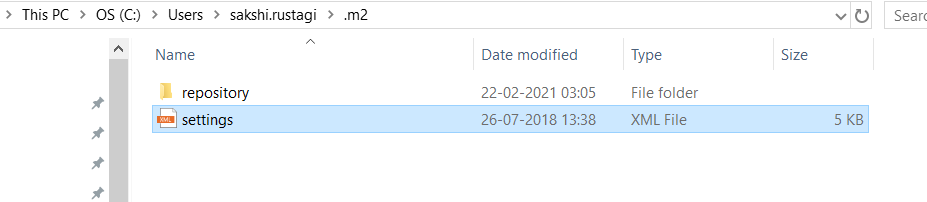
**Maven :**

Maven is a **build automation tool**. The build tool is used to set up everything which is required to run your java code independently. It takes your project’s Java source code, **compiles code**, **tests** it and **converts it into an executable Java program** : either a .jar or a .war file.

When maven build is executed, Maven automatically downloads all the dependency jars into the local repository. Usually, this folder **is** named ‘.m2’(in C:/Users/sakshi.rustagi).



Reference Link: <https://www.softwaretestinghelp.com/maven-project-setup-for-selenium-selenium-tutorial-24/>

**General Phrases used in Maven :**

* **groupId**: Generally, groupId refers to domain id. For best practices company name is used as groupId. It identifies the project uniquely.
* **artifactId**: It is basically the name of the Jar without version.
* **version**: This tag is used to create a version of the project.
* **Local repository**: Maven downloads all the required dependencies and stores in the local repository called m2.

**Path**: C:\Users\sakshi.rustagi\.m2

<dependency>

      <groupId>org.seleniumhq.selenium</groupId>

      <artifactId>selenium-java</artifactId>

      <version>3.3.1</version>

</dependency>

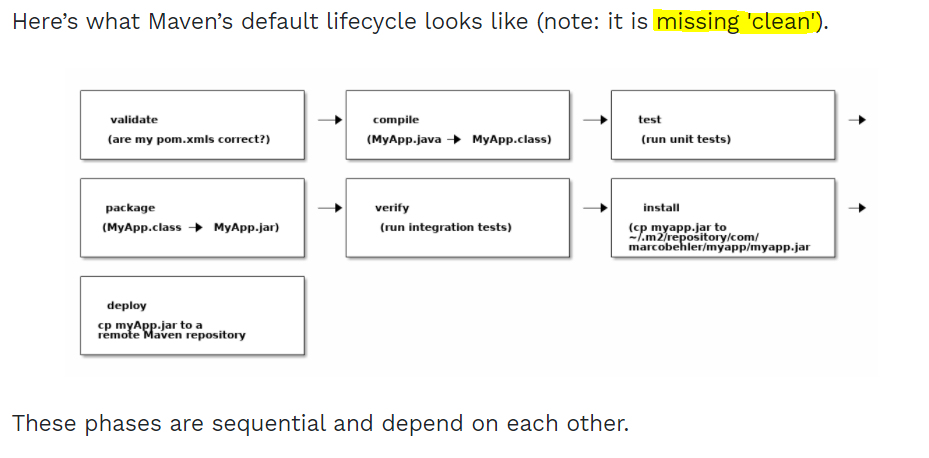
**Build Life Cycle :**

Basic maven phases are used as below.

* **clean**: Deletes all artifacts and targets which are created already.

Deletes the /target folder.

* **compile**: Used to compile the source code of the project(.java to .class files).
* **test**: Test the compiled code.
* **package**: Package is used to convert your .java source code into a .jar/.war file and puts it into the /target folder.
* **install**: First, it does a package(!). Then it takes that .jar/.war file and puts it into your local Maven repository, which lives in ~/.m2/repository.



**Installing Maven in Windows :**

[**https://mkyong.com/maven/how-to-install-maven-in-windows/**](https://mkyong.com/maven/how-to-install-maven-in-windows/)

**Steps :**

* Make sure JDK is installed, and JAVA\_HOME environment variable is configured.
* Visit [*Maven official website*](https://maven.apache.org/download.cgi), download the Maven zip file, for example : apache-maven-3.6.0-bin.zip.
* Add a MAVEN\_HOME system variables and point it to the Maven folder.



