dependency management
- Build lifecycle management
- Standard directory layout
- Plugin execution (testing, packaging, deployment, code generation, etc.)
- Repository management (local, remote, central)

2. Maven Project Structure
Maven enforces a **standard directory layout**:

```
src/
  main/
    java/        → application sources
    resources/   → config files, .properties, XML
  test/
    java/        → test sources
    resources/   → test configs
target/          → build output
pom.xml          → project configuration brain
```

This predictable layout allows plugins to work without extra configuration.

3. POM (pom.xml): The Heart of Maven
The **Project Object Model (POM)** is the configuration file that tells Maven how to behave.
Core identifiers

```
<groupId>com.example</groupId>
<artifactId>order-service</artifactId>
<version>1.0.0</version>
```

- **groupId** → Organization or company domain
- **artifactId** → The project/module name
- **version** → Follows semantic versioning

Together, they become a **coordinate**, like a GPS pin for locating your artifact.
Dependencies block

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.3.30</version>
```

```
    </dependency>
</dependencies>
```

4. Maven Build Lifecycles
Maven uses *lifecycles* to structure a project's build into phases.
You rarely run plugins directly; you run lifecycle **phases**.
Important lifecycles & phases

1. **clean lifecycle**
    - clean → removes target/
2. **default lifecycle** (the main one)
    - validate
    - compile
    - test
    - package
    - verify
    - install
    - deploy
3. **site lifecycle**
    - for generating documentation

Example
Running:
`mvn package`
Triggers:
validate → compile → test → package.

5. Packaging Types
Defines the *shape* of the final output.
Common types:

- jar
- war
- pom
- ear

Example:
`<packaging>jar</packaging>`

6. Maven Repositories
Repositories are library warehouses.
Types

- **Local repository**
  Stored on your machine: ~/.m2/repository
- **Central repository**
  Maven's official public repo
- **Remote repository**
  Company Nexus/Artifactory

Resolution logic
When Maven needs a dependency:

1. Check *local repo*
2. If missing, fetch from *remote/central*
3. Cache locally

7. Dependency Management Concepts

Maven's dependency model is a forest. To navigate it, understand these:

7.1 Transitive Dependencies

A depends on B
B depends on C
Maven auto-pulls C.

7.2 Dependency Scope

Controls **when** and **where** a dependency is available.

Scope	Meaning
compile	Available everywhere; default scope
provided	Expect provided by runtime (e.g., servlet API)
runtime	Needed at runtime but not compile-time
test	For testing only
system	Manually provided JARs (rare, discouraged)
import	Used in dependencyManagement of BOMs

Example:

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
</dependency>
```

7.3 Dependency Exclusions

Sometimes transitive dependencies conflict.

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

8. DependencyManagement vs Dependencies

A classic interview favorite.
dependencyManagement
Declares versions *without including them*.
Used by: multi-module projects or BOMs.
dependencies
Actually adds them.
Example:

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.16.1</version>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Child module can now use:

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
</dependency>
```

9. Maven Plugins
Plugins perform *actions* in each lifecycle phase.
Common plugins

- **maven-compiler-plugin**
- **maven-surefire-plugin** (unit tests)
- **maven-failsafe-plugin** (integration tests)
- **maven-jar-plugin**
- **maven-clean-plugin**

Example configuration:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.11.0</version>
            <configuration>
                <source>17</source>
                <target>17</target>
            </configuration>
        </plugin>
    </plugins>
</build>
```

10. Build Profiles
Profiles let Maven shape-shift based on environment: dev, test, prod.

Example

```
<profiles>
    <profile>
        <id>prod</id>
        <properties>
            <db.url>jdbc:mysql://prod-db</db.url>
        </properties>
        <activation>
            <activeByDefault>false</activeByDefault>
        </activation>
    </profile>
</profiles>
```

Run:

```
mvn package -P prod
```

11. Multi-Module Maven Projects

Used to manage large codebases.

Parent POM

```
<packaging>pom</packaging>
<modules>
    <module>service-api</module>
    <module>service-impl</module>
</modules>
```

Advantages:

- Centralized dependency versions
- Consistent plugin configurations
- Better project organization

12. Maven Coordinates & Versioning

Versions may be:

- **fixed** (1.4.3)
- **snapshot** (1.4.3-SNAPSHOT)
- **ranges** ([1.0,2.0))

Snapshot versions behave like a live wire: always latest development build.

13. Effective POM

Maven merges:

- your pom.xml
- parent POM
- super POM

To inspect the *final truth*:

```
mvn help:effective-pom
```

14. Maven Wrapper (mvnw)

Ensures same Maven version for all developers.

Initialize:

```
mvn -N io.takari:maven:wrapper
```
Run with:
```
./mvnw clean package
```

15. Common Interview Questions (With One-Liners)
Q: How does Maven resolve dependencies?
A: Local repo → remote repo → central repo, following nearest-wins strategy for conflict resolution.
Q: Difference between surefire and failsafe?
A: Surefire runs unit tests. Failsafe runs integration tests in verify phase.
Q: What is a snapshot?
A: A mutable version representing latest development artifacts, updated frequently.
Q: Why do we use dependencyManagement?
A: To centralize versions without forcing inclusion.
Q: How does Maven handle conflicting dependency versions?
A: Uses dependency mediation: nearest definition in the dependency tree wins.

16. Example: Typical Microservice pom.xml
A simple interview-worthy snippet:
```
<project>
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.demo</groupId>
    <artifactId>payment-service</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <properties>
        <java.version>17</java.version>
    </properties>

    <dependencies>
        <!-- Spring Boot starter -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- Testing -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</
artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</
artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

If you'd like, I can also prepare:
✓ Maven cheat sheet (one-page quick revision)
✓ Interview Q&A set (20–40 questions)
✓ A diagram explaining lifecycle flow
✓ Real-world scenarios (e.g., version conflicts, plugin overrides)
Just tell me the flavor you want.