MAVEN TUTORIAL

Apache Maven, is an innovative software project management tool, provides new concept of a project object model (**POM**) file to manage project's build, dependency and documentation. The most powerful feature is able to download the project dependency libraries automatically.

Maven tutorial provides basic and advanced concepts of **apache maven** technology. Our maven tutorial is developed for beginners and professionals.

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.Current version of Maven is 3.

Difference between Ant and Maven

Ant and **Maven** both are build tools provided by Apache. The main purpose of these technologies is to ease the build process of a project.

There are many differences between ant and maven that are given below:

Ant	Maven
Ant **doesn't has formal conventions**, so we need to provide information of the project structure in build.xml file.	Maven **has a convention** to place source code, compiled code etc. So we don't need to provide information about the project structure in pom.xml file.
Ant is **procedural**, you need to provide information about what to do and when to do through code. You need to provide order.	Maven is **declarative**, everything you define in the pom.xml file.
There is **no life cycle** in Ant.	There is **life cycle** in Maven.
It is **a tool** box.	It is **a framework**.
It is **mainly a build tool**.	It is **mainly a project management tool**.
The ant scripts are **not reusable**.	The maven plugins are **reusable**.
It is **less preferred** than Maven.	It is **more preferred** than Ant.

What is Maven?

Maven is a project management and comprehension tool. Maven provides developers a complete build lifecycle framework. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

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
- Reporing
- 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

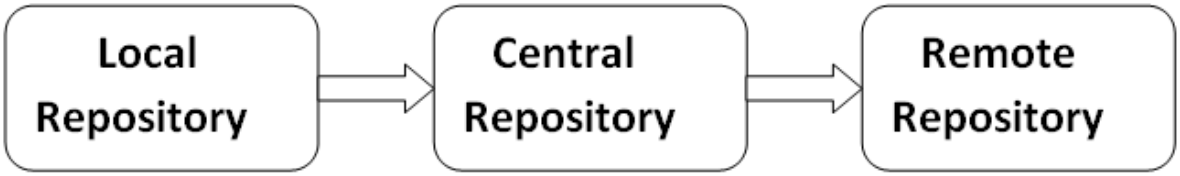
Maven Repository

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**.



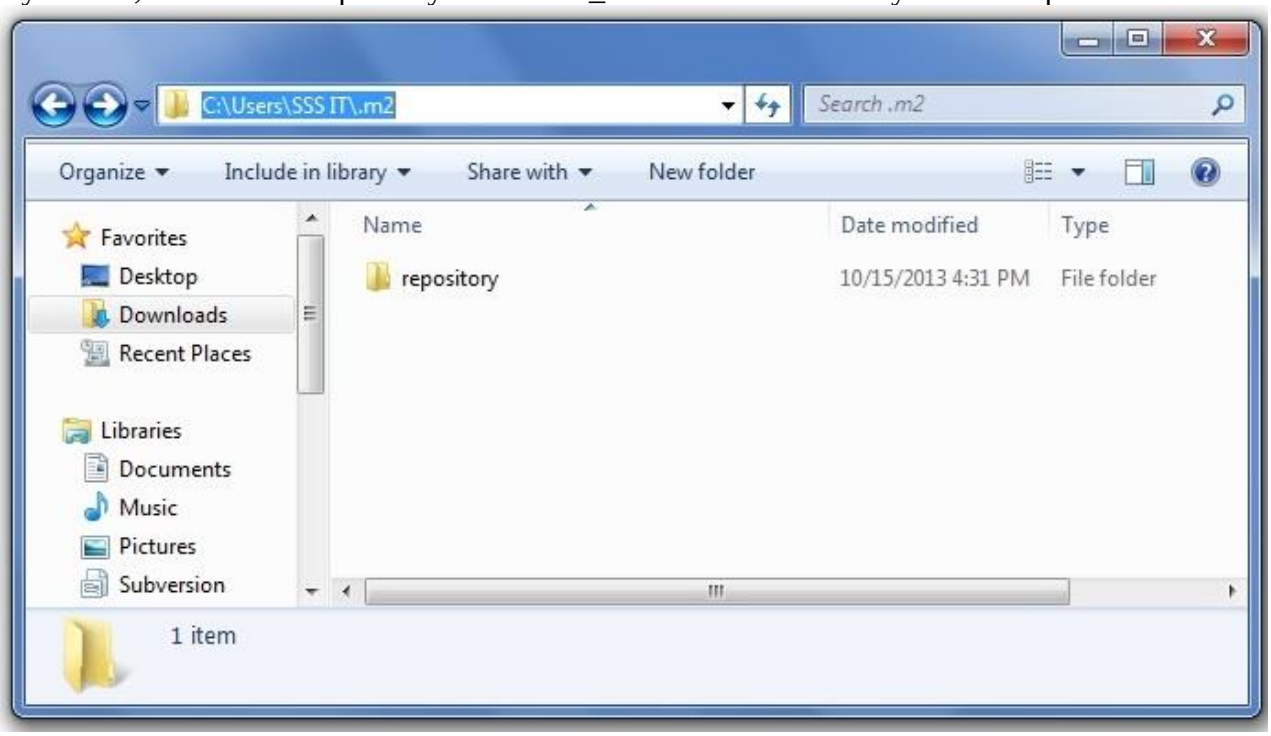
If dependency is not found in these repositories, maven stops processing and throws an error.