* In system variables, find PATH, clicks on the Edit... button. In “Edit environment variable” dialog, clicks on the New button and add this %MAVEN\_HOME%\bin

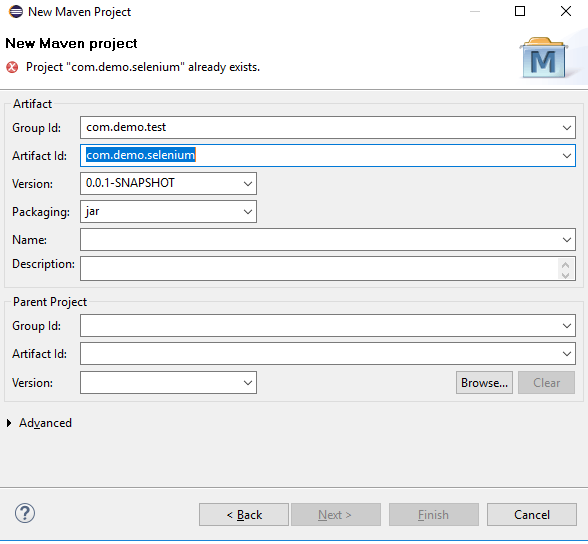


* For verification : start a new command prompt, type mvn –version.

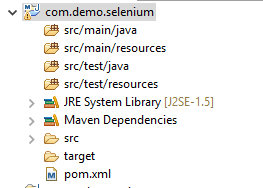
**Creating Maven project:**

In Eclipse, click on File -> New -> Project -> Maven -> Maven Project -> Next -> Check selects a simple project-> Next.

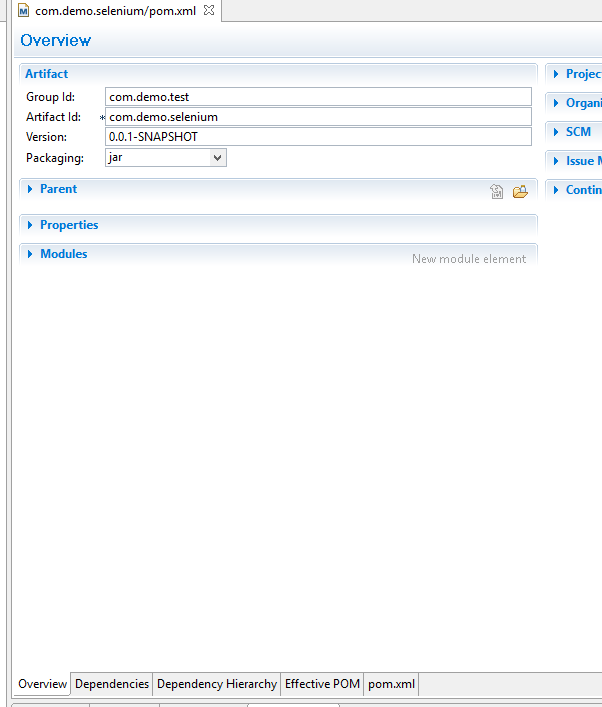
1. For a new project we need to enter its **Group Id, Artifact Id, Version and Packaging**. This can be any value as per your choice.



1. After finish, you will find the project structure is created like below. **pom.xml** is created which is used to download all dependencies.



