**EXPERIMENT # 4**

**JUnit – Unit Testing Framework**

**JUnit** is a [unit testing](http://en.wikipedia.org/wiki/Unit_testing) [framework](http://en.wikipedia.org/wiki/Software_framework) for the [Java programming language](http://en.wikipedia.org/wiki/Java_(programming_language)). JUnit has been important in the development of [test-driven development](http://en.wikipedia.org/wiki/Test-driven_development), and is one of a family of [unit testing](http://en.wikipedia.org/wiki/Unit_testing) frameworks collectively known as [xUnit](http://en.wikipedia.org/wiki/XUnit" \o "XUnit) that originated with [SUnit](http://en.wikipedia.org/wiki/SUnit" \o "SUnit).

JUnit is linked as a [JAR](http://en.wikipedia.org/wiki/JAR_(file_format)) at compile-time; the framework resides under packages junit.framework for JUnit 3.8 and earlier and under org.junitfor JUnit 4 and later.

**1. Introduction to unit testing**

**1.1. Unit testing**

A unit test is a piece of code written by a developer that executes a specific functionality in the code under test. Unit tests ensure that code is working as intended and validate that this is still the case after code changes.

**1.2. Unit testing with JUnit**

*JUnit* 4.x is a test framework which uses annotations to identify methods that are test methods. JUnit assumes that all test methods can be executed in an arbitrary order. Therefore tests should not depend on other tests.

**To write a test with JUnit**

* Annotate a method with @org.junit.Test
* Use a method provided by JUnit to check the expected result of the code execution versus the actual result

You can use Eclipse or the org.junit.runner.JUnitCore class to run the test.

**2. JUnit Annotations**

The following table gives an overview of the available annotations in JUnit 4.x.

**Table 1. Annotations**

| **Annotation** | **Description** |
| --- | --- |
| @Test public void method() | The annotation @Test identifies that a method is a test method. |
| @Before public void method() | Will execute the method before each test. This method can prepare the test environment (e.g. read input data, initialize the class). |
| @After public void method() | Will execute the method after each test. This method can cleanup the test environment (e.g. delete temporary data, restore defaults). |
| @BeforeClass public void method() | Will execute the method once, before the start of all tests. This can be used to perform time intensive activities, for example to connect to a database. |
| @AfterClass public void method() | Will execute the method once, after all tests have finished. This can be used to perform clean-up activities, for example to disconnect from a database. |
| @Ignore | Will ignore the test method. This is useful when the underlying code has been changed and the test case has not yet been adapted. Or if the execution time of this test is too long to be included. |
| @Test (expected = Exception.class) | Fails, if the method does not throw the named exception. |
| @Test(timeout=100) | Fails, if the method takes longer than 100 milliseconds. |

**3. Unit Testing of a Sample program using JUnit :**

The following program was implemented & tested using ECLIPSE & JUnit 4.0.

**MyClass.java ---**

package ankit.junit.first;

public class MyClass {

public int multiply(int x, int y) {

     return x \* y;

   }

public int add(int x, int y){

return x + y;

}

}

**MyClassTest.java ----**

package ankit.junit.first;

import static org.junit.Assert.\*;

import org.junit.Test;

public class MyClassTest {

@Test

public void testMultiply() {

MyClass tester = new MyClass();

     assertEquals("Result", 50, tester.multiply(10, 5));

}

@Test

public void testAdd() {

MyClass tester = new MyClass();

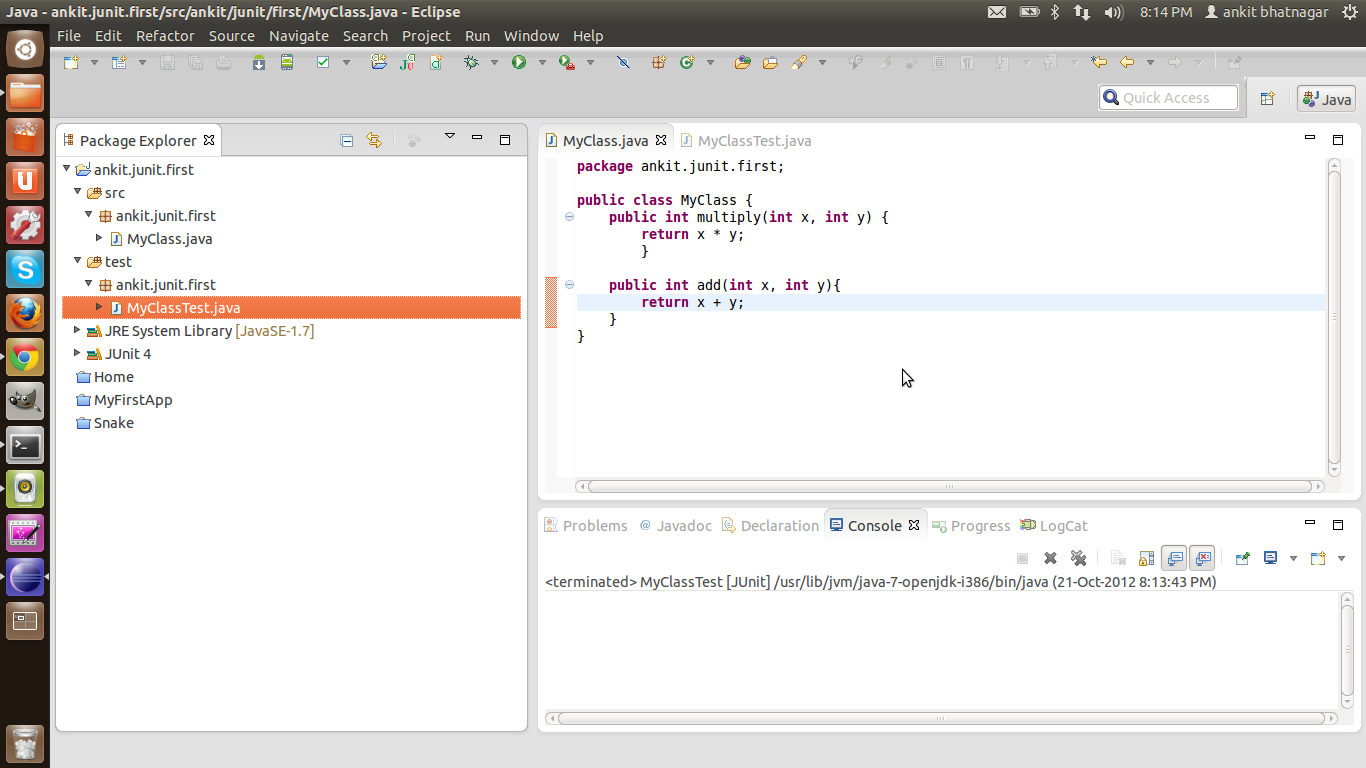
     assertEquals("Result", 15, tester.add(10, 5));