---

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 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">
5.
6.    <modelVersion>4.0.0</modelVersion>
7.
8.    <groupId>com.javatpoint.application1</groupId>
9.    <artifactId>my-application1</artifactId>
10.   <version>1.0</version>
11.   <packaging>jar</packaging>
12.
13.   <name>Maven Quick Start Archetype</name>
14.   <url>http://maven.apache.org</url>
15.
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>
24.
25. </project>
```

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

What is POM?

The easiest way to describe a POM in a maven project is, it is nothing but the core element of any maven project. Basically any maven project consists of one configurable file called pom.xml, which stands for the abbreviation **Project Object Model**. This pom.xml will always be located in the root directory of any maven project. This file represents the very basic and fundamental unit in maven.

The **pom.xml** basically contains the information related to the project which is built or to be built in. It contains all the necessary information about the configuration details, dependencies included and plug-ins included in the project. In simple, it contains the details of the build life cycle of a project.

Below are some of the configurations that can be handled in the pom.xml file :

- Dependencies used in the projects (Jar files)
- Plugins used (report plugin)
- Project version
- Developers involved in the project
- Mailing list
- Reporting

- Build profiles

What is Archetype?

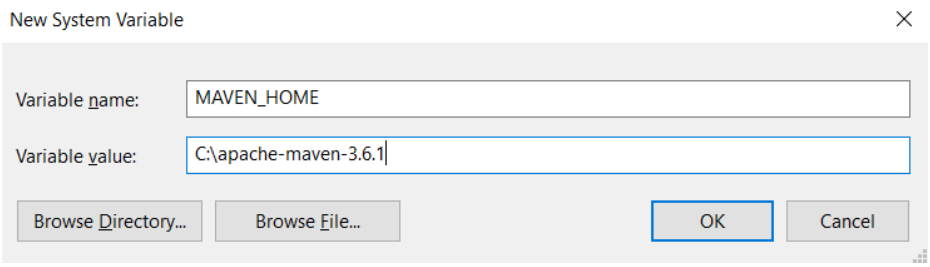
Archetype is a Maven plugin whose task is to create a project structure as per its template. We are going to use *quickstart* archetype plugin to create a simple java application here.

Download Maven and Set Up Maven Environment Variable

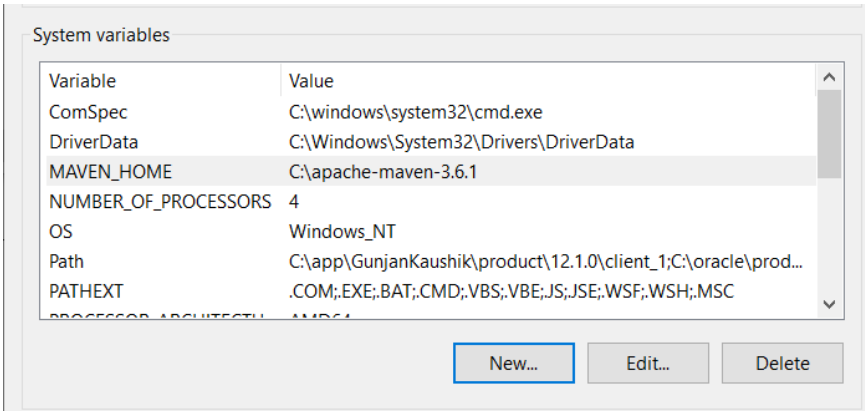
1. Maven can be downloaded from below location:
   https://Maven.apache.org/download.cgi

	Link	Checksums	Signature
Binary tar.gz archive	apache-maven-3.6.1-bin.tar.gz	apache-maven-3.6.1-bin.tar.gz.sha512	apache-maven-3.6.1-bin.tar.gz.asc
Binary zip archive	apache-maven-3.6.1-bin.zip	apache-maven-3.6.1-bin.zip.sha512	apache-maven-3.6.1-bin.zip.asc
Source tar.gz archive	apache-maven-3.6.1-src.tar.gz	apache-maven-3.6.1-src.tar.gz.sha512	apache-maven-3.6.1-src.tar.gz.asc
Source zip archive	apache-maven-3.6.1-src.zip	apache-maven-3.6.1-src.zip.sha512	apache-maven-3.6.1-src.zip.asc

2. Extract it to some location in your machine as per your convenience. For me it is lying at 'C:/apache-Maven-3.6.1'
3. You can set up the Maven Environment Variable similar to how we set up the Java Environment Variable in steps above.
4. Type 'Maven_HOME' in the Variable name box and 'C:\apache-Maven-3.6.1' in the Variable value box.



5. You'll now be able to see the newly created Maven variable under 'System Variables'.

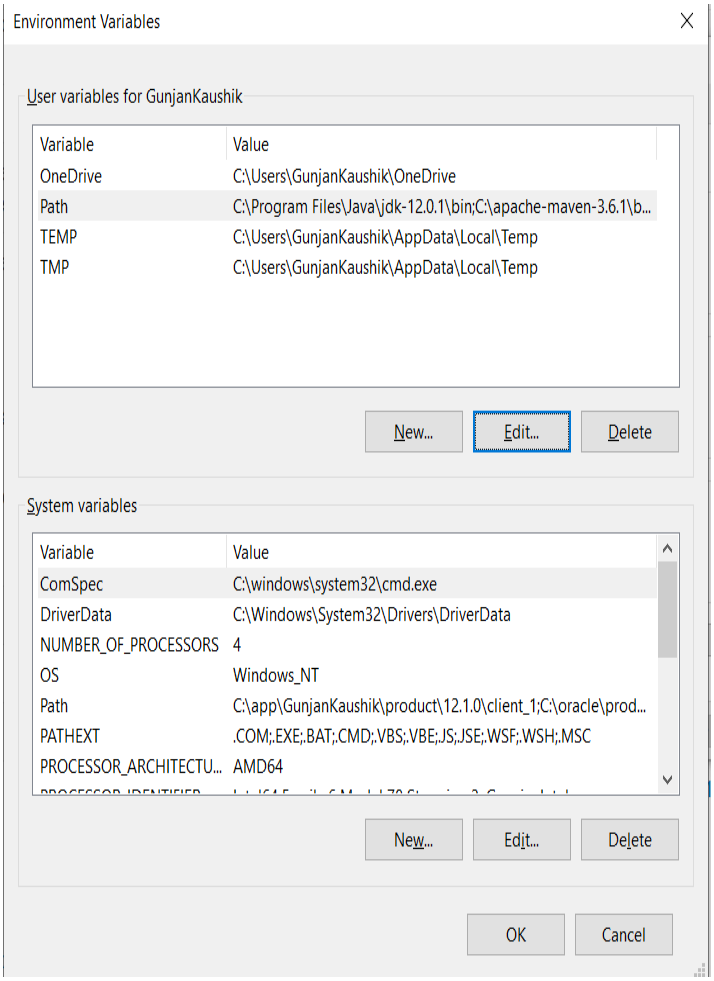


Step 4: Updating the Path Variable

In order to run Maven from the command line, we have to necessarily update the Path Variable with Maven's installation 'bin' directory.

1. Open system properties through My Computer.
2. Navigate to 'Advanced System Settings'.
3. Click on 'Environment Variables'.
4. Click on the Edit button under user variables.

5. Type ‘PATH’ in the Variable name box & ‘C:\apache-Maven-3.6.1\bin’ in the Variable value box.



Step 5: Testing the Maven Installation

Maven is now successfully installed in your system. Now let’s verify it from the Windows command line. Open command prompt and type mvn -version and hit Enter. Check to see the version of Maven installed in your system being displayed in the results.

Now you’re all set with Maven Installation now and can go ahead with creating projects using Maven.

Create your First Maven Project

Just like the Maven installation we discussed earlier in the Selenium Maven tutorial, you can also create a Maven project either through Eclipse IDE or through Command Line.

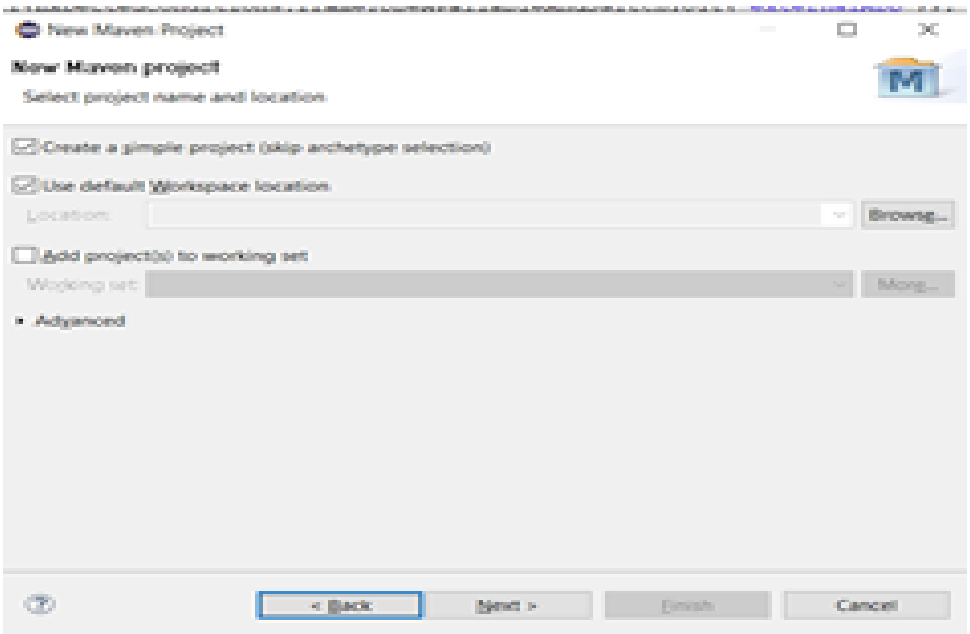
Creating Maven Project with Eclipse IDE

Below are the steps to create a Maven Project with Eclipse IDE:

Step 1: Create a new project from the Eclipse IDE.

Step 2: From the new project window expand Maven and select Maven Project and then click on Next.

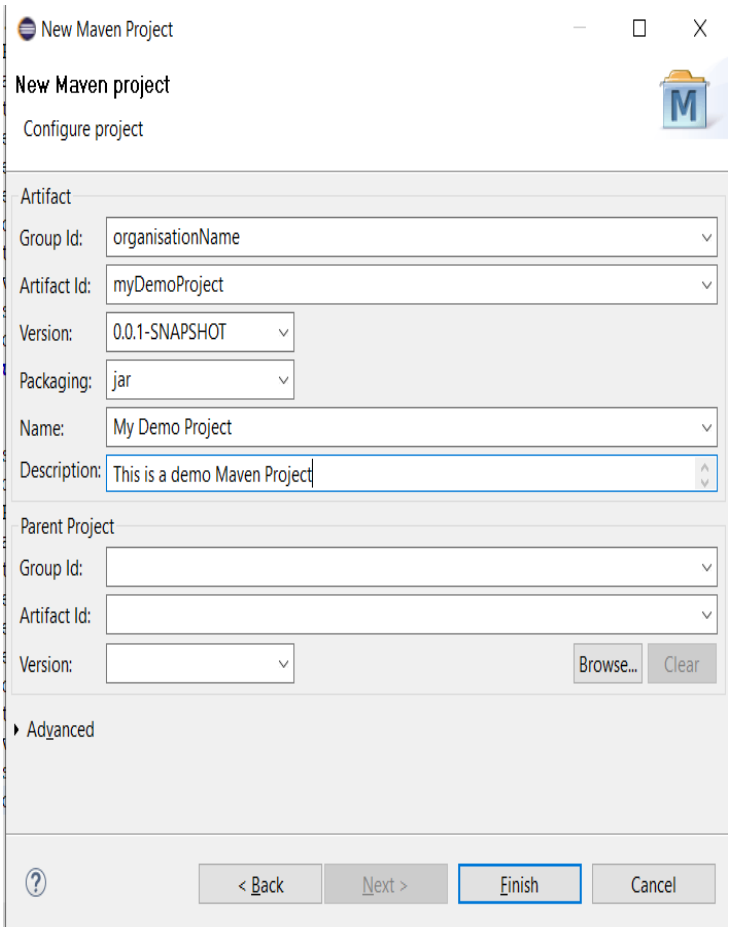
Step 3: You may create a simple project or just let go of this option. For now, we'll use a simple project which would create a simple Maven-enabled Java project.



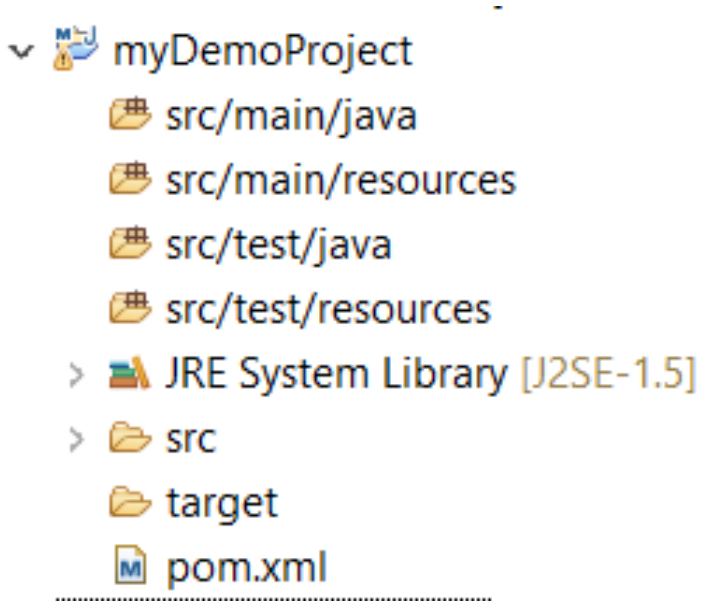
Step 4: Now upon clicking Next you'll need to type information of the Maven project is created. You may refer below descriptions to fill in the values:

Group Id- corresponds to your organization's name.
Artifact Id- refers to the project name.

The version can be chosen flexibly. If your project does not have any parent dependencies, you don't need to fill in the project dependencies. Just fill in the appropriate information and click on 'Finish'.



Step 5: Your Maven project has now been created!



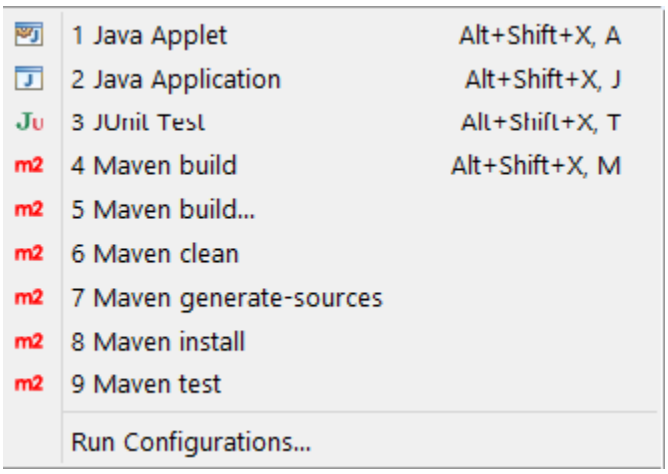
Note: Java code is placed in /src/main/java, resources are kept in /src/main/resources, testing code is placed in /src/test/java and the testing resources are placed in /src/test/resources.

To see the following directory structure you need to Create a New Java Project on your Eclipse IDE, and import the following project there.

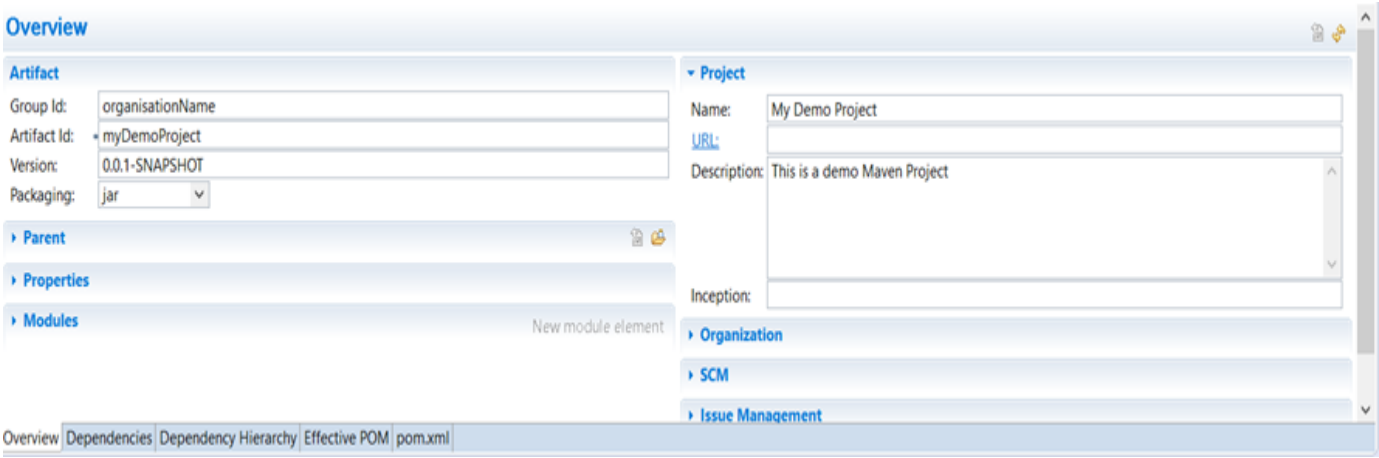
Table below will describe the main directories that are created as a part of the project.

Plugin name	Description
myDemoProject	Root folder of the project containing source and pom.xml
src\main\java	Contains all the java code under the package com.java.samples
src\main\test	Contains all the java test code under the package com.java.samples
src\main\resources	Contains all the resources like xml\properties file

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



Step 6: You may now open the pom.xml to view the structure set up by Maven. You'll see all the information that we entered in 'step 4' here. You can use the tabs at the bottom to change the view. The pom.xml tab has the pom XML code for the Maven project.



