STQA Mini Project No. 1

Title

Mini-Project 1: Create a small application by selecting relevant system environment/ platform and programming languages. Narrate concise Test Plan consisting features to be tested and bug taxonomy. Prepare Test Cases inclusive of Test Procedures for identified Test Scenarios. Perform selective Black-box and White-box testing covering Unit and Integration test by using suitable Testing tools. Prepare Test Reports based on Test Pass/Fail Criteria and judge the acceptance of application developed.

Problem Definition:

Perform Desktop Application testing using Automation Tool like JUnit generate Test Report by Using tool like Apache Maven.

Prerequisite:

Knowledge of Core Java, Basic Concepts of Unit Testing, Test Cases Writing using Junit etc tool

Software Requirements:

JDK 1.8, Eclipse java photon-R version, TestNG

Hardware Requirement:

PIV, 2GB RAM, 500 GB HDD, Lenovo A13-4089Model.

Learning Objectives:

We are going to learn how to Prepare Test Cases inclusive of Test Procedures for identified Test Scenarios. Perform selective Black-box and White-box testing covering Unit and Integration test by using suitable Testing tools. also Prepare Test Reports based on Test Pass/Fail Criteria

Outcomes:

You are able to understand Unit and Integration testing with Tool with Test Report.

Theory Concepts:

What is Unit Testing?

Unit Testing of software applications is done during the development (coding) of an application.

The objective of Unit Testing is to isolate a section of code and verify its correctness. In procedural programming a unit may be an individual function or procedure

The goal of Unit Testing is to isolate each part of the program and show that the individual parts are correct. Unit Testing is usually performed by the developer.

Unit Testing Tools

There are several automated tools available to assist with unit testing. We will provide a few examples below:

1. Jtest: Parasoft Jtest is an IDE plugin that leverages open-source frameworks (Junit, Mockito, PowerMock, and Spring) with guided and easy one-click actions for creating, scaling, and maintaining unit tests. By automating these time-consuming aspects of unit testing, it frees the

developer to focus on business logic and create more meaningful test suites.

2. Junit: Junit is a free to use testing tool used for Java programming language. It provides assertions to identify test method. This tool test data first and then inserted in the piece of code.
3. NUnit: NUnit is widely used unit-testing framework use for all .net languages. It is open source tool which allows writing scripts manually. It supports data-driven tests which can run in parallel.
4. JMockit: JMockit is open source Unit testing tool. It is code coverage tool with line and path metrics. It allows mocking API with recording and verification syntax. This tool offers Line coverage, Path Coverage, and Data Coverage.
5. EMMA: EMMA is an open-source toolkit for analyzing and reporting code written in Java language. Emma support coverage types like method, line, basic block. It is Java-based so it is without external library dependencies and can access to the source code.
6. PHPUnit: PHPUnit is a unit testing tool for PHP programmer. It takes small portions of code which is called units and test each of them separately. The tool also allows developers to use pre-define assertion methods to assert that system behave in a certain manner.

Those are just a few of the available unit testing tools. There are lots more, especially for C languages and Java, but you are sure to find a unit testing tool for your programming needs regardless of the language you use.

Extreme Programming & Unit Testing

Unit testing in Extreme Programming involves the extensive use of testing frameworks. A unit test framework is used in order to create automated unit tests. Unit testing frameworks are not unique to extreme programming, but they are essential to it. Below we look at some of what extreme programming brings to the world of unit testing:

- Tests are written before the code
- Rely heavily on testing frameworks
- All classes in the applications are tested
- Quick and easy integration is made possible

Bug taxonomy

Bug taxonomies help in providing fast and effective feedback so that they can easily identify possible reasons for failure of the software. Using bug taxonomy, a large number of potential bugs can be grouped into few categories.

Whenever a new bug is reported, using bug taxonomy, a tester can easily analyse and put that bug into any of these categories.

At the end of testing, Testers can understand the type of categories of bugs that frequently occurred and thereby in successive rounds of testing he can focus on writing more test cases that would help to detect such bugs. In addition, test leaders can guide their testers to focus on such frequently occurring bugs.

The summary of the Bug Taxonomy is given below,

- Requirements, Features, and Functionality Bugs
- Structural Bugs
- Data Bugs
- Coding Bugs
- Interface, Integration, and System Bugs
- Test and Test Design Bugs
- Testing and Design Style

What is Integration Testing?

In integration Testing, individual software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. integration Testing focuses on checking data communication amongst these modules. Hence it is also termed as 'I & T' (Integration and Testing), 'String Testing' and sometimes 'Thread Testing

Integration Test Case:

Integration Test Case differs from other test cases in the sense it **focuses mainly on the interfaces & flow of data/information between the modules**. Here priority is to be given for the **integrating links** rather than the unit functions which are already tested.

Sample Integration Test Cases for the following scenario: Application has 3 modules say 'Login Page', 'Mail box' and 'Delete mails' and each of them are integrated logically.

Here do not concentrate much on the Login Page testing as it's already been done in Unit Testing. But check how it's linked to the Mail Box Page.

Similarly Mail Box: Check its integration to the Delete Mails Module.

Test Case ID	Test Case Objective	Test Case Description	Expected Result
1	Check the interface link between the Login and Mailbox module	Enter login credentials and click on the Login button	To be directed to the Mail Box
2	Check the interface link between the Mailbox and Delete Mails Module	From Mail box select the an email and click delete button	Selected email should appear in the Deleted/Trash folder

Desktop Application Testing by Using Junit Tool

What is Junit?

JUnit is a framework for implementing testing in Java.

It provides a simple way to explicitly test specific areas of a Java program, it is extensible and can be employed to test a hierarchy of program code either singularly or as multiple units.

Why use a testing framework? Using a testing framework is beneficial because it forces you to explicitly declare the expected results of specific program execution routes. When debugging it is possible to write a

test which expresses the result you are trying to achieve and then debug until the test comes out positive. By having a set of tests that test all the core components of the project it is possible to modify specific areas of the project and immediately see the effect the modifications have on the other areas by the results of the test, hence, side-effects can be quickly realized.

JUnit promotes the idea of first testing then coding, in that it is possible to setup test data for a unit which defines what the expected output is and then code until the tests pass. It is believed by some that this practice of "test a little, code a little, test a little, code a little..." increases programmer productivity and stability of program code whilst reducing programmer stress and the time spent debugging.

JUnit is a simple open source Java testing framework used to write and run repeatable automated tests.

It is an instance of the xUnit architecture for unit testing framework. Eclipse supports creating test cases and running test suites, so it is easy to use for your Java applications.

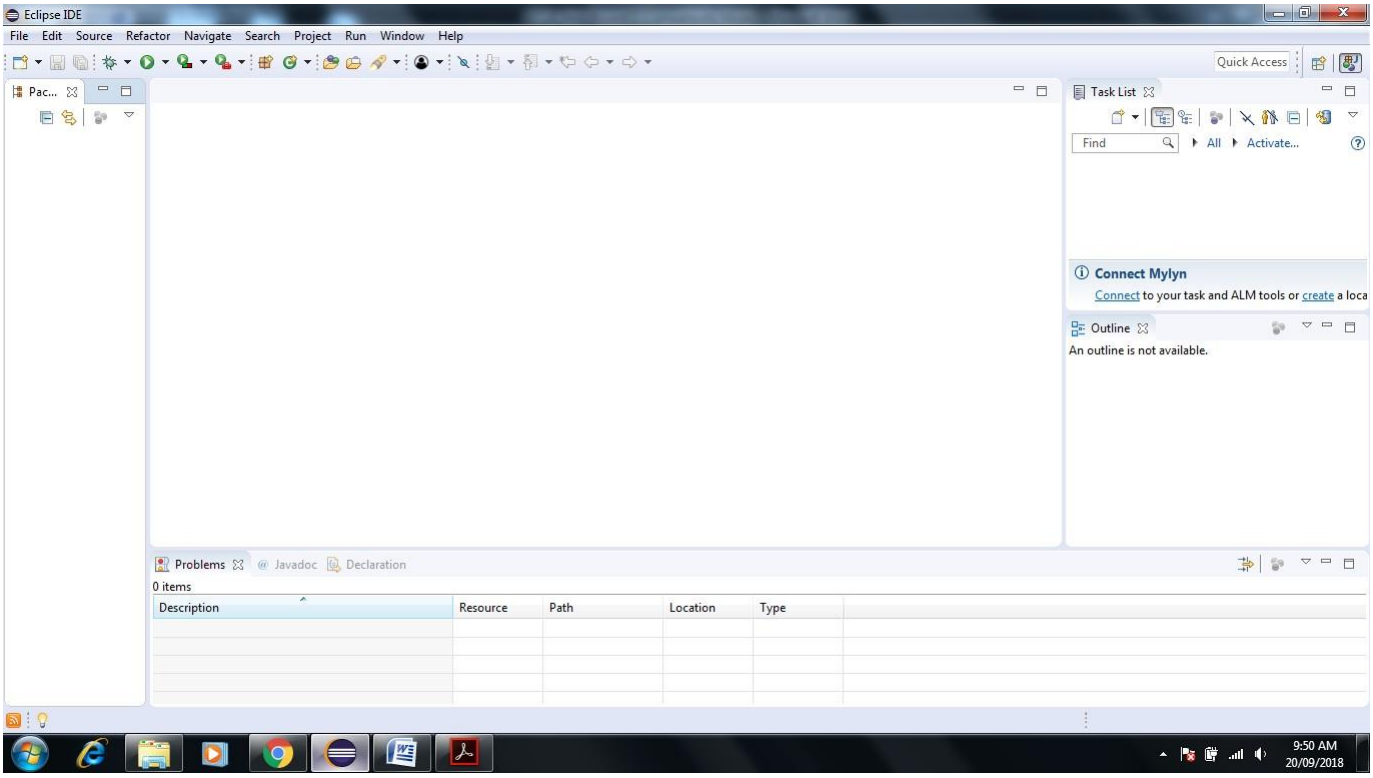
JUnit features include:

- Assertions for testing expected results
- Test fixtures for sharing common test data
- Test suites for easily organizing and running tests
- Graphical and textual test runners

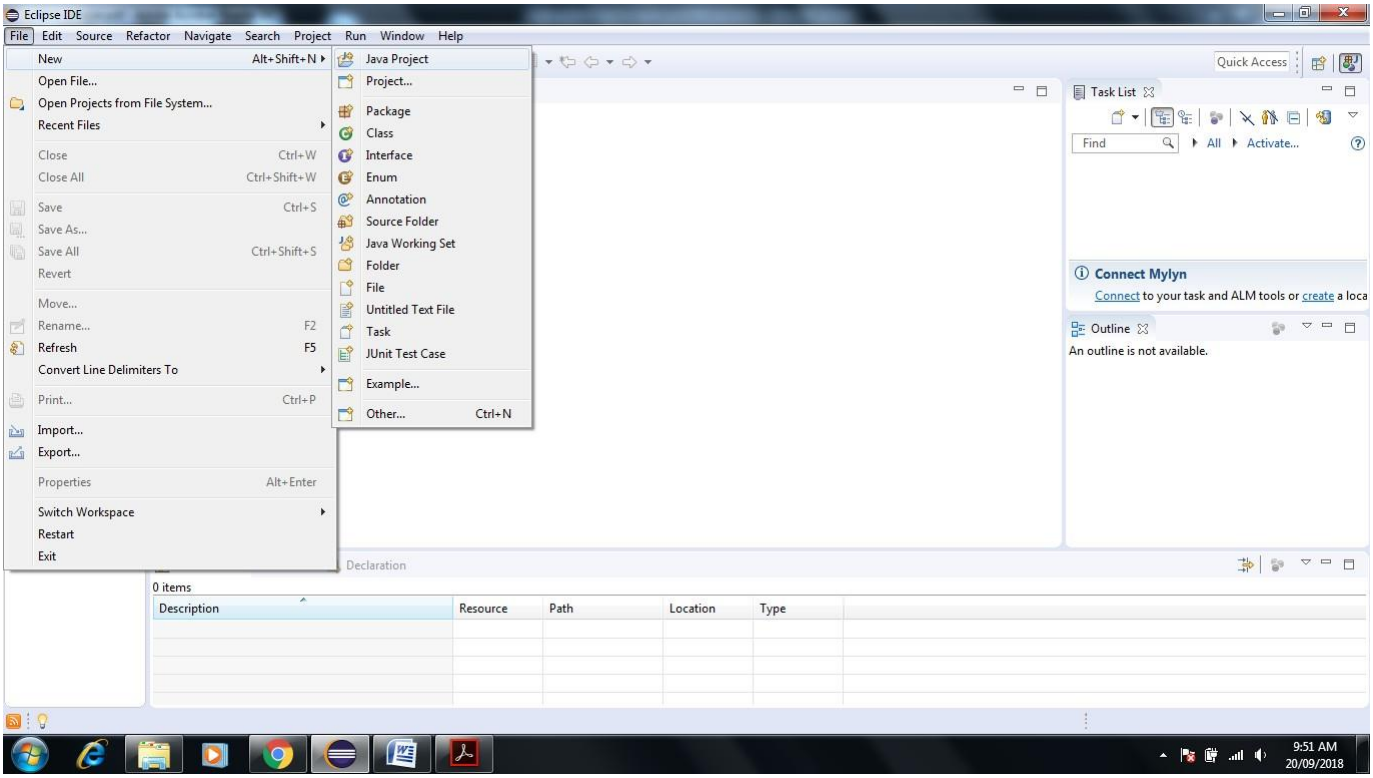
How to Create Simple Junit Test in Eclipse IDE

1. Download JDK 1.8 and Eclipse latest version here we are using eclipse-java-photon-R-win32.

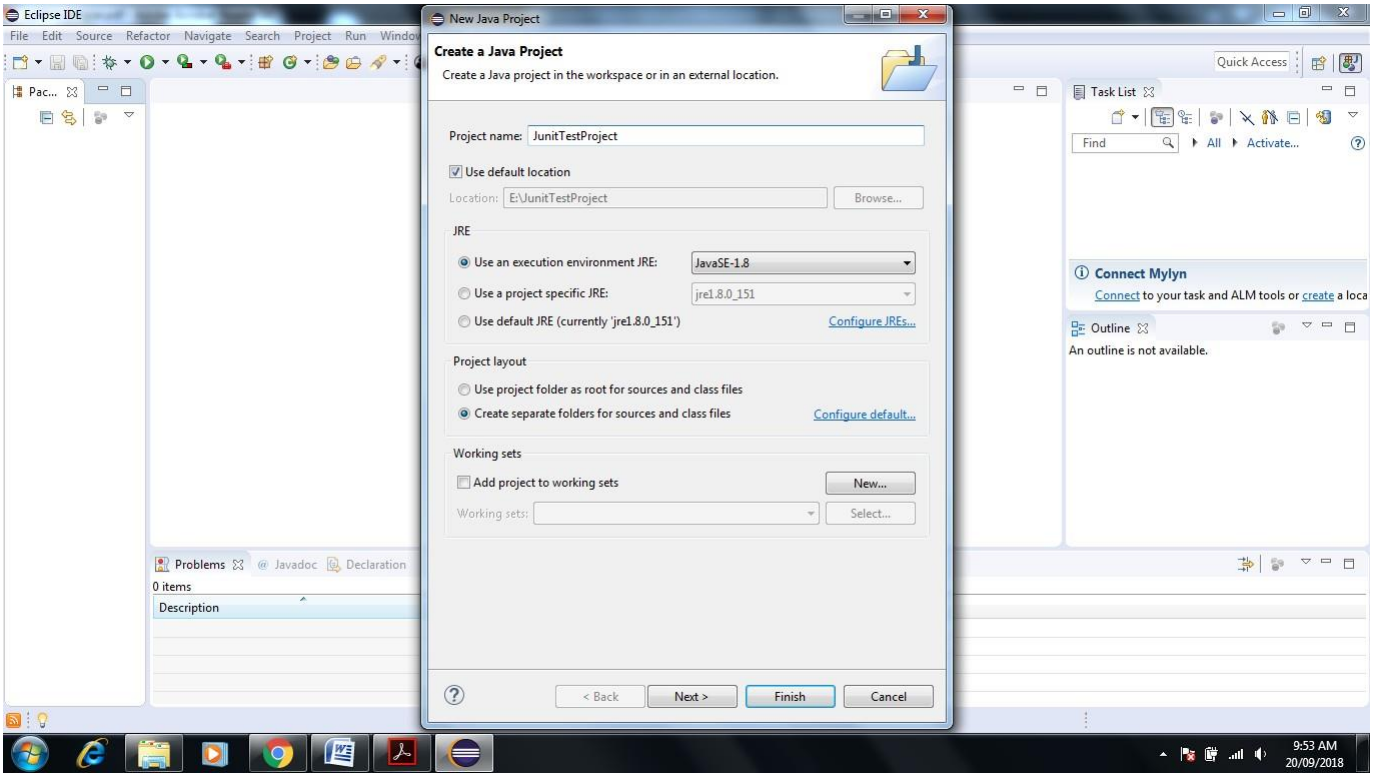
2. Open Eclipse IDE



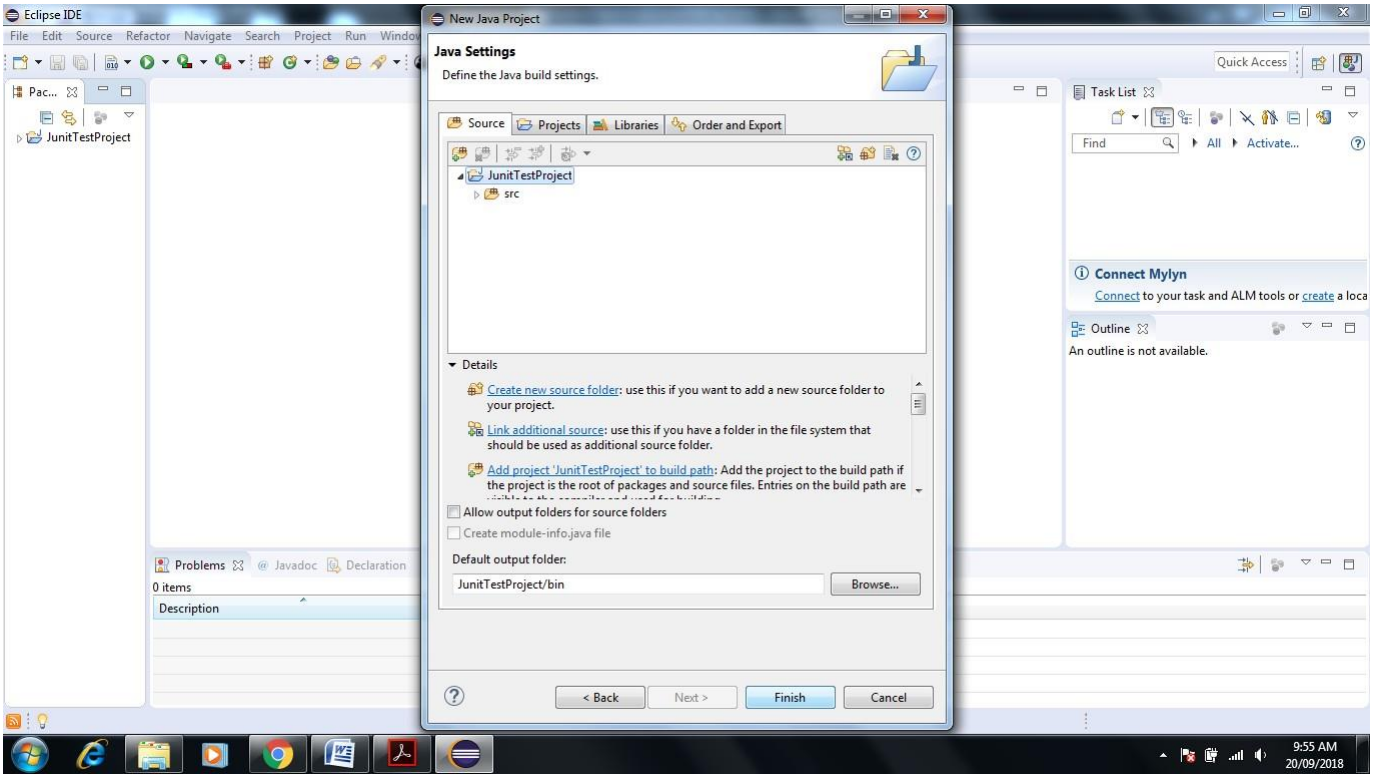
3. Go to File and Select New -> Create New Java Project



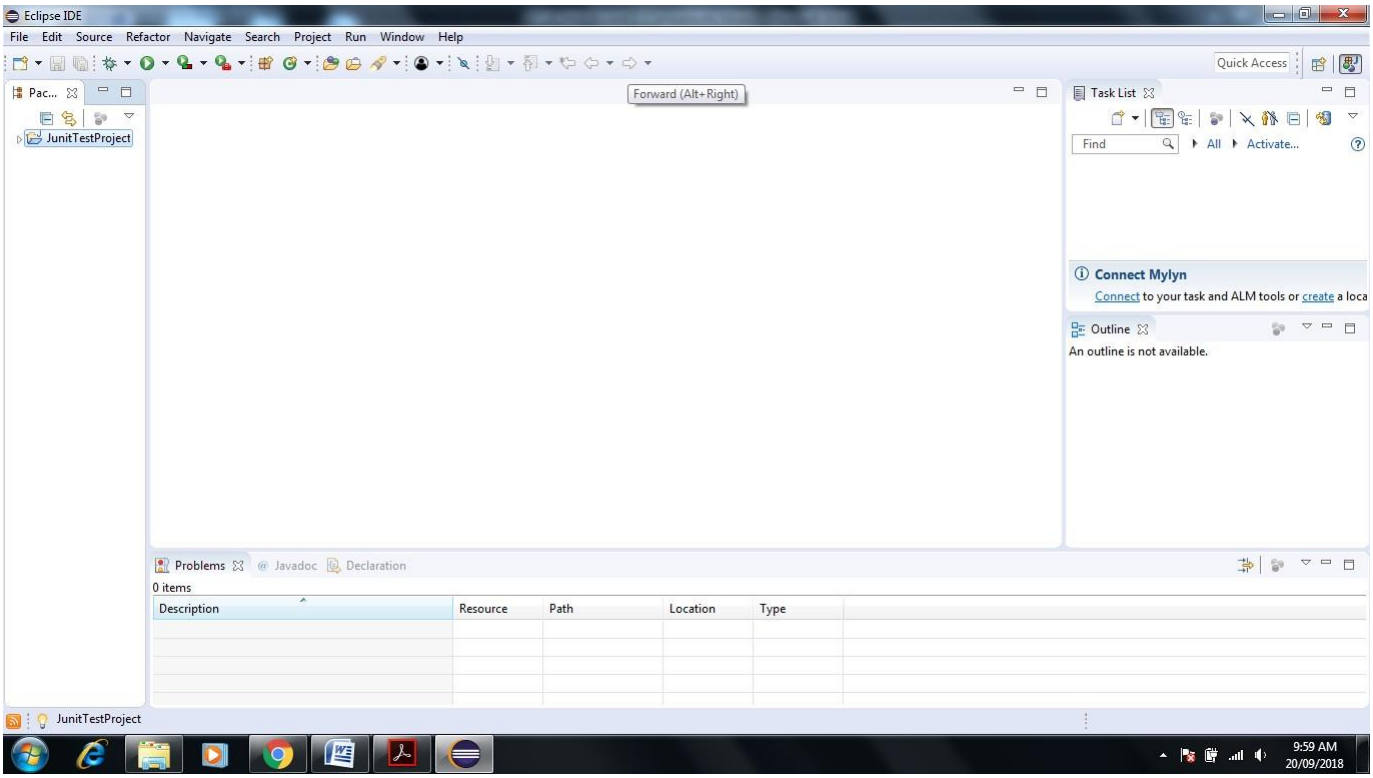
4. Give JunitTestProject name to the project and check use project folder as root for source and class files



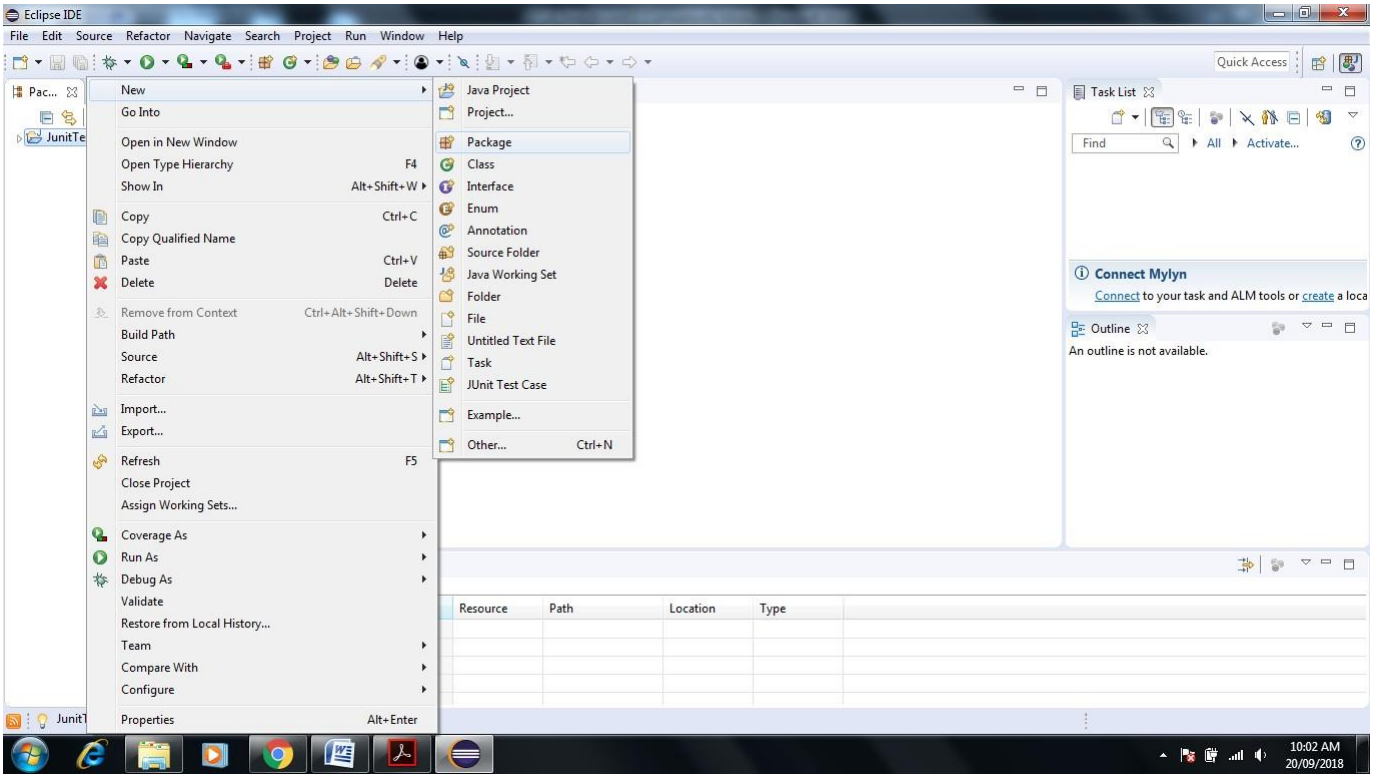
5. Click on Next-> Next Screen will Appear-> Click Finish



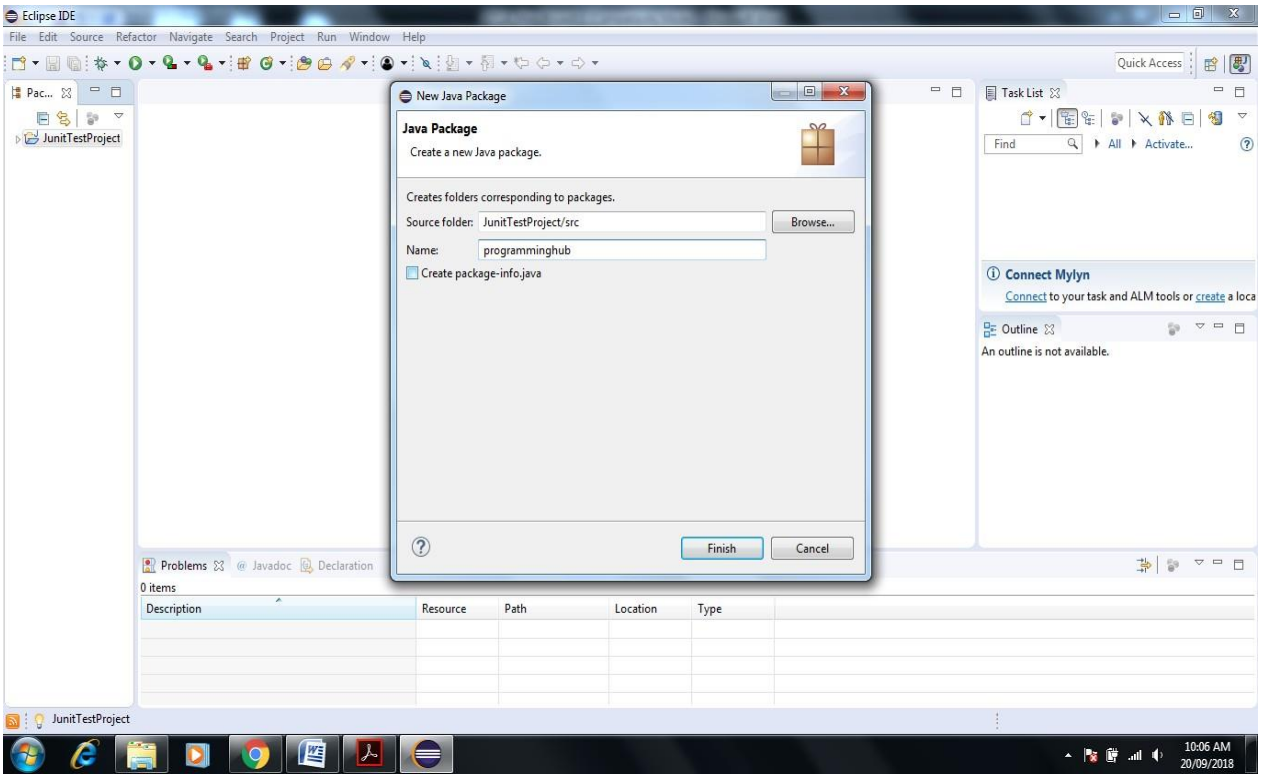
6. Next Screen Shown JunitTestProject Folder in Project Explorer



