Maven

Maven is a powerful project management tool that is based on POM (project object model). It is used for projects build, dependency and documentation.

It simplifies the build process like ANT. But it is too much advanced than ANT.

## **Understanding the problem without Maven**

There are many problems that we face during the project development. They are discussed below:

**1) Adding set of Jars in each project:** In case of struts, spring, hibernate frameworks, we need to add set of jar files in each project. It must include all the dependencies of jars also.

**2) Creating the right project structure:** We must create the right project structure in servlet, struts etc, otherwise it will not be executed.

**3) Building and Deploying the project:** We must have to build and deploy the project so that it may work.

## **What it does?**

Maven simplifies the above mentioned problems. It does mainly following tasks.

1. It makes a project easy to build
2. It provides uniform build process (maven project can be shared by all the maven projects)
3. It provides project information (log document, cross referenced sources, mailing list, dependency list, unit test reports etc.)
4. It is easy to migrate for new features of Maven

Apache Maven helps to manage

* Builds
* Documentation
* Reporting
* SCMs
* Releases
* Distribution

## **What is Build Tool**

A build tool takes care of everything for building a process. It does following:

* Generates source code (if auto-generated code is used)
* Generates documentation from source code
* Compiles source code
* Packages compiled code into JAR of ZIP file
* Installs the packaged code in local repository, server repository, or central repository

# How to install Maven on windows

You can download and install maven on windows, linux and MAC OS platforms. Here, we are going to learn how to install maven on windows OS.

To install maven on windows, you need to perform following steps:

1. Download maven and extract it
2. Add JAVA\_HOME and MAVEN\_HOME in environment variable
3. Add maven path in environment variable
4. Verify Maven

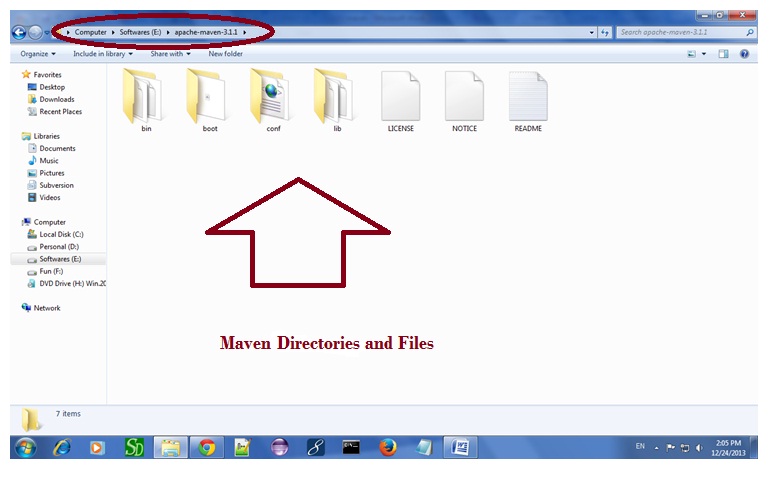
## **1) Download Maven**

To install maven on windows, you need to download apache maven first.

Download Maven latest Maven software from [Download latest version of Maven](http://maven.apache.org/download.cgi)

For example: **apache-maven-3.1.1-bin.zip**

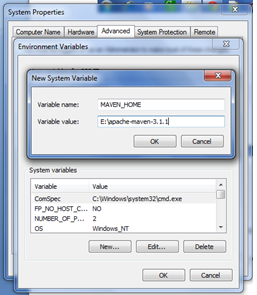
Extract it. Now it will look like this:



## **2) Add MAVEN\_HOME in environment variable**

Right click on **MyComputer** -> **properties** -> **Advanced System Settings** -> **Environment variables** -> **click new button**

Now **add MAVEN\_HOME** in variable name and path of maven in variable value. It must be the home directory of maven i.e. outer directory of bin. For example: **E:\apache-maven-3.1.1** .It is displayed below:



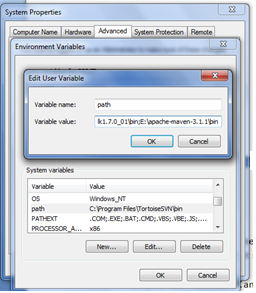
Now click on **OK** button.