```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apac
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>organisationName</groupId>
4   <artifactId>myDemoProject</artifactId>
5   <version>0.0.1-SNAPSHOT</version>
6   <name>My Demo Project</name>
7   <description>This is a demo Maven Project</description>
8 </project>
```

Overview | Dependencies | Dependency Hierarchy | Effective POM | pom.xml

The Maven Project is now ready to be used.

Next, let us see how we can create a Maven project using Command-Line.

Step 1: Open a Command Prompt and navigate to the folder where you want to set up your project. Once navigated, type below command:

mvn archetype:generate -DgroupId=demoProject -DartifactId=DemoMavenProject -DarchetypeArtifactId=Maven-archetype-quickstart -DinteractiveMode=false

Here, DgroupId is the organization name, DartifactId is the project name and DarchetypeArtifactId is the type of Maven project.

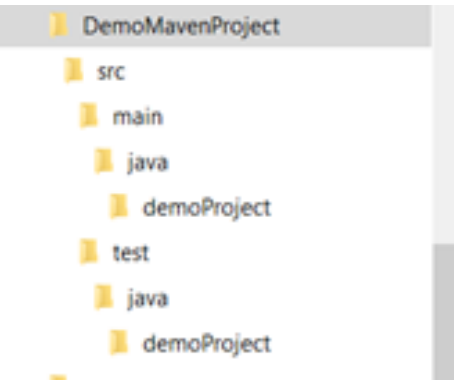
On clicking Enter, your Maven project will be created.