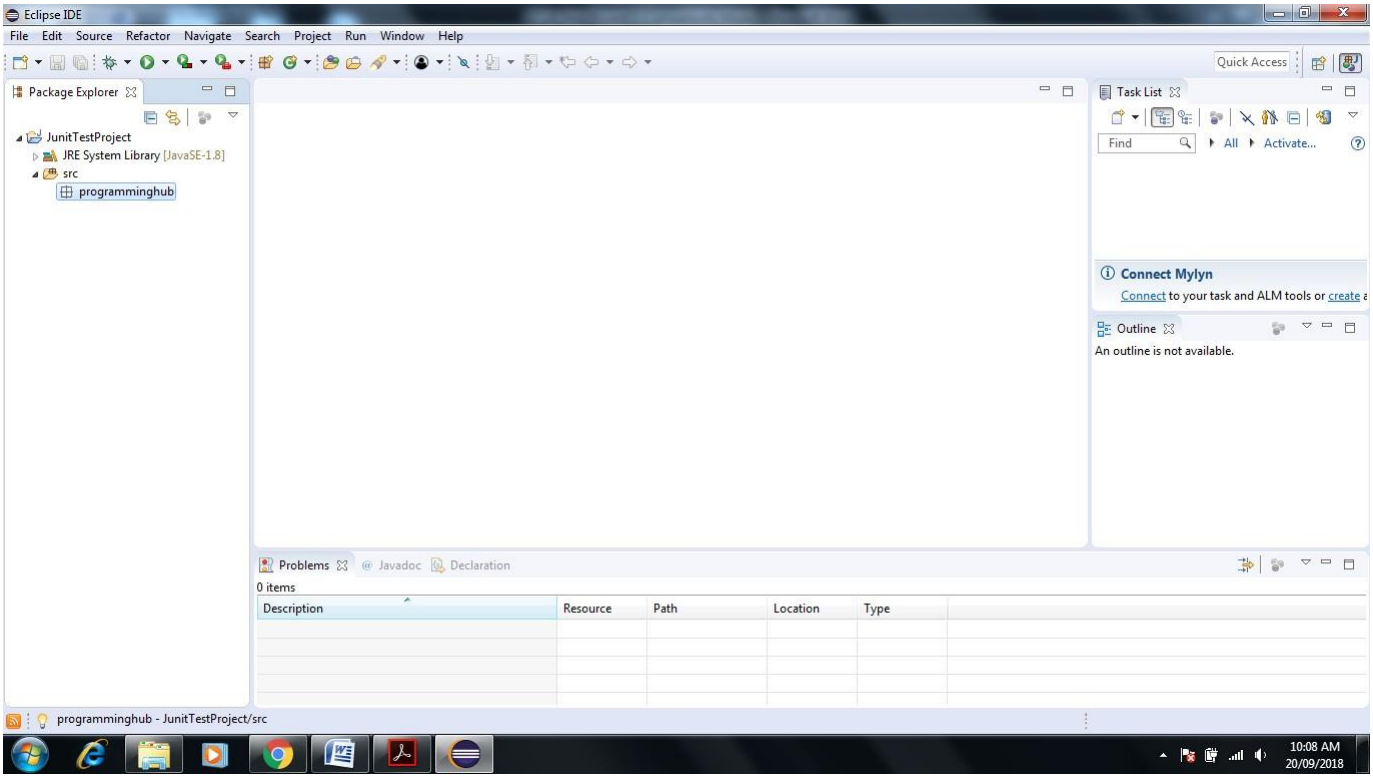
7. Right Click on Folder name JunitTestProject->New->Package



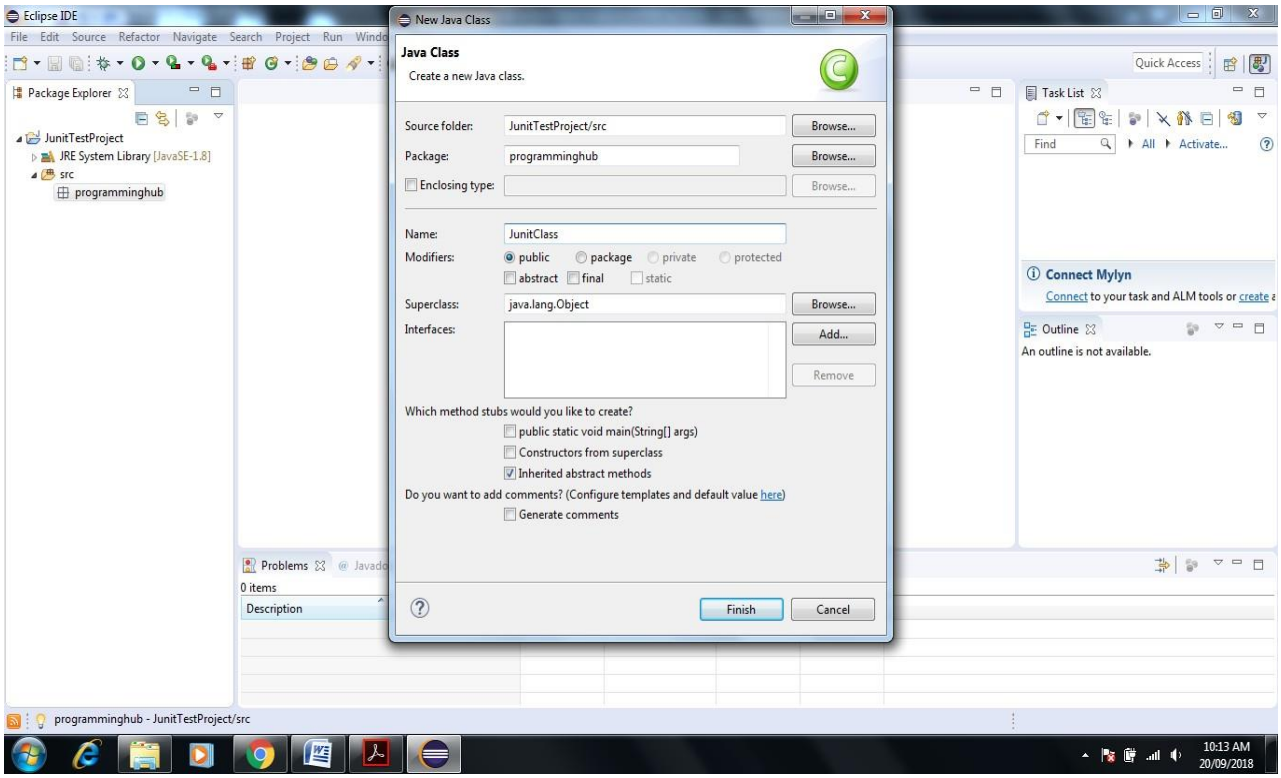
8. Name package as programming hub-> Click on Finish



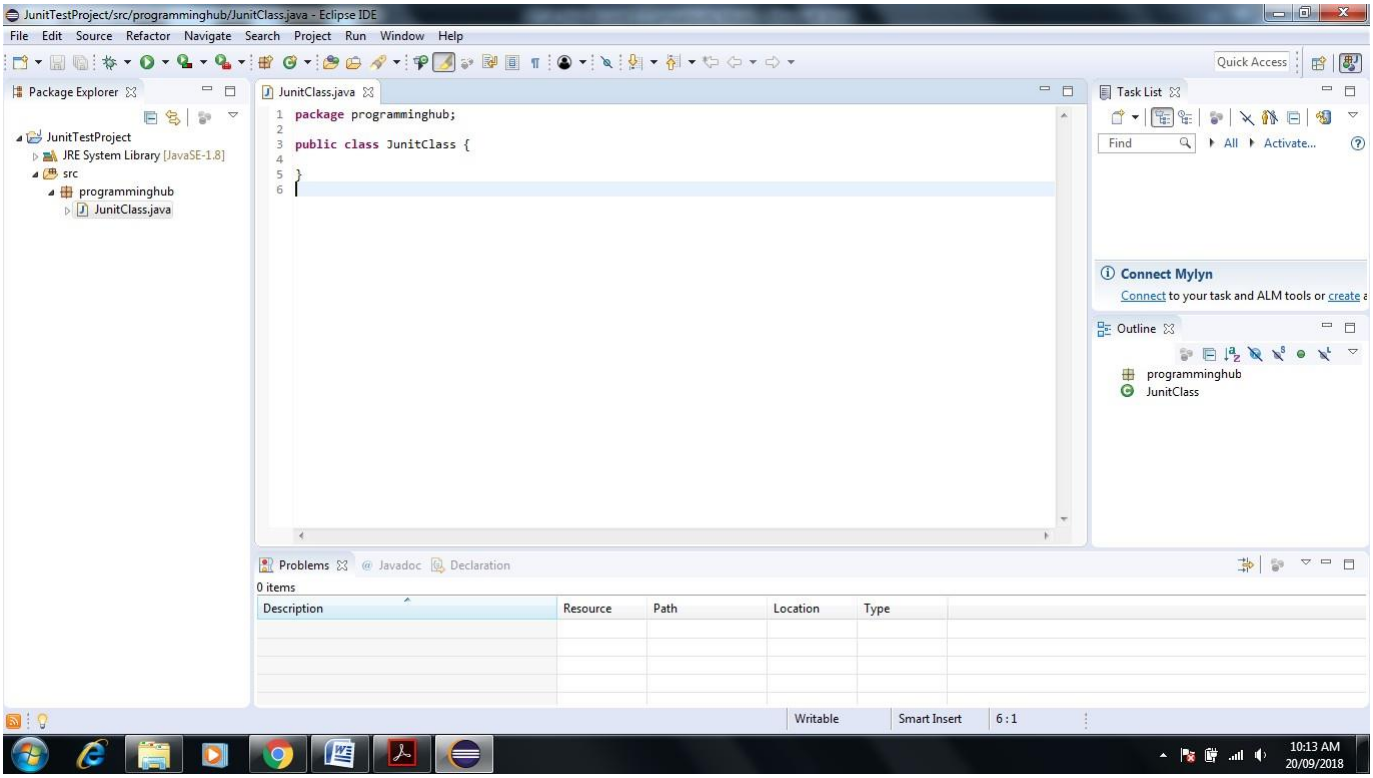
9. See the Programming hub package see in project Explorer Screen of Eclipse



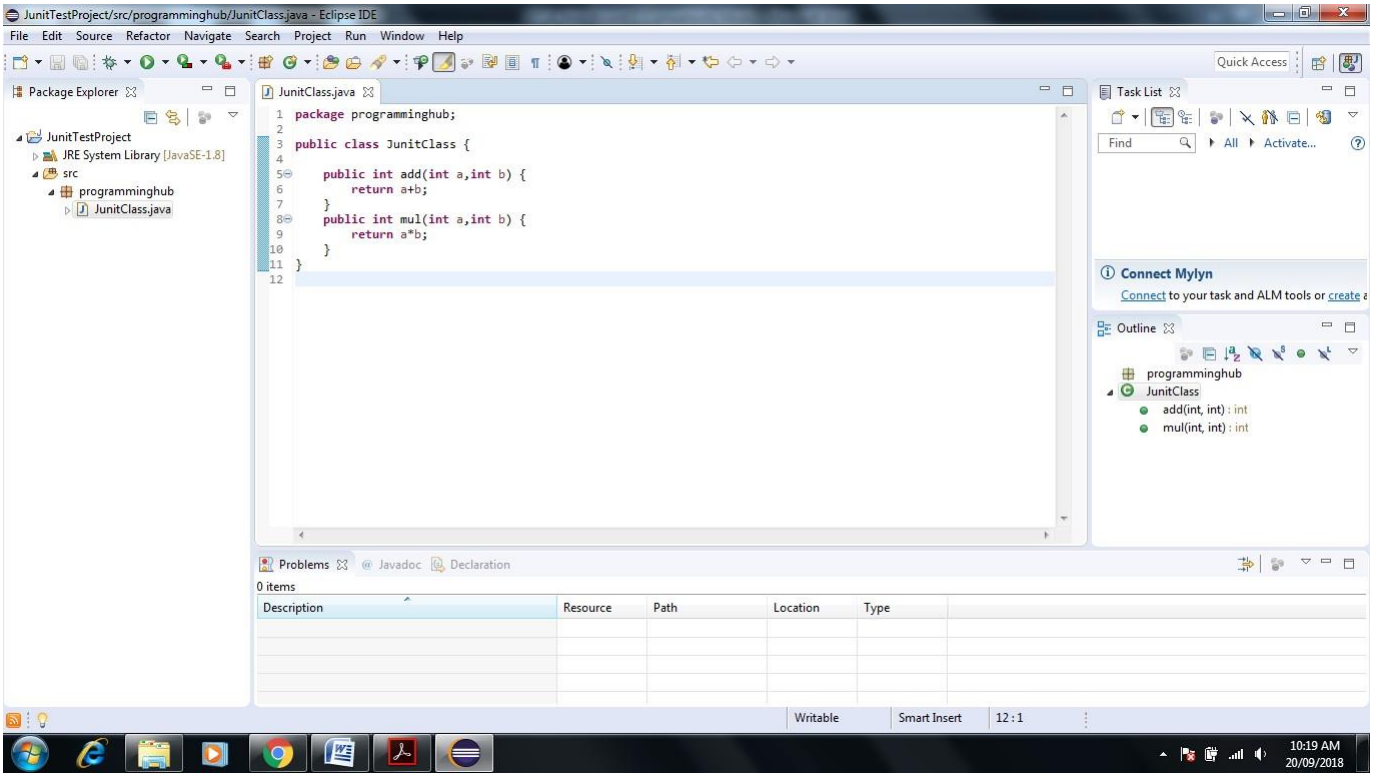
10. Right Click on Programminghub Package->New->Class give the name JunitClass->Click Finish.



11. Next screen will appear

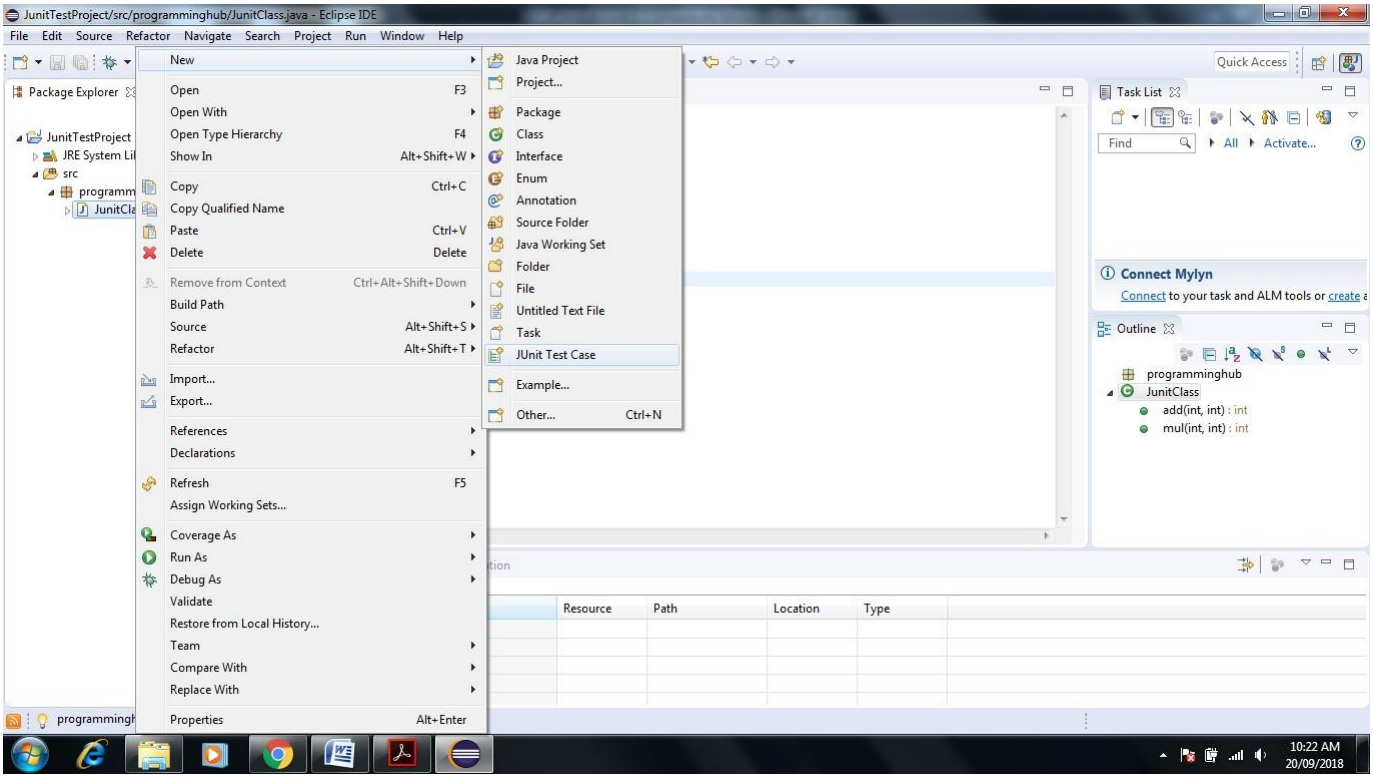


12 Write a small program with only two functions Add and Multiplication

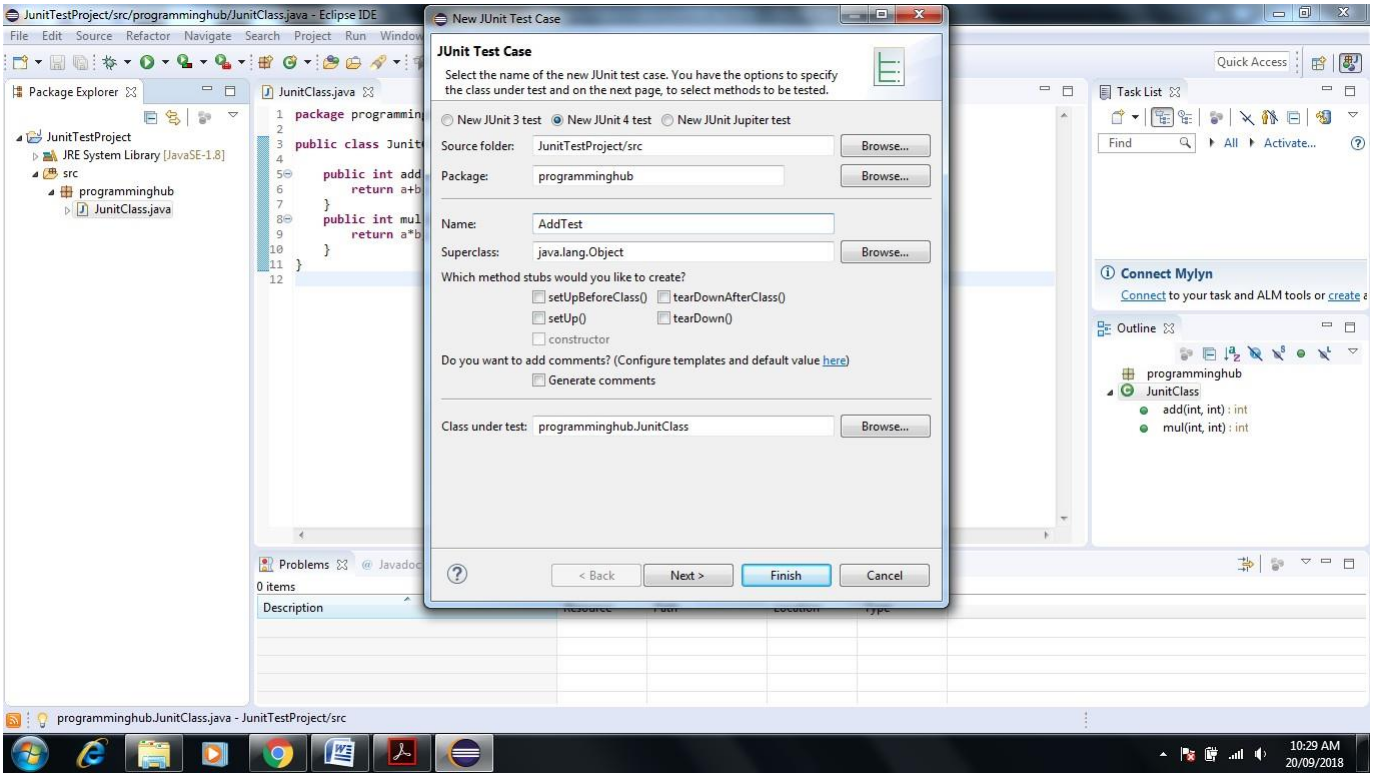


13. Write Test Cases for Java Program

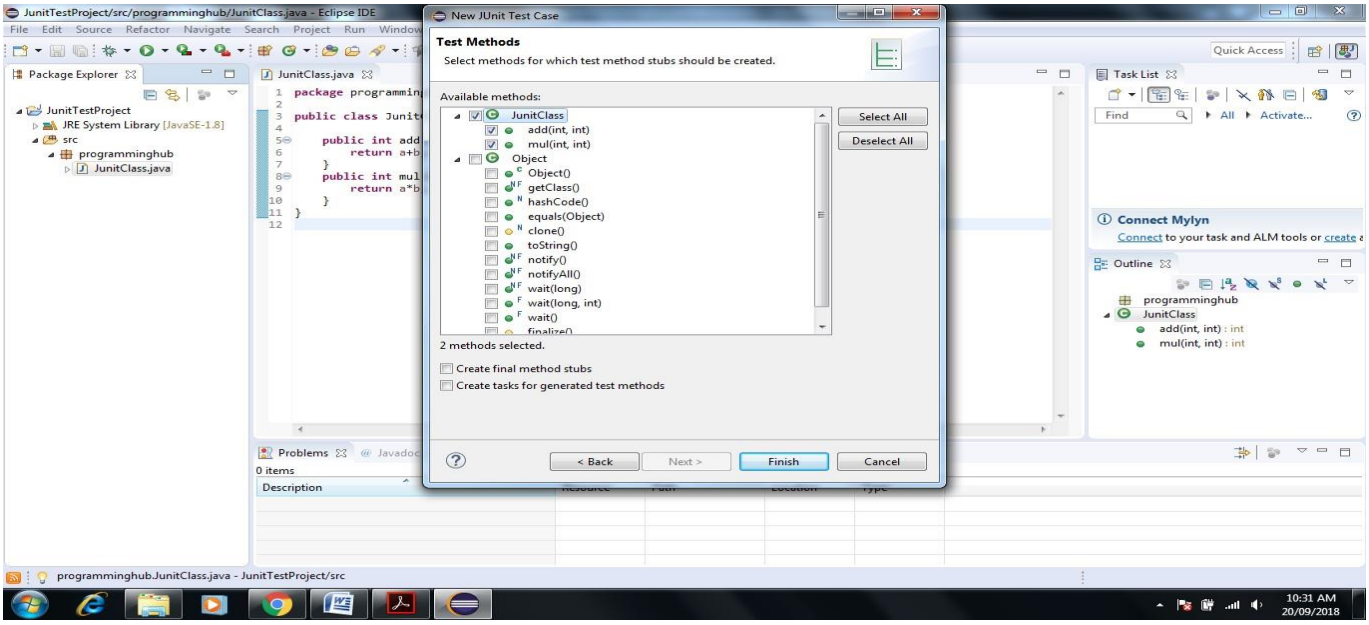
Right click on Junitclass-> New-> Click on Junit Test Cases



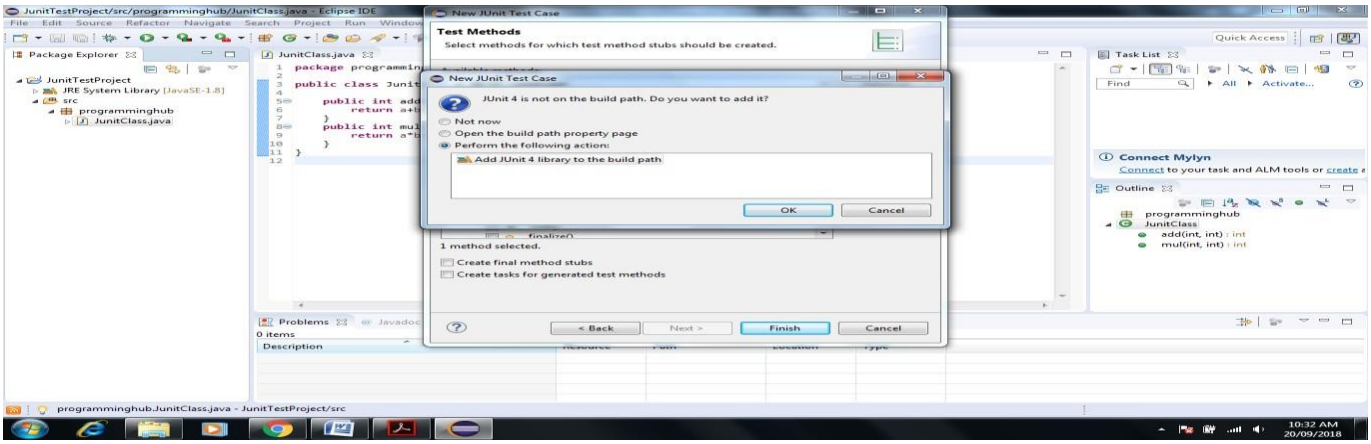
14. Name test suite as AddTest and choose New Junit4 test



