Day1: Maven Build Tool

What are build tools?

> Build tools are programs that automate the creation of executable applications from source code (e.g., .apk for an Android app). Building incorporates compiling, linking and packaging the code into a usable or executable form.
>
> Basically build automation is the act of scripting or automating a wide variety of tasks that software developers do in their day-to-day activities like:
>
> 1. Downloading dependencies.
> 2. Compiling source code into binary code.
> 3. Packaging that binary code.
> 4. Running tests.
> 5. Deployment to production systems.

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

Maven

Maven is a tool that can now be used for building and managing any Java-based project. It makes the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project.

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

Dependency is defined as a state of needing something or someone.

When you rely on coffee to get you through the day, this is an example of you having dependency on caffeine (coffee).

In Maven, a dependency is just another archive—JAR, ZIP, and so on—which our current project needs in order to compile, build, test, and/or run. These project dependencies are collectively specified in the pom. xml file, inside of a <dependencies> tag.
More details will be discussed later in these notes.

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.

Maven's Objectives

Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal, Maven deals with several areas of concern:

- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Encouraging better development practices

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

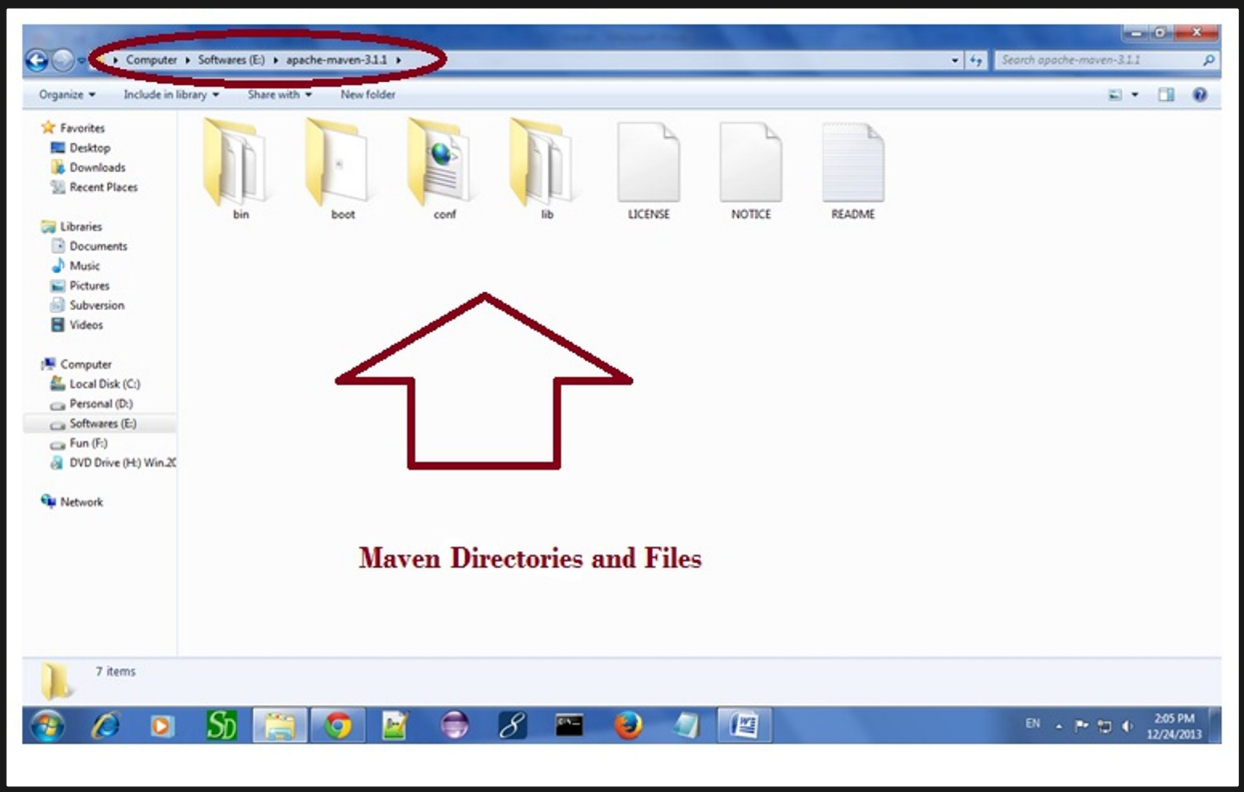
1. Download maven and extract it
2. Add JAVA_HOME and MAVEN_HOME in environment variable
3. Add maven path in environment variable
4. Verify Maven