```
C:\>mvn archetype:generate -DgroupId=demoProject -DartifactId=DemoMavenProject -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
C:\
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< org.apache.maven:standalone-pom >-------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:3.1.2:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:3.1.2:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO]
[INFO] --- maven-archetype-plugin:3.1.2:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: C:\
[INFO] Parameter: package, Value: demoProject
[INFO] Parameter: groupId, Value: demoProject
[INFO] Parameter: artifactId, Value: DemoMavenProject
[INFO] Parameter: packageName, Value: demoProject
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\DemoMavenProject
[INFO] ----------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ----------------------------------------------------------------------------
[INFO] Total time:  36.016 s
[INFO] Finished at: 2019-08-23T13:44:03+05:30
[INFO] ----------------------------------------------------------------------------

C:\>
```

Step 2: You can go to the project location to see the newly created Maven project. You can open the pom.xml file, which is in the project folder, by default the POM is generated like this:

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>demoProject</groupId>
  <artifactId>DemoMavenProject</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>DemoMavenProject</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Step 3: You can view the default folder structure of your Maven project.



Now that we know how to create a Maven project, let's try to integrate Selenium with Maven. But before we do that, we need to understand the various Dependencies that would help with this integration.

Selenium Maven Dependency For Your Automation Project

All the external libraries that are used in a project are called dependencies. Maven has an excellent feature that automatically downloads required libraries from its central repository, which makes it easy as you don't have to store them locally. Below is an example of writing a Selenium Maven dependency in your pom.xml:

```
<dependency>
  <groupId>org.Seleniumhq.Selenium</groupId>
  <artifactId>Selenium-java</artifactId>
  <version>4.0.0-alpha-1</version>
  </dependency>
```

On adding the above Selenium Maven dependency, Maven will download the Selenium java library into our local Maven repository.

There is another Selenium Maven dependency that can be added to pom.xml based on the requirement. Few examples that you might have to use in our Selenium project are :

TestNG Selenium Maven Dependency:

This would import the testing framework dependency for Java.

```
<dependency>
 <groupId>org.testng</groupId>
 <artifactId>testng</artifactId>
 <version>6.14.3</version>
 <scope>test</scope>
</dependency>
```

A Complete Guide For Your First TestNG Automation Script

Apache POI Selenium Maven dependency:

This would download the libraries required to access Microsoft format files.

```
<dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi</artifactId>
        <version>3.17</version>
</dependency>


<dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi-ooxml</artifactId>
        <version>3.17</version>
</dependency>
```

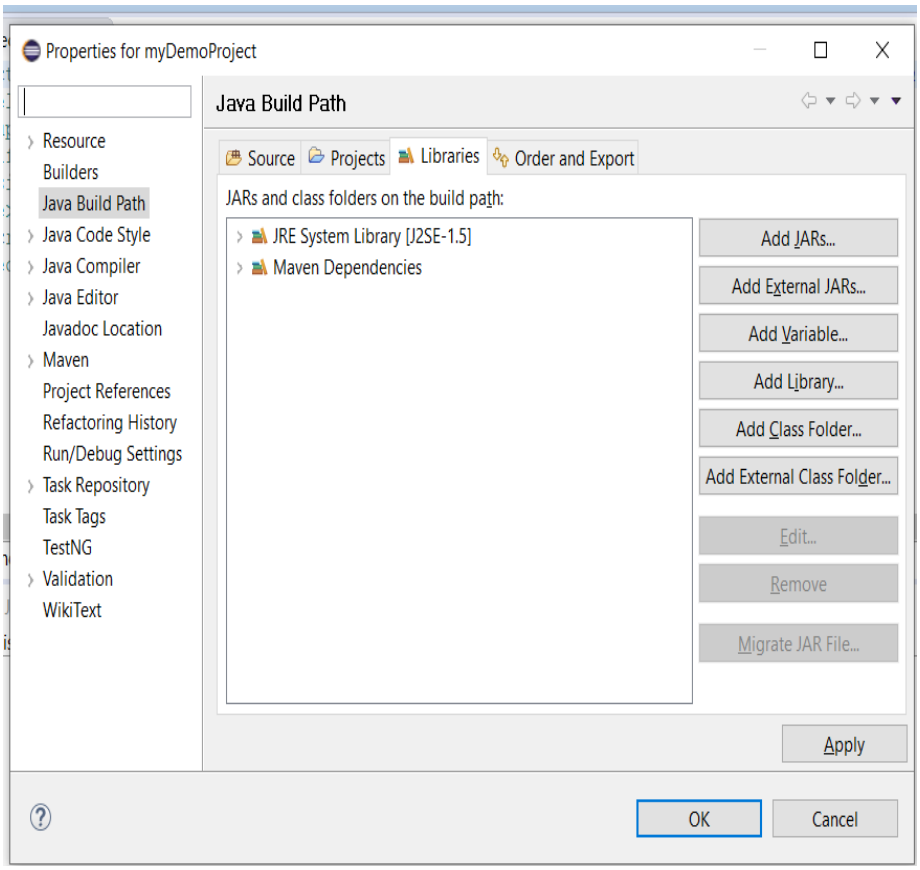
view rawMavendependency.java hosted with ♡ by **GitHub**

You can add these Selenium Maven dependencies in your pom.xml as shown 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">
```

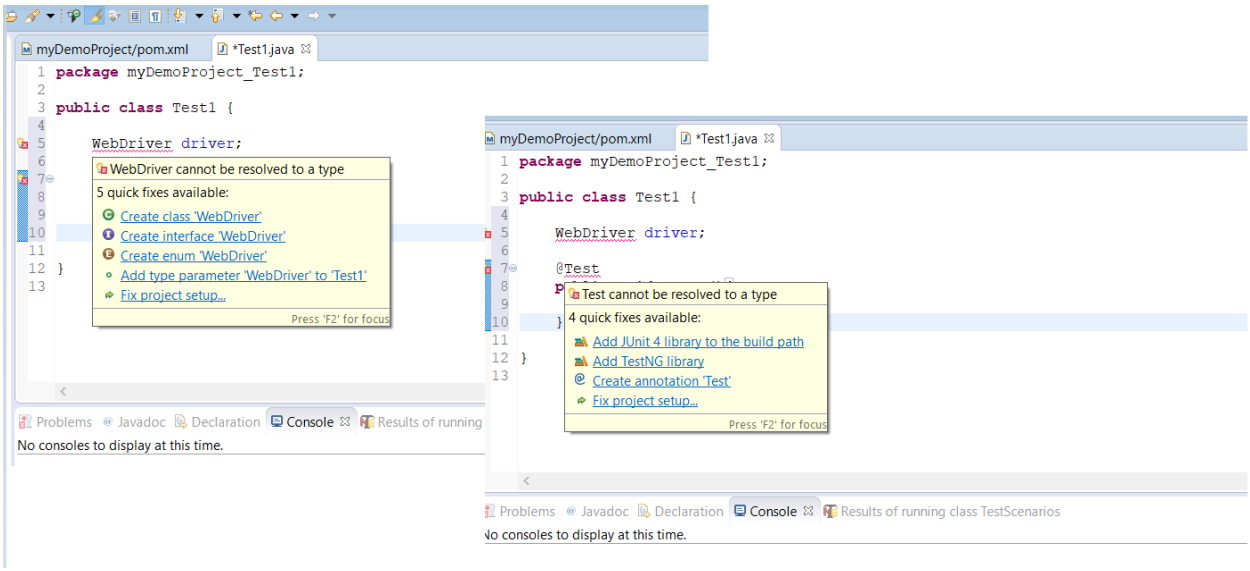
```
    <modelVersion>4.0.0</modelVersion>
    <groupId>organisationName</groupId>
    <artifactId>myDemoProject</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <dependencies>

        <dependency>
                <groupId>org.Seleniumhq.Selenium</groupId>
                <artifactId>Selenium-java</artifactId>
                <version>4.0.0-alpha-1</version>
        </dependency>

        <dependency>
                <groupId>org.testng</groupId>
                <artifactId>testng</artifactId>
                <version>6.9.10</version>
                <scope>test</scope>
        </dependency>

    </dependencies>
</project>
```