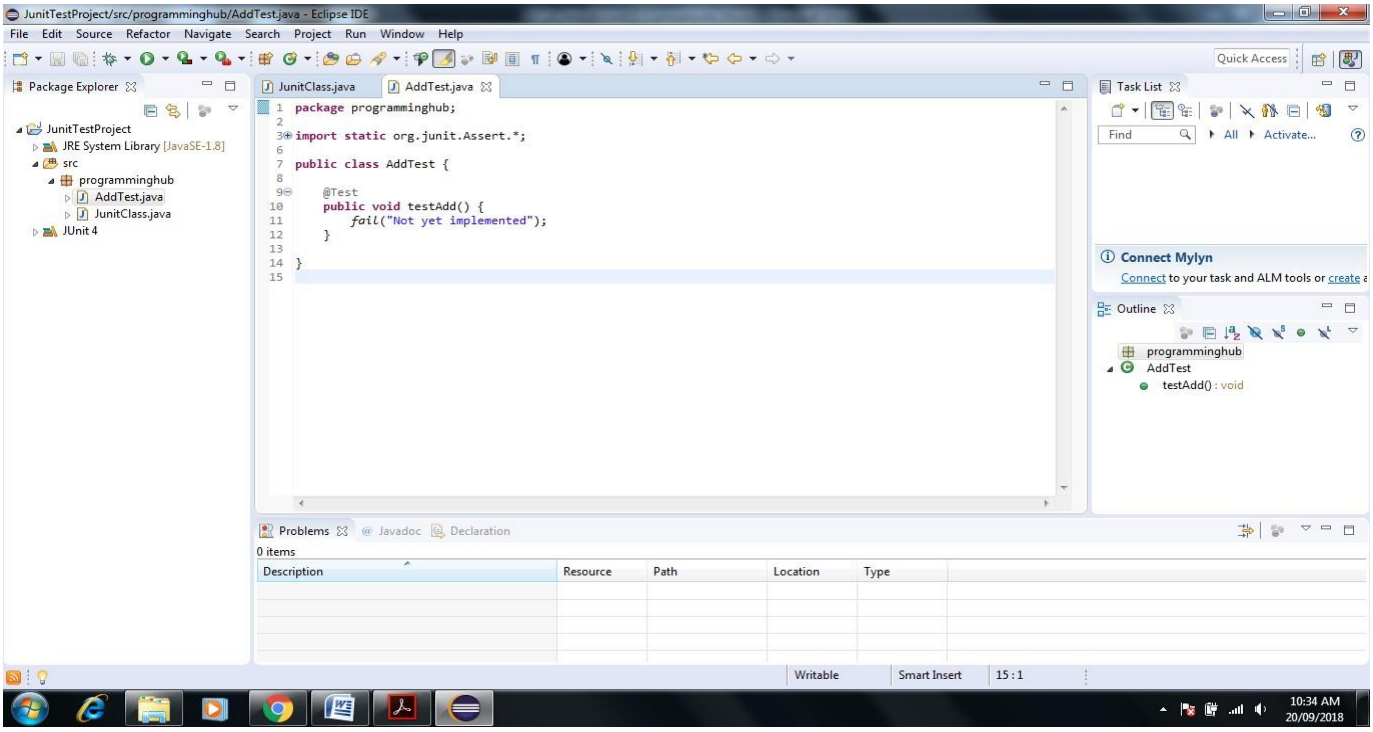
15. Click on add Checkbox



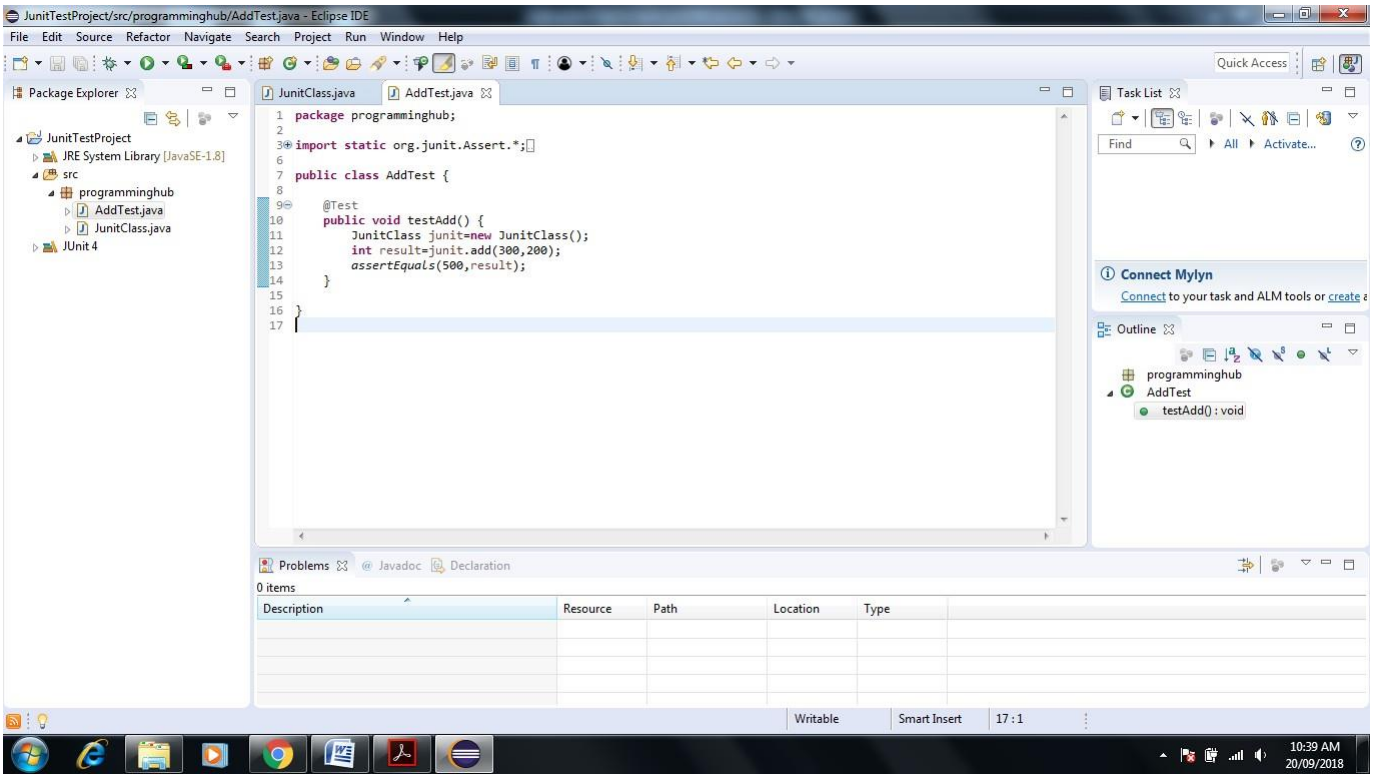
16. Click on Next-> Ok



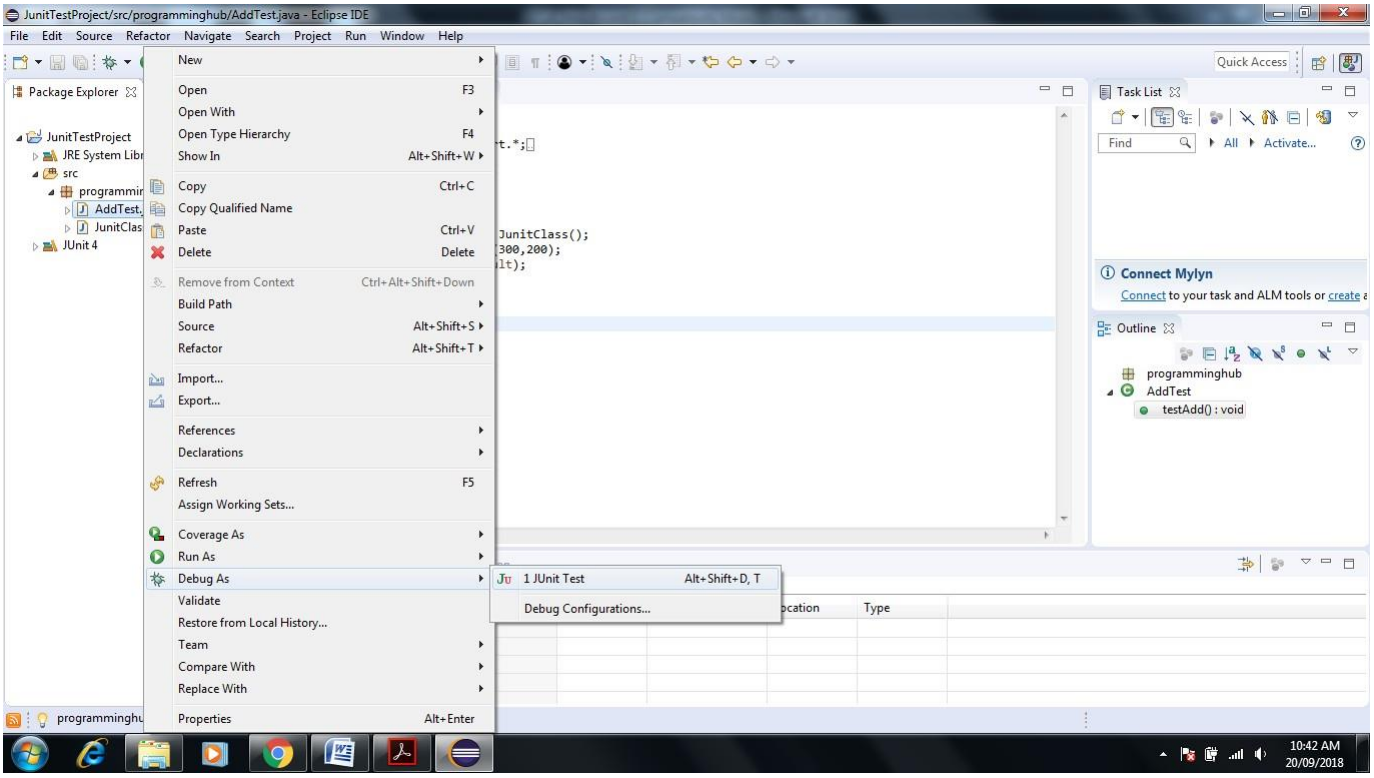
17. Next screen will appear



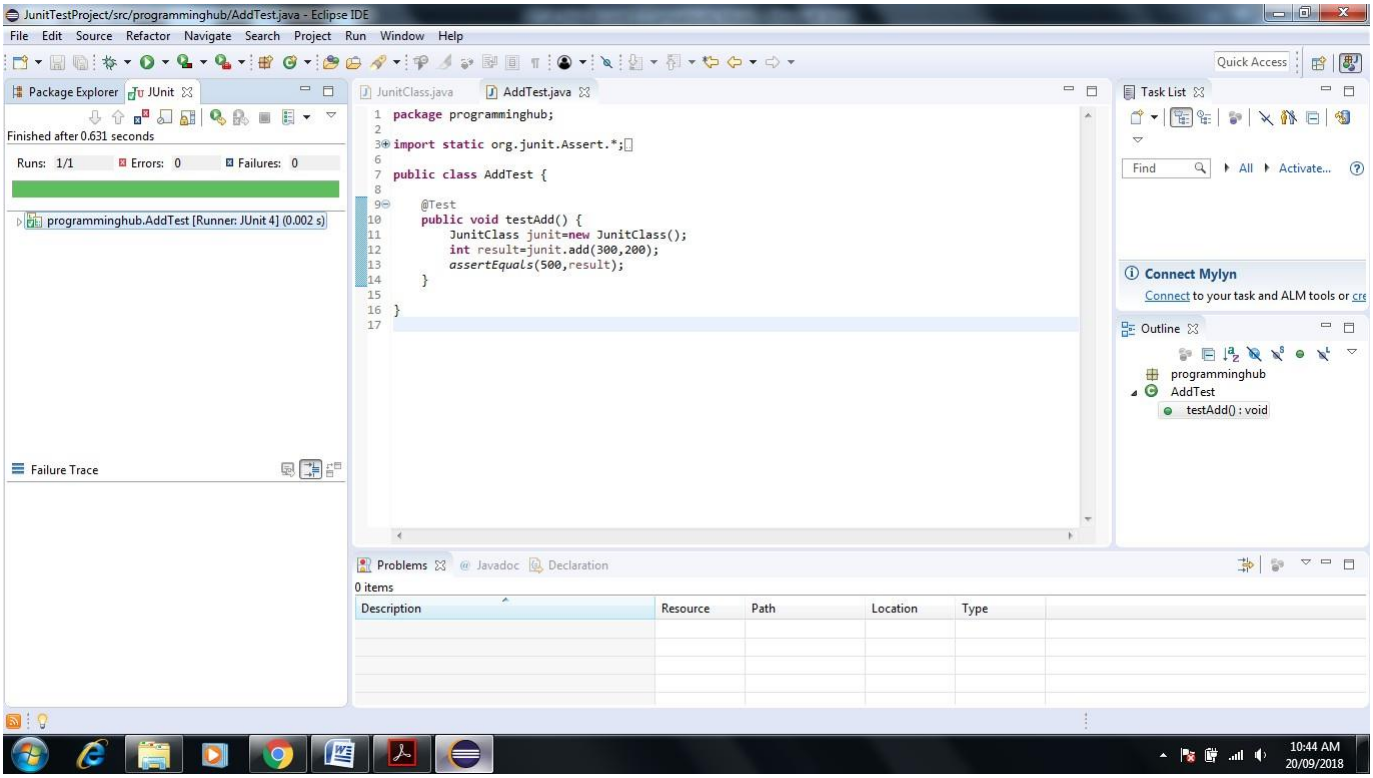
18. Write a code for Test case addition of two number inside AddTest



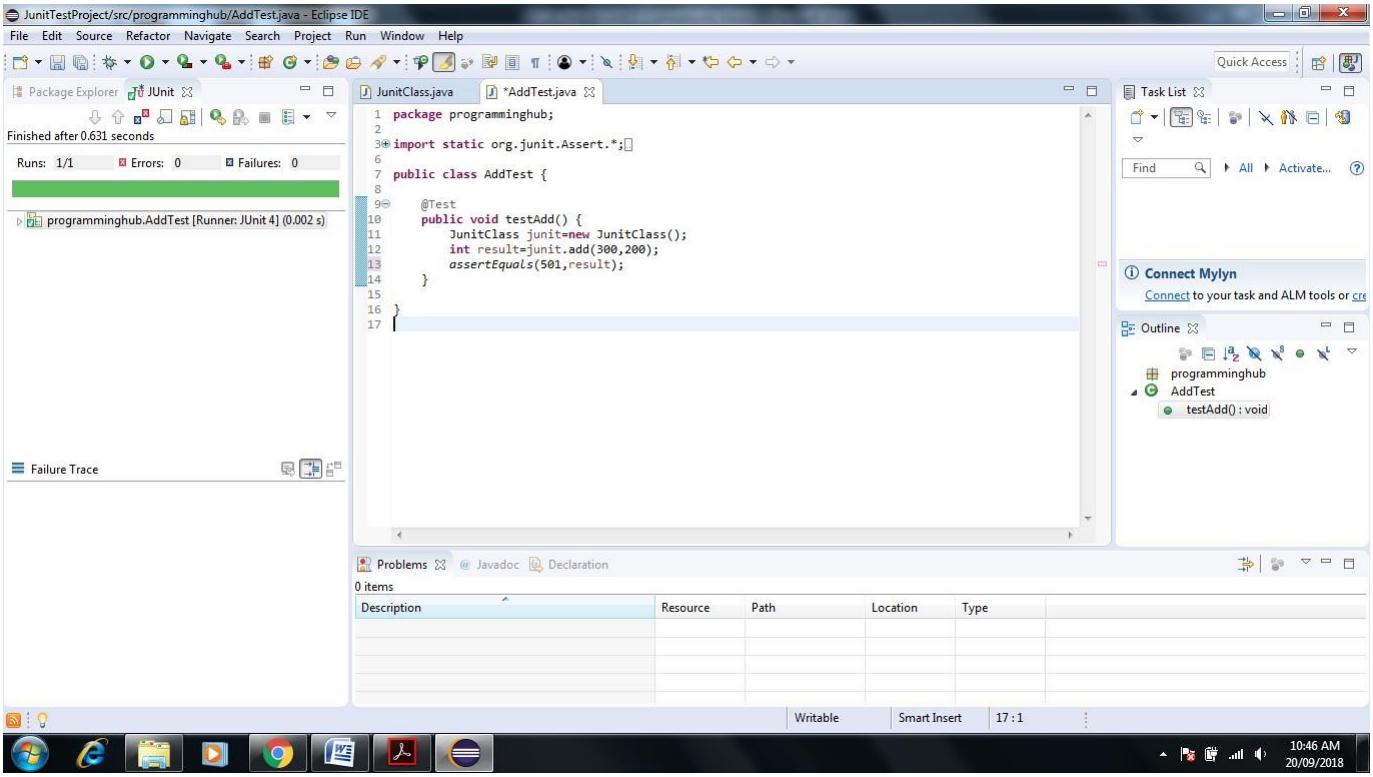
19. Let us run AddTest test case. Right click AddTest-> Debug As->JUnit Test



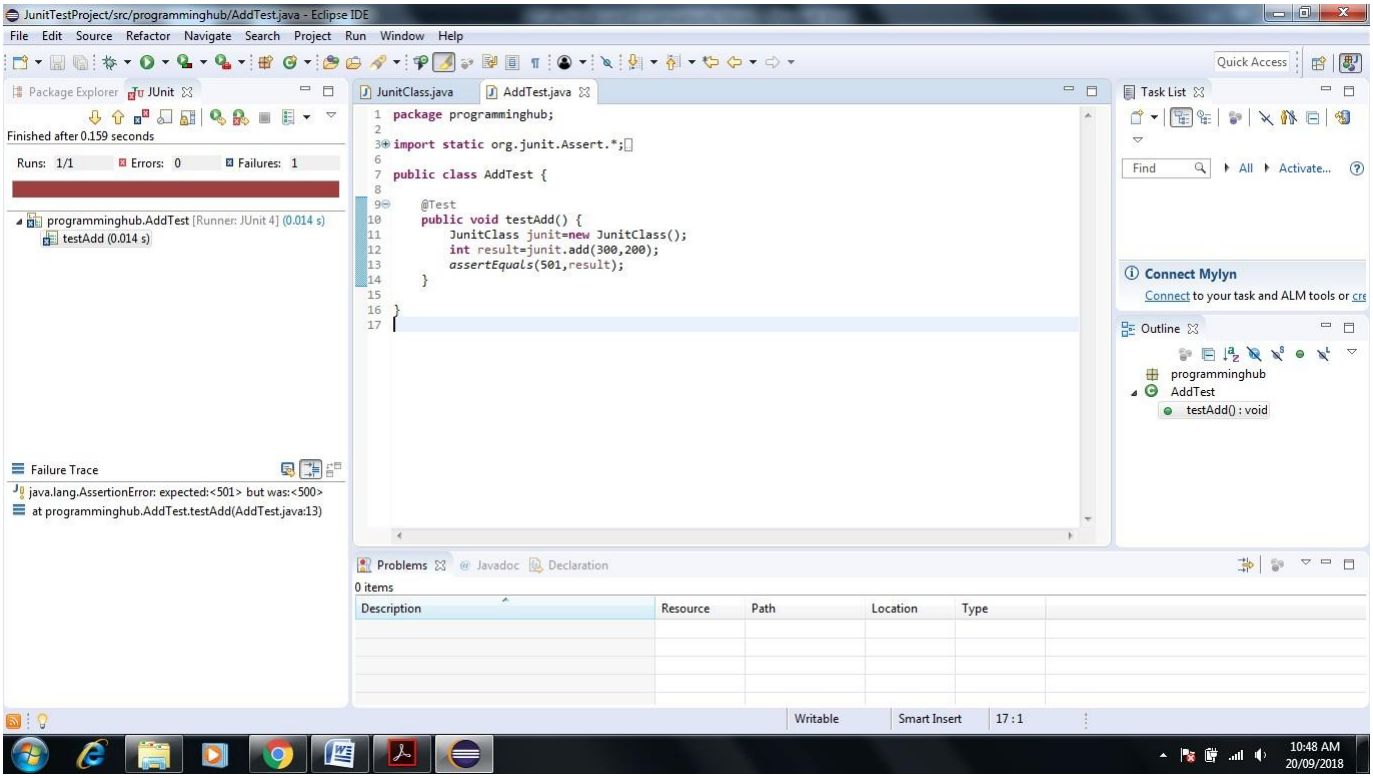
20. Result of test case is as follows. It shows 0 error and 0 failure and green color test bar which means that test case has run successfully(Green Color Bar Indicate)



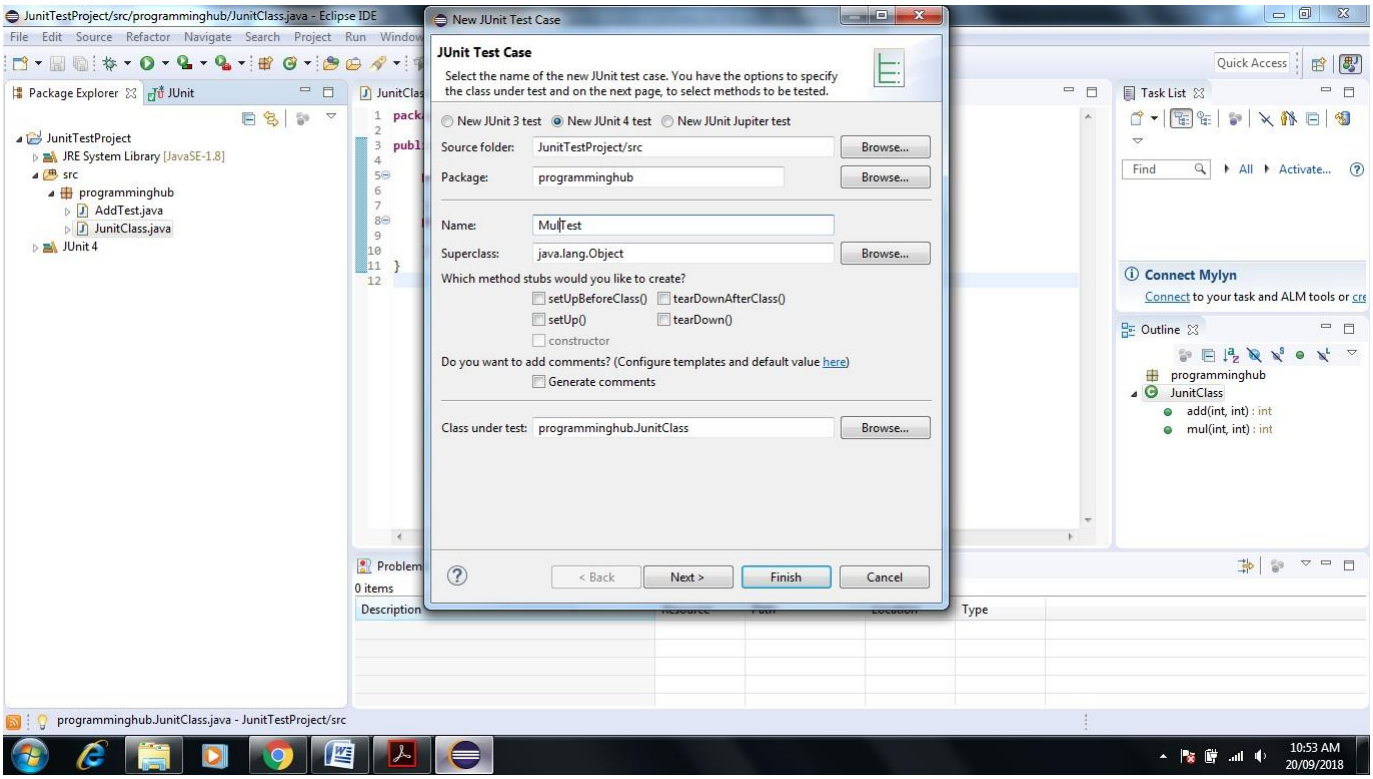
21. Let us purposely give wrong input in assertEquals method or unexpected result here we write 501 instead of 500 indicate wrong addition result



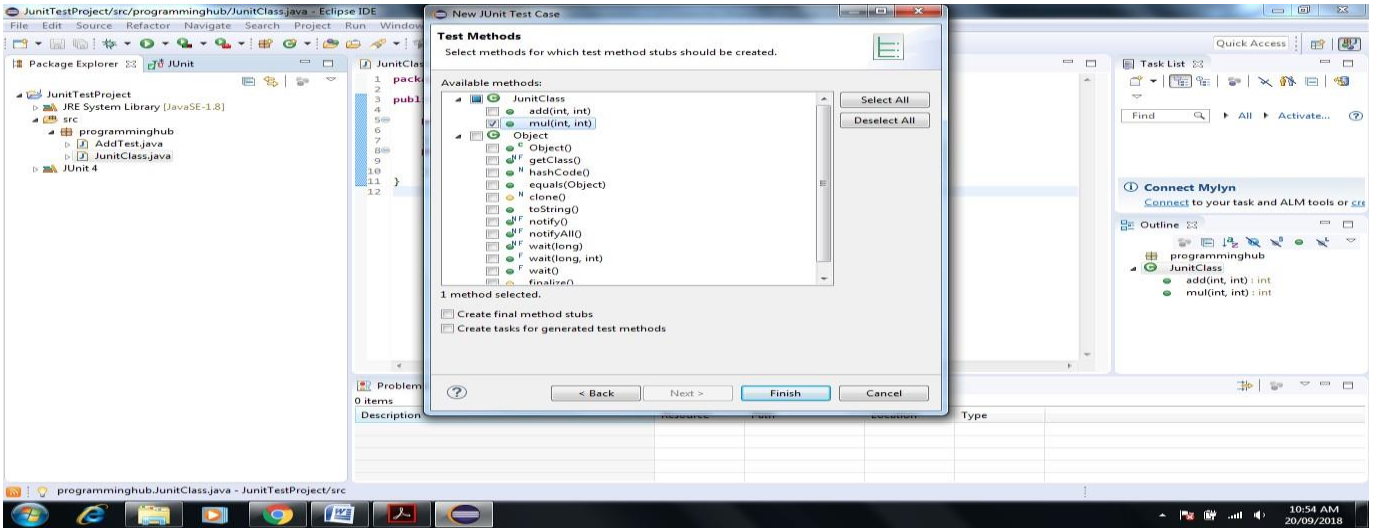
22. Now test case should fail.(Brown Color Bar Indicate) So again run AddTest as follows



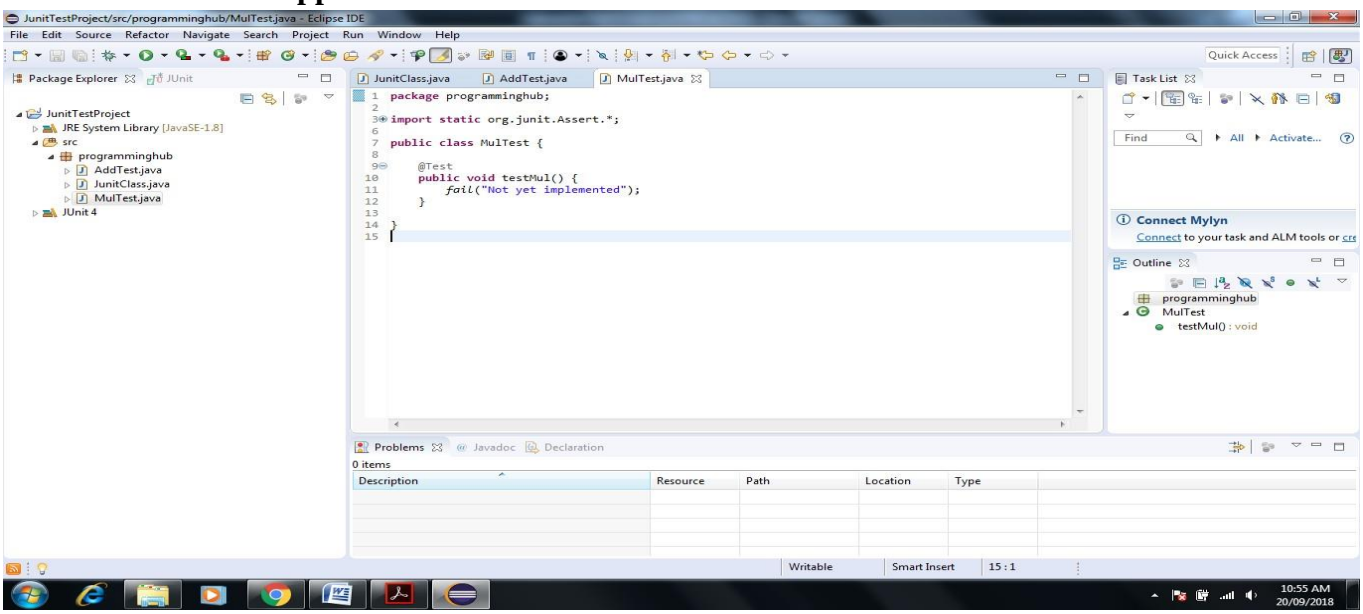
23. Similarly you can Create Test case for Multiplication Function
Click on Project Explorer Screen-> Right Click on JunitClass->New->JUnit Test Case-> Give name MulTest.