## **3) Add Maven Path in environment variable**

Click on new tab if path is not set, then set the path of maven. If it is set, edit the path and append the path of maven.

Here, we have installed JDK and its path is set by default, so we are going to append the path of maven.

The path of maven should be **%maven home%/bin**. For example, **E:\apache-maven-3.1.1\bin** .



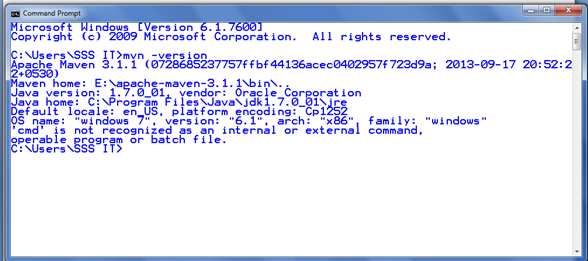
## **4)Verify maven**

To verify whether maven is installed or not, open the command prompt and write:

1. mvn −version

Now it will display the version of maven and jdk including the maven home and java home.

Let's see the output:



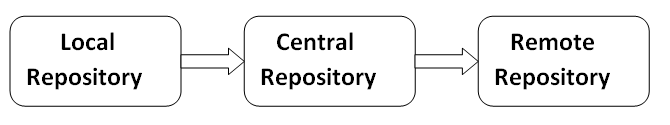
Maven Repository

A **maven repository** is a directory of packaged JAR file with pom.xml file. Maven searches for dependencies in the repositories. There are 3 types of maven repository:

1. Local Repository
2. Central Repository
3. Remote Repository

Maven searches for the dependencies in the following order:

**Local repository** then **Central repository** then **Remote repository**.

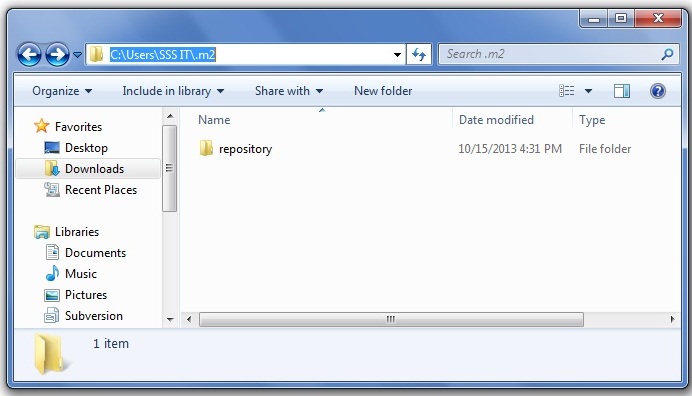


If dependency is not found in these repositories, maven stops processing and throws an error.

## **1) Maven Local Repository**

Maven **local repository** is located in your local system. It is created by the maven when you run any maven command.

By default, maven local repository is %USER\_HOME%/.m2 directory. For example: **C:\Users\SSS IT\.m2**.



## **Update location of Local Repository**

We can change the location of maven local repository by changing the **settings.xml** file. It is located in **MAVEN\_HOME/conf/settings.xml**, for example: **E:\apache-maven-3.1.1\conf\settings.xml**.

Let's see the default code of settings.xml file.

*settings.xml*

1. ...
2. **<settings** xmlns="http://maven.apache.org/SETTINGS/1.0.0"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd"**>**
5. <!-- localRepository
6. | The path to the local repository maven will use to store artifacts.
7. |
8. | Default: ${user.home}/.m2/repository
9. **<localRepository>**/path/to/local/repo**</localRepository>**
10. --**>**
12. ...
13. **</settings>**

Now change the path to local repository. After changing the path of local repository, it will look like this:

*settings.xml*

1. ...
2. **<settings** xmlns="http://maven.apache.org/SETTINGS/1.0.0"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd"**>**
5. **<localRepository>**e:/mavenlocalrepository**</localRepository>**
7. ...
8. **</settings>**

As you can see, now the path of local repository is e:/mavenlocalrepository.