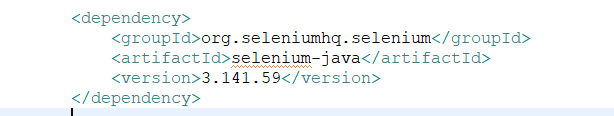
1. Project POM.xml will look like this:



1. We can add dependencies for the jars that we need to install.

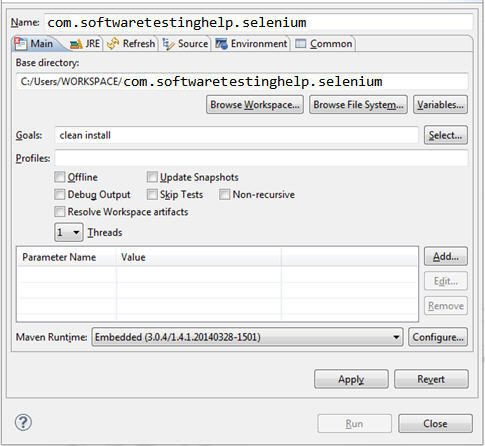
**Central repository for downloading jars :** <http://repo1.maven.org/maven2/org/>

e.g., for downloading selenium jars:



1. These jars will be automatically downloaded in .m2 folder.
2. Build the project: Right click on Project name -> Run As-> Maven Build.

In goals add “**clean install**” and click on Run.



**For clean** : mvn clean  
**For compile** : mvn compile

**For test** : mvn test

**For creating jar/war package** : mvn package  
**For install** : mvn install

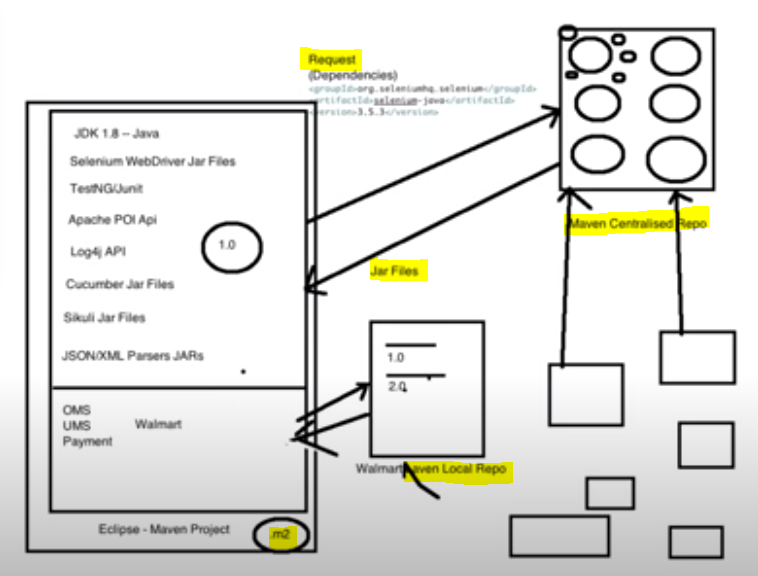
**The project will run, and jar of the project will also be created.**

If we do ‘**mvn install’** then jar and pom file will be copied over to local repository as well.

Installing D:\java\_workspace\MavenUnderstading\target\MavenUnderstading-0.0.1-SNAPSHOT.jar to C:\Users\sakshi.rustagi\.m2\repository\Maven\MavenUnderstading\0.0.1-SNAPSHOT\MavenUnderstading-0.0.1-SNAPSHOT.jar

[INFO] Installing D:\java\_workspace\MavenUnderstading\pom.xml to C:\Users\sakshi.rustagi\.m2\repository\Maven\MavenUnderstading\0.0.1-SNAPSHOT\MavenUnderstading-0.0.1-SNAPSHOT.pom

**Explanation :**



Different companies(e.g., Selenium, TestNG, Apache POI) have put their tools of different versions on **Maven Central Repository**.

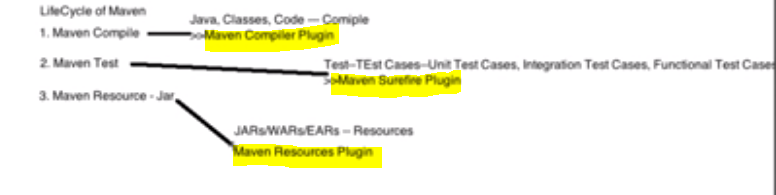
To download these, we send request in the form of **Dependencies** in our **pom.xml** file and receive response in the form of **JAR files**.

