==========

Build Tools

==========

-> Build tools are used to automate application build process

a) Compile The Source Code

b) Download Required Dependencies (Ex: springboot, hibernate, junit, log4j, kafka...)

c) Execute Unit Test Cases (JUnits)

d) Package our application as jar / war

JAR ---> Java Archieve ---> It is package format for java standalone application

WAR ---> Web Archieve ---> It is package format for java web applications

-> Instead of doing the above build steps manually, we can take the help of Build Tools to automate that process.

-> We have below build tools for java applications

1) Ant (Outdated)

2) Maven

3) Gradle

==========

Maven

==========

-> Maven is a free and open source software given by Apache Organization

-> Maven s/w is developed using Java programming language

-> Maven is used to perform Build Automation for java projects

-> Maven is called as Java Build Tool

Note: Maven is used as a build tool for java projects.

==========

What we can do using maven

==========

1) We can create initial project folder structure using maven

2) We can download "project dependencies" using maven

(ex : springboot, hibernate, kafka, redis, email, log4j, junit, security...)

-> To develop one java project we will use several frameworks like spring, hibernate etc along with Java

-> We need to download those frameworks and we should add to our java project

-> These frameworks we are using in our project are called as our project dependencies

-> Instead of we are downloading dependencies, we can tell to maven s/w to download dependencies

Note: Required dependencies we will add in "pom.xml" file then maven s/w will download them

-> pom stands for project object model

-> When we create maven project then pom.xml file will be created automatically

-> pom.xml will act as input file for maven software

3) We can compile project source code using maven

4) We can package java project as jar or war file using maven

==========

Maven Installation

==========

1) Download and install Java software

-> When we install java we will below 2 things

a) JDK (Java Development Kit)

b) JRE (Java Runtime Environment)

-> JDK contains set of tools to develop java programs

-> JRE contains platform/environment which is used to run java programs

Link To Download Java : https://www.oracle.com/in/java/technologies/javase/javase8-archive-downloads.html

2) Set JAVA_HOME in Environment Variables (System Env Variables)

User Environment Variables: Specific to particular account which logged in our PC

System Envrionment Variables : For All Accounts

Note: Environment Variables will be used by operating system

JAVA_HOME = C:\Program Files\Java\jdk1.8.0_202

3) Set Path for JAVA (Go to System Env Variables -> Env Variables -> System Variables -> Select Path and Click on Edit then add JDK path like below)

Path = C:\Program Files\Java\jdk1.8.0_202\bin

4) Verify Java installation by executing below command in "Command Prompt"

> java -version

Note: It should dipslay java version which we have installed

5) Download Maven software from Apache website

Link To download Maven : https://maven.apache.org/download.cgi

File Name : apache-maven-3.8.5-bin.zip (Binary Archive)

6) Extract Maven Zip file -> Copy extracted maven folder and paste it in "C" drive

7) Set MAVEN_HOME in System Environment Variables

MAVEN_HOME = C:\apache-maven-3.8.5

8) Set Path for Maven in System Environment Variables

Path : C:\apache-maven-3.8.5\bin

9) Open Command Prompt and verify Maven Installaton using below command

> mvn -version

==========

Maven Terminology

==========

archetype

groupId

artifactId

packaging

version

-> Archteype represents what type of project we want to create

=> maven-archetype-quickstart : It represents java standalone application

=> maven-archetype-webapp : It represents java web application

Note: Maven providing 1500+ archetypes

-> groupId represents company name or project name

-> artifactId represents project name or project module name

-> packaging represents how we want to package our java application (jar or war)

-> version represents project version

SNAPSHOT : Under development

RELEASE : Development completed

==========

Creating stand-alone application using maven

==========

1) Create one folder for maven practise

2) Open Command prompt from that folder

3) Execute below command to create maven project

>> mvn archetype:generate -DgroupId=in.ashokit -DartifactId=01-Maven-App -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

4) Once project created then verify project folder structure

01-Maven-App

- src
  - main
    - java
  - test
    - java
- pom.xml

src/main/java : Application source code (.java files)

src/test/java : Application Unit Test code (.java files)

pom.xml : Project Object Model (Maven configuration file)

5) We can add dependencies in pom.xml file

6) We can find maven dependencies in www.mvnrepository.com website

7) Add below dependency in pom.xml file

```
<dependency>
```

```
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>5.3.23</version>
    </dependency>
```

8) Execute maven goal

```
> mvn compile
```

==========

How maven will downoad dependencies

==========

-> Maven will download dependencies using repository

-> In Maven we have 3 types of repositories

1) Central Repository

2) Remote Repository

3) Local Repository

-> central repository is maintaining by apache organization

-> every company will maintain their own remote repository (Ex: Nexus, JFrog)

-> Local repository will be created in our system (Location : C://users/<uname>/.m2)

-> When we add dependency in pom.xml, maven will search for that dependency in local repository. If it is available it will add to project build path.

-> If dependency not available in local repository then maven will connect to Central Repository or Remote Repository based on our configuration.

Note: By default maven will connect with central repository. If we want to use remote reposiotry then we need to configure remote repository details.

Note: Every software company will maintain their own remote repositories (Ex: Nexus / JFrog)

==========

Configuring Remote Repository

==========

```
<repositories>
    <repository>
      <id>id</id>
      <url>jfrong-repo-url/</url>
    </repository>
</repositories>
```

==========

Maven Goals

==========

-> To perform project build activities maven prorvided several goals for us

clean

compile

test

package

install

-> clean goal is used to delete target folder

-> compile goal is used to compile project source code. Compiled code will be stored in target folder

compile

.java ----------> .class

-> test goal is used to execute unit test code of our application (junit code)

-> package goal is used to generate jar or war file for our application based on packaging type available in pom.xml file.