## **2) Maven Central Repository**

Maven **central repository** is located on the web. It has been created by the apache maven community itself.

The path of central repository is: <http://repo1.maven.org/maven2/>.

The central repository contains a lot of common libraries that can be viewed by this url <http://search.maven.org/#browse>.

## **3) Maven Remote Repository**

Maven **remote repository** is located on the web. Most of libraries can be missing from the central repository such as JBoss library etc, so we need to define remote repository in pom.xml file.

Let's see the code to add the jUnit library in pom.xml file.

*pom.xml*

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0"
2. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4. http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
6. **<modelVersion>**4.0.0**</modelVersion>**
8. **<groupId>**com.javatpoint.application1**</groupId>**
9. **<artifactId>**my-application1**</artifactId>**
10. **<version>**1.0**</version>**
11. **<packaging>**jar**</packaging>**
13. **<name>**Maven Quick Start Archetype**</name>**
14. **<url>**http://maven.apache.org**</url>**
16. **<dependencies>**
17. **<dependency>**
18. **<groupId>**junit**</groupId>**
19. **<artifactId>**junit**</artifactId>**
20. **<version>**4.8.2**</version>**
21. **<scope>**test**</scope>**
22. **</dependency>**
23. **</dependencies>**
25. **</project>**

You can search any repository from Maven official website **mvnrepository.com**.

# Maven pom.xml file

**POM** is an acronym for **Project Object Model**. The pom.xml file contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc.

Maven reads the pom.xml file, then executes the goal.

Before maven 2, it was named as project.xml file. But, since maven 2 (also in maven 3), it is renamed as pom.xml.

## **Elements of maven pom.xml file**

For creating the simple pom.xml file, you need to have following elements:

|  |  |
| --- | --- |
| **Element** | **Description** |
| **project** | It is the root element of pom.xml file. |
| **modelVersion** | It is the sub element of project. It specifies the modelVersion. It should be set to 4.0.0. |
| **groupId** | It is the sub element of project. It specifies the id for the project group. |
| **artifactId** | It is the sub element of project. It specifies the id for the artifact (project). An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, source and binary distributions, and WARs. |
| **version** | It is the sub element of project. It specifies the version of the artifact under given group. |

*File: pom.xml*

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0"
2. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4. http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
6. **<modelVersion>**4.0.0**</modelVersion>**
7. **<groupId>**com.javatpoint.application1**</groupId>**
8. **<artifactId>**my-app**</artifactId>**
9. **<version>**1**</version>**
11. **</project>**

## **Maven pom.xml file with additional elements**

Here, we are going to add other elements in pom.xml file such as:

|  |  |
| --- | --- |
| **Element** | **Description** |
| **packaging** | defines packaging type such as jar, war etc. |
| **name** | defines name of the maven project. |
| **url** | defines url of the project. |
| **dependencies** | defines dependencies for this project. |
| **dependency** | defines a dependency. It is used inside dependencies. |
| **scope** | defines scope for this maven project. It can be compile, provided, runtime, test and system. |

*File: pom.xml*

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0"
2. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4. http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
6. **<modelVersion>**4.0.0**</modelVersion>**
8. **<groupId>**com.javatpoint.application1**</groupId>**
9. **<artifactId>**my-application1**</artifactId>**
10. **<version>**1.0**</version>**
11. **<packaging>**jar**</packaging>**
13. **<name>**Maven Quick Start Archetype**</name>**
14. **<url>**http://maven.apache.org**</url>**
16. **<dependencies>**
17. **<dependency>**
18. **<groupId>**junit**</groupId>**
19. **<artifactId>**junit**</artifactId>**
20. **<version>**4.8.2**</version>**
21. **<scope>**test**</scope>**
22. **</dependency>**
23. **</dependencies>**
25. **</project>**

# Maven Example

We can create a simple maven example by executing the **archetype:generate** command of **mvn tool**.

To create a simple java project using maven, you need to open command prompt and run the **archetype:generate** command of mvn tool.