These JAR files will be stored in our system at folder : “C:/Users/Sakshi.rustagi/**.m2**”

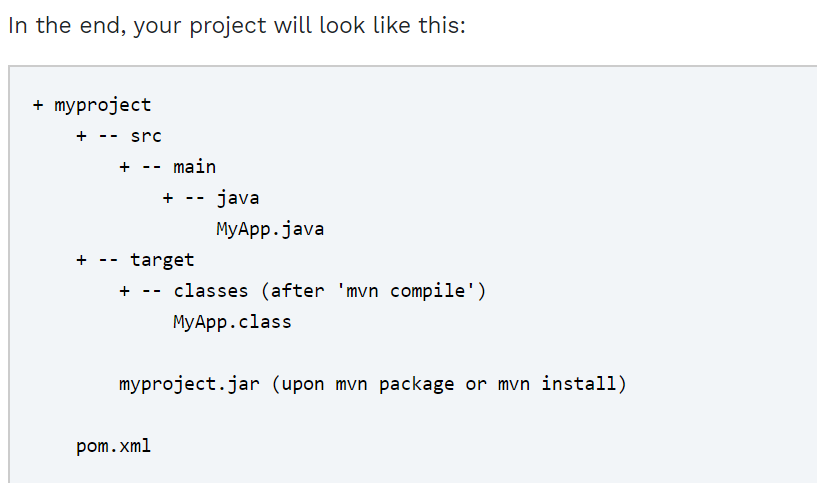
Now companies, they create their own local repositories as well (like **Nexus** in case of ION) and from there also we can download required Jars and use in our project.

**Different LifeCycles of Maven :**

* Maven Compile : **maven-compiler-plugin**
* Maven Test : **maven-surefire-plugin**
* Maven Resource(jar and build generation) : **maven-resources-plugin**



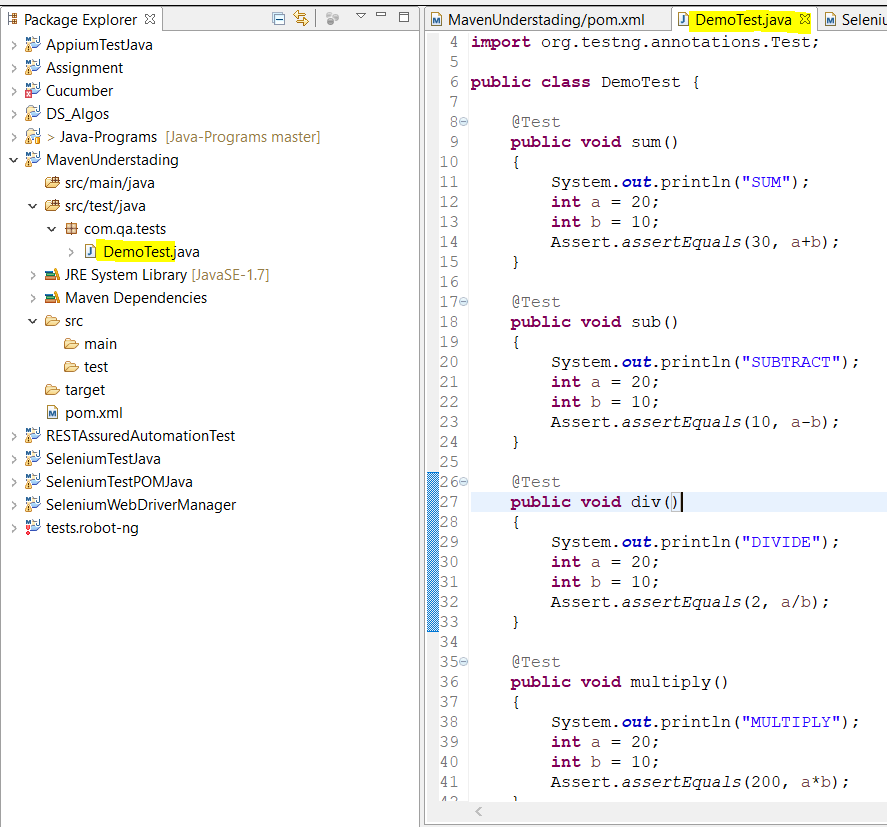
Maven considers these plugins by default. But if we want to enhance them then we need to add these in pom.xml with enhancement.



