|  |
| --- |
| Department of Software Engineering  Mehran University of Engineering and Technology, Jamshoro |

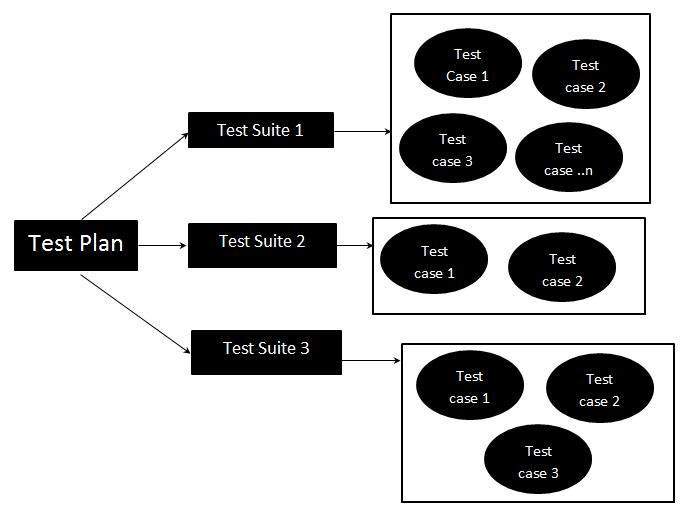
|  |  |  |  |
| --- | --- | --- | --- |
| Course: SW426 - Software Quality Engineering | | | |
| Instructor | Rabia Iftikhar | **Practical/Lab No.** | 06 |
| Date | 18-08-2020 | **CLOs** | CLO-3 |
| Signature |  | **Assessment Score** | 1 Mark |

|  |  |
| --- | --- |
| Topic | Creating test suites using JUnit |
| Objectives | * To learn creating test suites and test runners * To explore Junit framework |

|  |
| --- |
| Lab Discussion: Theoretical concepts and Procedural steps |

**Test Suites**

A test case answers the question: What am I going to test? You develop test cases to define the things that you must validate to ensure that the system is working correctly and is built with a high level of quality. A test suite is a collection of test cases that are grouped for test execution purposes. If each test case represents a piece of a scenario, such as the elements that simulate completing a transaction, use a test suite.



In Junit, test suite allows us to aggregate all test cases from multiple classes in one place and run it together.

To run the suite test, you need to annotate a class using below-mentioned annotations:

1. @Runwith(Suite.class)
2. @SuiteClasses(test1.class,test2.class……) or

@Suite.SuiteClasses ({test1.class, test2.class……})

With above annotations, all the test classes in the suite will start executing one by one.

**Steps to create Test Suite and Test Runner**

**Step 1:** Create a simple test class (e.g. MyFirstClassTest) and add a method annotated with @test.

**import** **static** org.junit.Assert.\*;

**import** org.junit.Test;

**public** **class** MyFirstClassTest {

@Test

**public** **void** myFirstMethod() {

String s= "F16-Software SQE course";

*assertEquals*("F16-Software SQE course", s);

*assertEquals*(2, 1+1);

System.*out*.println(s);

}

}

**Step 2:** Create another test class to add (e.g. MySecondClassTest) and create a method annotated with @test.

import static org.junit.Assert.\*;

import org.junit.Test;

public class MySecondClassTest {

@Test

public void mySecondMethod() {

assertEquals(2, 1);

assertEquals(5, 2+2);

assertEquals(-8, -12+4);

}

}

**Step 3:** To create a testSuite you need to first annotate the class with @RunWith(Suite.class) and @SuiteClasses(class1.class2…..).

import static org.junit.Assert.\*;

import org.junit.Test;

import org.junit.runner.\*;

import org.junit.runners.Suite;

import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)

@SuiteClasses({MyFirstClassTest.class, MySecondClassTest.class})

public class TestSuiteExm {

@Test

public void test() {

}

}

**Step 4:** Create a Test Runner class to run our test suite as given below;

import org.junit.runner.\*;

import org.junit.runner.notification.Failure;

public class TestRunner {

public static void main (String[] args){

Result result= JUnitCore.runClasses(TestSuiteExm.class);

for (Failure failure: result.getFailures()){

System.out.println(failure.toString());