#### **Syntax**

The **syntax** to generate the project architecture is given below:

1. mvn archetype:generate -DgroupId=groupid -DartifactId=artifactid
2. -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=booleanValue

#### **Example**

The **example** to generate the project architecture is given below:

1. mvn archetype:generate -DgroupId=com.javatpoint -DartifactId=CubeGenerator
2. -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

#### **Note: Here, we are using maven-archetype-quickstart to create simple maven core project. if you use maven-archetype-webapp, it will generate a simple maven web application.**

#### **Output**

Now it will **generate following code in the command prompt**:

1. mvn archetype:generate -DgroupId=com.javatpoint -DartifactId=Cub
2. eGenerator -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=fa
3. lse
4. [INFO] Scanning for projects...
5. [INFO]
6. [INFO] ------------------------------------------------------------------------
7. [INFO] Building Maven Stub Project (No POM) 1
8. [INFO] ------------------------------------------------------------------------
9. [INFO]
10. [INFO] **>>>** maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom **>>**
11. **>**
12. [INFO]
13. [INFO] **<<<** **maven-archetype-plugin:2.2:generate** (default-cli) @ standalone-pom **<<**
14. **<**
15. [INFO]
16. [INFO] --- maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom --
17. -
18. [INFO] Generating project in Batch mode
19. Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/mav
20. en-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.jar
21. Downloaded: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/mave
22. n-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.jar (5 KB at 3.5 KB/se
23. c)
24. Downloading: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/mav
25. en-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.pom
26. Downloaded: http://repo.maven.apache.org/maven2/org/apache/maven/archetypes/mave
27. n-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.pom (703 B at 0.9 KB/s
28. ec)
29. [INFO] -------------------------------------------------------------------------
30. ---
31. [INFO] Using following parameters for creating project from Old (1.x) Archetype:
32. maven-archetype-quickstart:1.0
33. [INFO] -------------------------------------------------------------------------
34. ---
35. [INFO] Parameter: groupId, Value: com.javatpoint
36. [INFO] Parameter: packageName, Value: com.javatpoint
37. [INFO] Parameter: package, Value: com.javatpoint
38. [INFO] Parameter: artifactId, Value: CubeGenerator
39. [INFO] Parameter: basedir, Value: C:\Users\SSS IT
40. [INFO] Parameter: version, Value: 1.0-SNAPSHOT
41. [INFO] project created from Old (1.x) Archetype in dir: C:\Users\SSS IT\CubeGene
42. rator
43. [INFO] ------------------------------------------------------------------------
44. [INFO] BUILD SUCCESS
45. [INFO] ------------------------------------------------------------------------
46. [INFO] Total time: 10.913s
47. [INFO] Finished at: Thu Dec 26 16:45:18 IST 2013
48. [INFO] Final Memory: 9M/25M
49. [INFO] ------------------------------------------------------------------------
50. 'cmd' is not recognized as an internal or external command,
51. operable program or batch file.

#### **Generated Directory Structure**

Now go to the current directory from where you have executed the mvn command. For example: **C:\Users\SSS IT\CubeGenerator**. You will see that a simple java project is created that has the following directory:

1. CubeGenerator
2. -src
3. --main
4. ---java
5. ----com
6. -----javatpoint
7. ------App.java
8. --test
9. ---java
10. ----com
11. -----javatpoint
12. ------AppTest.java
13. -pom.xml

As you can see, there are created 3 files pom.xml, App.java and AppTest.java. Let's have a quick look at these files:

#### **1) Automatically Generated pom.xml file**

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0"
2. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4. http://maven.apache.org/maven-v4\_0\_0.xsd"**>**
6. **<modelVersion>**4.0.0**</modelVersion>**
7. **<groupId>**com.javatpoint**</groupId>**
8. **<artifactId>**CubeGenerator**</artifactId>**
9. **<packaging>**jar**</packaging>**
10. **<version>**1.0-SNAPSHOT**</version>**
11. **<name>**CubeGenerator**</name>**
12. **<url>**http://maven.apache.org**</url>**
13. **<dependencies>**
14. **<dependency>**
15. **<groupId>**junit**</groupId>**
16. **<artifactId>**junit**</artifactId>**
17. **<version>**3.8.1**</version>**
18. **<scope>**test**</scope>**
19. **</dependency>**
20. **</dependencies>**
21. **</project>**