**Example :**

Suppose we have created new project with a pom and a Java file :





**To run test through TestNG :** Project Right Click -> Run As -> TestNG Test

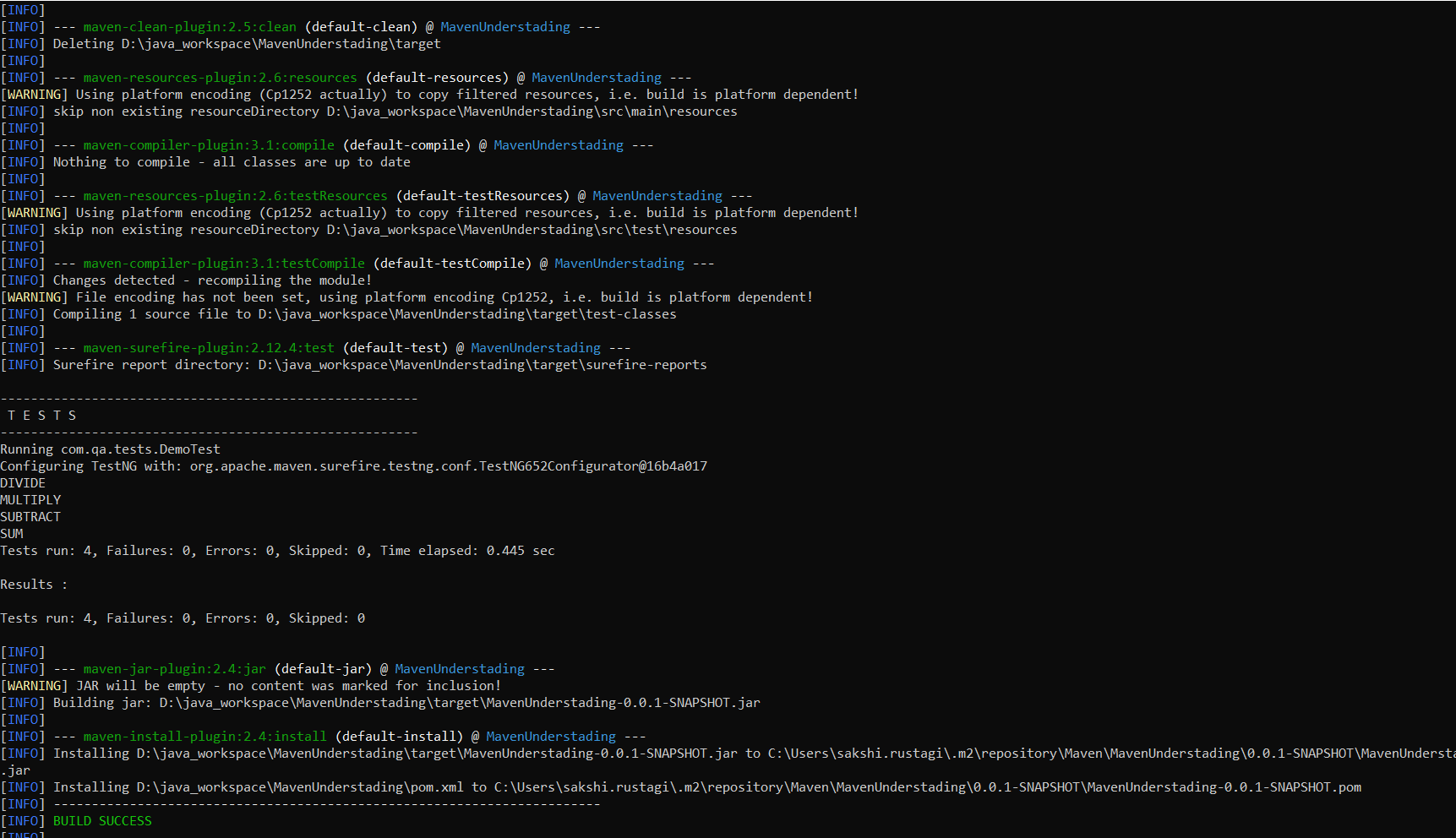
**To run test through Maven :** Project Right Click -> Run As -> Maven test