1) Download Maven

Download Maven latest Maven software from Download latest version of Maven

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

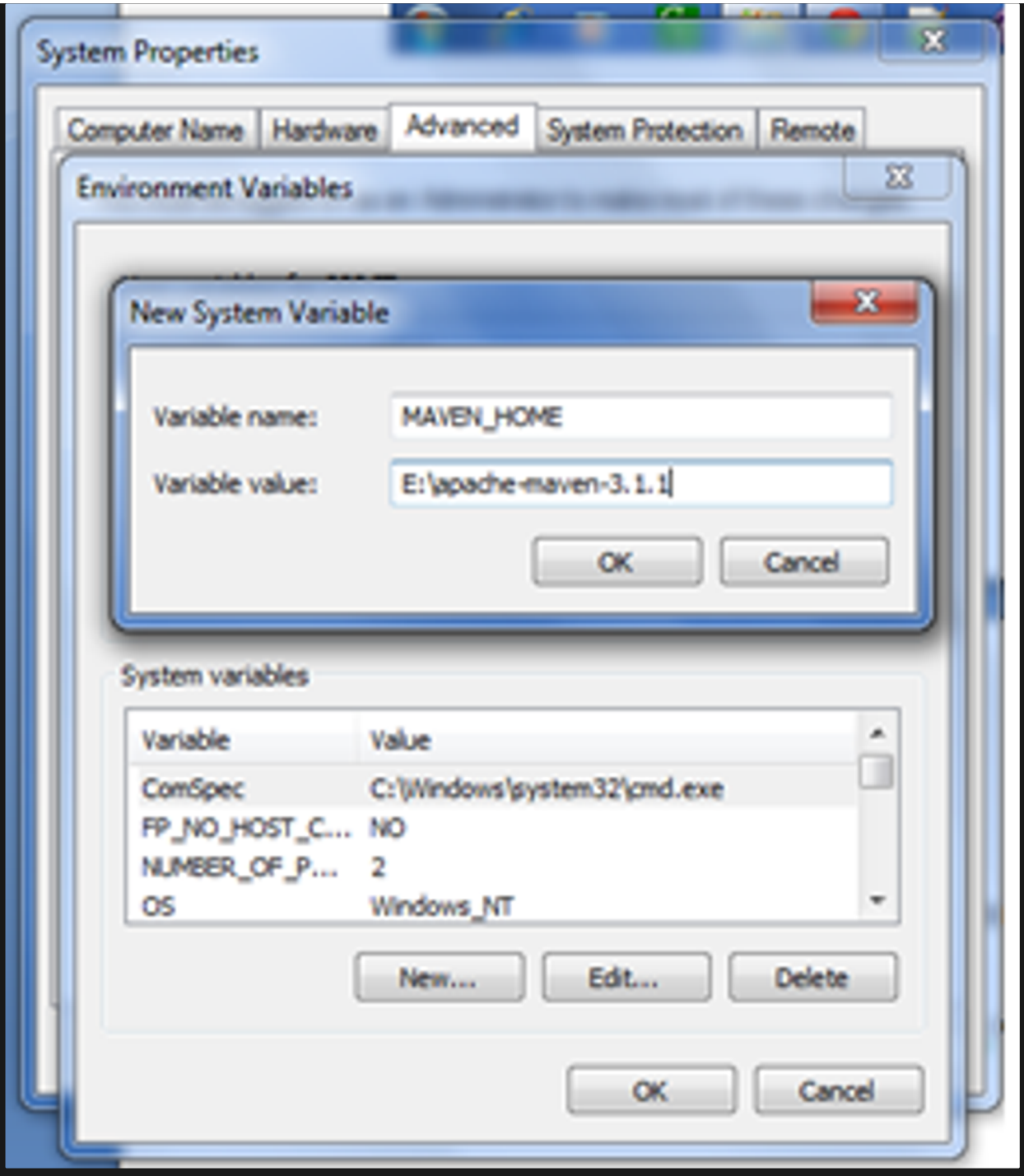
Extract it. Now it will look like this:



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.

