Date: 18-03-25

1. **Maven Life Cycle**
   Maven operates through a structured build life cycle, consisting of several phases:
   - **Clean**: Eliminates the target directory along with compiled files.
   - **Validate**: Confirms the project's correctness.
   - **Compile**: Transforms source code into executable format.
   - **Test**: Executes unit tests to verify functionality.
   - **Package**: Assembles the project into a JAR or WAR file.
   - **Verify**: Checks the package's integrity before proceeding.
   - **Install**: Places the package in the local repository for use.
   - **Deploy**: Uploads the package to a remote repository for distribution.

     To execute Maven's build life cycle, use the command:

     mvn clean install

     This will run the relevant phases, ensuring your project is compiled, tested, and installed properly.
2. **What is pom.xml file and why do we use it?**
   The pom.xml file (Project Object Model) is the core configuration file for a Maven project. It defines essential details, dependencies, plugins, and project structure, acting as a blueprint for the build process.

- **Dependency Management**: It specifies the required libraries and their versions, ensuring consistency across different environments.
- **Build Configuration**: Defines build instructions, packaging details, and execution processes.
- **Plugin Management**: Enables automation of tasks like compiling, testing, and deploying.
- **Project Metadata**: Stores information like the project name, version, and author details.
- **Effortless Integration**: Ensures compatibility with CI/CD pipelines, making software development more streamlined.

  In essence, pom.xml simplifies and standardizes project builds while keeping everything organized.

  Example:

```
<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0.0</version>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
```

```
        <artifactId>spring-core</artifactId>
        <version>5.3.9</version>
      </dependency>
    </dependencies>
</project>
```

3. **How Dependencies work?**

   In Maven, dependencies are external libraries or modules that a project requires to function properly. They are managed through the pom.xml file, allowing Maven to automatically download and integrate them.

   Example:

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

4. **Check the maven Repository**

   Maven repositories are used to store dependencies.

   - Local Repository (~/.m2/repository/)
   - Central Repository (Maven's default repository)
   - Remote Repository (hosted by organizations)

   Command to Check Local Repo:

   ls ~/.m2/repository/

5. **How all modules build using maven**

   In Adobe Experience Manager (AEM), Maven is used to build multi-module projects efficiently. AEM projects typically follow a structured approach using the AEM Project Archetype, which helps generate a well-organized project with multiple modules.

   Navigate to the **parent project directory** and run:

   mvn clean install

   This command compiles, tests, and packages all modules in the project.

6. **Can we build a specific module**

   Yes! In **Adobe Experience Manager (AEM)**, you can build a specific module within a multi-module Maven project instead of building the entire project.

   If you are in the **parent project** but want to build only one module, use:

   mvn clean install --projects module-name

   Example:

   mvn clean install --projects core

7. **Role of ui.apps and ui.content and ui.frontend folder**

   - **ui.apps**: Stores AEM components, templates, and configurations.

- **ui.content**: Manages content structure, including pages and assets.

- **ui.frontend**: Handles frontend resources like JavaScript and CSS for client libraries.

8. **Why we are using run mode?**

   In Adobe Experience Manager (AEM), Run Modes allow configuration-based customization for different environments (e.g., development, staging, production). They enable dynamic settings, ensuring that AEM behaves optimally across various deployment scenarios without manual changes.

9. **What is Publish Environment**

   In Adobe Experience Manager (AEM), the Publish Environment is responsible for delivering content to end users. It hosts the published version of a website, making it accessible to visitors. Unlike the Author Environment, where content is created and edited, the Publish Environment is optimized for performance, security, and scalability.

10. **Why are we using Dispatcher**

    In Adobe Experience Manager (AEM), the Dispatcher is used for caching, load balancing, and security. It helps improve website performance by serving cached content, reduces the load on AEM instances, and protects against unauthorized access.

11. **From where can we access crx/de?**

    You can access CRXDE Lite in Adobe Experience Manager (AEM) through your browser by navigating to:

    https://<host>:<port>/crx/de

    Replace <host> and <port> with your AEM instance details.