Maven and AEM Core Concepts

1 Maven Lifecycle

Maven follows a structured build lifecycle composed of phases:

- **validate** – Validates the project structure and metadata.
- **compile** – Compiles source code into bytecode.
- **test** – Runs unit tests using frameworks like JUnit.
- **package** – Bundles the compiled code into JAR/WAR files.
- **verify** – Performs integration checks.
- **install** – Places the package into the local repository.
- **deploy** – Deploys to a remote repository for sharing.

Command:

```
mvn clean install
```

2 What is `pom.xml` and Why We Use It

`pom.xml` (Project Object Model) is the configuration file used by Maven. It contains:

- Project metadata: `groupId`, `artifactId`, `version`
- Dependencies, build plugins, modules
- Repository links and properties

Example:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>demo-app</artifactId>
  <version>1.0.0</version>
</project>
```

3 How Dependencies Work

Dependencies are declared in `pom.xml` and resolved automatically by Maven from repositories.

Example:

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-core</artifactId>
  <version>5.3.9</version>
</dependency>
```

Command to view dependency tree:

```
mvn dependency:tree
```

4 Maven Repository Structure

Maven supports:

- **Local Repository** – `~/.m2/repository`
- **Central Repository** – `https://repo.maven.apache.org`
- **Remote/Private Repository** – Nexus, Artifactory

Repository Configuration:

```
<repositories>
  <repository>
    <id>central</id>
    <url>https://repo.maven.apache.org/maven2</url>
```

```
  </repository>
</repositories>
```

5 Multi-module Maven Builds

Multi-module projects use a parent POM:

```
<modules>
  <module>core</module>
  <module>ui.apps</module>
</modules>
```

Command to build all modules:

```
mvn clean install
```

Each module has a child `pom.xml` that inherits the parent.

6 Building a Specific Module

To build a specific module:

```
mvn clean install -pl module-name -am
```

`-pl` (project list) specifies the module. `-am` builds dependencies too.

7 Roles of `ui.apps`, `ui.content`, `ui.frontend`

- **ui.apps** – Contains OSGi components, Java code, servlets, Sling models.
- **ui.content** – Contains AEM content packages (templates, pages).
- **ui.frontend** – Holds static frontend assets (JS, CSS, React/Vue code).

These modules are built into deployable content packages using tools like FileVault (vlt) or `content-package-maven-plugin`.

8 Why We Use Run Modes

Run Modes in AEM help configure different environments (e.g., dev, stage, prod).

Example:

```
crx-quickstart/install.author.dev/
```

Packages placed under that path apply only for that specific run mode.

9 What is the Publish Environment

The Publish instance in AEM is the public-facing server responsible for:

- Serving activated content to end-users
- Handling content delivery via dispatcher

Authors activate content from Author to Publish using replication agents.

10 Why We Use Dispatcher

Dispatcher is AEM's caching and load balancing tool. It:

- Caches static resources to reduce load
- Filters request/response to secure endpoints
- Works with Apache HTTPD or IIS

Dispatcher Configuration: `dispatcher.any`, `filters`, `cache`, etc.

11 Accessing `/crx/de`

You can access the CRX repository console via:

URL:

```
http://localhost:4502/crx/de
```

Login with AEM credentials (e.g., `admin/admin`). CRX DE provides a tree view of JCR repository, node management, and content debugging.