#### **2) Automatically Generated App.java file**

1. **package** com.javatpoint;
2. /\*\*
3. \* Hello world!
4. \*
5. \*/
6. **public** **class** App
7. {
8. **public** **static** **void** main( String[] args )
9. {
10. System.out.println( "Hello World!" );
11. }
12. }

#### **3) Automatically Generated AppTest.java file**

1. **package** com.javatpoint;
3. **import** junit.framework.Test;
4. **import** junit.framework.TestCase;
5. **import** junit.framework.TestSuite;
6. /\*\*
7. \* Unit test for simple App.
8. \*/
9. **public** **class** AppTest
10. **extends** TestCase
11. {
12. /\*\*
13. \* Create the test case
14. \*
15. \* @param testName name of the test case
16. \*/
17. **public** AppTest( String testName )
18. {
19. **super**( testName );
20. }
21. /\*\*
22. \* @return the suite of tests being tested
23. \*/
24. **public** **static** Test suite()
25. {
26. **return** **new** TestSuite( AppTest.**class** );
27. }
28. /\*\*
29. \* Rigourous Test :-)
30. \*/
31. **public** **void** testApp()
32. {
33. assertTrue( **true** );
34. }
35. }

## **Compile the Maven Java Project**

To compile the project, go to the project directory,

for example: **C:\Users\SSS IT\CubeGenerator** and write the following command on the command prompt:

1. mvn clean compile

Now, you will see a lot of execution on the command prompt. If you check your project directory, **target directory** is created that contains the class files.

## **Run the Maven Java Project**

To run the project, go to the project directory\target\classes,

for example: **C:\Users\SSS IT\CubeGenerator\target\classes** and write the following command on the command prompt:

1. java com.SampleDemo.App

Now, you will see the output on the command prompt:

## **Output of the maven example**

1. Hello World!

## **How to build the maven project or how to package maven project?**

The **mvn package** command completes the build life cycle of the maven project such as:

1. validate
2. compile
3. test
4. package
5. integration-test
6. verify
7. install
8. deploy

You need to execute the following command on the command prompt to package the maven project:

1. mvn **package**

Now you will see that **a jar file is created** inside the project/target directory.

You can also run the maven project by the jar file. To do so, go to the maven project directory, for example: **C:\Users\SSS IT\CubeGenerator** and execute the following command on the cmd:

1. java -classpath target\CubeGenerator-1.0-SNAPSHOT.jar;.; com.SampleDemo.App

Now you will see the following output:

1. Hello World!

# Maven Build Life Cycle

What is build life cycle? The sequence of steps which is defined in order to execute the tasks and goals of any maven project is known as build life cycle in maven. Maven 2.0 version is basically a build life cycle oriented and clearly says that these steps are well defined to get the desired output after the successful execution of the build life cycle.

Maven comes with 3 built-in build life cycles as shown below :

* **Clean** - this phase involves cleaning of the project (for a fresh build & deployment)
* **Default** - this phase handles the complete deployment of the project
* **Site** - this phase handles the generating the java documentation of the project.

Now we will dig more into the detailed phases involved in the above mentioned built-in build life cycles.

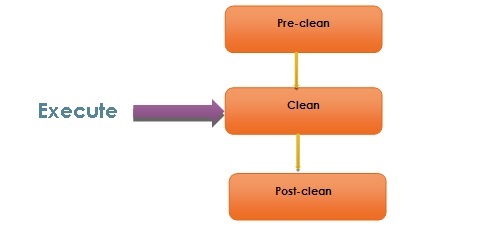
## **Maven: Build Life Cycle of clean phase**

As mentioned above, this clean phase is used to clean up the project and make it ready for the fresh compile and deployment. The command used for the same is mvn post-clean. When this command is invoked, maven executes the below tasks via executing the below commands internally :

1. mvn pre-clean
2. mvn clean
3. mvn post-clean

This maven's clean is a goal and on executing it cleans up the output directory (target folder) by deleting all the compiled files.