In case you want to validate how these Selenium Maven dependency import the required libraries, you'll have to go back to the demo project and see what libraries are present by default. For doing the same you need to right-click on your project name and Configure the Build Path for you project and check under the Libraries tab of the newly opened window:



Here you can see the default libraries present in your project for this Selenium Maven tutorial.

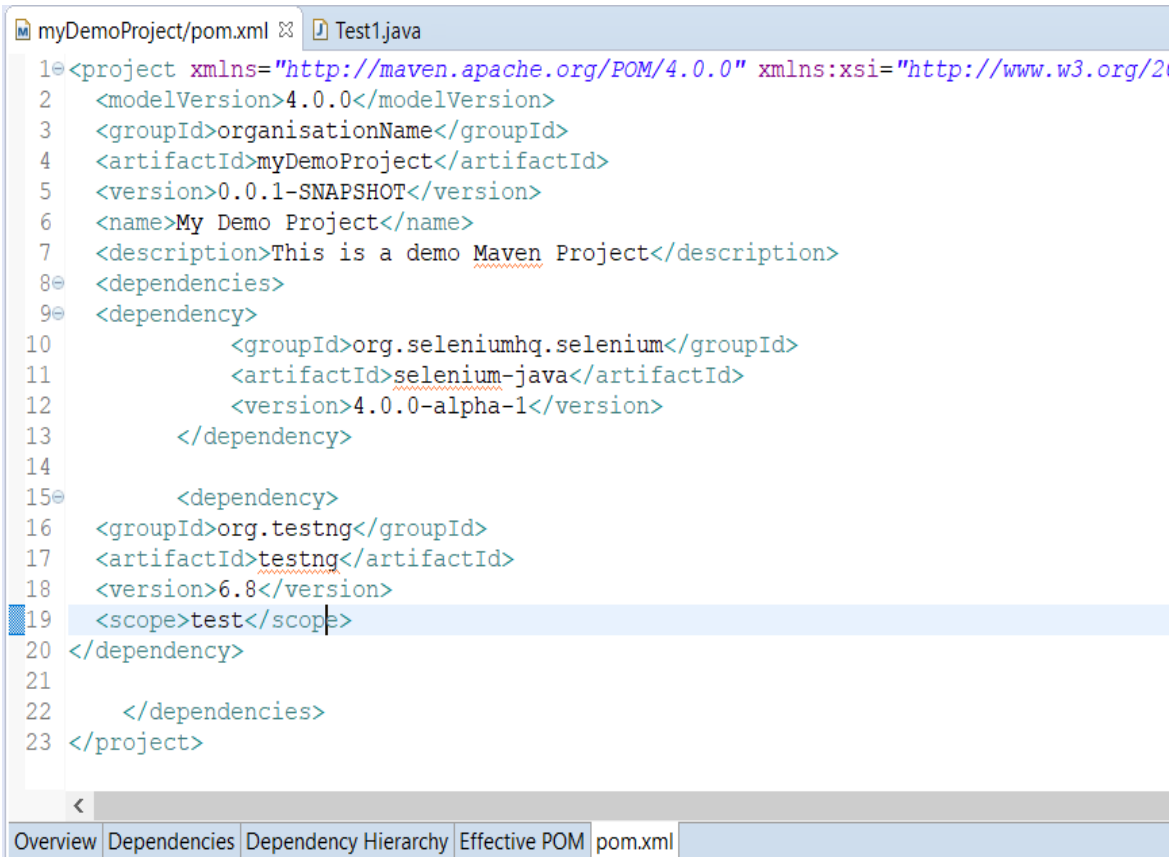
For a much clearer vision, I'll create a sample class to use Selenium components and some testNG annotations. Please note that I'll create the sample test class under the src/test/java folder of our project structure. You can clearly see below that the error correction can be done after adding the libraries.



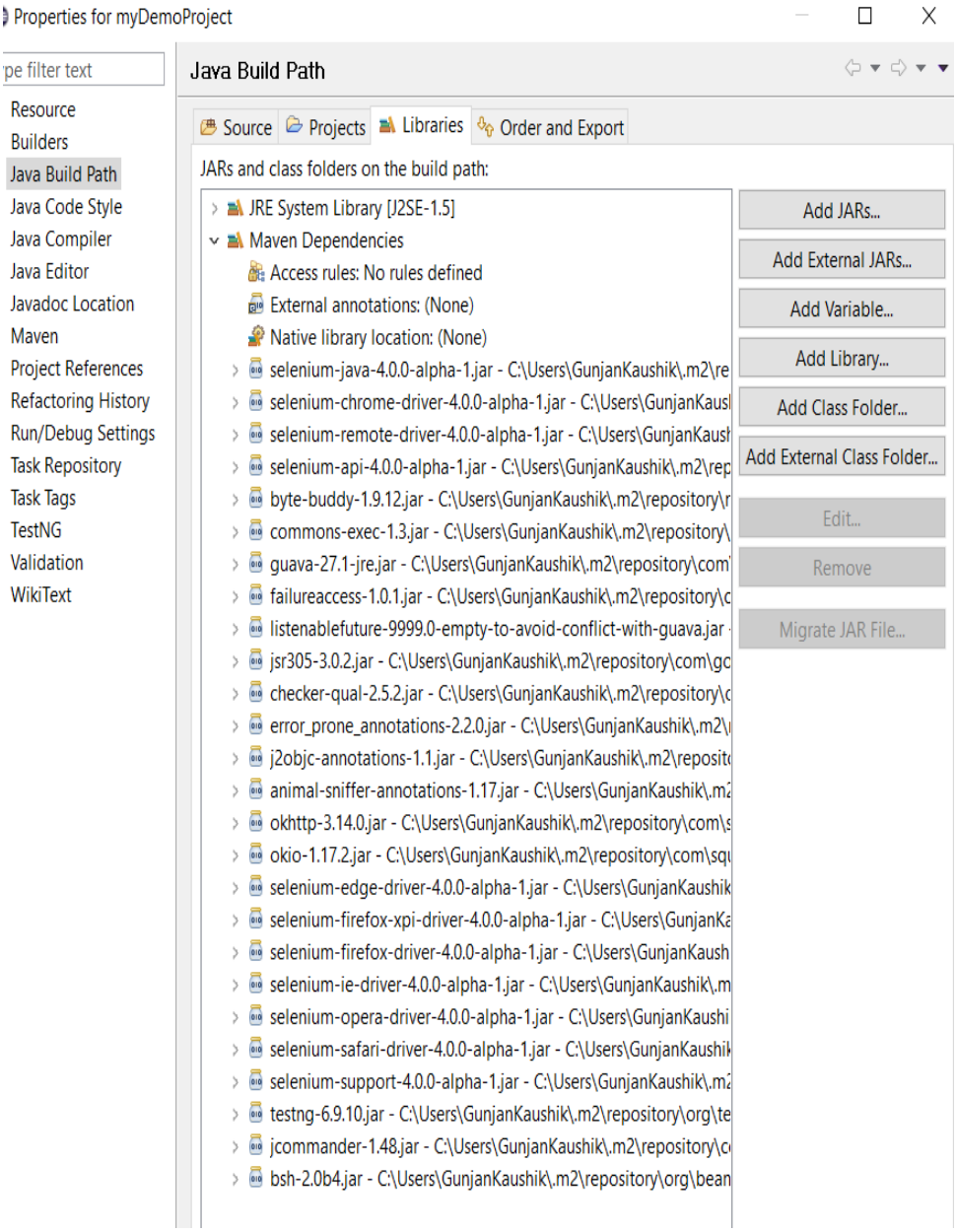
So, now rather than manually adding libraries by configuring the project build path in this Selenium Maven tutorial, I'll write the dependencies for the project in pom.xml, and Maven will directly download those from its repository. This saves the trouble of doing it manually and reducing the chance of missing out on adding some of the jars. So, here is how you add the dependencies:

The pom.xml before the Selenium Maven dependency is added:

Pom.xml after the Selenium Maven dependency is added:



After you have saved and refreshed your project, check the build path and see the Selenium and testNG libraries being added Selenium Maven dependency.