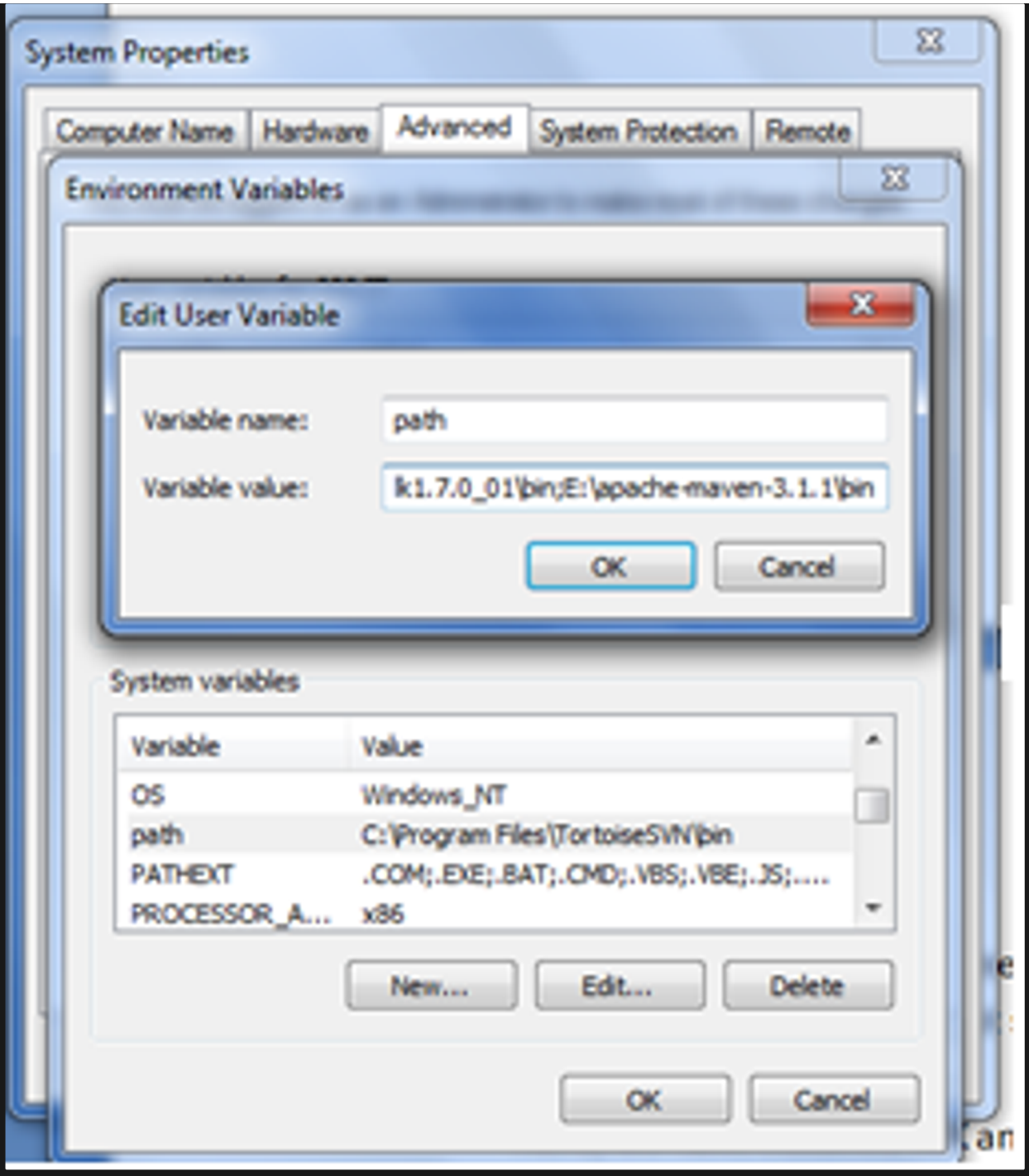
---

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** .