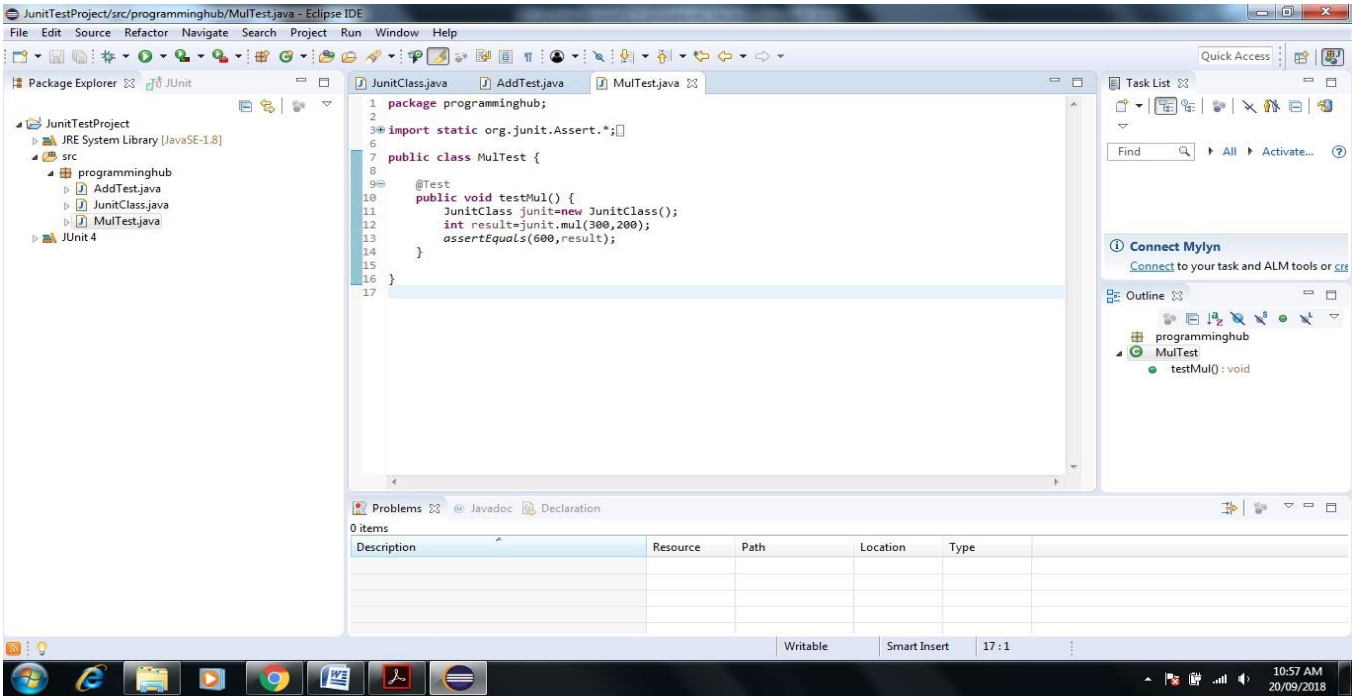
24. Click on Next ->Select Mul Check Box -> Click Finish



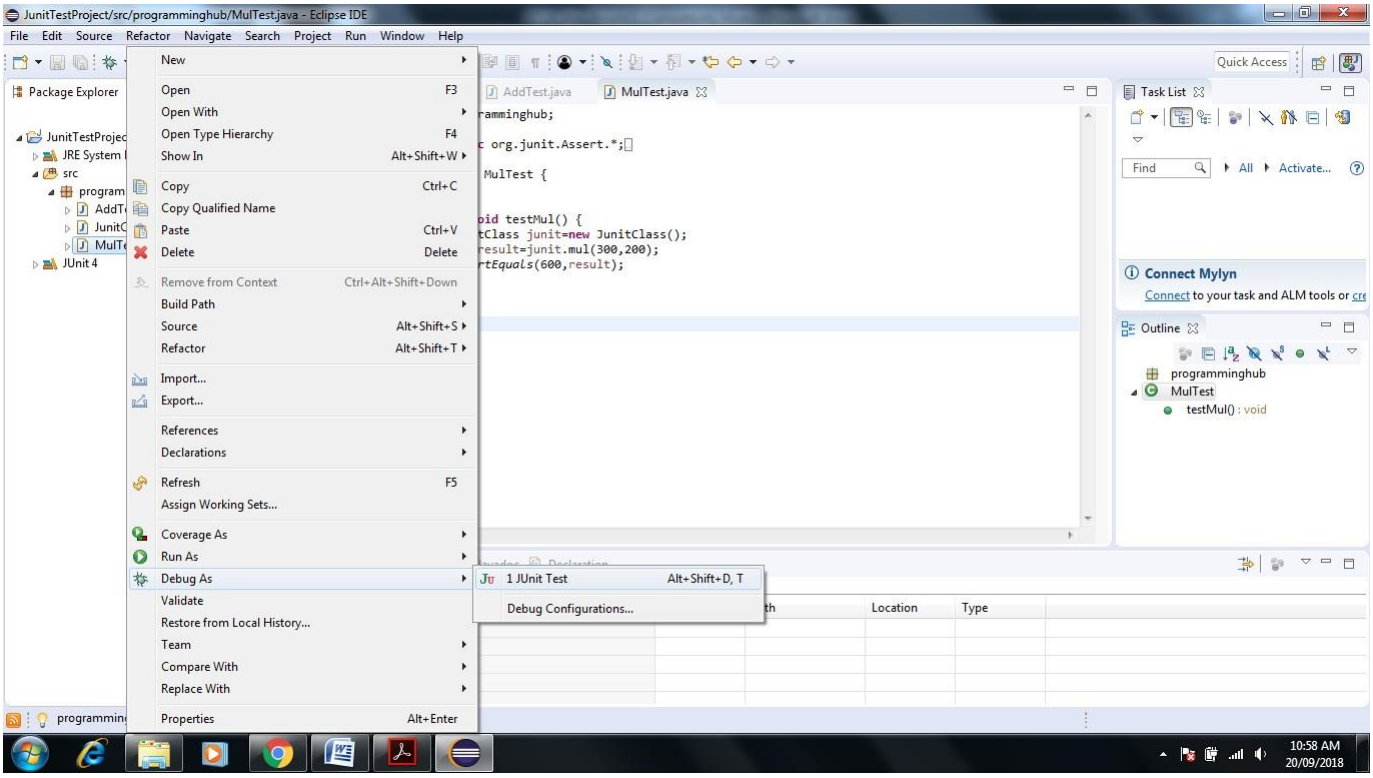
25. Next Screen will appear



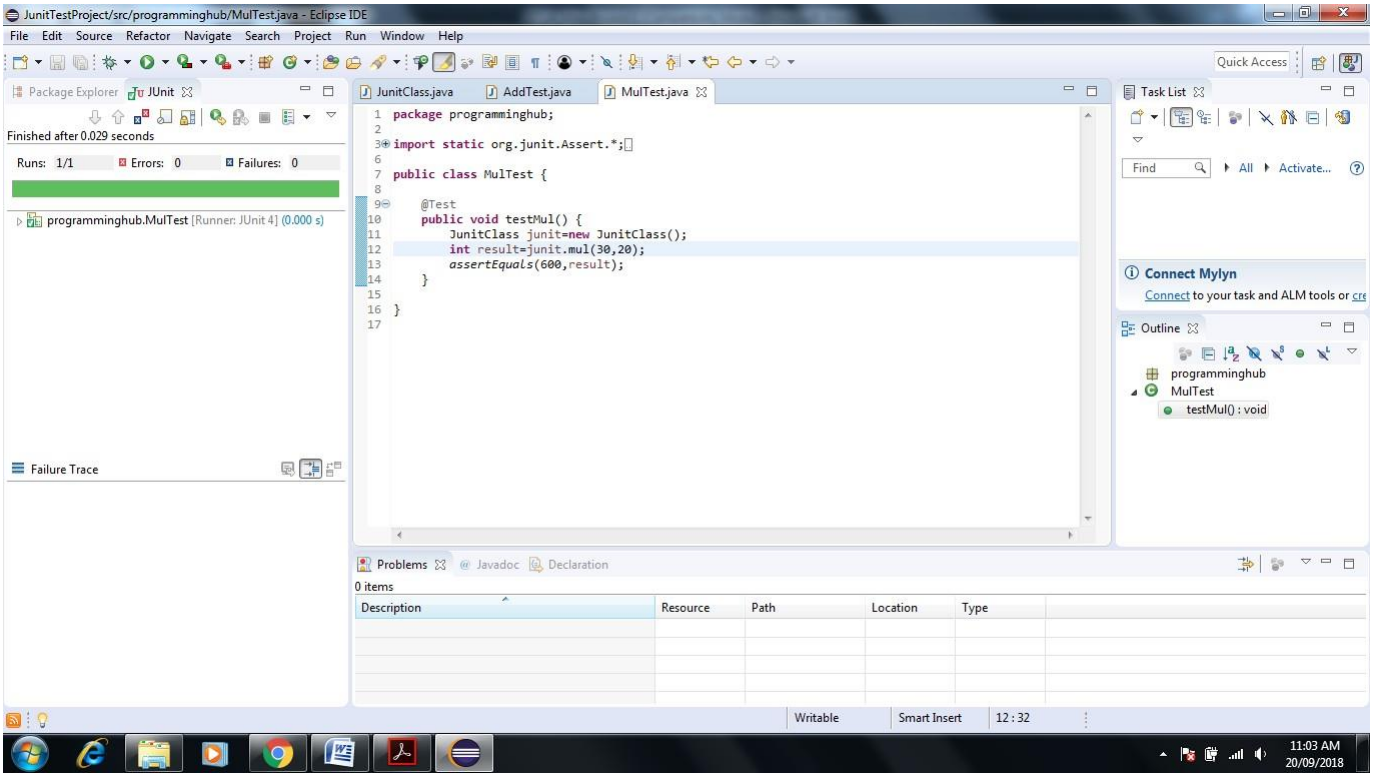
26. Write a Test Case Code inside MulTest method



27. Right Click on MulTest->Debug->JUnit Test

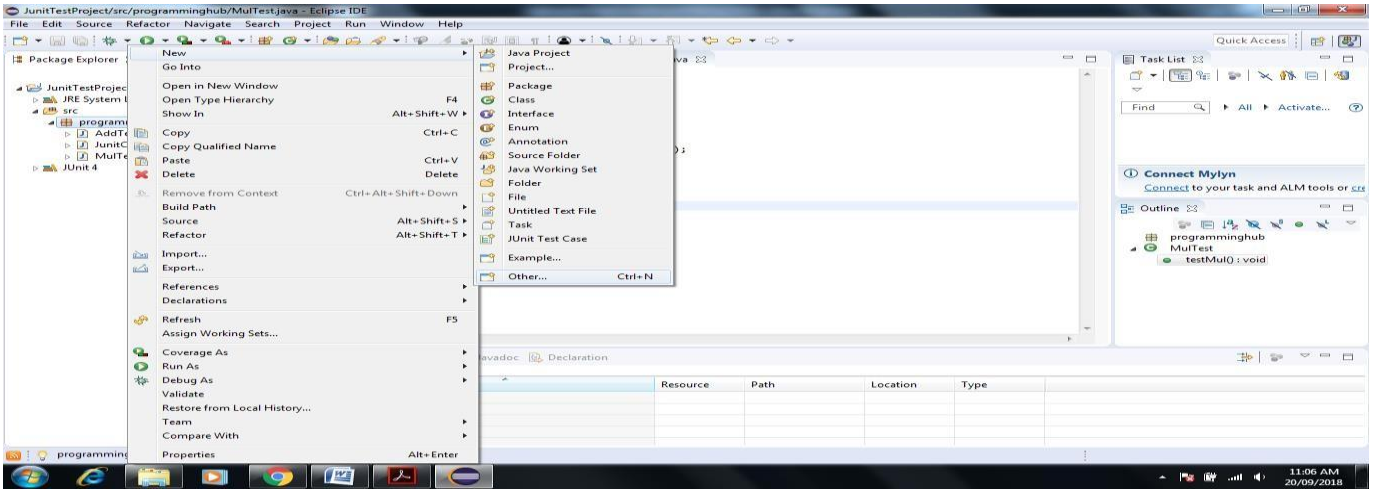


28. Execute Test

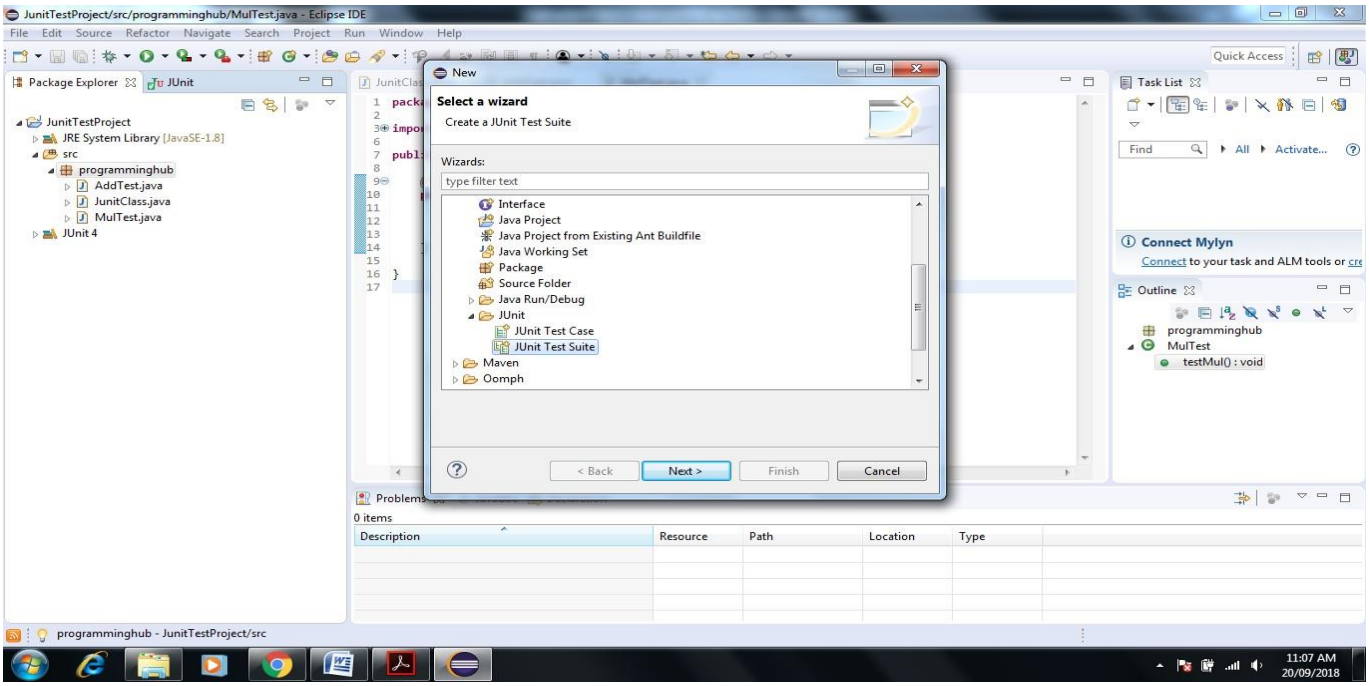


Test Suite – it is used to test multiple test cases at one time.

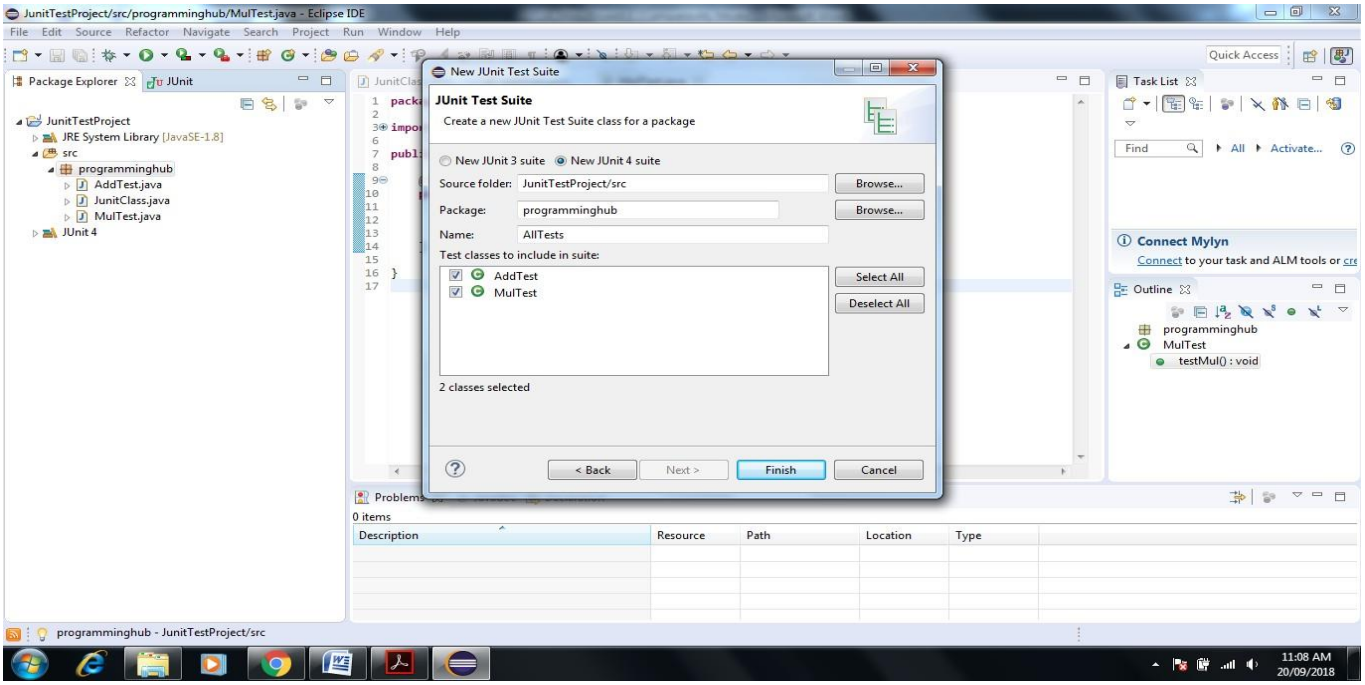
29. Now let us create Test Suite both add and mul test cases in one time



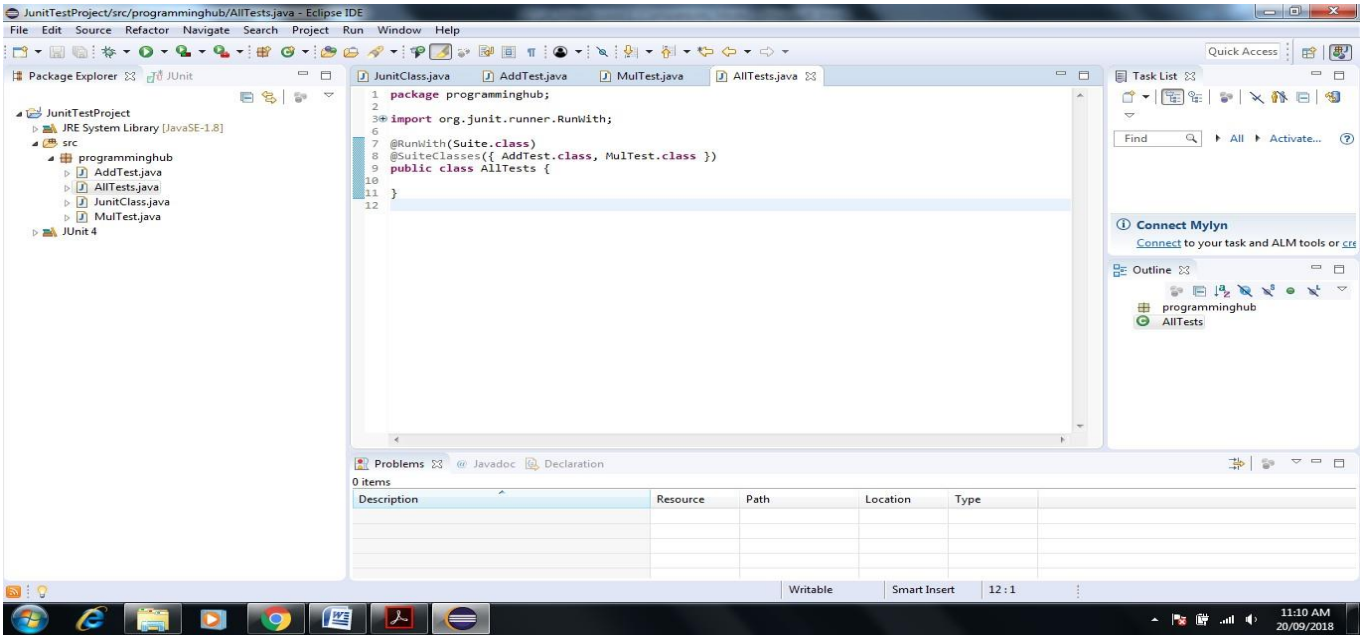
30. Click on Package name->New->Other->JUnit->JUnit Test Suite->Next



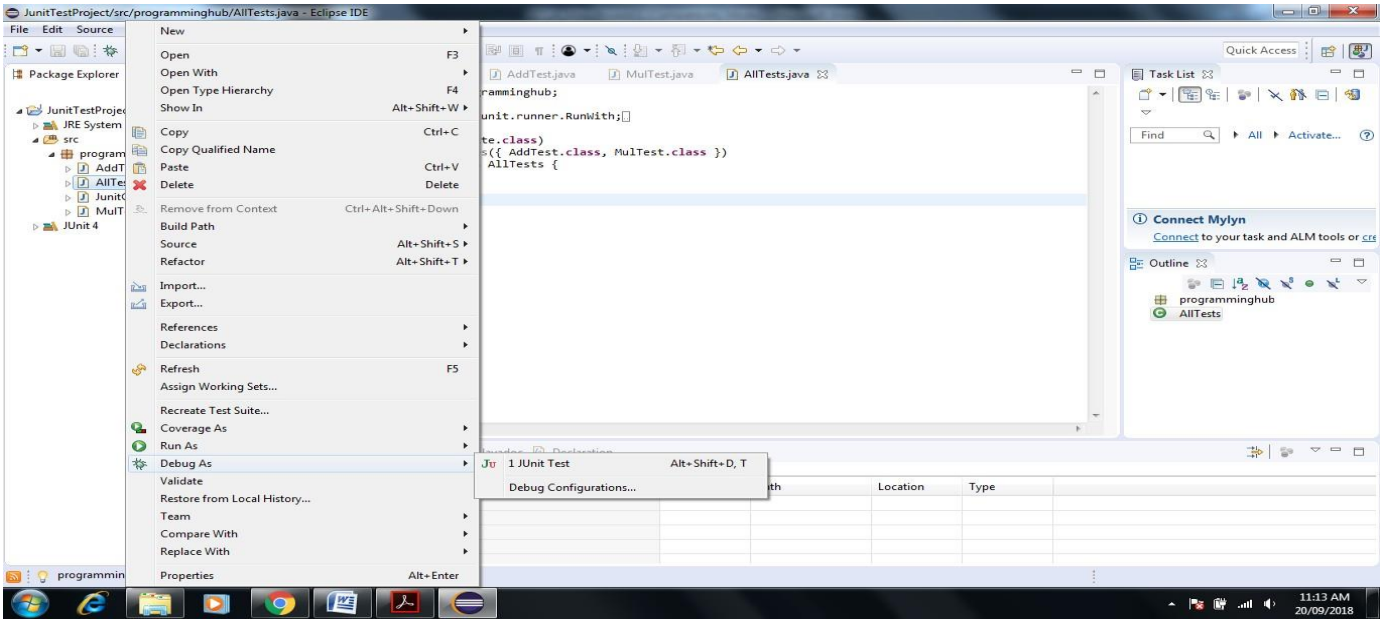
31. Click on Finsh



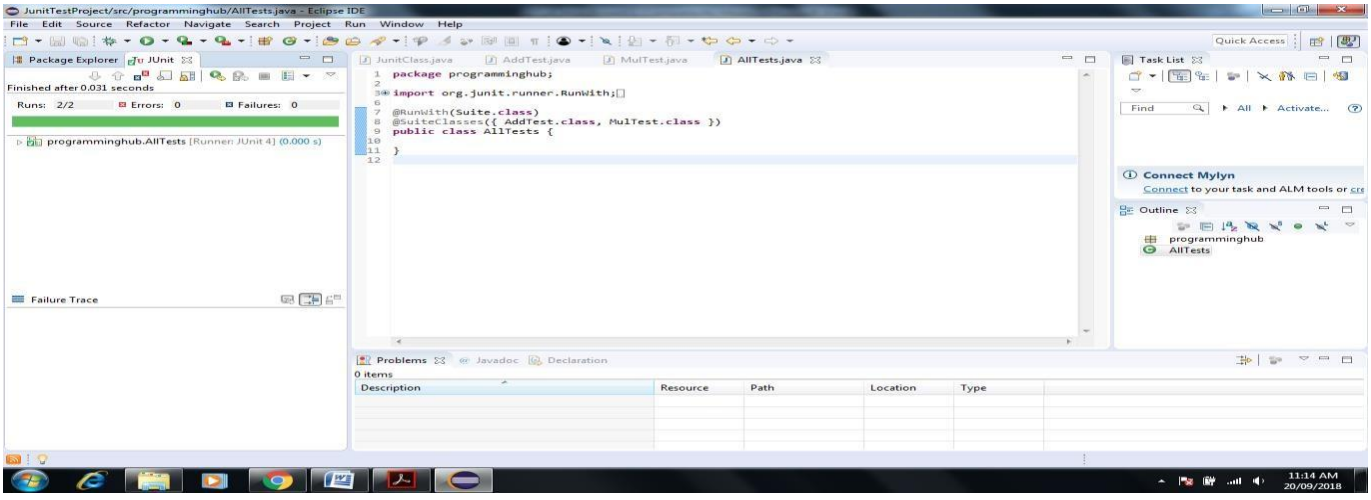
32. Next Screen Appear that automatically create Test Suite for Add and Mul



33. Execute Test Suite Right Click on All Test ->Debug->JUnit Test



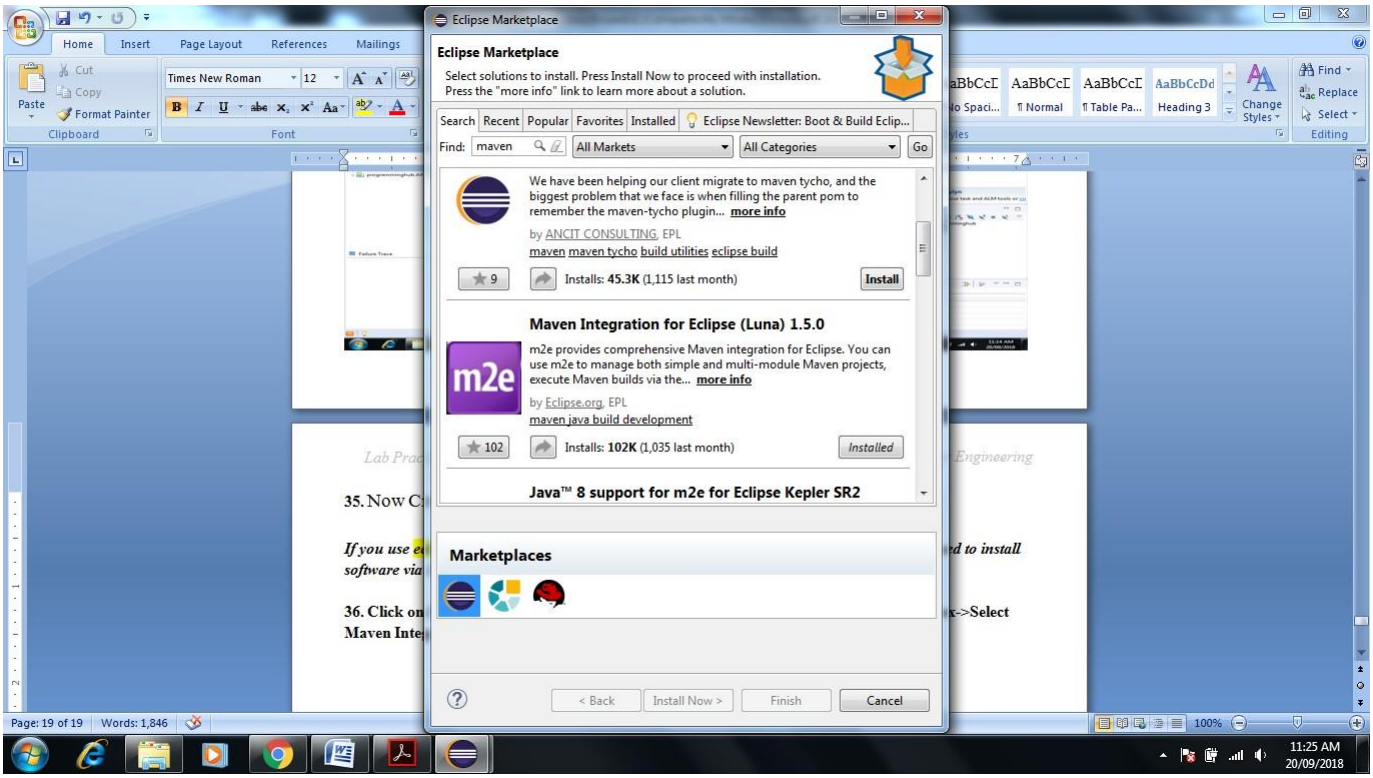
34. Test Suite Executed successfully Test suite fails even if a single test case among all fails.



