# Maven

**What is Maven?**

1. Maven is a build tool that helps in project management. It helps in documenting and building the project.
2. Build Tool
   1. Essential for the process of building.
   2. Generating source code
   3. Generating documentation from source code
   4. Compiling of source code
   5. Packaging of the compiled codes into JAR files
   6. Installing the packaged code in local repository, server or central repository.
3. Maven is written in Java or C# and is based on the POM (project object model)
4. POM – Project Object Model
   1. It is an XML file that has all the information regarding project and configuration details.
   2. Resides in the base directory of the project as - pom.xml file
   3. It also includes certain goals and plugins used by Maven in the project.
   4. Maven looks for pom in current directory while executing a task or a goal.
5. Use to build and manage any Java project.

**Links:**

Maven Repo - <https://mvnrepository.com/>

**Need for Maven:**

1. Getting the JAR files for each project as there may be different versions of separate packages.
2. To download dependencies visiting of the official website of different software is not needed. We can visit mvnrepository.com
3. Helps in creating right project structure.
4. Helps to build and deploy the project.

**What exactly it does?**

1. Simplifies the process of building the project.
2. The task of downloading the jar files and other dependencies does not need to be done manually.
3. Provides easy access to all the information.
4. Apache Maven helps to manage all process, such as building, documenting etc.

**Code Structure:**

Graphical user interface, application

Description automatically generated with medium confidence

# Maven

Pom.xml

<dependencies>

<dependency>

</dependency>

<dependency>

</dependency>

</dependencies>

* Internally creates maven local repository.
* It will download all libraries from the remote maven repository. For the first time we need internet.
* After the dependencies are downloaded on the local repository, the project will refer the local repo going forward.
* If we change the version, it will go to the remote maven repository.

**Plugins:**

* Project related configs can be added via plugins

<plugins>

<plugin>

//compile entire project

</plugin>

<plugin>

//start and stop server every time

</plugin>

</plugins>

Maven Plugins - https://maven.apache.org/plugins/

Demo

**Creating maven project:**

* They are templates provided by maven.
* You can skip the archtype.
* GroupId – represents company name
* Artifact name – name of project
* Packaging : A jar file or war file or pom
* Parent Project : If we want to inherit any parent project

**jar**

Java archive – or jar – is one of the most popular packaging types. Projects with this packaging type produce a compressed zip file with the .jar extension. It may include pure Java classes, interfaces, resources, and metadata files

**war**

Simply put, a web application archive – or war – contains all files related to a web application. It may include Java servlets, JSPs, HTML pages, a deployment descriptor, and related resources. Overall, war has the same goal bindings as a jar, but with one exception —the package phase of the war has a different goal, which is war.

**pom**

Among all packaging types, pom is the simplest one. It helps to create aggregators and parent projects.

**Understanding folder structure:**

* Provides default structure
* src/main/java – development source code
* src/main/resources – If any resources are needed we can keep it here
* src/test/java – We add test cases here
* src/test/resources – resources needed for test
* src, target – will be used by maven at runtime

**pom.xml**

* Different tags

Eg:

**int** userLogin(String username, String password) {

ResourceBundle rb = ResourceBundle.*getBundle*("config");

String un = rb.getString("un");

String pwd = rb.getString("pwd");

System.***out***.println(un);

System.***out***.println(pwd);

**if**(username.equals(un) && password.equals(pwd))

**return** 1;

**return** 0;

}

**Maven Build Life cycle:**

* Validate – Validate the code
* Compile – Compile the code
* Test – Test the code
* Package – Package the code, either jar or war
* Integration Test -
* Verify
* Install – Install in local repo
* Deploy – Install in remote rep’s to use by other projects

**Running Maven project:**

* **Run As -> Maven build, install ..**
* It will execute the previous phases if we are executing the phase that comes later
* Goal – Is like target(clean, test..)
* Run As -> Maven clean
* If we run and get error, we should set the JDK
* After making changes, we can update project
* Install will create jar file.

Push to Maven - https://jenkov.com/tutorials/maven/publish-to-central-maven-repository.html

**Simple calculator Example**

<!-- https://mvnrepository.com/artifact/junit/junit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

**TDD Overview:A diagram of a test

Description automatically generated**