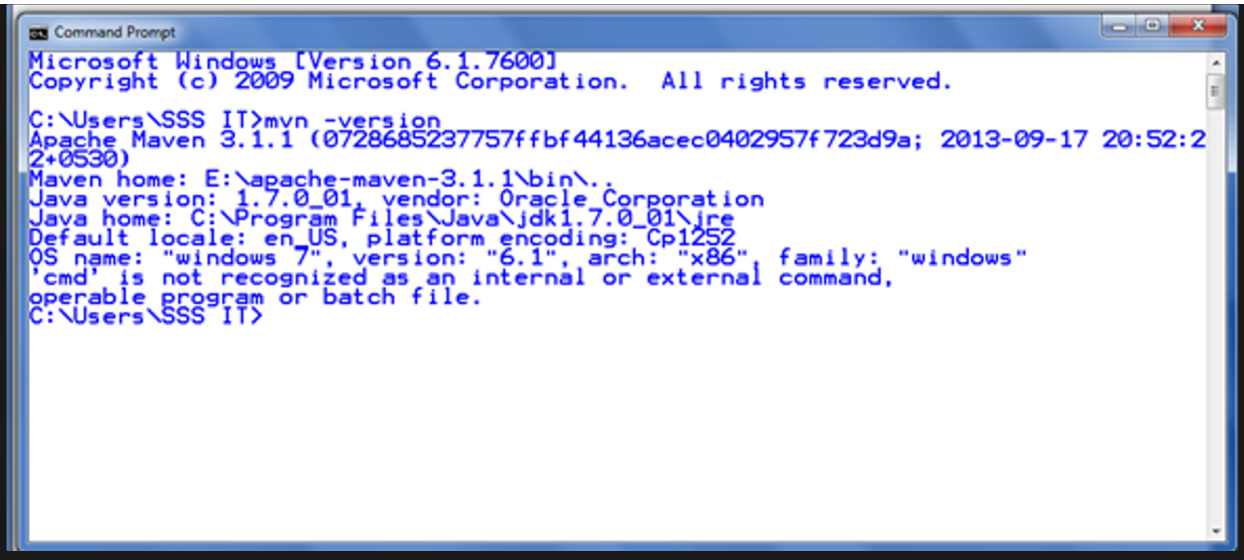
4)Verify maven

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:



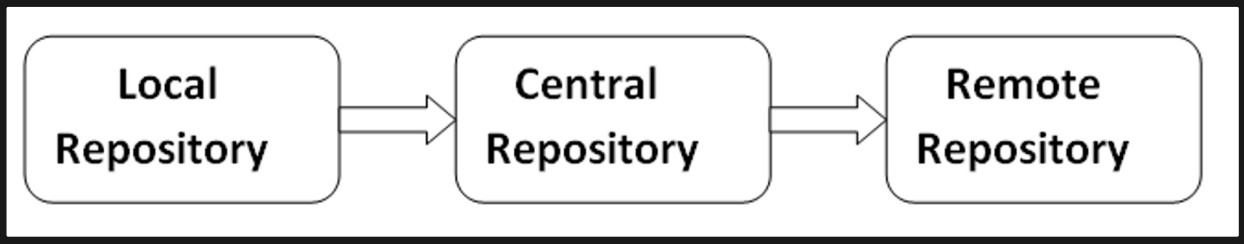
Maven Repository

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**.