Now running though command line :

* **mvn clean install**

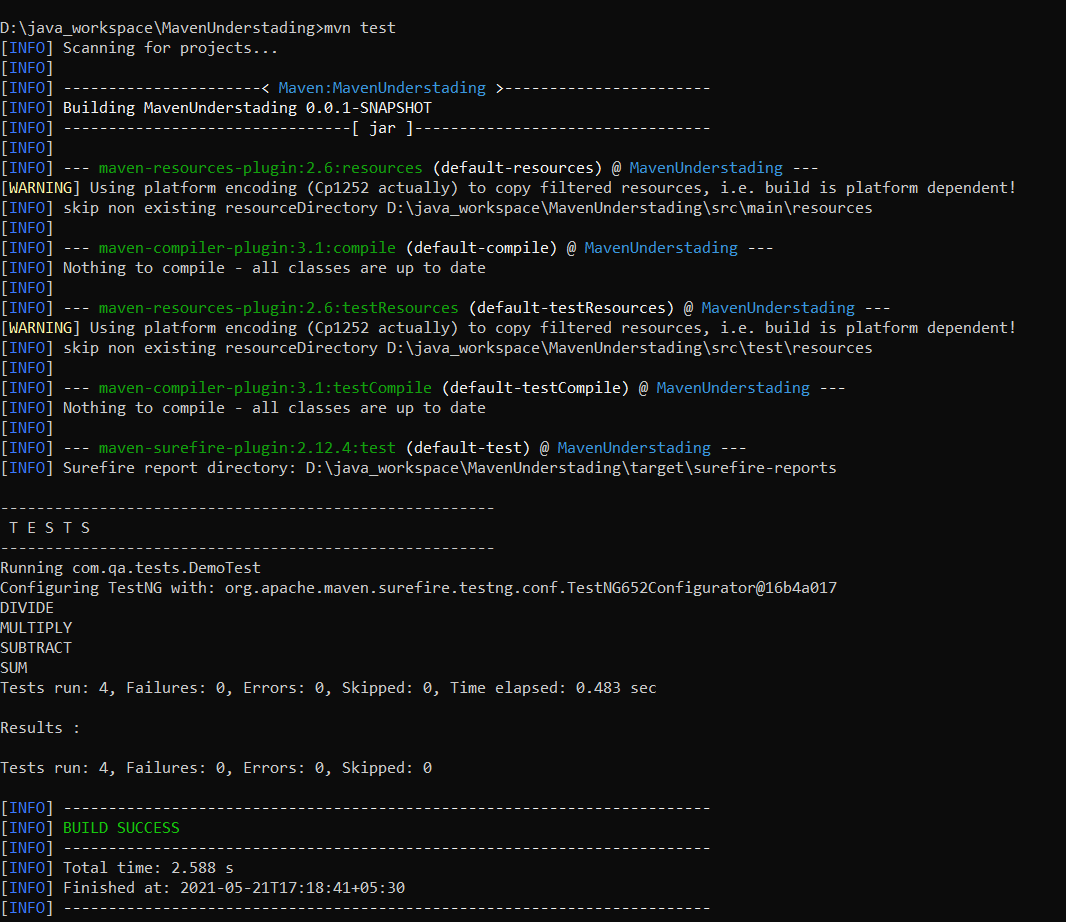
This will execute all the lifecycles i.e., clean, then compile, then run all the tests, generating the package jar/war file and install.

This will execute all the tests inside the first class of project(**src/test/java**) folder.