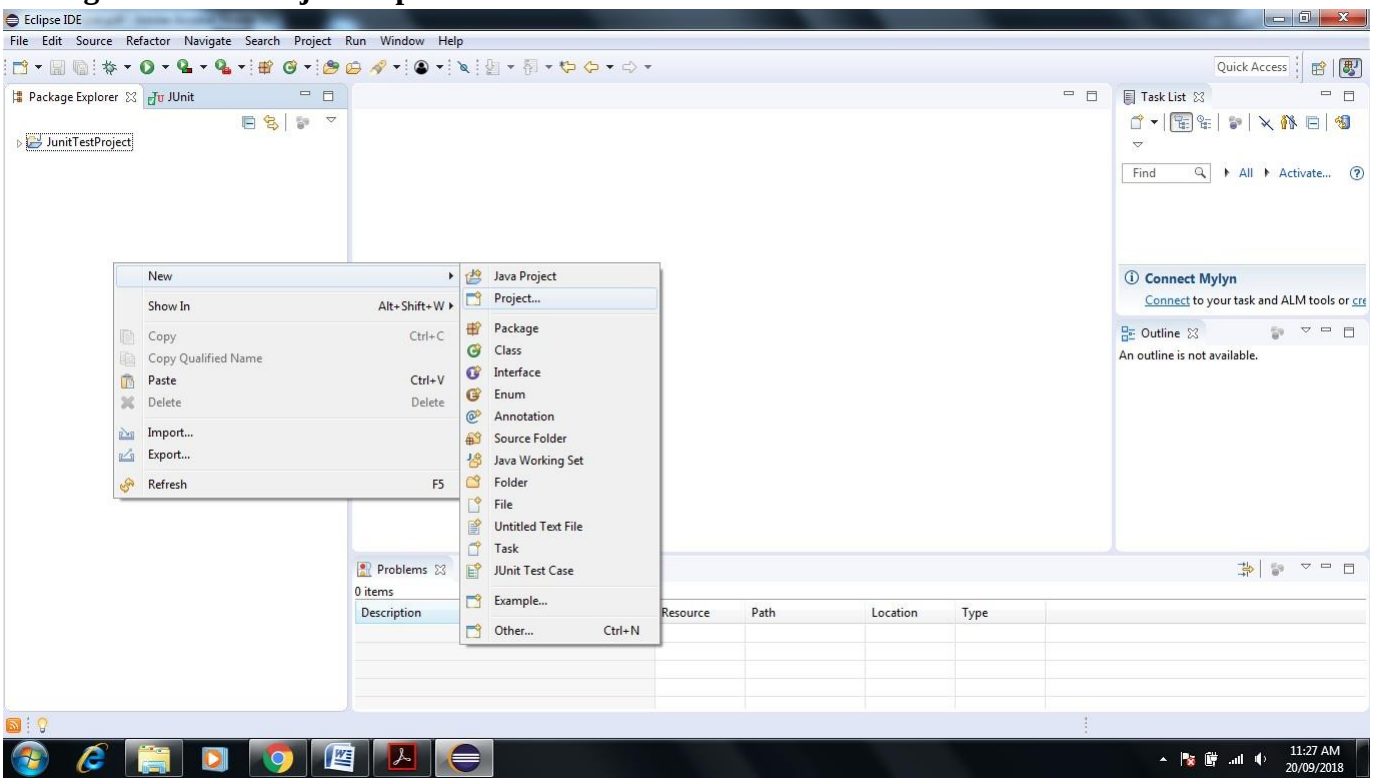
35. Now Create **Test Report** Using **Apache Maven**

If you use eclipse-java-photon-R-win32 Version it include Maven in built installed so no need to install software via Eclipse help Install Software Option

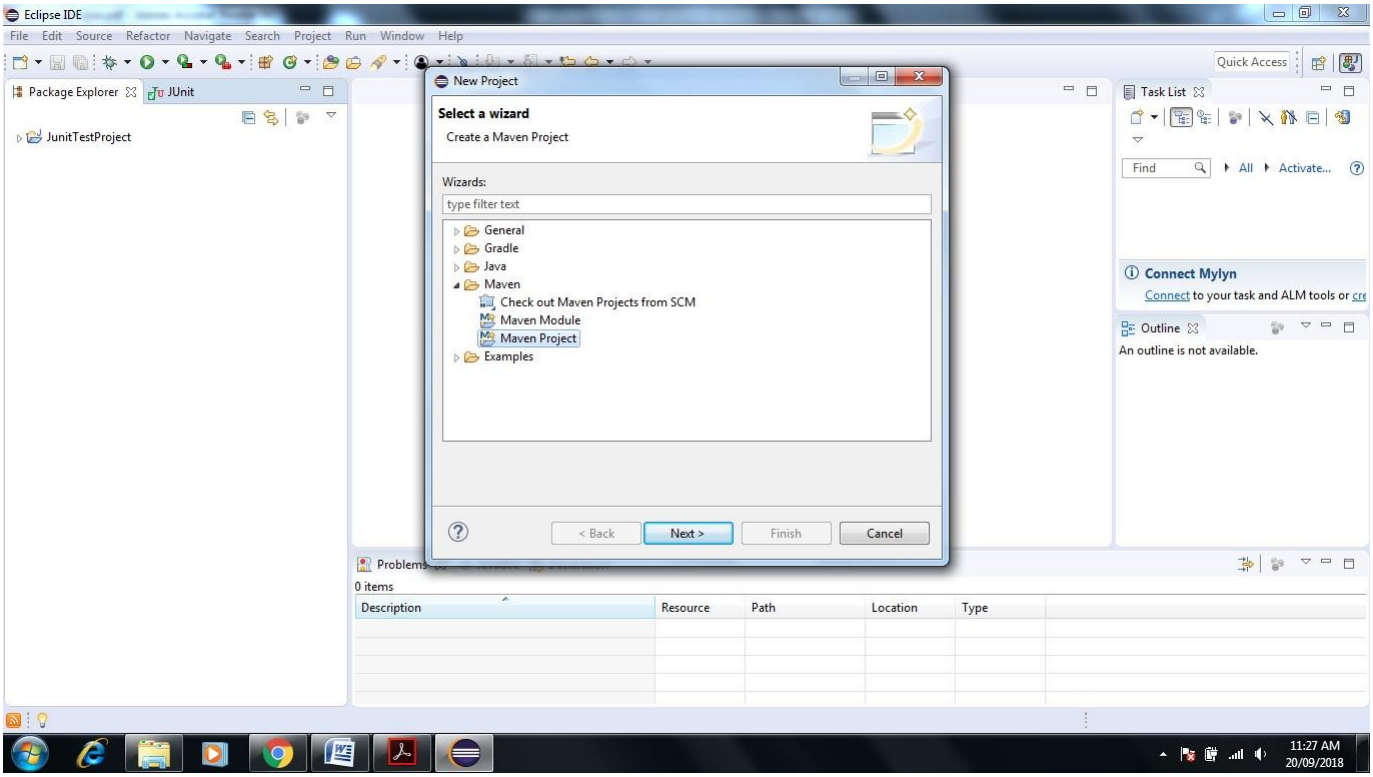
36. Click on Help in Eclipse->Eclipse Marketplace->Enter Maven Keyword in Search box->Select Maven Integration version as per requirement->Click on Install



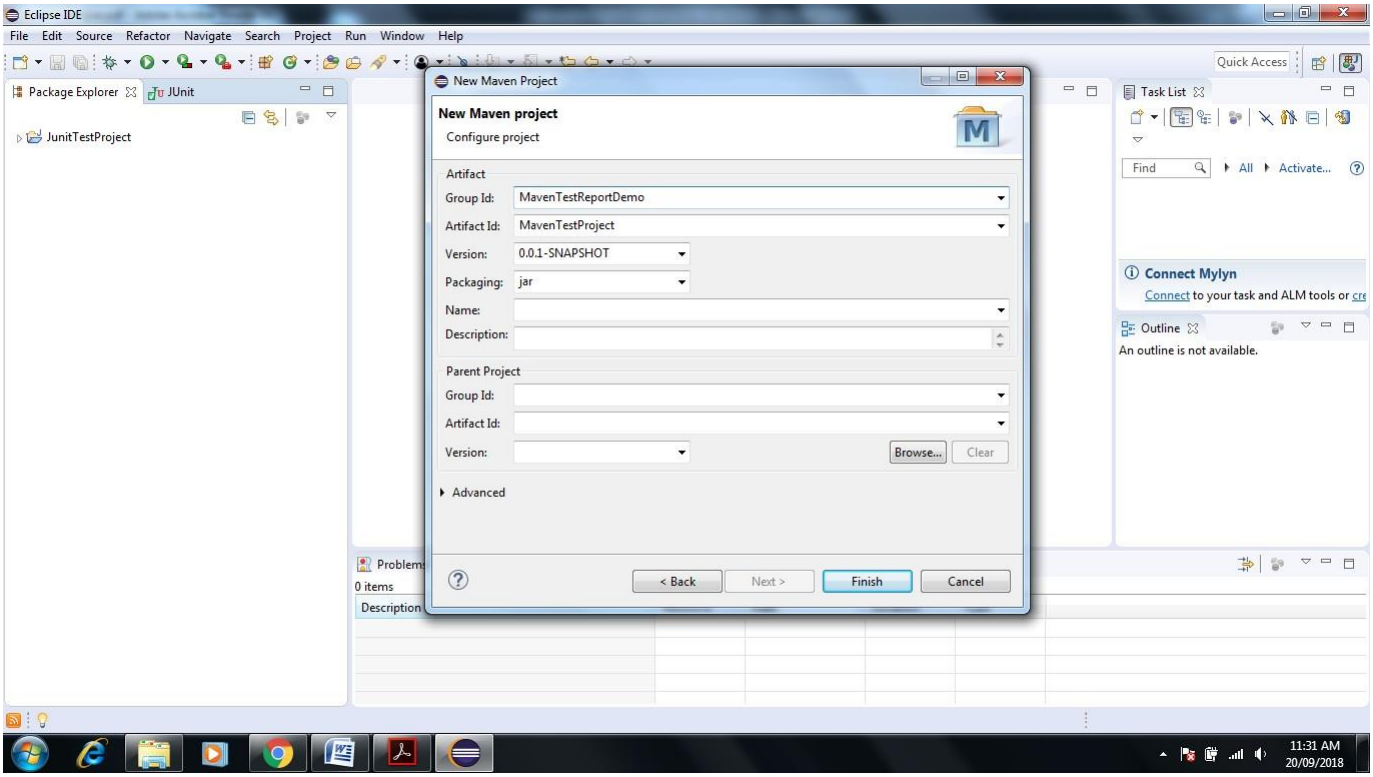
37. Right Click in Project Explorer Window



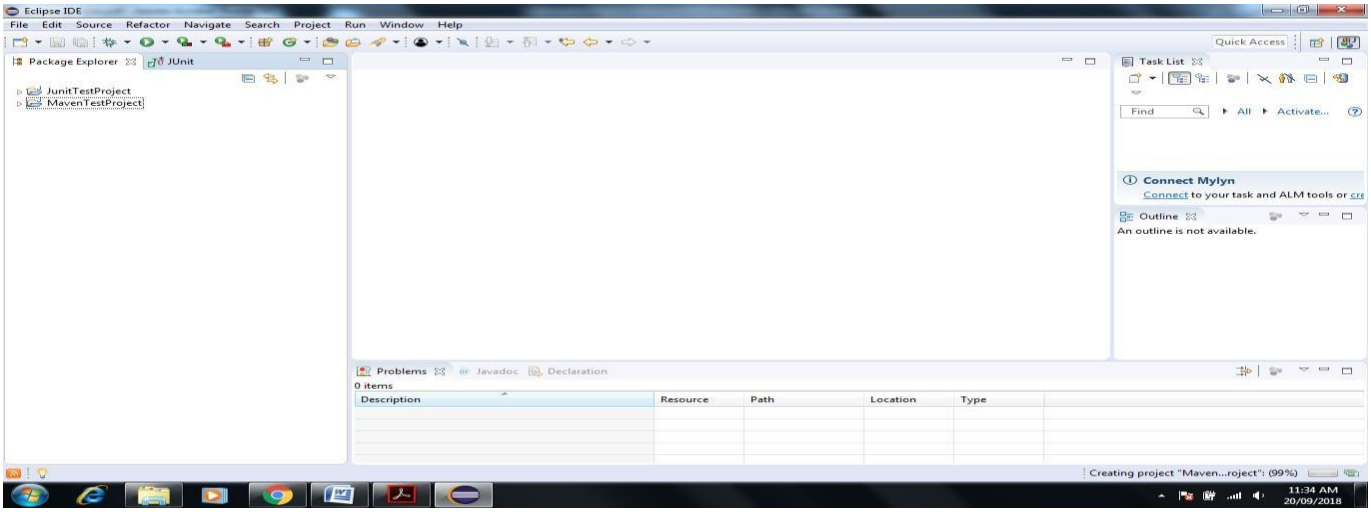
38. Go to Maven Project-> Click Next



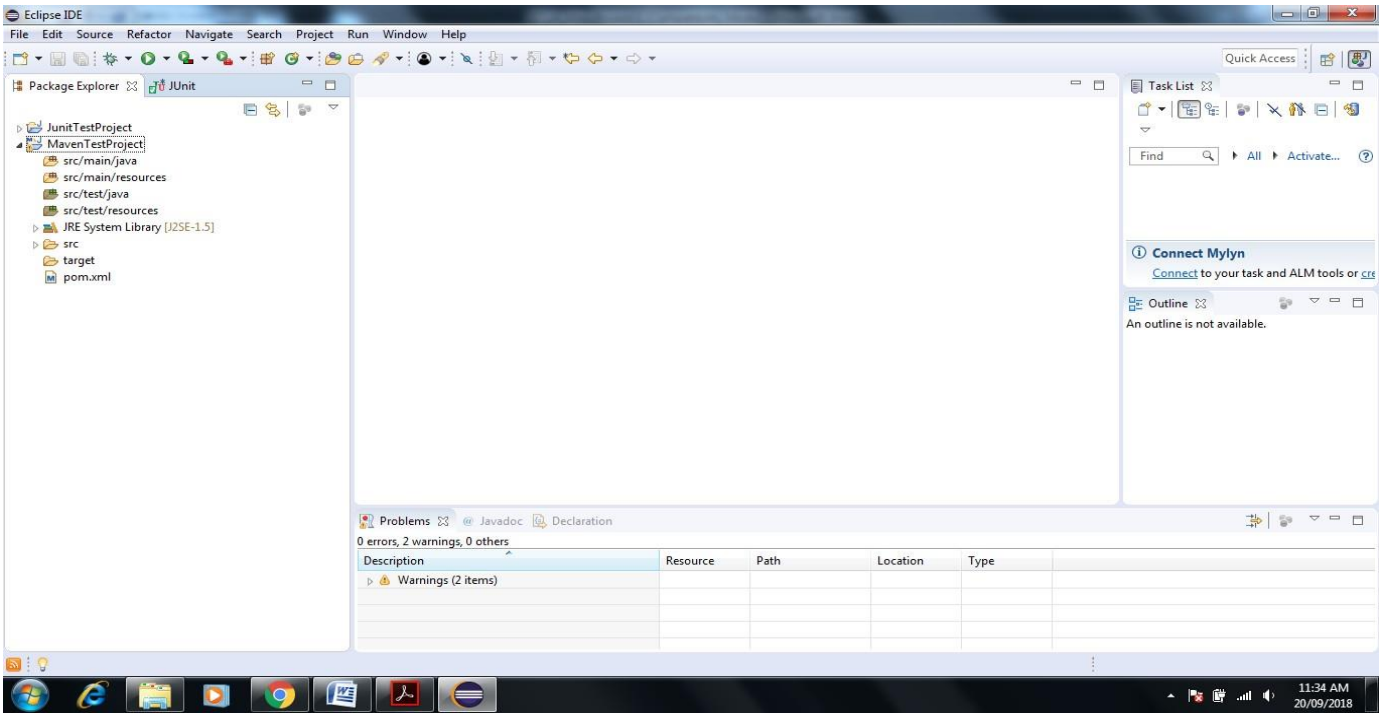
39. Select Check Box Create Simple Project-> Click Next-> Give Group Id and Artifact name



40. Click on Finish-> Next Screen Appear

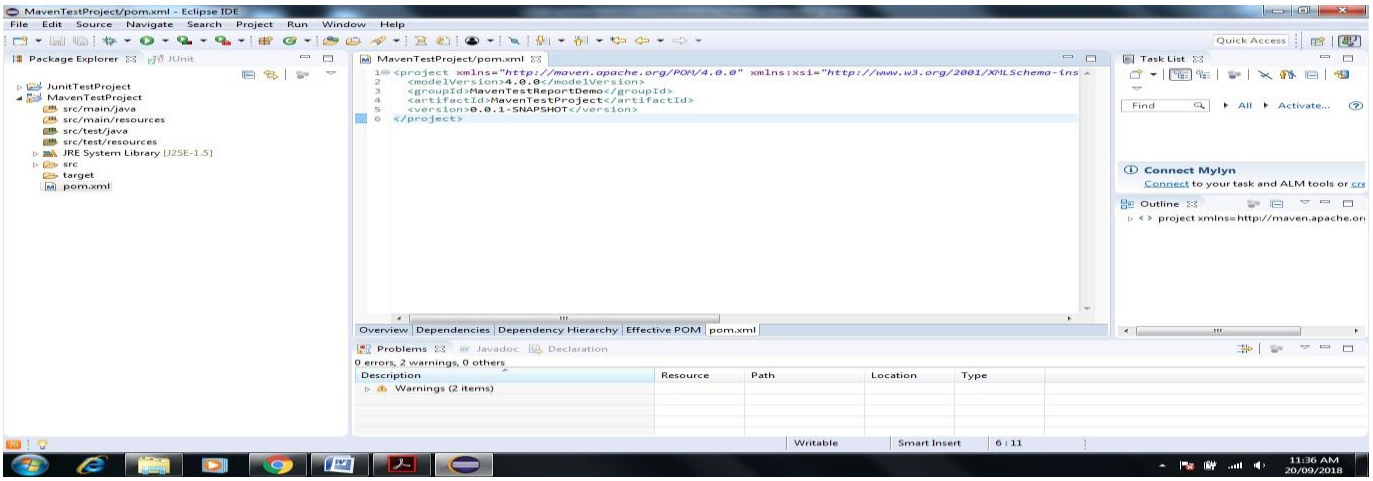


41. MavenTestProject shown Pom.xml file doble click on same

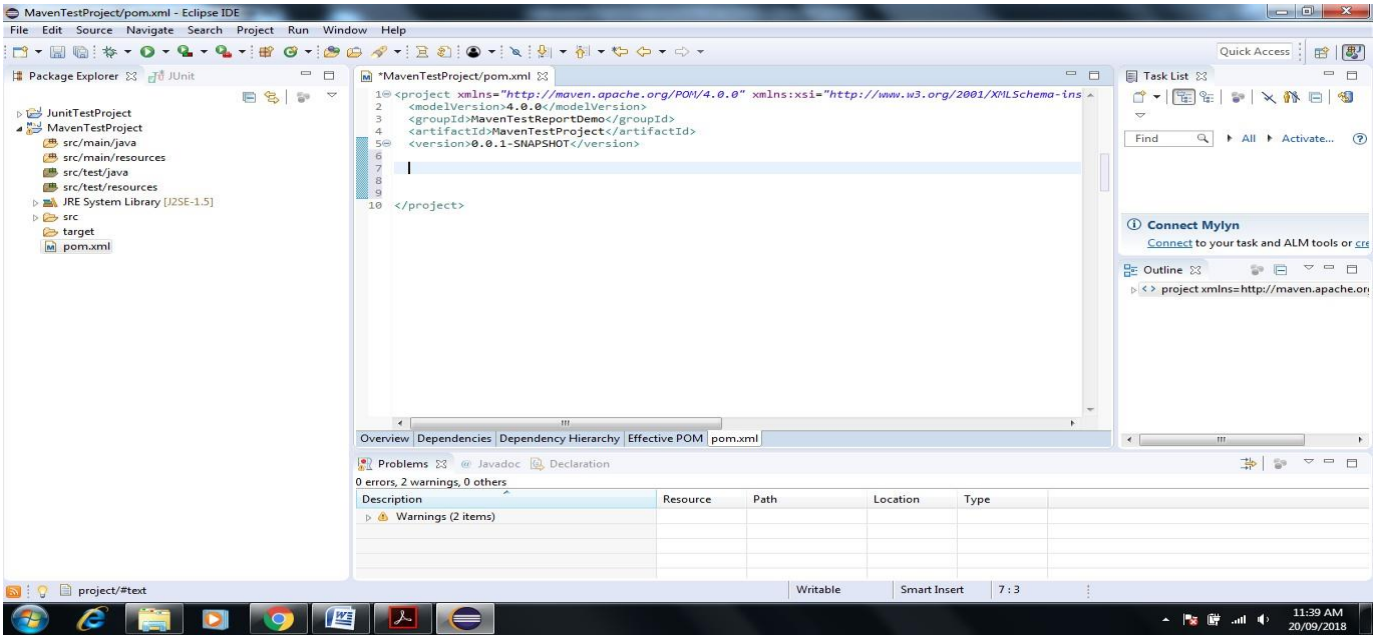


42. it shown some description like

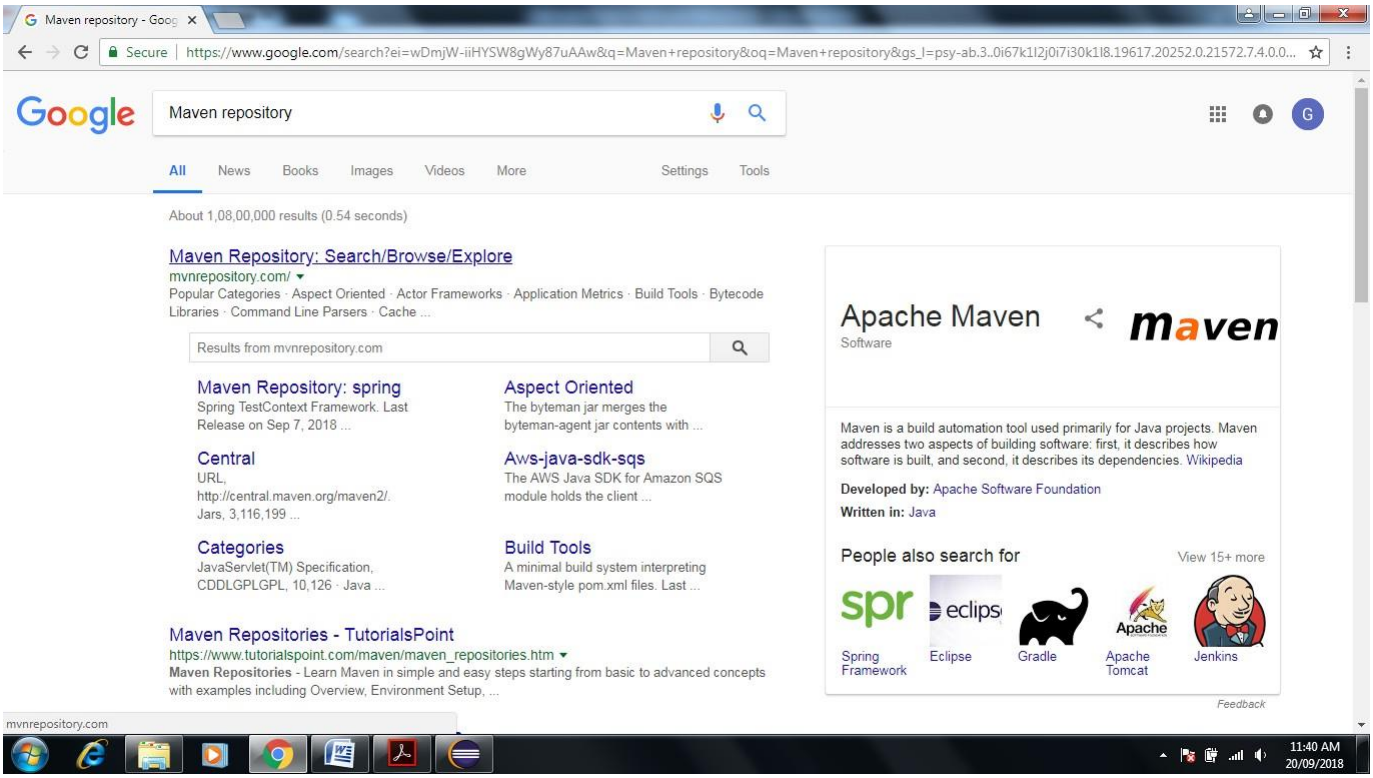
```
<modelVersion>4.0.0</modelVersion>
<groupId>MavenTestReportDemo</groupId>
<artifactId>MavenTestProject</artifactId>
<version>0.0.1-SNAPSHOT</version>
```



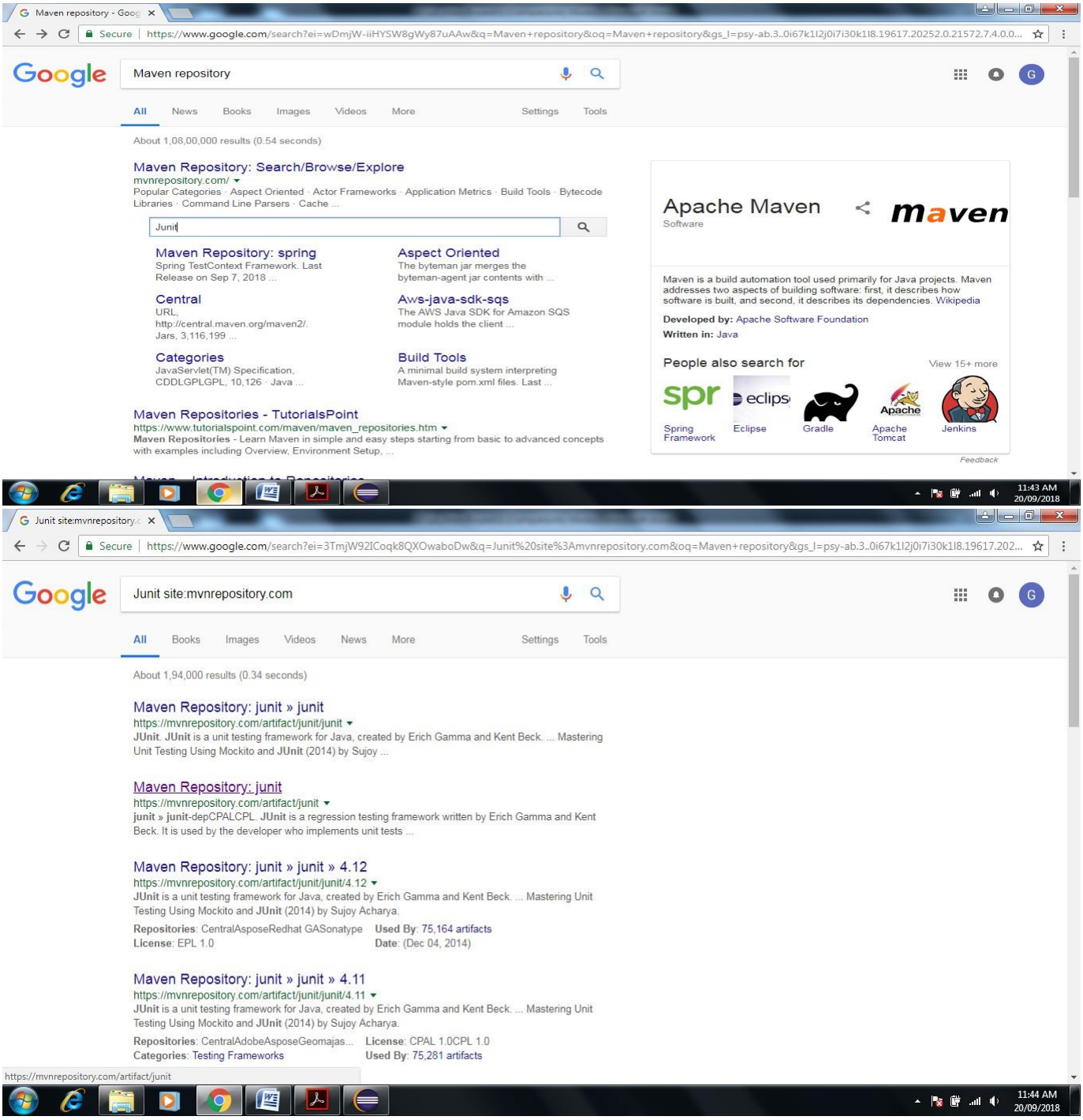
43. We add dependencies to pom.xml of Junit and Selenium



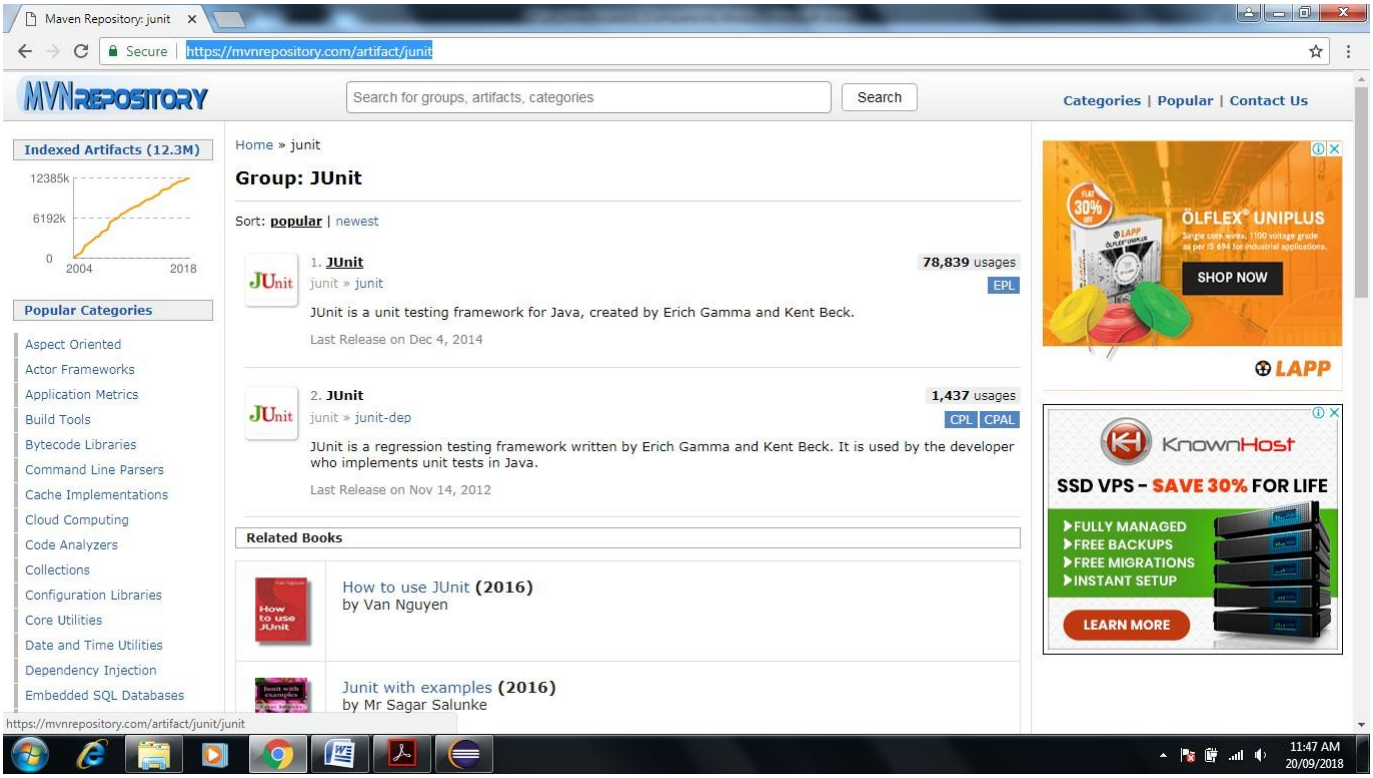
44. To add dependency❼ Go to Google.com->Enter Maven repository-> in Search box on S❽Enter Junit



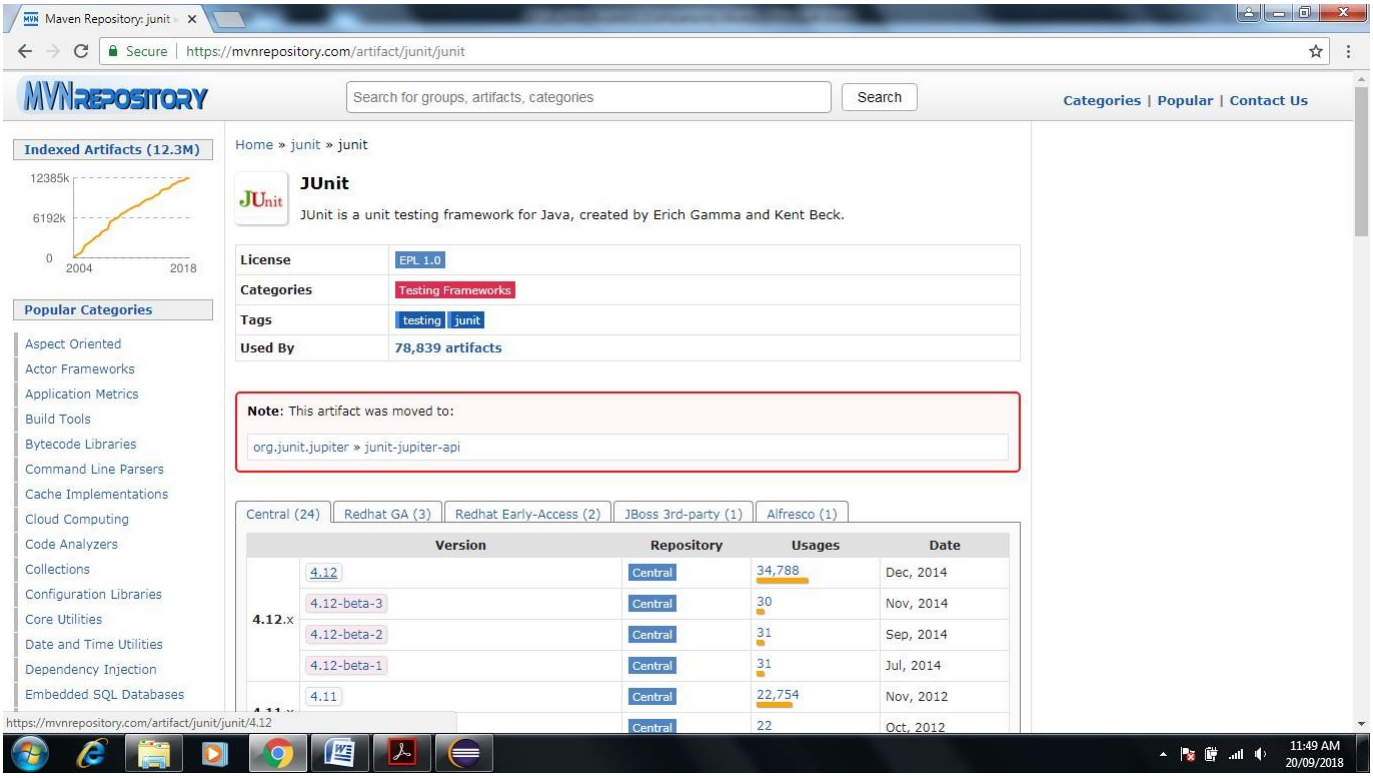
45. after Enter keyword Junit inside Seach box then Enter->it shown another Site Maven Repository for Junit Select that site.