Also, you can now go to your test class and see the different options to correct the error thrown in this Selenium Maven tutorial:

```
myDemoProject/pom.xml    Test1.java
1  package myDemoProject_Test1;
2
3  public class Test1 {
4
5      WebDriver driver;
6
7
8
9
10
11
12 }
13
```

WebDriver cannot be resolved to a type

10 quick fixes available:

- Import 'WebDriver' (org.openqa.selenium)
- Create class 'WebDriver'
- Create interface 'WebDriver'
- Change to 'WebDriverException' (org.openqa.selenium)
- Change to 'WebDriverInfo' (org.openqa.selenium)
- Change to 'WebDriverWait' (org.openqa.selenium.support.ui)
- Change to 'WrapsDriver' (org.openqa.selenium)
- Create enum 'WebDriver'

Press 'F2' for focus

You can simply Import the WebDriver and you'd be good to go. Similarly for @Test annotation just import the testng.annotations.

```
myDemoProject/pom.xml    Test1.java
1  package myDemoProject_Test1;
2
3  import org.openqa.selenium.WebDriver;
4  import org.testng.annotations.Test;
5
6  public class Test1 {
7
8      WebDriver driver;
9
10     @Test
11     public void setUp(){
12         //your code here
13     }
14
15 }
16
```

You can add more dependencies like apache POI, extent reporting, commons email, or anything that might be specific to your project in a similar fashion to your pom.xml.

Now that I am done with configuring the project, I'll run the project to see if the tests work fine.

Maven Lifecycle In Selenium

There is a maven lifecycle in Selenium that every Maven build follows. The different methods are simply goals. Before going ahead I'll explain what these goals are.

Open a command line in your system and type mvn and then Enter.

```
C:\Users\GunjanKaushik>mvn
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.152 s
[INFO] Finished at: 2019-08-26T14:43:51+05:30
[INFO] ------------------------------------------------------------------------
[ERROR] No goals have been specified for this build. You must specify a valid lifecycle phase or a goal in the format <p
lugin-prefix>:<goal> or <plugin-group-id>:<plugin-artifact-id>[:<plugin-version>]:<goal>. Available lifecycle phases are
: validate, initialize, generate-sources, process-sources, generate-resources, process-resources, compile, process-class
es, generate-test-sources, process-test-sources, generate-test-resources, process-test-resources, test-compile, process-
test-classes, test, prepare-package, package, pre-integration-test, integration-test, post-integration-test, verify, ins
tall, deploy, pre-clean, clean, post-clean, pre-site, site, post-site, site-deploy. -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/NoGoalSpecifiedException

C:\Users\GunjanKaushik>
```

You can see a Build failure message with error being displayed saying that no goal has been defined in this Selenium Maven tutorial. As you parse through this message you can see the different goals that can be defined for our Maven project. I'll quickly go through these default goals before going into detail about the goals required for the Selenium project.

1. validate: would check if our project is correct and all the required information available
2. compile: would compile the project source code
3. test: would unit test the compiled source code of our project
4. package: would package the compiled code into the distributable formats, like JAR
5. integration-test: would deploy the package into an environment where we would run the integration tests
6. verify: would verify if the package is valid
7. install: would locally install the package
8. deploy: would be used in integration or release environment by copying the final project into a remote repository where it can be accessed by other projects or developers
9. clean: cleans up previous build artifacts
10. site: creates site documentation for the project

Of the above-said default goals, three are crucial for Selenium test automation. There are clean, install, and tests.

You can either use these goals alone or use it as a combination like clean-install for this Selenium Maven tutorial.

Clean- It would clean the target folder, i.e. the folder where the previous build's libraries, build files(war, tar or jar files), reports, output files, etc are saved. On executing mvn -clean this target folder will be deleted.

Install- It would install all the dependencies, in case of missing jars, and create the deployment file(war/jar files in case of JAVA) and then it'll run the test.

Test- It will simply run the test without creating any deployment file.

When using Eclipse IDE, you can directly use any of these three goals by right-clicking on your pom.xml, then Run As and selecting any of the options.

Maven Clean

I'll start with selecting Maven **clean** in this Selenium Maven tutorial, you can see the output below:

```
myDemoProject/pom.xml    Test1.java
9
10    @Test
11    public void setUp(){
12
13        System.out.println("Tihs is our first maven execution!!!");
14        //your code here
15    }
16

Problems  Javadoc  Declaration  Console  Results of running class TestScenarios
<terminated> C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (26-Aug-2019, 3:09:03 pm)
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building My Demo Project 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ myDemoProject ---
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0/plexus-utils-3.0.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0/plexus-utils-3.0.pom (4 KB at 1.5 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/sonatype/spice/spice-parent/16/spice-parent-16.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/sonatype/spice/spice-parent/16/spice-parent-16.pom (9 KB at 21.2 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/sonatype/forge/forge-parent/5/forge-parent-5.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/sonatype/forge/forge-parent/5/forge-parent-5.pom (9 KB at 20.0 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0/plexus-utils-3.0.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0/plexus-utils-3.0.jar (221 KB at 130.6 KB/sec)
[INFO] Deleting C:\Users\GunjanKaushik\workspace\myDemoProject\target
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 7.387 s
[INFO] Finished at: 2019-08-26T15:09:18+05:30
[INFO] Final Memory: 9M/121M
[INFO] ------------------------------------------------------------------------
```

So here the task of clean, which is to delete the target folder has been successfully completed and hence our Build is successful.

Maven Install

Before going to the second task of **install**, you need to add a plugin called Maven Compiler plugin.Without it the Selenium test automation build will fail. This plugin is used to identify the specific location of the compiler. You can just add the below plugin in your pom.xml and refresh the project before executing Maven install.

```
<build>

    <plugins>
```

```
        <plugin>
        <groupId>org.apache.Maven.plugins</groupId>
        <artifactId>Maven-compiler-plugin</artifactId>
        <configuration>
        <compilerVersion>1.5</compilerVersion>
        <source>1.5</source>
        <target>1.5</target>
        </configuration>
        </plugin>
        </plugins>
    </build>
```


Upon adding this piece, your pom.xml would look like below:

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="ht
  <modelVersion>4.0.0</modelVersion>
  <groupId>organisationName</groupId>
  <artifactId>myDemoProject</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>My Demo Project</name>
  <description>This is a demo Maven Project</description>
  <dependencies>
  <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>4.0.0-alpha-1</version>
        </dependency>
        <dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>6.8</version>
  <scope>test</scope>
</dependency>
    </dependencies>
    <build>
   <plugins>
          <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <compilerVersion>1.5</compilerVersion>
        <source>1.5</source>
        <target>1.5</target>
      </configuration>
    </plugin>
    </plugins>
  </build>