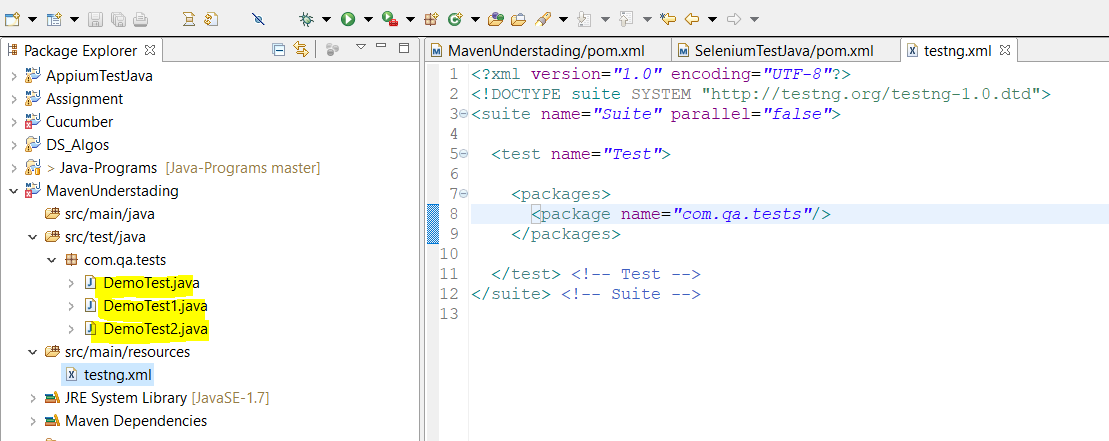
* **mvn test**

If we want to run just tests, then use this command.

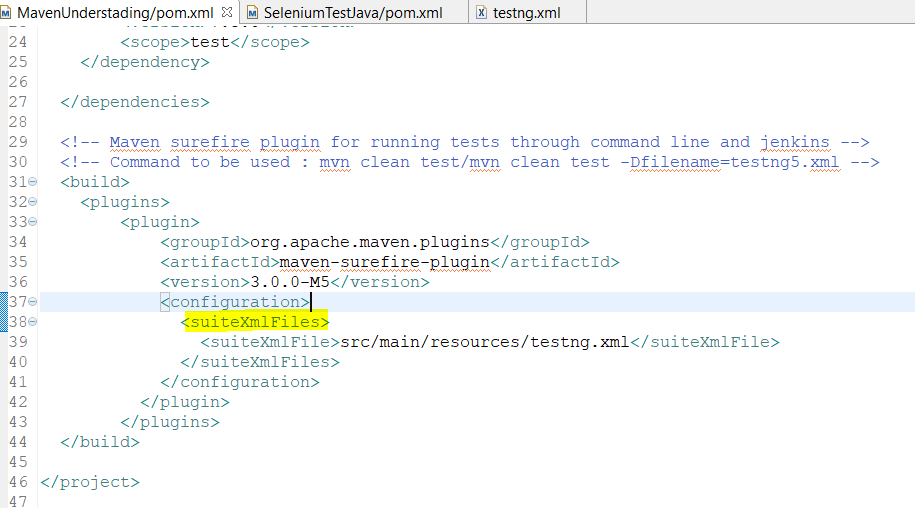


If we create two more class files with 4 tests each in it and run **mvn test**, then we observe that it only runs **one class file** with 4 tests and not all class files thus making it to 12 tests.