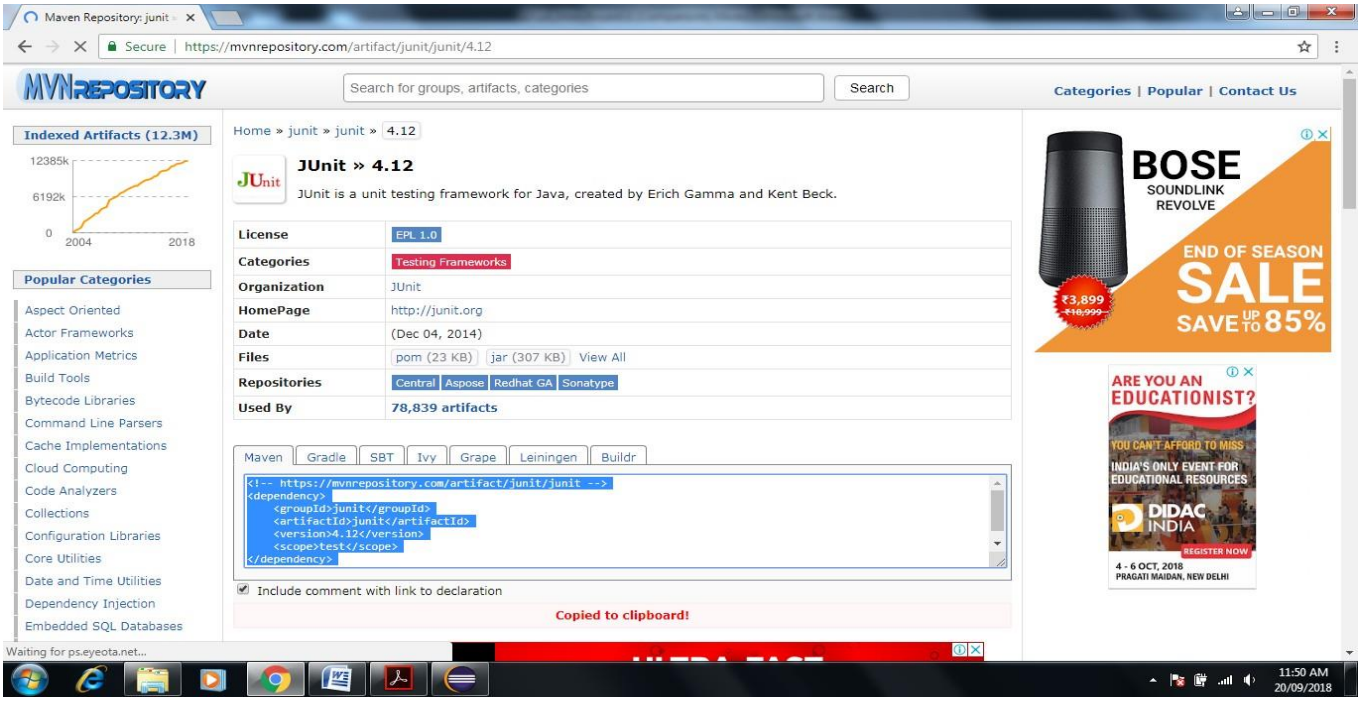
46. Click on Maven Repository-JUnit it open another site-(https://mvnrepository.com/artifact/junit)



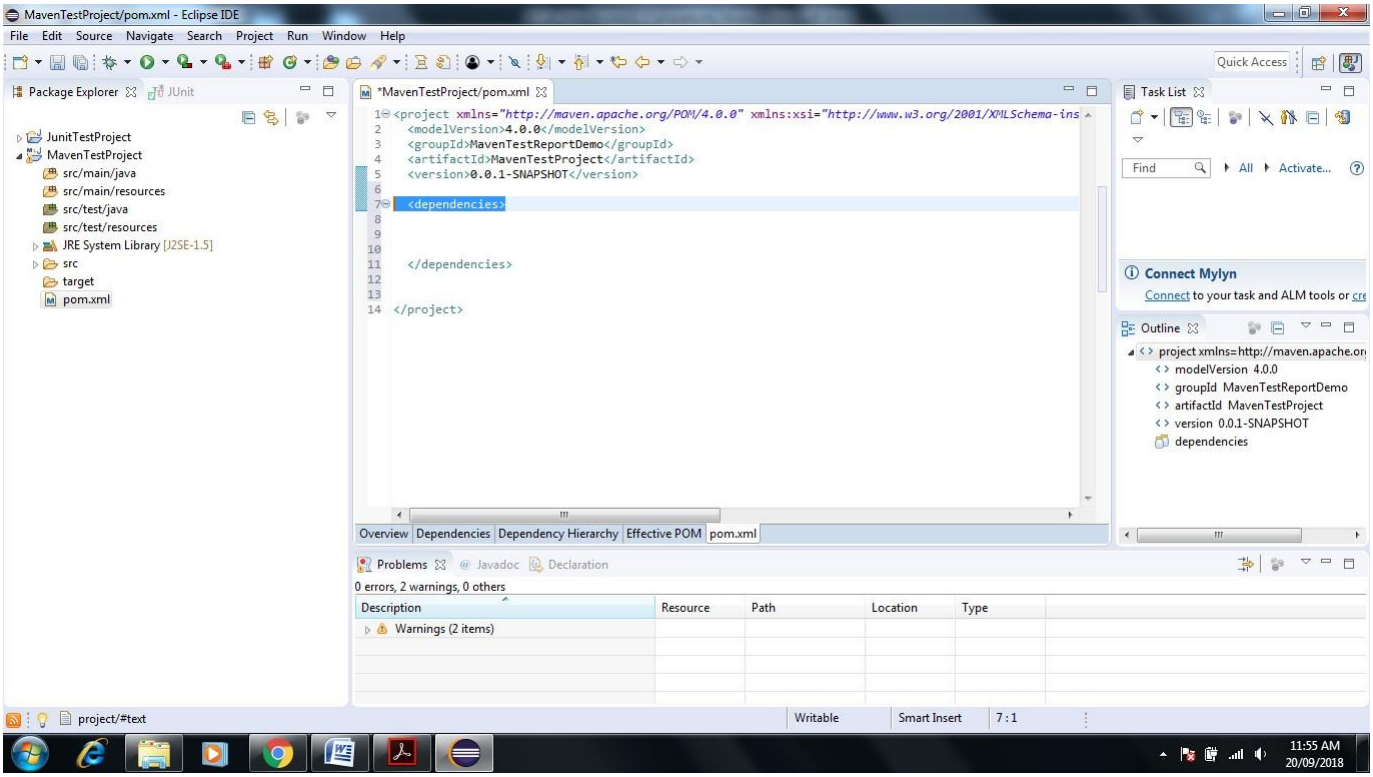
47. Click on JUnit-> Open and click on latest version as shown below (here 4.12x)