</project>
```

Now, go to Maven Install just like you did for Maven Clean in this Selenium Maven tutorial and observe the console output for the build installation:

Problems @ Javadoc Declaration Console TestNG

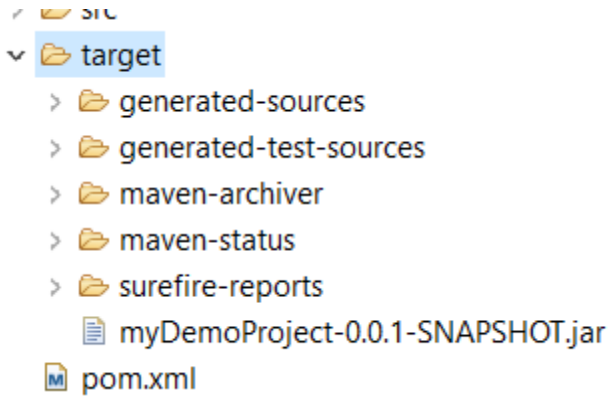
```
<terminated> C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (26-Aug-2019, 6:16:38 pm)
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building My Demo Project 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ myDemoProject ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ myDemoProject ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ myDemoProject ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ myDemoProject ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ myDemoProject ---
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-booter/2.12.4/surefire-booter-2.12.4.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-booter/2.12.4/surefire-booter-2.12.4.pom (3 KB at 1.2 
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-api/2.12.4/surefire-api-2.12.4.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-api/2.12.4/surefire-api-2.12.4.pom (3 KB at 7.5 KB/sec
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/maven-surefire-common/2.12.4/maven-surefire-common-2.12.4.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/maven-surefire-common/2.12.4/maven-surefire-common-2.12.4.pom (
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugin-tools/maven-plugin-annotations/3.1/maven-plugin-annotations-3.1.
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plugin-tools/maven-plugin-annotations/3.1/maven-plugin-annotations-3.1.p
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugin-tools/maven-plugin-tools/3.1/maven-plugin-tools-3.1.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plugin-tools/maven-plugin-tools/3.1/maven-plugin-tools-3.1.pom (16 KB at
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/reporting/maven-reporting-api/2.0.9/maven-reporting-api-2.0.9.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/reporting/maven-reporting-api/2.0.9/maven-reporting-api-2.0.9.pom (2 KB 
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/reporting/maven-reporting/2.0.9/maven-reporting-2.0.9.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/reporting/maven-reporting/2.0.9/maven-reporting-2.0.9.pom (2 KB at 4.3 K
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-toolchain/2.0.9/maven-toolchain-2.0.9.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-toolchain/2.0.9/maven-toolchain-2.0.9.pom (4 KB at 10.4 KB/sec)
```

Problems @ Javadoc Declaration Console TestNG

```
<terminated> C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (26-Aug-2019, 6:16:38 pm)
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running myDemoProject.Test1.Test1
Configuring TestNG with: org.apache.maven.surefire.testng.conf.TestNG652Configurator@368102c8
Tihs is our first maven execution!!!
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 4.534 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ myDemoProject ---
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.5/maven-archiver-2.5.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.5/maven-archiver-2.5.pom (5 KB at 12.9 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/2.1/plexus-archiver-2.1.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/2.1/plexus-archiver-2.1.pom (3 KB at 8.2 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-io/2.0.2/plexus-io-2.0.2.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-io/2.0.2/plexus-io-2.0.2.pom (2 KB at 5.0 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-components/1.1.19/plexus-components-1.1.19.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-components/1.1.19/plexus-components-1.1.19.pom (3 KB at 8.4
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus/3.0.1/plexus-3.0.1.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus/3.0.1/plexus-3.0.1.pom (19 KB at 55.9 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-interpolation/1.15/plexus-interpolation-1.15.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-interpolation/1.15/plexus-interpolation-1.15.pom (1018 B at
[INFO] Downloading: https://repo.maven.apache.org/maven2/commons-lang/commons-lang/2.1/commons-lang-2.1.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/commons-lang/commons-lang/2.1/commons-lang-2.1.pom (10 KB at 30.2 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/classworlds/classworlds/1.1-alpha-2/classworlds-1.1-alpha-2.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/classworlds/classworlds/1.1-alpha-2/classworlds-1.1-alpha-2.jar (37 KB at 52.5 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.5/maven-archiver-2.5.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.5/maven-archiver-2.5.jar (22 KB at 59.0 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-interpolation/1.15/plexus-interpolation-1.15.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-interpolation/1.15/plexus-interpolation-1.15.jar (60 KB at 1
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/2.1/plexus-archiver-2.1.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/2.1/plexus-archiver-2.1.jar (181 KB at 247.1 KB/sec
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-io/2.0.2/plexus-io-2.0.2.jar
```

```
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-diges
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest
[INFO] Installing C:\Users\GunjanKaushik\workspace\myDemoProject\target\myDemoProject-0.0
[INFO] Installing C:\Users\GunjanKaushik\workspace\myDemoProject\pom.xml to C:\Users\Gunj
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 46.804 s
[INFO] Finished at: 2019-08-26T18:17:28+05:30
[INFO] Final Memory: 16M/256M
[INFO] ------------------------------------------------------------------------
```

You can see in the console output that the Maven installer executed the tests as well. To see the installed directories in your local system you refresh your project and see the directories generated. In the below snapshot you can see all the files generated(a jar file as well since this is a simple JAVA program) as a result of Maven install. You can share this jar file directly for others to execute.



Maven Test

Similarly, we can do **Maven Test** in this Selenium Maven tutorial and see the build results in the console:

```
Javadoc  Declaration  Console  TestNG
<terminated> C:\Program Files\Java\jre1.8.0_201\bin\javaw.exe (26-Aug-2019, 7:37:28 pm)
[INFO] ------------------------------------------------------------------------
[INFO] Building My Demo Project 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ myDemoProject ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ myDemoProject ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ myDemoProject ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ myDemoProject ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ myDemoProject ---
[INFO] Surefire report directory: C:\Users\GunjanKaushik\workspace\myDemoProject\target\surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running myDemoProject Test1.Test1
Configuring TestNG with: org.apache.maven.surefire.testng.conf.TestNG652Configurator@368102c8
Tihs is our first maven execution!!!
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 6.268 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 26.914 s
[INFO] Finished at: 2019-08-26T19:37:57+05:30
```

The above steps show how execution can be done through Eclipse IDE, but in real-time project execution is done mostly through the command line So now we would use these goals from the command line. Before proceeding onto these tasks make sure that your current working directory in cmd points to your local workspace location. If we do not do so our Maven command would not be able to find the desired pom.xml and hence our Selenium test automation build would fail.

Let us first do the Maven Clean using the command prompt in this Selenium Maven tutorial. We simply need to write 'mvn clean':

```
C:\Users\GunjanKaushik\workspace\myDemoProject>mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------< organisationName:myDemoProject >-------------------
[INFO] Building My Demo Project 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ myDemoProject ---
[INFO] Deleting C:\Users\GunjanKaushik\workspace\myDemoProject\target
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.414 s
[INFO] Finished at: 2019-08-26T20:08:57+05:30
[INFO] ------------------------------------------------------------------------
```

You can see the target folder getting deleted. The next step in this Selenium Maven tutorial is to do Maven install which is done by typing 'mvn install' in the command prompt.