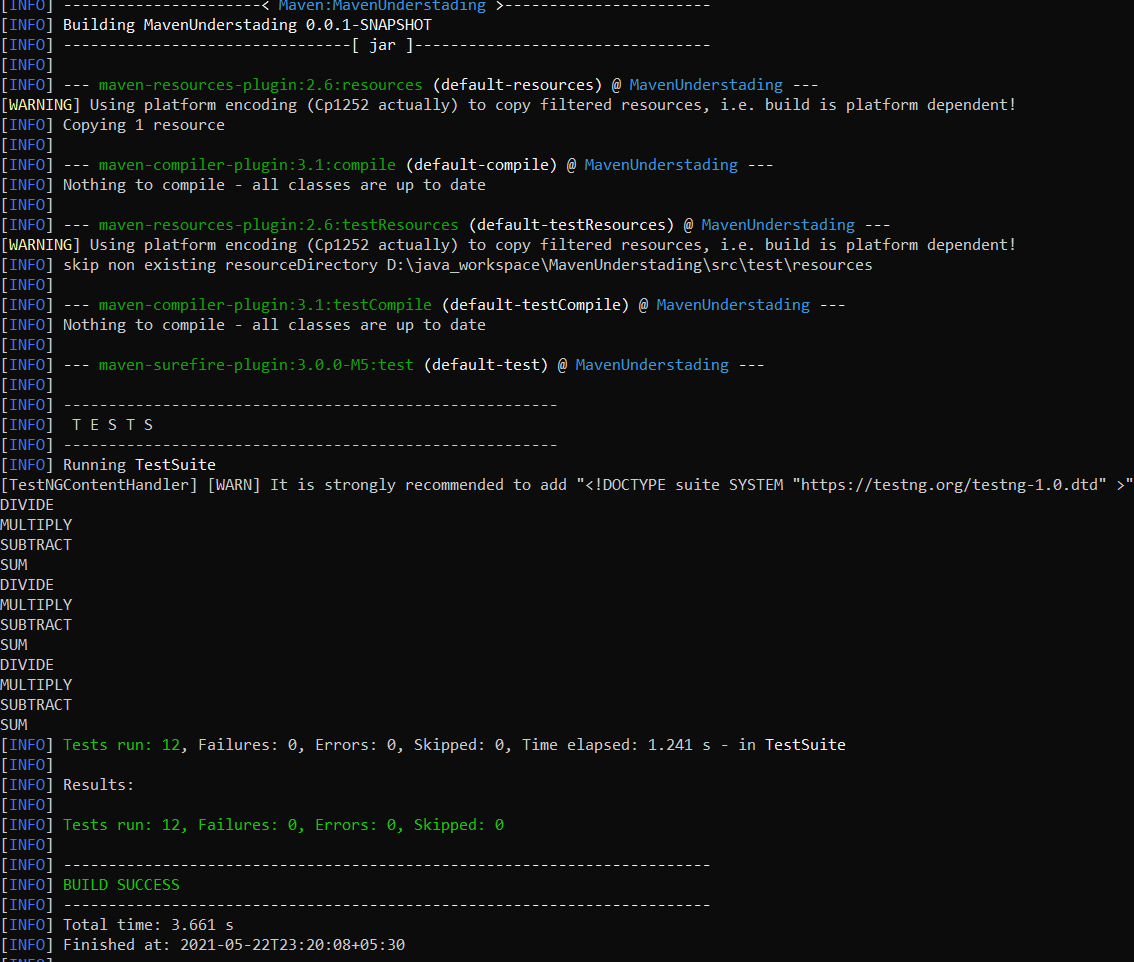
If we want to run all 12 tests, then we need to enhance our “**maven-surefire-plugin**” to include “**testng.xml**” file.

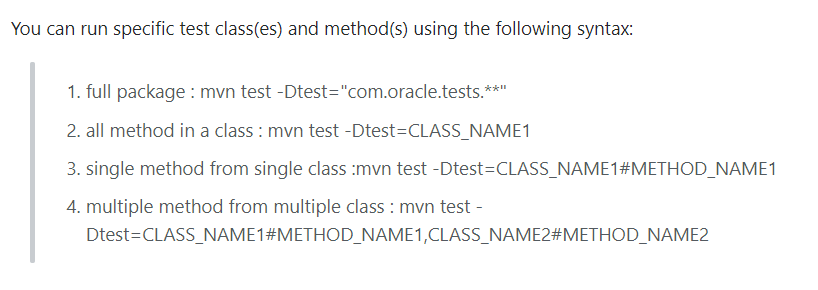


Add “**maven-surefire plugin**” in order to incorporate “**suiteXmlFiles**” tag.



Now run “**mvn test -Dtestng.dtd.http=true**”, all 12 tests will be executed.





* **mvn package -DskipTests**

If you want to generate just build without executing tests.

This is used by developers for generating the build when they do not want all the test cases written by them to be run.