48. Copy above dependency to paste in pom.xml in Maven in Eclipse

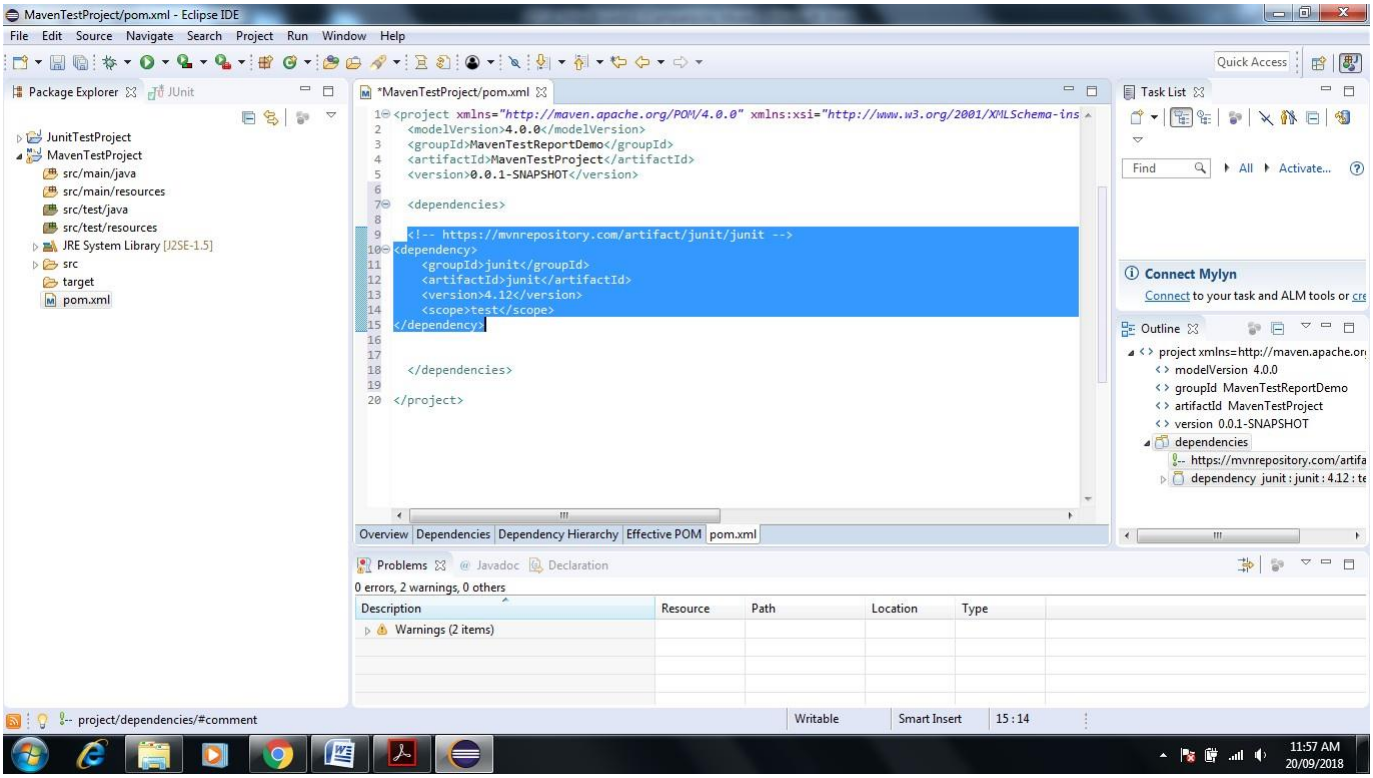


49. Add <dependencies> tag before pasting as shown below

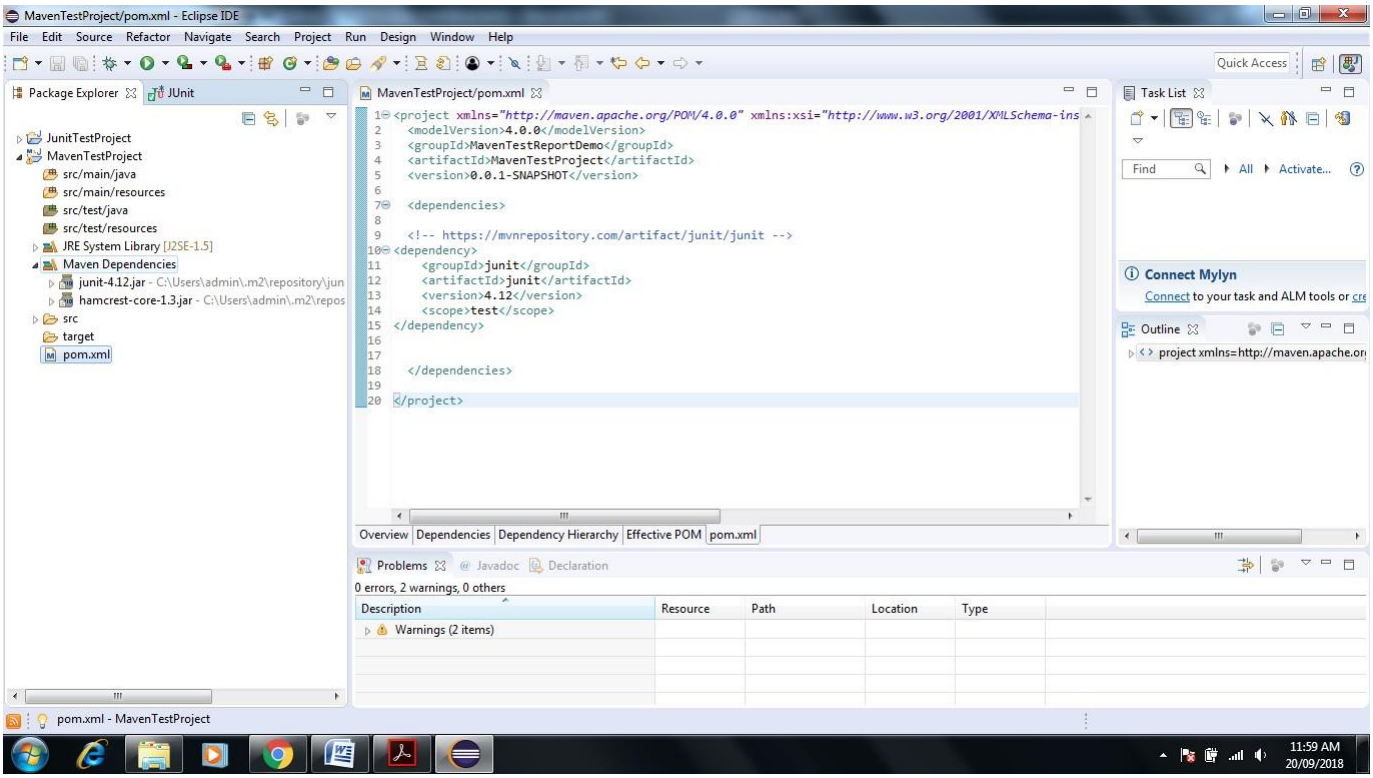


50. Now Paste the above code in between <dependencies> tag then save pom.xml file

```
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

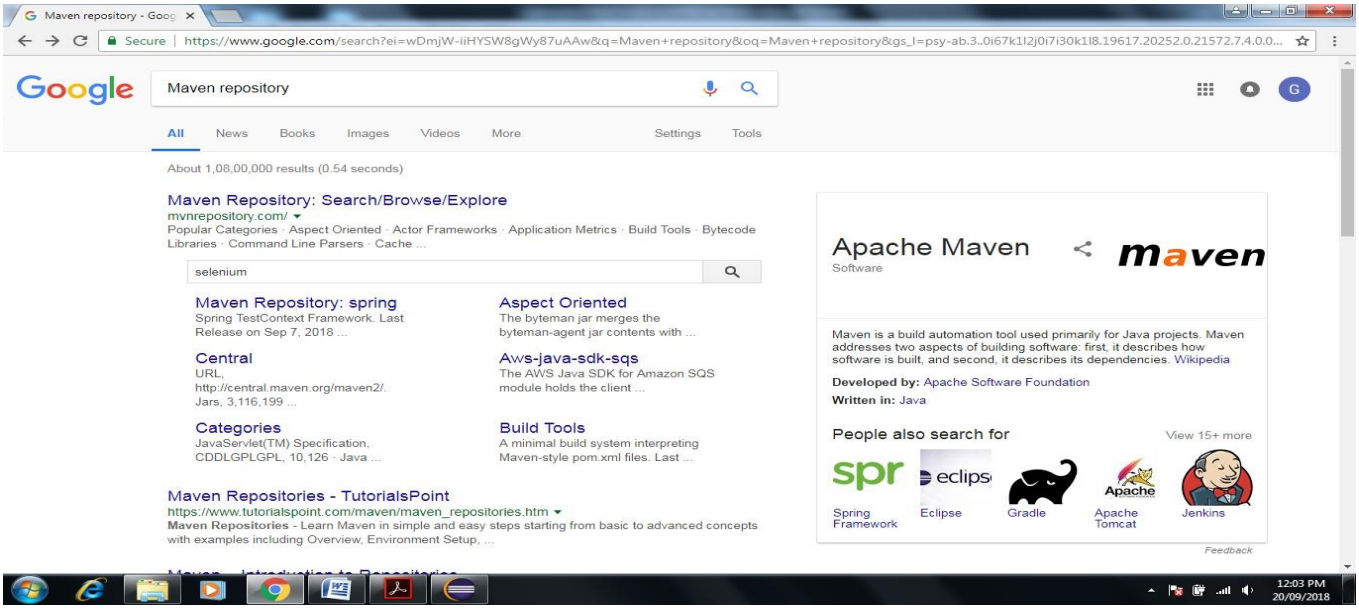


51. Now it It gets reflected in Maven by adding Junit jars

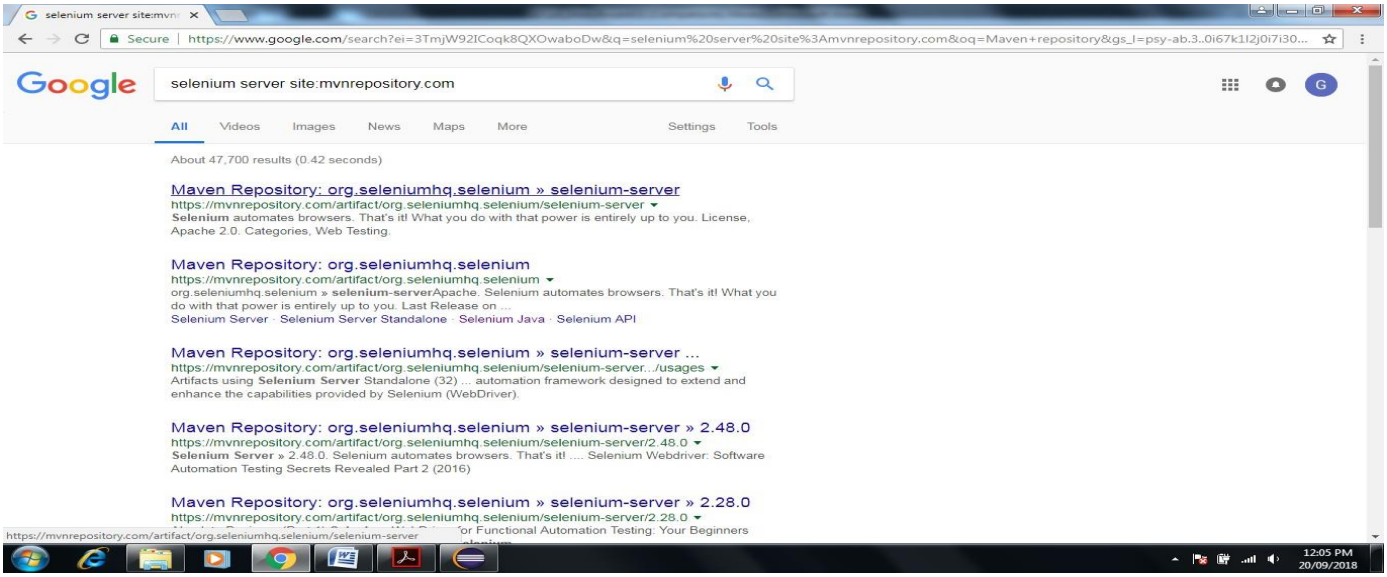


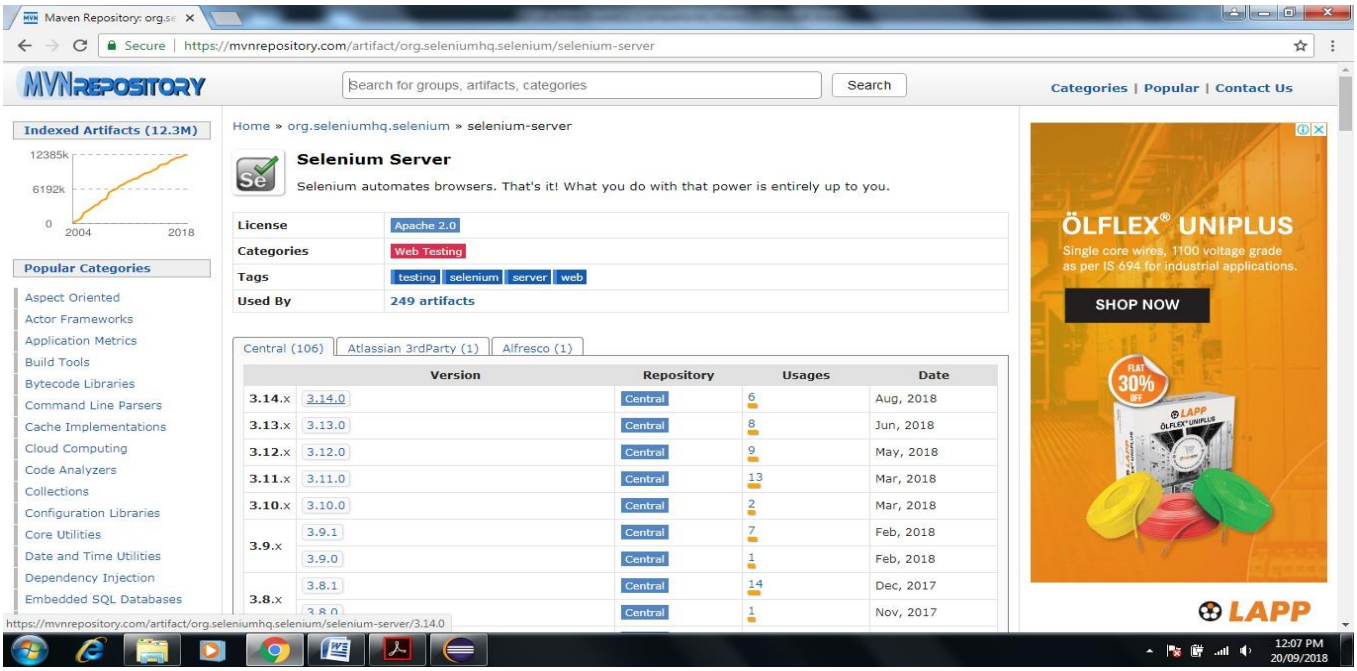
52. Same process can be repeated for Selenium server

Go to Google-> Enter Maven Repository->Enter Selenium Server in Search box->Enter

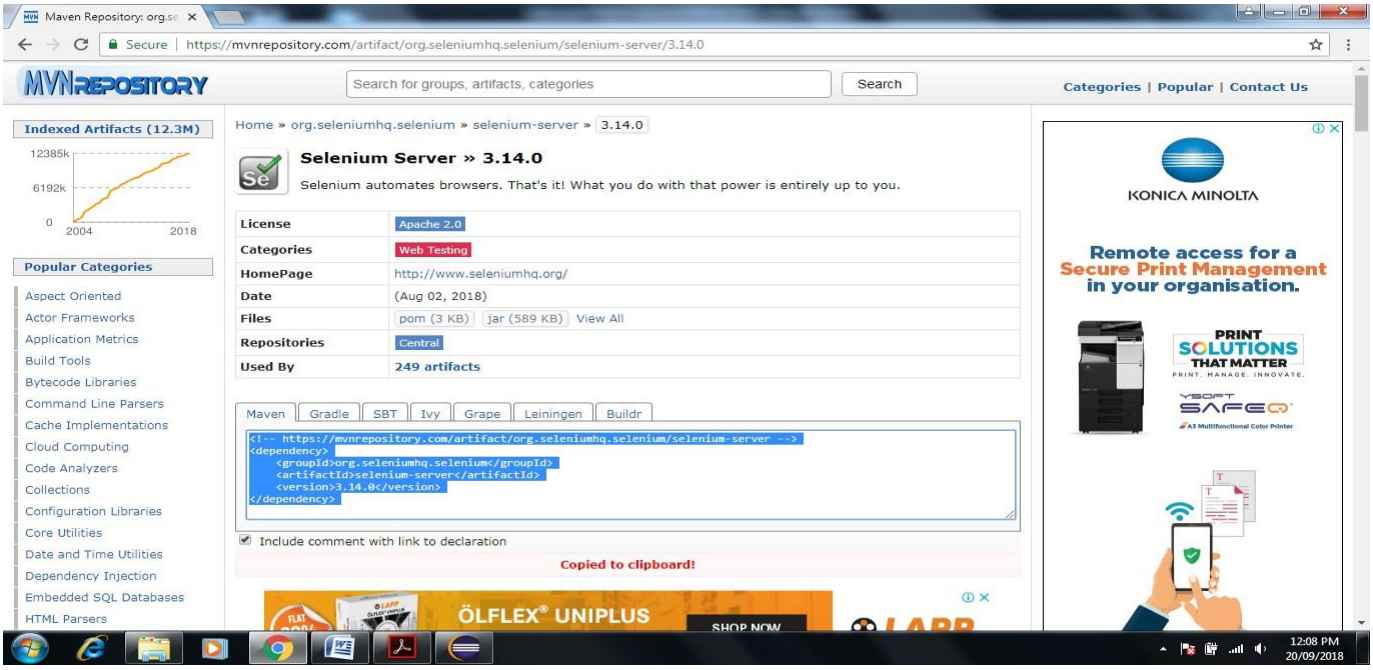


53. Click on First Link of Website-> Click on latest version

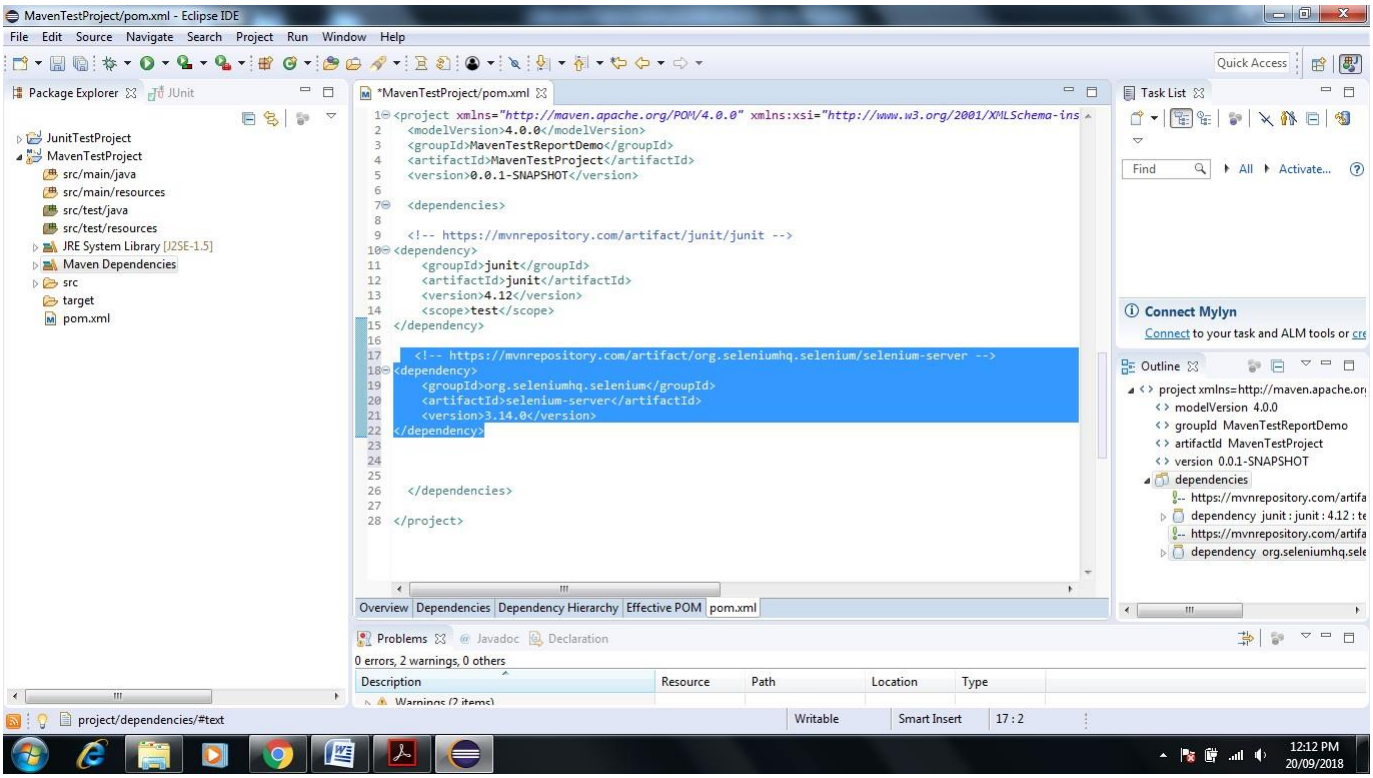




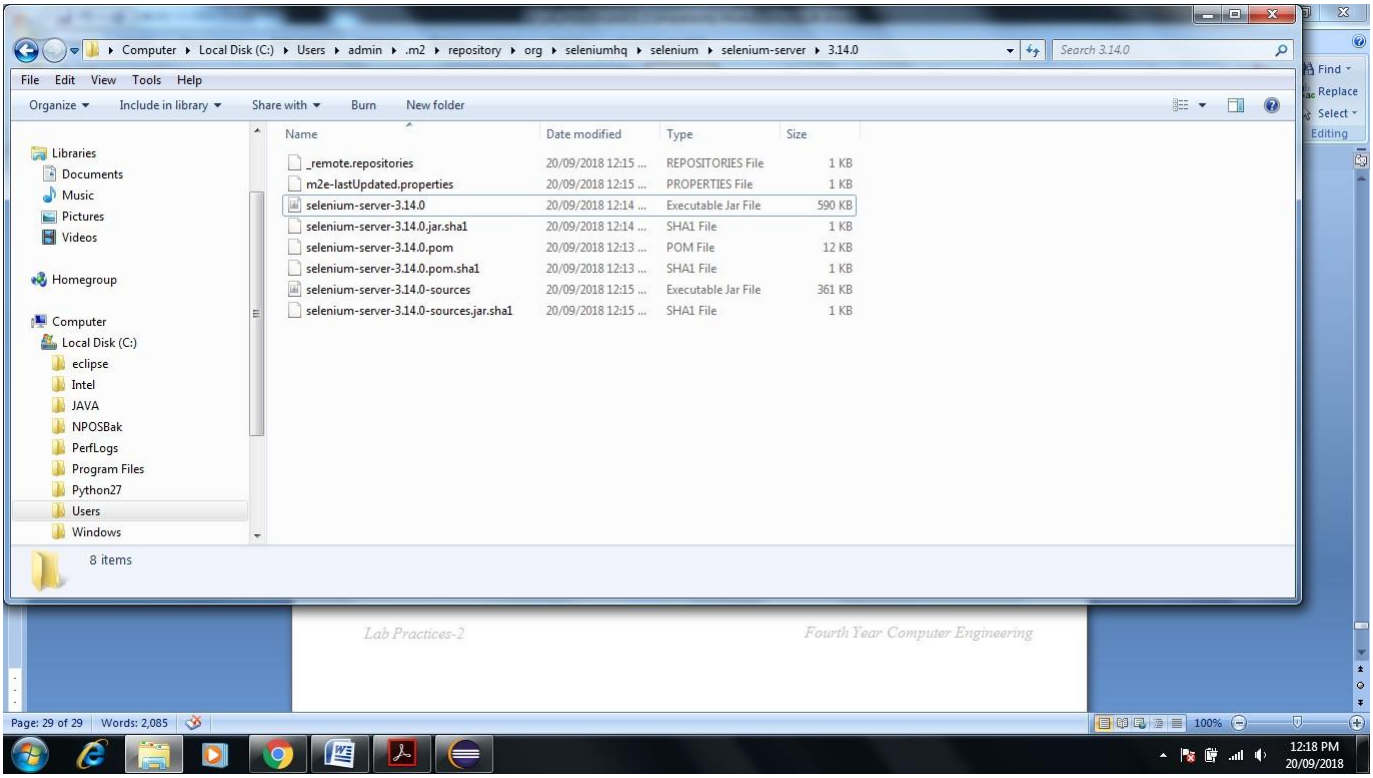
54. Copy Code in Maven Tab



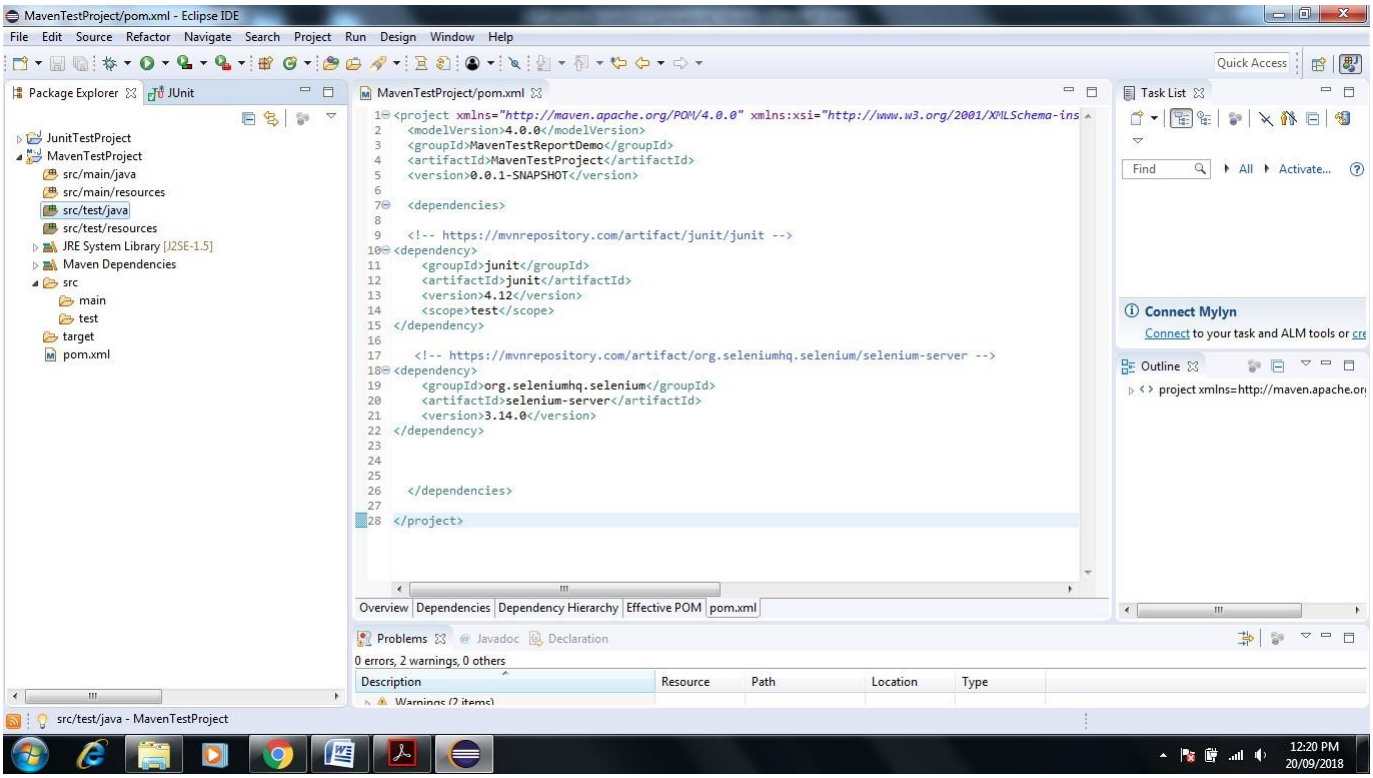
55. Paste in pom.xml file in between <dependencies> tag



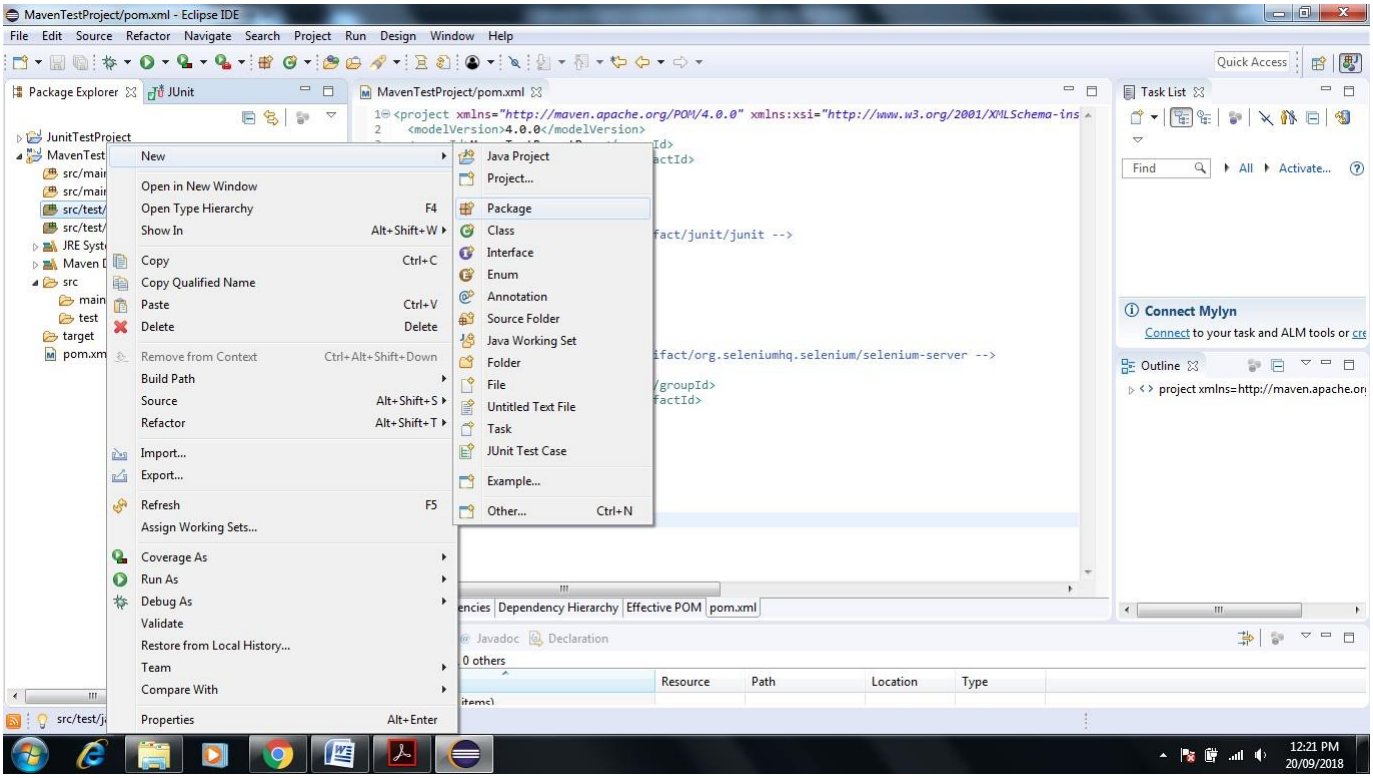
56. Now Go to C:\Users\admin\.m2\repository\org\seleniumhq\selenium\selenium-server\3.14.0 Check the latest selenium server version.



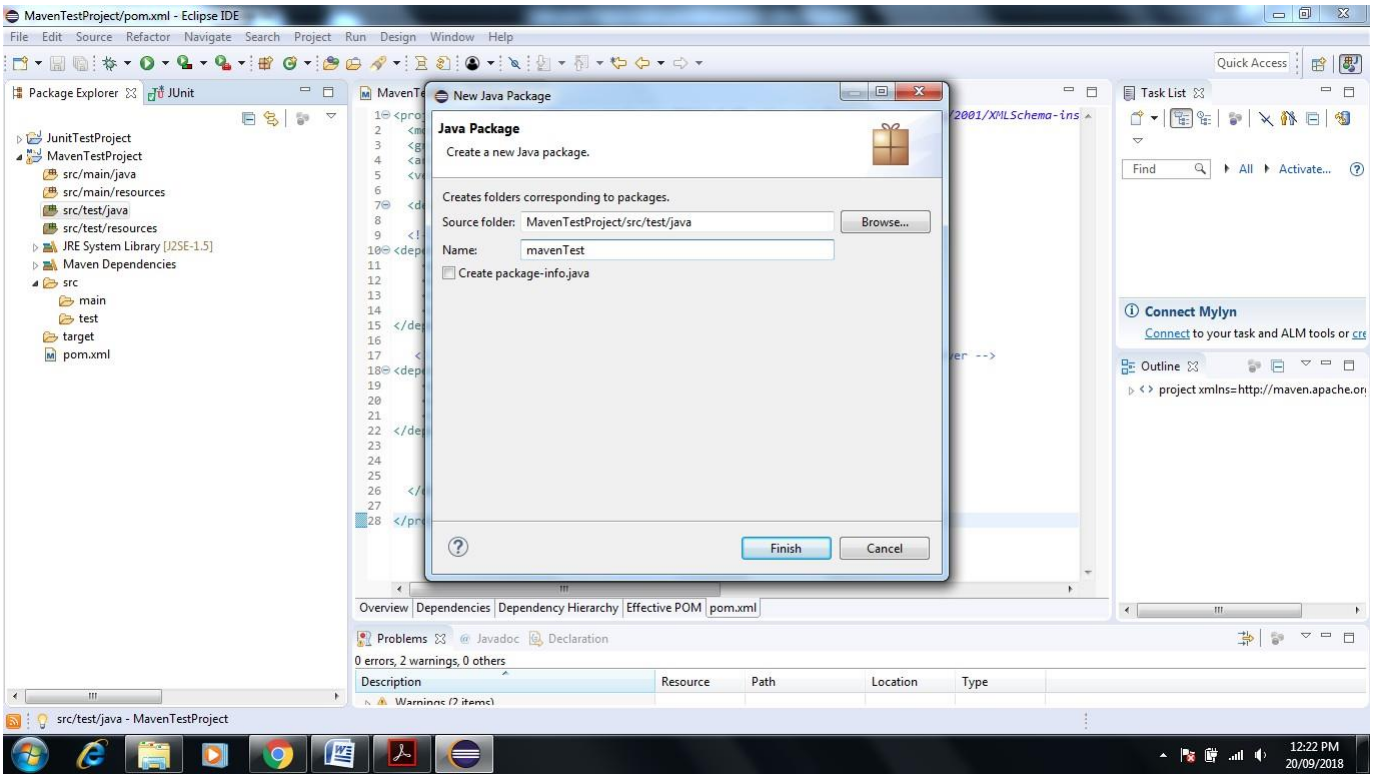
57. Now go to Eclipse -> Click on Maven Test Project->Right Click on src/test/java



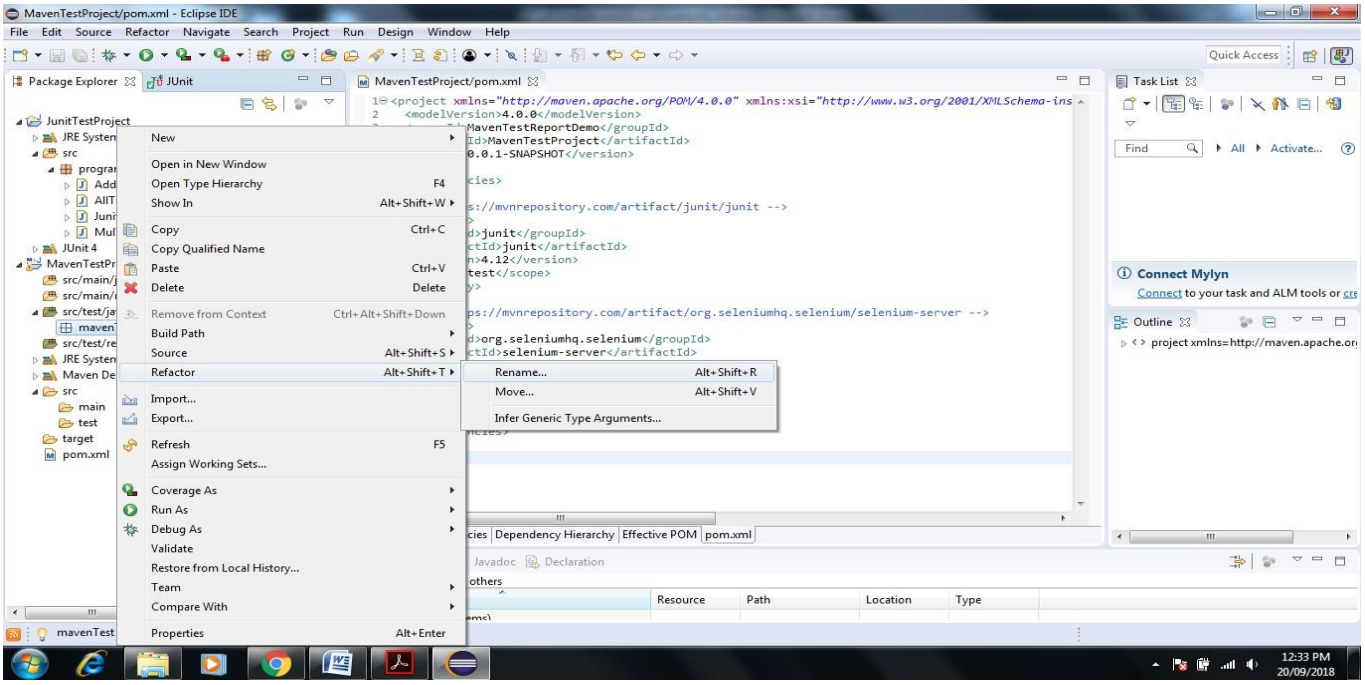
58. Click New->Package



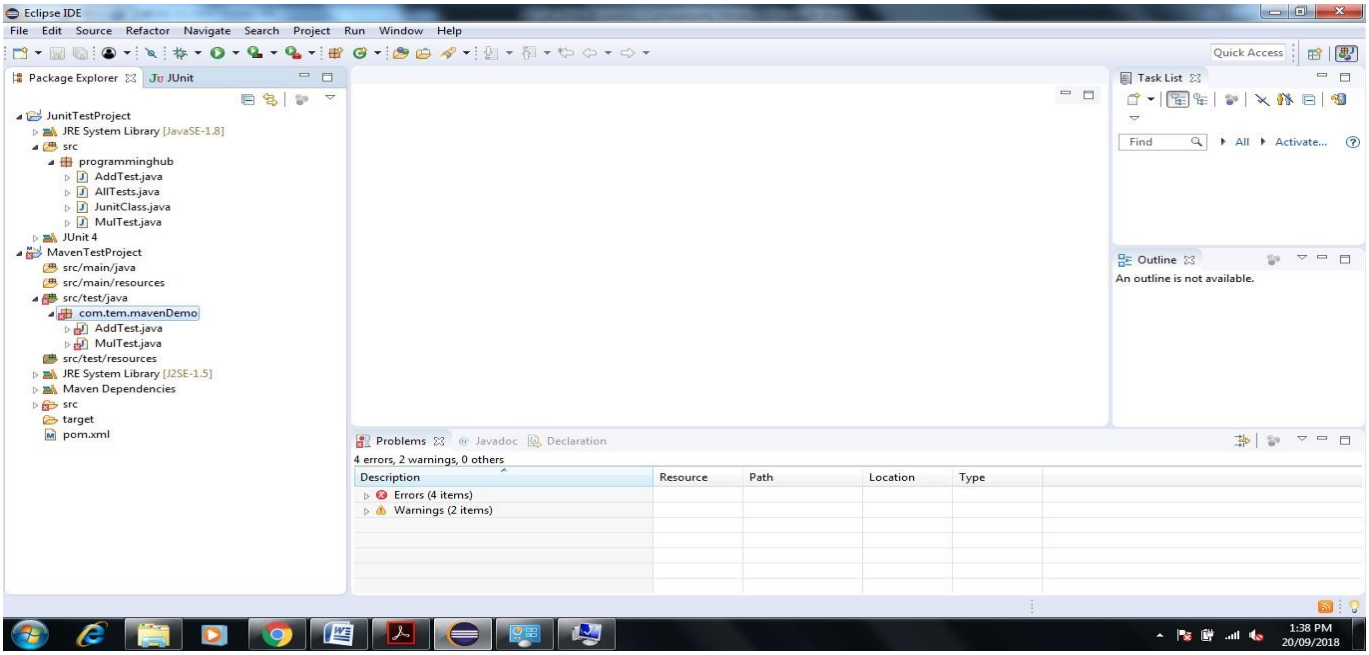
59. Give name to package mavenTest



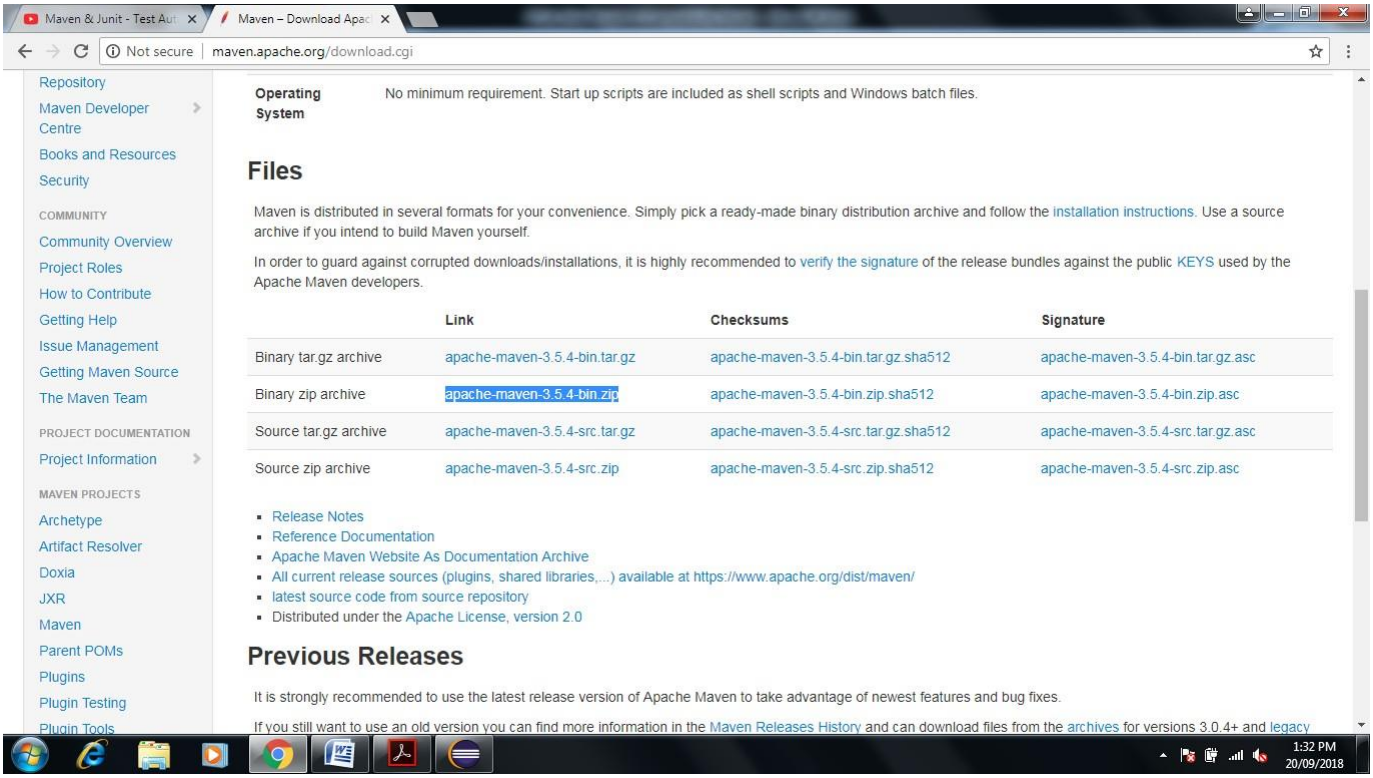
60. It shows the manvenTest Package under src/test/java folder now rename same by right click on mavenTest Click on Refactor->Rename->give another name com.tem.mavenDemo->Click on ok



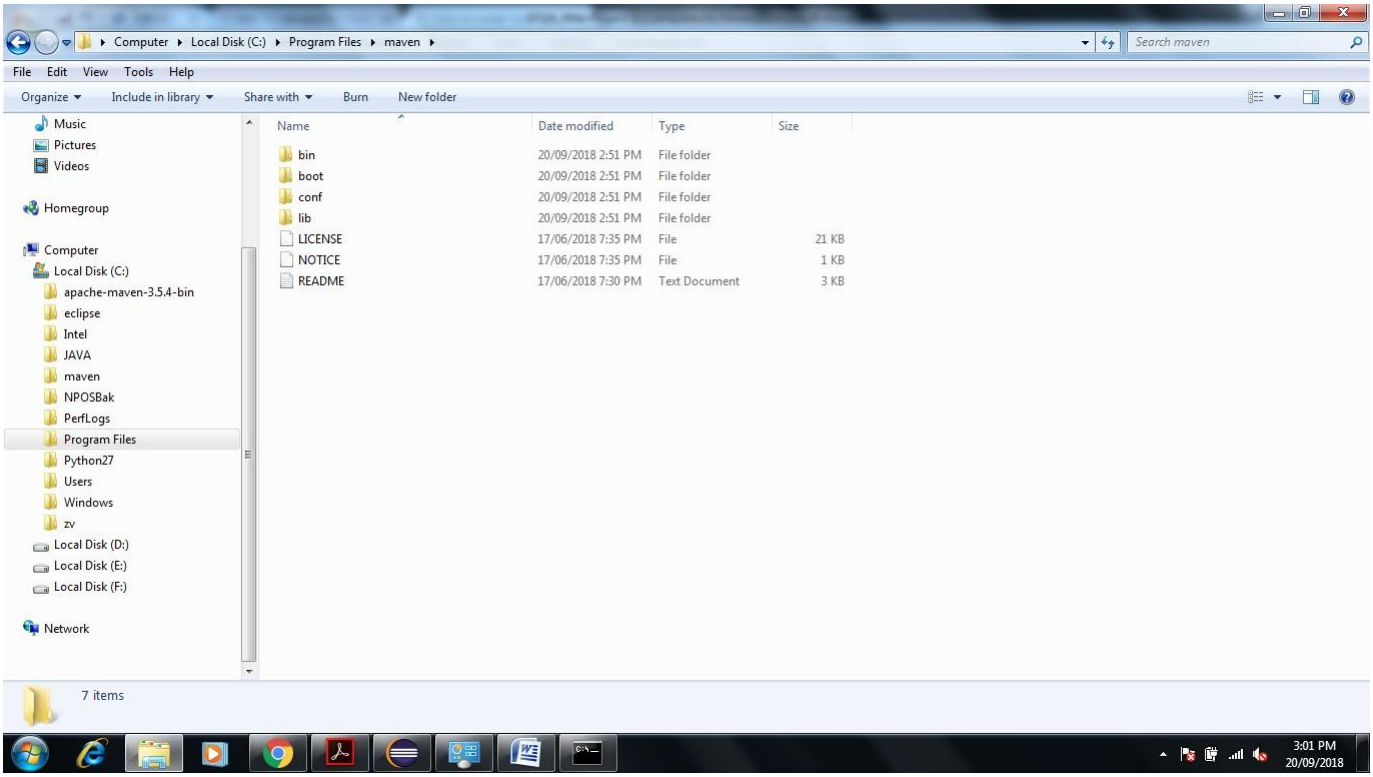
61. rename as com.tem.mavenDemo



62. Download Apache Maven Select that binary apache-maven-3.5.4-bin