```
C:\Users\GunjanKaushik\workspace\myDemoProject>mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------< organisationName:myDemoProject >-------------------
[INFO] Building My Demo Project 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-resources-plugin/2.6/maven-resources-plugin-2.6.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-resources-plugin/2.6/maven-resources-plugin-2.6.pom (8.1 kB at 4.4 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/23/maven-plugins-23.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/23/maven-plugins-23.pom (9.2 kB at 24 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/22/maven-parent-22.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/22/maven-parent-22.pom (30 kB at 46 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/apache/11/apache-11.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/apache/11/apache-11.pom (15 kB at 37 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-resources-plugin/2.6/maven-resources-plugin-2.6.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-resources-plugin/2.6/maven-resources-plugin-2.6.jar (30 kB at 44 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-compiler-plugin/3.1/maven-compiler-plugin-3.1.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-compiler-plugin/3.1/maven-compiler-plugin-3.1.pom (10 kB at 20 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/24/maven-plugins-24.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/24/maven-plugins-24.pom (11 kB at 27 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/23/maven-parent-23.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/23/maven-parent-23.pom (33 kB at 82 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/apache/13/apache-13.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/apache/13/apache-13.pom (14 kB at 35 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-compiler-plugin/3.1/maven-compiler-plugin-3.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-compiler-plugin/3.1/maven-compiler-plugin-3.1.jar (43 kB at 89 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-surefire-plugin/2.12.4/maven-surefire-plugin-2.12.4.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-surefire-plugin/2.12.4/maven-surefire-plugin-2.12.4.pom (10 kB at 25 kB/s)
```

```
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-testng/2.12.4/surefire-testng-2.12.4.jar
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/1.0.4/plexus-utils-1.0.4.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-grouper/2.12.4/surefire-grouper-2.12.4.jar (38 kB at 39 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/wagon/wagon-provider-api/1.0-beta-2/wagon-provider-api-1.0-beta-2.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-testng-utils/2.12.4/surefire-testng-utils-2.12.4.jar (24 kB at 16 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-testng/2.12.4/surefire-testng-2.12.4.jar (34 kB at 22 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-artifact/2.0/maven-artifact-2.0.jar (79 kB at 48 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/1.0.4/plexus-utils-1.0.4.jar (164 kB at 77 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/wagon/wagon-provider-api/1.0-beta-2/wagon-provider-api-1.0-beta-2.jar (46 kB at 37 kB/s)

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running myDemoProject_Test1.Test1
Configuring TestNG with: org.apache.maven.surefire.testng.conf.TestNG652Configurator@51016012
Tihs is our first maven execution!!!
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.594 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ myDemoProject ---
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.5/maven-archiver-2.5.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.5/maven-archiver-2.5.pom (4.5 kB at 11 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/2.1/plexus-archiver-2.1.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/2.1/plexus-archiver-2.1.pom (2.8 kB at 6.7 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0/plexus-utils-3.0.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0/plexus-utils-3.0.pom (4.1 kB at 11 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/sonatype/spice/spice-parent/16/spice-parent-16.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/sonatype/spice/spice-parent/16/spice-parent-16.pom (8.4 kB at 14 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/sonatype/forge/forge-parent/5/forge-parent-5.pom
```

```
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/2.1/plexus-archiver-2.1.jar (184 kB at 155 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/commons-lang/commons-lang/2.1/commons-lang-2.1.jar (208 kB at 116 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0/plexus-utils-3.0.jar (226 kB at 87 kB/s)
[INFO] Building jar: C:\Users\GunjanKaushik\workspace\myDemoProject\target\myDemoProject-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ myDemoProject ---
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.5/plexus-utils-3.0.5.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.5/plexus-utils-3.0.5.pom (2.5 kB at 5.8 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus/3.1/plexus-3.1.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus/3.1/plexus-3.1.pom (19 kB at 47 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.pom (1.1 kB at 2.4 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-components/1.1.7/plexus-components-1.1.7.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-components/1.1.7/plexus-components-1.1.7.pom (5.0 kB at 13 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus/1.0.8/plexus-1.0.8.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus/1.0.8/plexus-1.0.8.pom (7.2 kB at 15 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-container-default/1.0-alpha-8/plexus-container-default-1.0-alpha-8.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-container-default/1.0-alpha-8/plexus-container-default-1.0-alpha-8.pom (7.3 kB at 15 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.5/plexus-utils-3.0.5.jar
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.jar (12 kB at 22 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.5/plexus-utils-3.0.5.jar (230 kB at 134 kB/s)
[INFO] Installing C:\Users\GunjanKaushik\workspace\myDemoProject\target\myDemoProject-0.0.1-SNAPSHOT.jar to C:\Users\GunjanKaushik\.m2\repository\organisationName\myDemoProject\0.0.1-SNAPSHOT\myDemoProject-0.0.1
-SNAPSHOT.jar
[INFO] Installing C:\Users\GunjanKaushik\workspace\myDemoProject\pom.xml to C:\Users\GunjanKaushik\.m2\repository\organisationName\myDemoProject\0.0.1-SNAPSHOT\myDemoProject-0.0.1-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  02:53 min
[INFO] Finished at: 2019-08-26T20:23:34+05:30
[INFO] ------------------------------------------------------------------------
```

You can see the console output is almost similar to what was there through Eclipse execution. Let us now perform a Maven test by typing 'mvn test' which would result in the test being executed without building jar files.

```
C:\Users\GunjanKaushik\workspace\myDemoProject>mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------< organisationName:myDemoProject >-------------------
[INFO] Building My Demo Project 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ myDemoProject ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ myDemoProject ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ myDemoProject ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ myDemoProject ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ myDemoProject ---
[INFO] Surefire report directory: C:\Users\GunjanKaushik\workspace\myDemoProject\target\surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running myDemoProject_Test1.Test1
Configuring TestNG with: org.apache.maven.surefire.testng.conf.TestNG652Configurator@51016012
Tihs is our first maven execution!!!
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 7.345 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  36.314 s
[INFO] Finished at: 2019-08-26T20:27:22+05:30
[INFO] ------------------------------------------------------------------------
```

Now that you know the basic goals for executing our automation test you're good to go to run your automation scripts through Maven!

Maven Surefire Plugin

By now you have read this term in this Selenium Maven tutorial quite some time in your console output logs, so I'll shed some light on it. The surefire plugin helps Maven to identify the tests and is used with whichever framework your project is built on. To add Surefire plugin to your pom.xml use below code snippet:

```
<properties>
<suiteXmlFile>src/main/resources/testng.xml</suiteXmlFile>
</properties>



<build>
<plugin>
<groupId>org.apache.Maven.plugins</groupId>
<artifactId>Maven-surefire-plugin</artifactId>
<version>2.17</version>
<configuration>
<suiteXmlFiles>
<suiteXmlFile>${suiteXmlFile}</suiteXmlFile>
</suiteXmlFiles>
</configuration>
</plugin>
</build>
```


Your pom.xml should look like below:

myDemoProject/pom.xml Test1.java

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSc
2    <modelVersion>4.0.0</modelVersion>
3    <groupId>organisationName</groupId>
4    <artifactId>myDemoProject</artifactId>
5    <version>0.0.1-SNAPSHOT</version>
6    <name>My Demo Project</name>
7    <description>This is a demo Maven Project</description>
8    <properties>
9  <suiteXmlFile>src/main/resources/testng.xml</suiteXmlFile>
10 </properties>
11   <dependencies>
12   <dependency>
17         <dependency>
23     </dependencies>
24     <build>
25    <plugins>
26            <plugin>
27                <artifactId>maven-compiler-plugin</artifactId>
28                <version>3.1</version>
29                <configuration>
30                    <source>1.7</source>
31                    <target>1.7</target>
32                    <fork>true</fork>
33                    <executable>C:\Program Files\Java\jdk-12.0.1\bin\javac</executable>
34                </configuration>
35            </plugin>
36
37            <plugin>
38                <groupId>org.apache.maven.plugins</groupId>
39                <artifactId>maven-surefire-plugin</artifactId>
40                <version>2.17</version>
41                <configuration>
42                    <suiteXmlFiles>
43                    <suiteXmlFile>${suiteXmlFile}</suiteXmlFile>
44                    </suiteXmlFiles>
45                </configuration>
46                </plugin>
```

Here I've put in testng.xml in src/main/resources and thereby giving its path in properties.

Now I'll run Maven test from eclipse in this Selenium Maven tutorial and see the results:

```
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-grouper/2.17/surefi
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-grouper/2.17/surefir

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running TestSuite
Tihs is our first maven execution!!!
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.997 sec - in TestSuite

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 38.307 s
[INFO] Finished at: 2019-08-26T20:56:26+05:30
[INFO] Final Memory: 10M/85M
[INFO] ------------------------------------------------------------------------
```

You can now check the reports that have been generated by default and in a similar way by typing ‘mvn test’ can execute this suite from the command prompt.