Note: jar or war file will be created in target folder.

-> install goal is used to install our project as a dependency in maven local repository.

Note: Every maven goal is associated with maven plugin. When we execute maven goal then respective maven plugin will execute to perform the operation.

Syntax : mvn <goal-name>

Note: We need to execute maven goals from project folder

==========

Creating web application using maven

==========

>> mvn archetype:generate -DarchetypeArtifactId=maven-archetype-webapp -DgroupId=in.ashokit -DartifactId=flipkart_app -DinteractiveMode=false

==========

Working with Maven Using IDE (STS / Eclipse

==========

-> Every IDE will have Maven plugin by default (no need to install)

-> Using IDE we can create maven project directley

-> Create Standalone project using IDE (quick-start archetype)

-> Create Web application using IDE (web-app archetype)

-> Right Click on Project -> Build Path -> Configure Build Path -> Libraries -> Select JRE -> Edit -> Configure JDK

Note: If JDK not displaying, then add JDK using 'Installed-JRES' button (we have to select jdk folder from our Program Files where we have installed java)

Note: For web application we need to add 'servlet-api' dependency in pom.xml

```
<dependency>
```

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

-> Right click on Project -> Run As -> Maven Build -> Enter Maven Goals (Ex: clean package)

==========

What is Exclusion In Maven

==========

-> Exclusion is used to remove un-wanted child dependencies from build path

Ex: When we add 'spring-context' we are getting other dependencies also like 'spring-core', 'spring-beans', 'spring-aop' etc..

-> I want to use 'spring-context' but i don't want 'spring-aop' in build path then we can exclude 'spring-aop' from 'spring-context' like below

```
    <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.3.23</version>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework</groupId>
                    <artifactId>spring-aop</artifactId>
```

```
                    </exclusion>
                </exclusions>
            </dependency>
```

==========

Maven Multi - Module Project

==========

-> A project contains collection of Modules like below

1) User Module

2) Admin Module

3) Reports Module

5) Payment Module

6) Products Module etc....

-> It is not recommended to develop all the modules code in single project. Maintenence will become very difficult

-> To simplify the development we will use 'Maven-Multi-Module' concept

-> In Maven Multi Module, We will create project with Parent - Child relation

-> One Parent Project can have multiple child projects

-> Child Projects are called as 'Maven Modules'

-> When we execute Maven Goal in Parent Project then it will execute that goal in all the child projects also.

Note: Parent Project Packaging type should be 'POM'

==========

Creating Maven Multi Module Project in IDE

==========

1) Create Maven Project with Packaging Type as 'pom'

2) Right Click on the Project -> New -> Select Others -> Search For Maven Module -> Create the Module

3) After Module got created, check parent project pom.xml and child project pom.xml

==========

Parent Project pom.xml looks like below

==========

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.flipkart</groupId>
        <artifactId>Flipkart_App</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>pom</packaging>
```

```
        <modules>
                <module>ADMIN</module>
                <module>REPORTS</module>
        </modules>

</project>
```

==========

Child Project pom.xml looks like below

==========

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.flipkart</groupId>
    <artifactId>Flipkart_App</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <artifactId>REPORTS</artifactId>
</project>
```

-> Right click on Parent Project -> Run as -> Maven Build -> Enter Goals and Execute.

==========

What is Dependency Scope in the Maven

==========

-> 'Scope' represents when that dependency should be used by Maven in Build Process

There are 6 scopes:

compile : This is the default scope, used if none is specified. Compile dependencies are available in all classpaths of a project. Furthermore, those dependencies are propagated to dependent projects.

provided : This is much like compile, but indicates you expect the JDK or a container to provide the dependency at runtime. For example, when building a web application for the Java Enterprise Edition, you would set the dependency on the Servlet API and related Java EE APIs to scope provided because the web container provides those classes. A dependency with this scope is added to the classpath used for compilation and test, but not the runtime classpath. It is not transitive.

runtime : This scope indicates that the dependency is not required for compilation, but is for execution. Maven includes a dependency with this scope in the runtime and test classpaths, but not the compile classpath.

test : This scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases. This scope is not transitive. Typically this scope is used for test libraries such as JUnit and Mockito. It is also used for non-test libraries such as Apache Commons IO if those libraries are used in unit tests (src/test/java) but not in the model code (src/main/java).

system: This scope is similar to provided except that you have to provide the JAR which contains it explicitly. The artifact is always available and is not looked up in a repository.

import : This scope is only supported on a dependency of type pom in the <dependencyManagement> section. It indicates the dependency is to be replaced with the effective list of dependencies in the specified POM's <dependencyManagement> section. Since they are replaced, dependencies with a scope of import do not actually participate in limiting the transitivity of a dependency.

==========

What we have learnt in Maven

==========

1) What is Build Tool

2) Why we need Build Tool

3) What is Maven

4) Maven Setup in Windows OS (JAVA_HOME, JAVA PATH, MAVEN_HOME, MAVEN PATH)

5) Maven Terminology

6) Standalone Application Creation using Command Prompt & IDE

7) Web Application Creation using Command Prompt & IDE

8) Maven pom.xml file

9) How to add dependencies in 'pom.xml' file (www.mvnrepository.com)

10) Maven Goals Execution in command prompt & in IDE

11) Maven Repositories (local, remote, central)

12) Working with Remote Repository (Ex: Nexus)

13) Maven Dependency Exclusion

14) Maven Multi Module Project

15) Maven Dependency Scopes