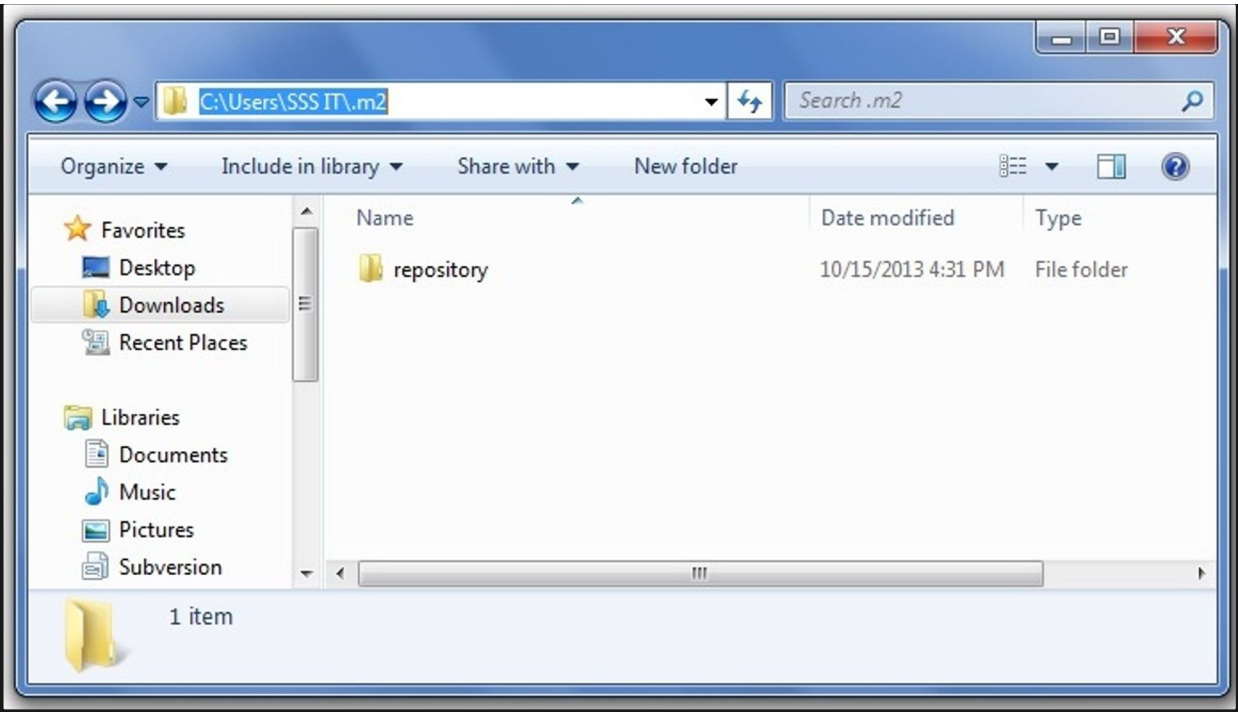
If dependency is not found in these repositories, maven stops processing and throws an error.

---

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

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd"> <!-- localRepository | The path to the
local repository maven will use to store artifacts. | | Default:
${user.home}/.m2/repository <localRepository>/path/to/local/repo</localRepository> -->
... </settings>
```

```
<! -- Now change the path to local repository. After changing the path of local
repository, it will look like this: --> <! -- settings.xml --> <settings
xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
<localRepository>e:/mavenlocalrepository</localRepository> ... </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 central.sonatype.com

---

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 adding the MySQL JDBC driver jar file in our application class path using Maven

pom.xml

```
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd"> <modelVersion>4.0.0</modelVersion>
<groupId>com.masai</groupId> <artifactId>application1</artifactId> <version>0.0.1-
SNAPSHOT</version> <properties> <maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target> </properties> <dependencies> <!--
https://mvnrepository.com/artifact/mysql/mysql-connector-java --> <dependency>
<groupId>mysql</groupId> <artifactId>mysql-connector-java</artifactId>
<version>8.0.28</version> </dependency> </dependencies> </project>
```

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

Note: after changing the Java version, you need to update the maven project.