**NOTE:** Whenever a maven command for any life cycle is invoked, maven executes the phases till and up to the invoked phase. E.g. when 'mvn clean' is invoked, maven will execute only the phase clean. But, no compile/deployment/site phase is invoked.



## **Maven: Build Lifecycle(Default)**

Below is the list of phases in the build lifecycle (default) of maven. These phases will be invoked through the maven commands.

|  |  |
| --- | --- |
| **Lifecycle Phase** | **Description** |
| validate | Validates and ensures that the project is fine and perfect considering all the required information is made available for the build |
| generate-sources | Generating any source code to include the same in the compilation process |
| process-sources | Processing the source code in case some filter needs to be applied |
| generate-sources | Generating any source code to include the package |
| process-resources | Process of copying the resources to the destination folder and getting ready for the packaging |
| compile | Compilation of the project source code. |
| process-classes | To perform the bytecode enhancements for the class files generated from the compilation |
| generate-test-sources | Copying and processing the resources in the test destination directory. |
| test-compile | Compile the source code in the test destination directory |
| test | Executing/running the tests using some suitable test framework. Note: these test cases are not considered for packaging and deploying |
| prepare-package | To perform any final changes/validations before it is sent for final packaging. |
| package | Packaging the successfully compiled and tested code to some distributable format - JAR, WAR, EAR |
| pre-integration-test | To perform actions before integration tests are executed. This may require to set up some environmental changes for the app. |
| integration-test | Deploy the application to an environment where integration tests can be run. |
| post-integration-test | Basically, cleaning up the environment which was made in pre-integration-test phase. |
| verify | Performs action for a quality check and ensures the required criteria being met |
| install | Installing the application in the local repository. Any other project can use this as a dependency. |
| deploy | The final package will be copied to a remote repository, may be as a formal release and also made available to the other developers too. |

## **Maven: Site Lifecycle(site)**

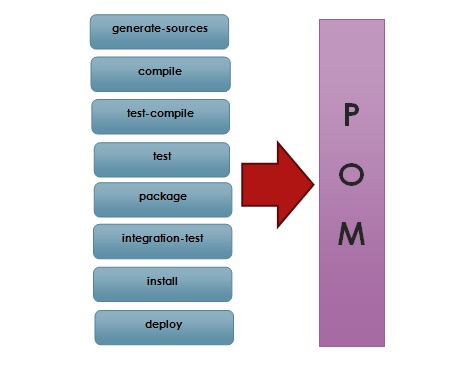
Apart from cleaning, compiling the source code, building a deployable format of the application, maven has phase which does more than these phases. This phase is one of the vital features provided by maven which generates the detailed documentation of any java project.

This project documentation has a dedicated phases involved as listed below :

* pre-site
* site
* post-site
* site-deploy

The command used in maven to generate javadocs for a given project is '*mvn site*'. Basically, when this command is invoked, maven calls '*Doxia*' document generation and other report generating plugins.

**Doxia** is basically a framework used for content generation by maven. This generates contents both in static and dynamic ways.



create database SampleDB;

use SampleDB;

create table emp(id int(10),name varchar(40),age int(3));

insert into emp values(1,'Geetha',32);

insert into emp values(2,'Parth joshi',22);

select \* from emp;

**package** demo;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.ResultSet;

**import** java.sql.Statement;

//import java.sql.\*;

/\*\*

\* Hello world!

\*

\*/

**public** **class** App

{

**public** **static** **void** main( String[] args )

{

System.***out***.println( "Hello World!" );

**try**{

//Class.forName("com.mysql.jdbc.Driver");

Connection con=DriverManager.*getConnection*("jdbc:mysql://localhost:3306/SampleDB","root","root@123");

//here sonoo is database name, root is username and password

Statement stmt=con.createStatement();

ResultSet rs=stmt.executeQuery("select \* from emp");

**while**(rs.next())

System.***out***.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));

con.close();

}

**catch**(Exception e){ System.***out***.println(e);}

}

}