# AEM - Task 1

**Maven Life Cycle**

Maven follows a structured build lifecycle with several key phases:

- **validate**: Ensures the project structure and configuration are correct.
- **compile**: Translates source code into bytecode.
- **test**: Executes unit tests.
- **package**: Bundles the compiled code into a JAR/WAR file.
- **verify**: Runs integration tests to validate the build.
- **install**: Saves the package to the local repository for reuse.
- **deploy**: Uploads the built package to a remote repository for distribution.

**What is the pom.xml File and Why is it Used?**

The pom.xml (Project Object Model) is the core configuration file in a Maven project. It defines metadata, dependencies, build settings, plugins, and the overall project structure. Key advantages of using pom.xml include:

- Centralized dependency management.
- Automated build execution.
- Project structure definition and configuration.
- Integration of various plugins for build and deployment tasks.

**How Dependencies Work in Maven**

Maven manages dependencies through the <dependencies> section in pom.xml. It automatically downloads the required libraries from repositories and resolves transitive dependencies (dependencies of other dependencies).

**Example:**

<dependencies>

 <dependency>

```
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.12.0</version>
  </dependency>
</dependencies>
```

**Maven Repositories**

Maven retrieves dependencies from different types of repositories:

- **Local Repository (~/.m2/repository)**: Stores cached dependencies locally.
- **Central Repository (Maven Central)**: Default global repository for commonly used dependencies.
- **Remote Repositories**: Custom repositories used in enterprise projects.

To check dependencies in a Maven project, use:

```
mvn dependency:tree
```

This command displays the project's dependency hierarchy.

**Building Multiple Modules with Maven**

Maven supports multi-module builds using a parent pom.xml, which references all modules:

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

To build all modules, run:

```
mvn clean install
```

**Building a Specific Module**

To build a specific module along with its dependencies, use:

mvn clean install -pl module-name -am

- -pl: Specifies the module to build.
- -am: Ensures dependencies of the module are built as well.

**Role of ui.apps, ui.content, and ui.frontend Folders**

- **ui.apps**: Contains application configurations, templates, and components.
- **ui.content**: Holds content packages, pages, and assets.
- **ui.frontend**: Includes front-end resources such as JavaScript, React, and CSS files.

**Why Use Run Modes in AEM?**

Run modes allow AEM to apply environment-specific configurations, helping in:

- Defining settings for different environments (development, staging, production).
- Enhancing performance and security.

**Examples of Run Modes:**

- author mode – For content authors to create and manage content.
- publish mode – For serving content to live users.

**What is the Publish Environment?**

The publish environment is where finalized AEM content is delivered to end users. It retrieves content from the author environment and makes it accessible to website visitors.

**Why Use the Dispatcher in AEM?**

The Dispatcher is AEM's caching and load-balancing tool that:

- Enhances performance by caching pages.
- Protects AEM instances from high traffic loads.
- Improves security by filtering unwanted requests.

**Accessing CRX/DE**

CRX/DE (Content Repository eXtreme Developer Environment) can be accessed at:
http://localhost:4502/crx/de/index.jsp
It allows developers to explore, modify, and manage AEM Content and configurations.