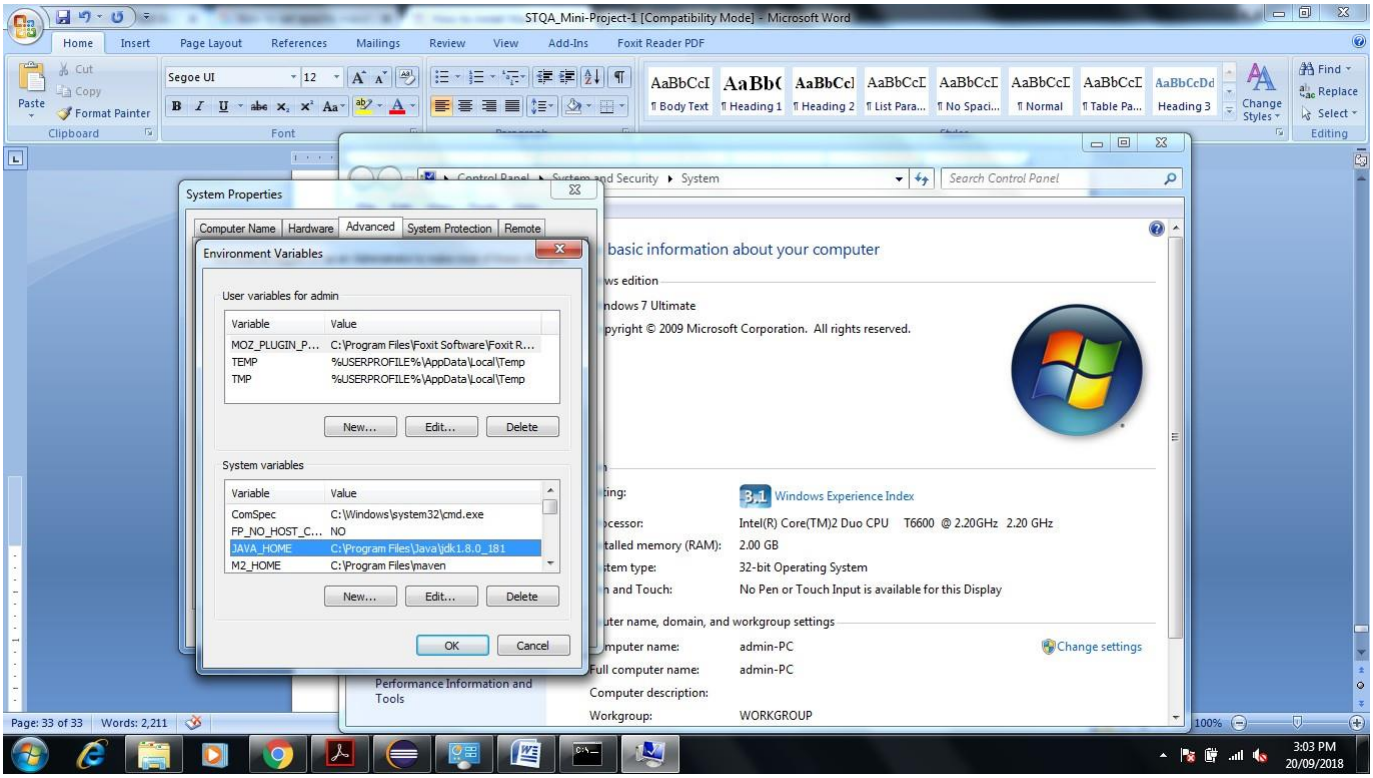
63. after Download->go to Program File->create one folder give name maven-> now extract the downloaded file in maven folder



64. Environment Setup Very Important Steps to Generate Report

1. **JDK and JAVA_HOME**

Make sure JDK is installed, and "**JAVA_HOME**" variable is added as Windows environment variable. Our JDK installed in Program File ->JAVA->JDK 1.8.0



2. Set Path of Add **M2_HOME** and **MAVEN_HOME**
   Create new system variable name M2_HOME and **MAVEN_HOME separately set variable value➐ C:\Program Files\maven**

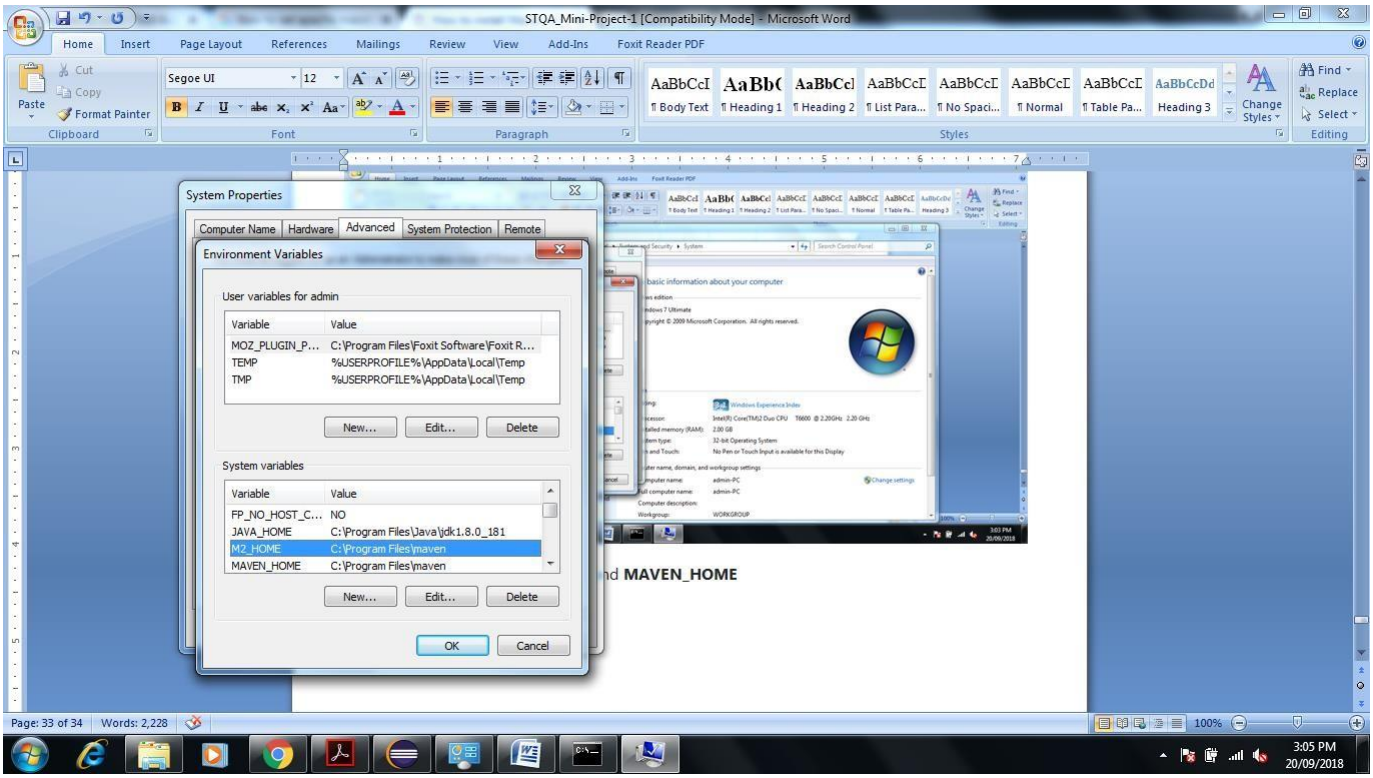


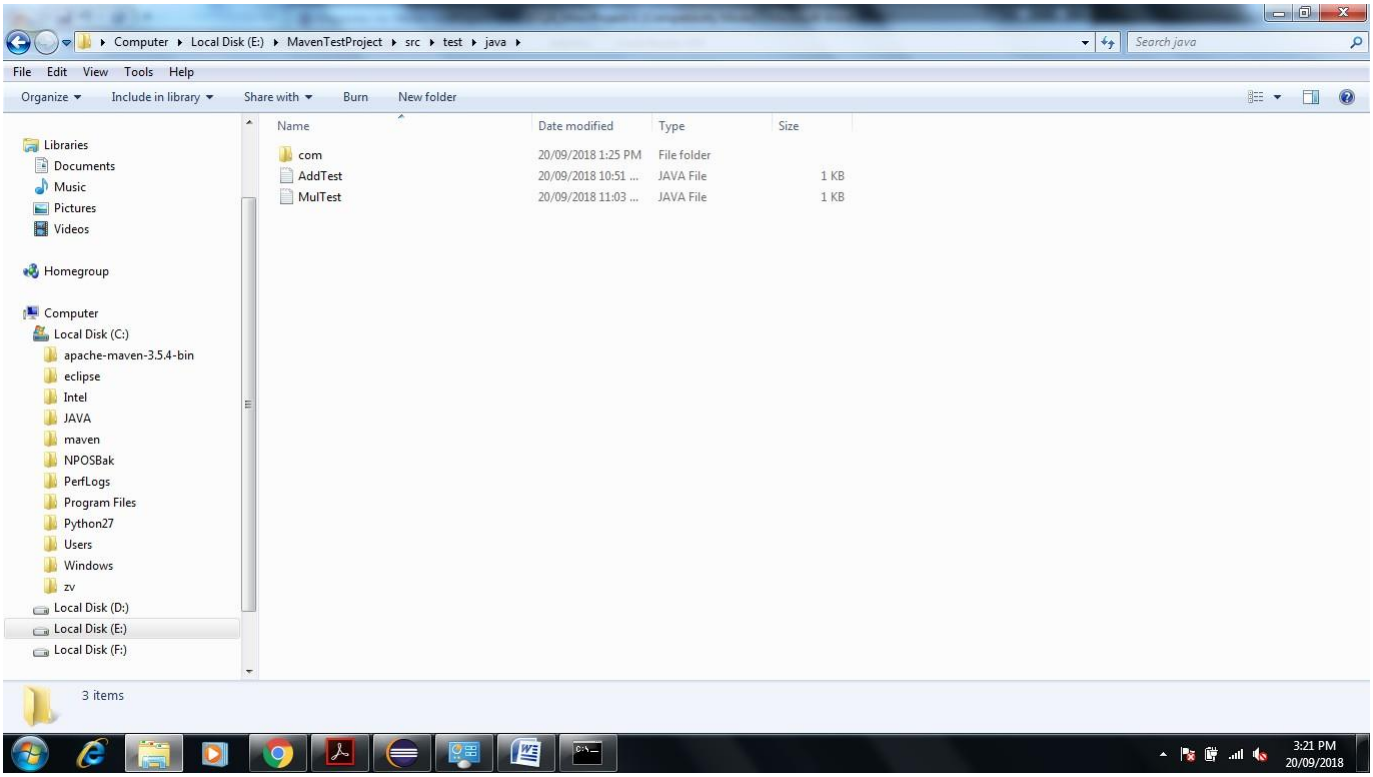
Figure Shows the Path of M2_HOME & MAVEN_HOME same.

3. **Update PATH Variable as per folowing**

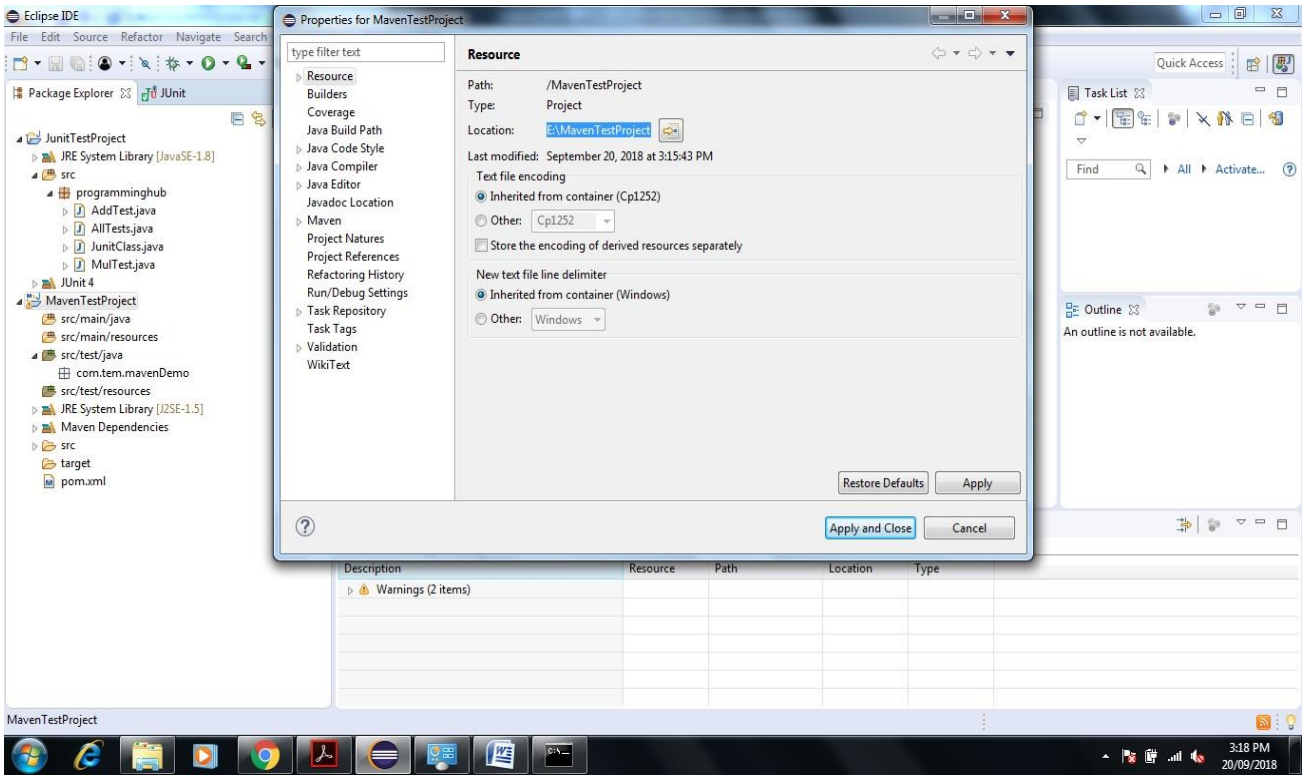
C:\Program Files\maven%MAVEN_HOME%\bin;%M2_HOME%\bin;

4. Verification

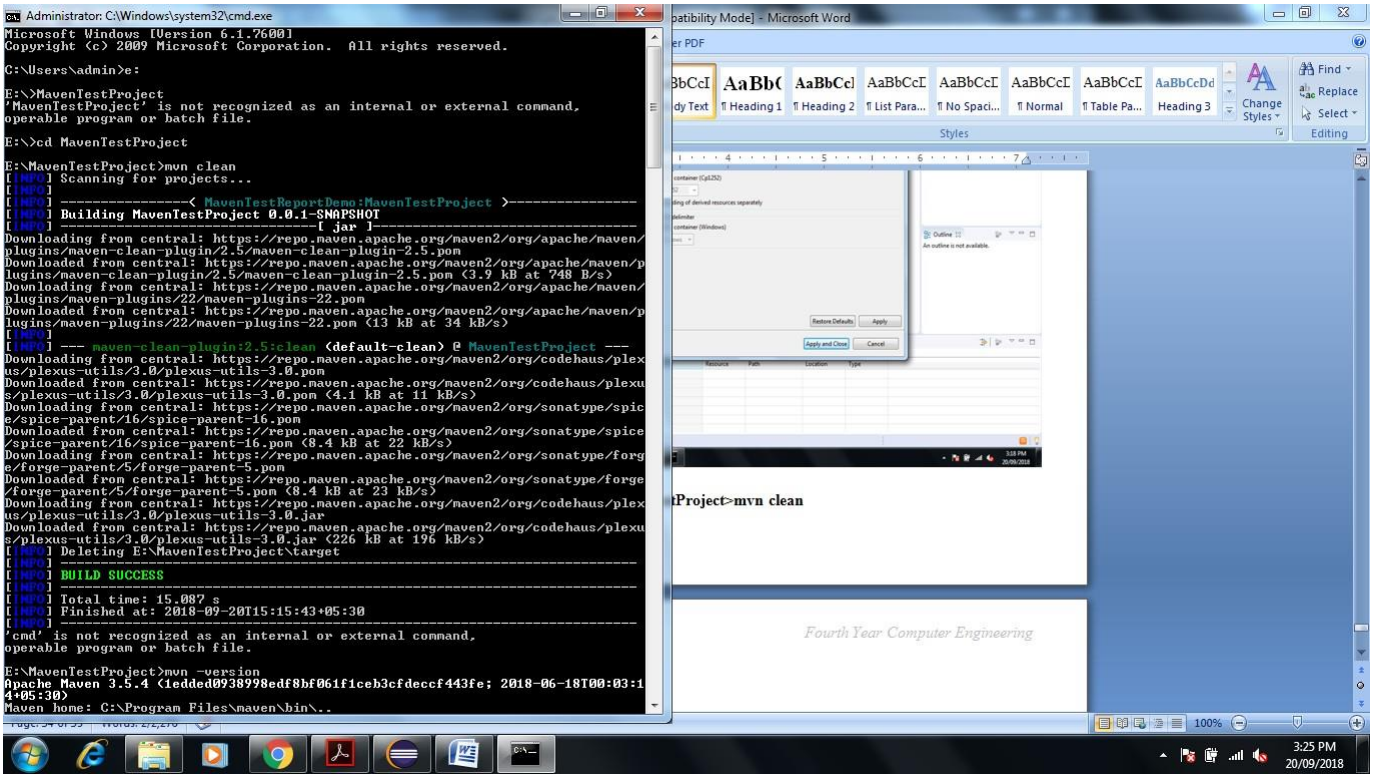
Now copy Previous Created JUnit Test case java file Add Test and Mul Test Paste Externally in E:\MavenTestProject\src\test\java



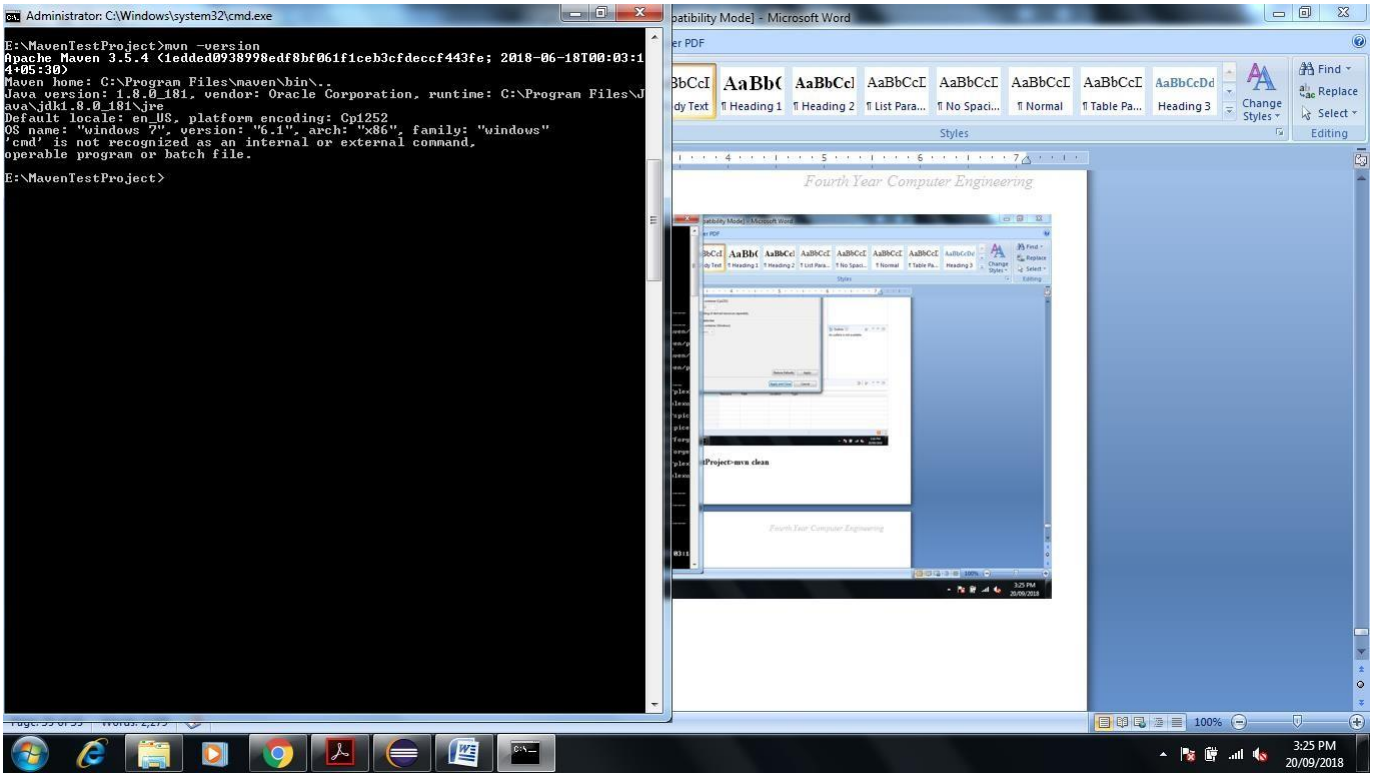
Now Open Eclipse Right Click on MavenTestProject->Properties➐Resources➐Copy Path Project Folder



Now go to command prompt➐E:\MavenTestProject>mvn clean



Enter E:\MavenTestProject>mvn –version



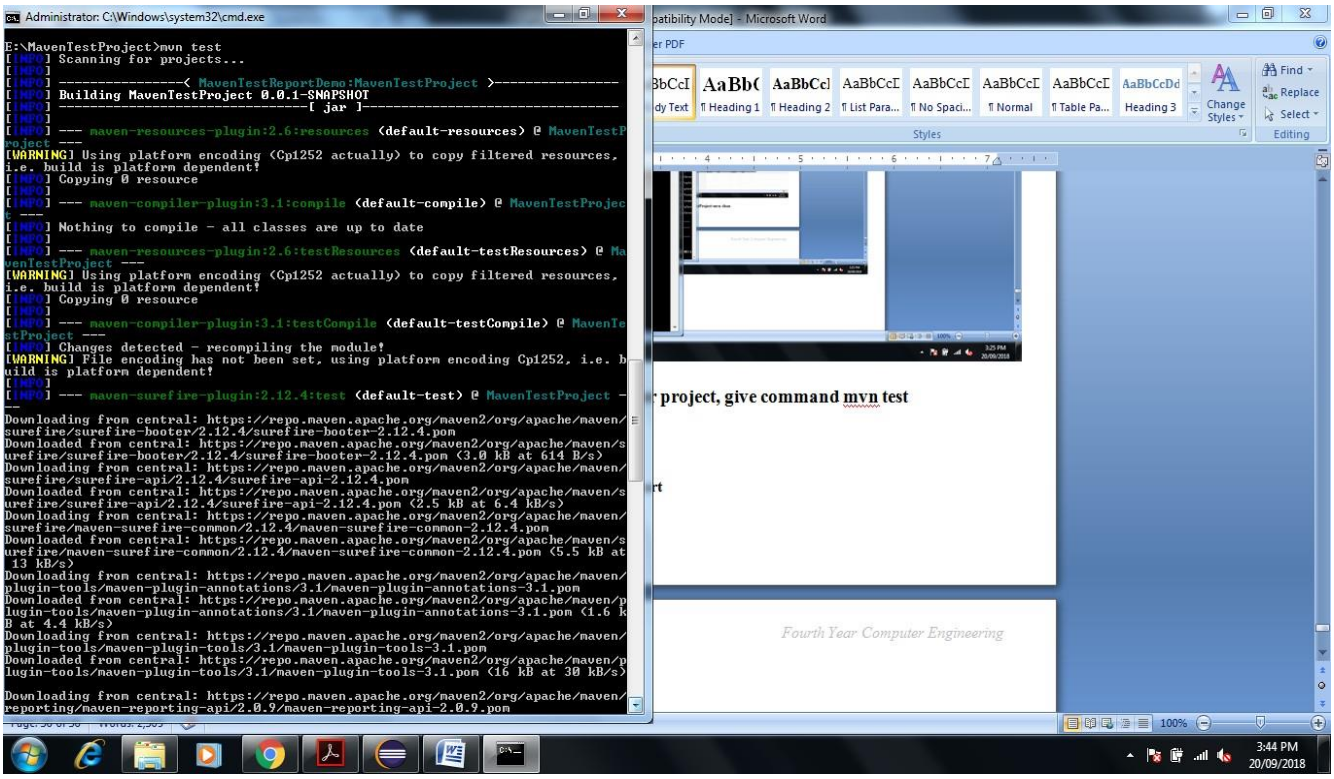
To run test suite or all test cases under project, give command mvn test

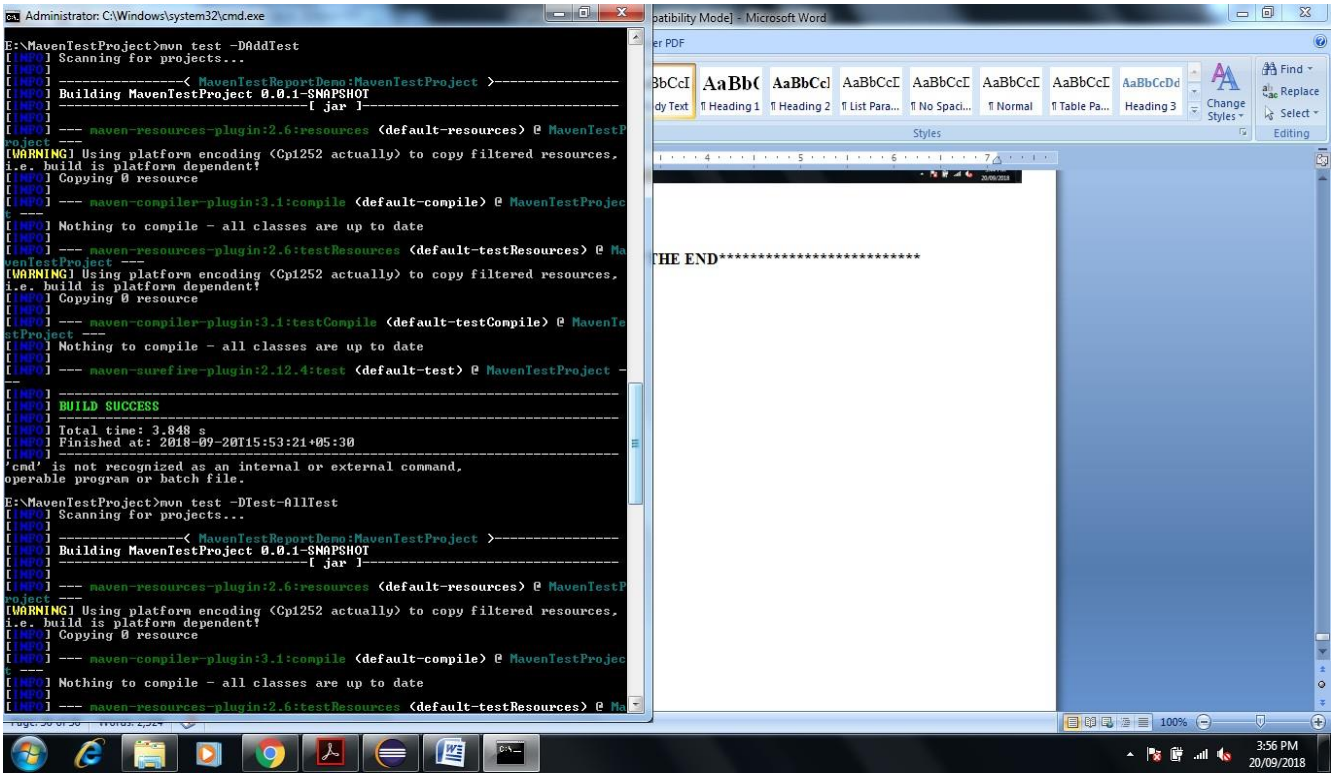
Enter E:\MavenTestProject>mvn test

This Command is Used to See the Test Report

You can even run individual test cases. Give command mvn test –Dtesttestcasename

Eg. mvn test –Dtest-AllTest





Conclusion

In this way using JUnit and Maven Automation tool we are Perform Unit Testing and Prepare Test Report of same.

Assignment Question

1. Write any Five Tool for White Box and Black Box Testing Purpose.