}

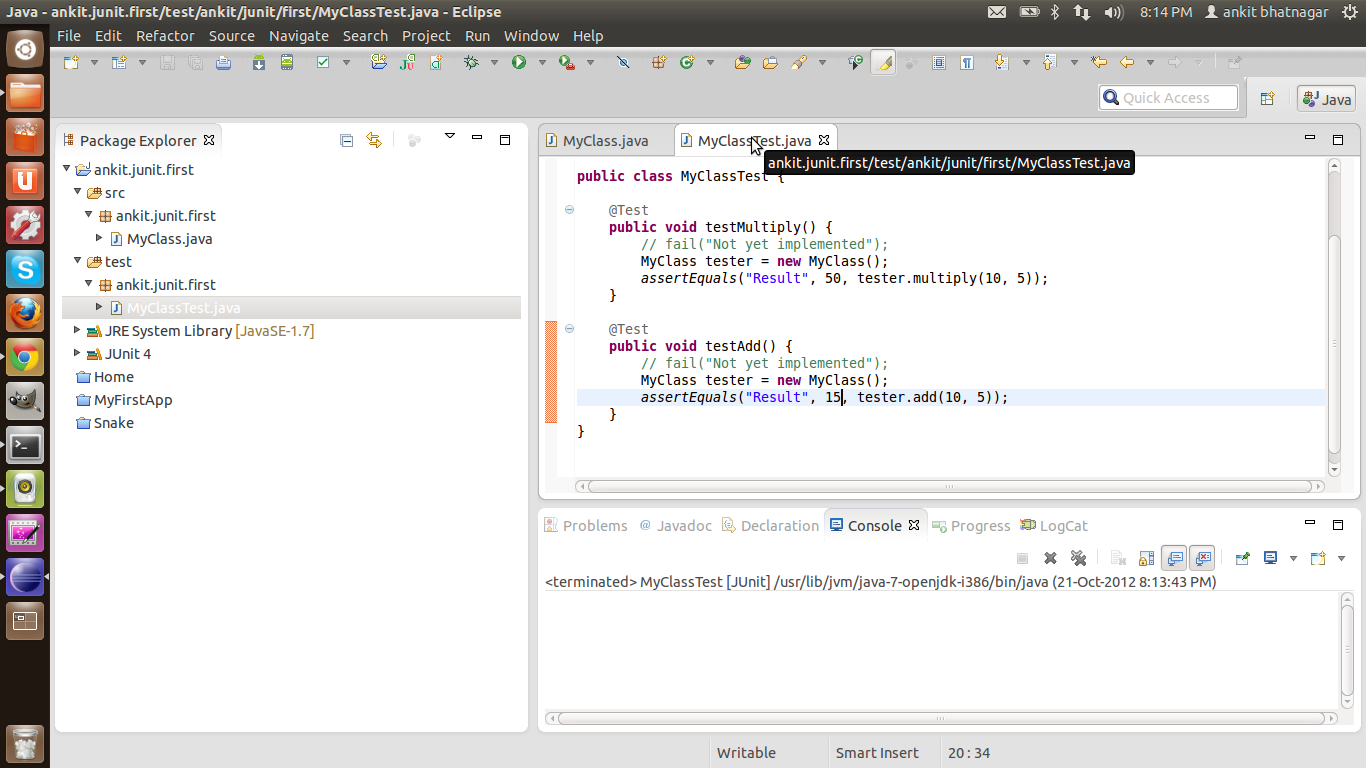
}

**4. Screenshots of the Testing procedure:**

(1) Implementation of MyClass.java --->



(2) Implementation of MyClassTest.java --->



(3) JUnit test results --->