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:

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

```
<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.javatpoint.application1</groupId> <artifactId>my-app</artifactId>
<version>1</version> </project>
```

Maven pom.xml file with additional elements

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

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

```
<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.javatpoint.application1</groupId> <artifactId>my-application1</artifactId>
<version>1.0</version> <packaging>jar</packaging> <name>Maven Quick Start Archetype</name> <url>http://maven.apache.org</url> <dependencies> <dependency>
<groupId>junit</groupId> <artifactId>junit</artifactId> <version>4.8.2</version>
<scope>test</scope> </dependency> </dependencies> </project>
```

Maven Eclipse Example

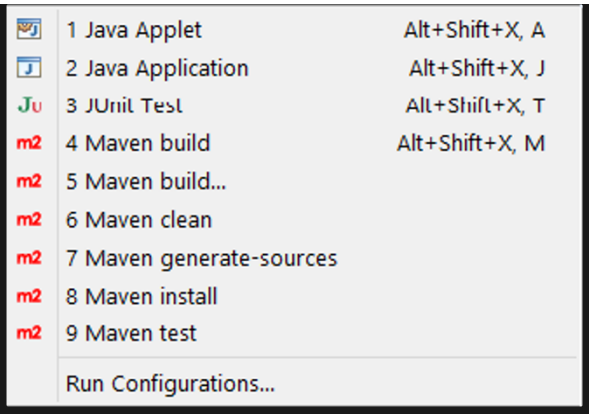
In eclipse, click on File menu → New → Project → Maven → Maven Project. → Next → Next → Next. Now write the group Id, artifact Id, Package as shown in below figure → finish.

Now you will see a maven project with complete directory structure. All the files will be created automatically such as Hello Java file, pom.xml file, test case file etc.

Now you can see the code of App.java file and run it. It will be like the given code:

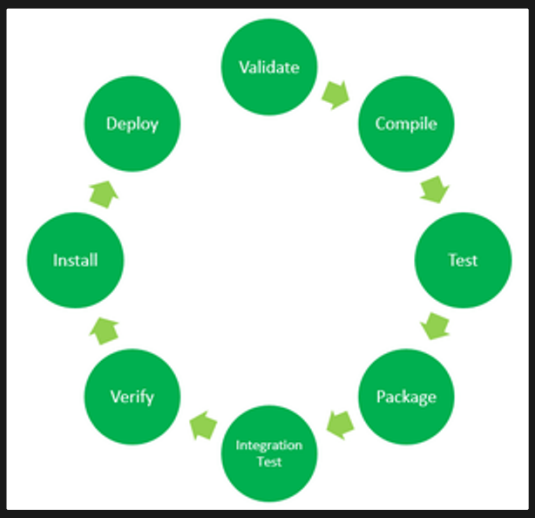
```
public class App { public static void main( String[] args ) { System.out.println( "Hello World!" ); } }
```

If you right click on the project → Run As, you will see the maven options to build the project.



Maven Lifecycle:

Below is a representation of the default Maven lifecycle and its 8 steps: Validate, Compile, Test, Package, Integration test, Verify, Install and Deploy.



The default Maven lifecycle consists of 8 major steps or phases for compiling, testing, building and installing a given Java project as specified below:

1. **Validate:** This step validates if the project structure is correct. For example – It checks if all the dependencies have been downloaded and are available in the local repository.
2. **Compile:** It compiles the source code, converts the .java files to .class and stores the classes in target/classes folder.
3. **Test:** It runs unit tests for the project.
4. **Package:** This step packages the compiled code in distributable format like JAR or WAR.
5. **Integration test:** It runs the integration tests for the project.
6. **Verify:** This step runs checks to verify that the project is valid and meets the quality standards.
7. **Install:** This step installs the packaged code to the local Maven repository.

8. **Deploy:** It copies the packaged code to the remote repository for sharing it with other developers.

Maven follows a sequential order to execute the commands where if you run step *n*, all steps preceding it (Step 1 to *n-1*) are also executed. For example – if we run the Installation step (Step 7), it will validate, compile, package and verify the project along with running unit and integration tests