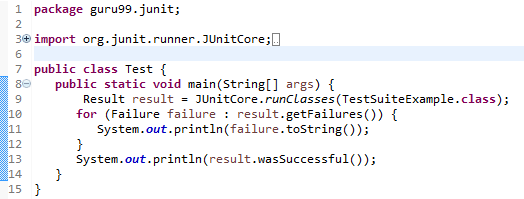
}

System.out.println(result.wasSuccessful());

}

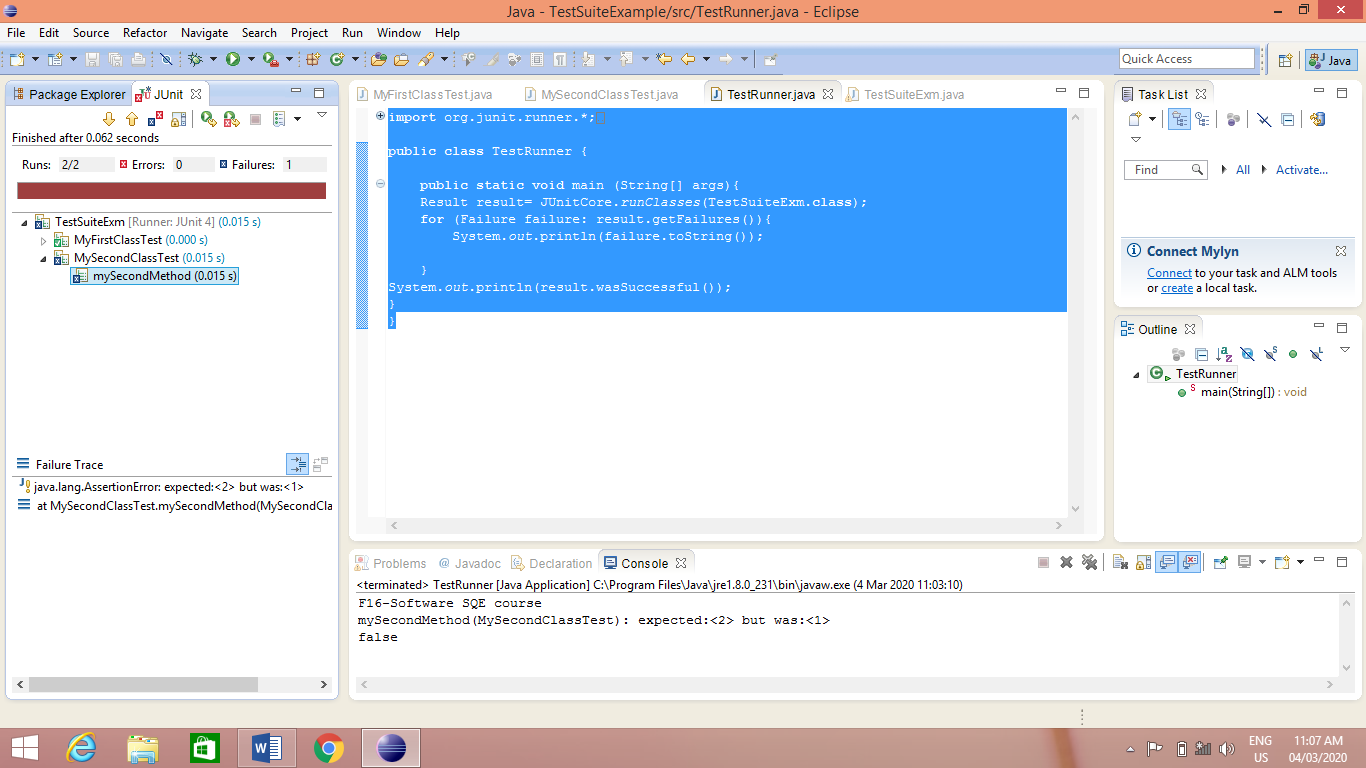
}

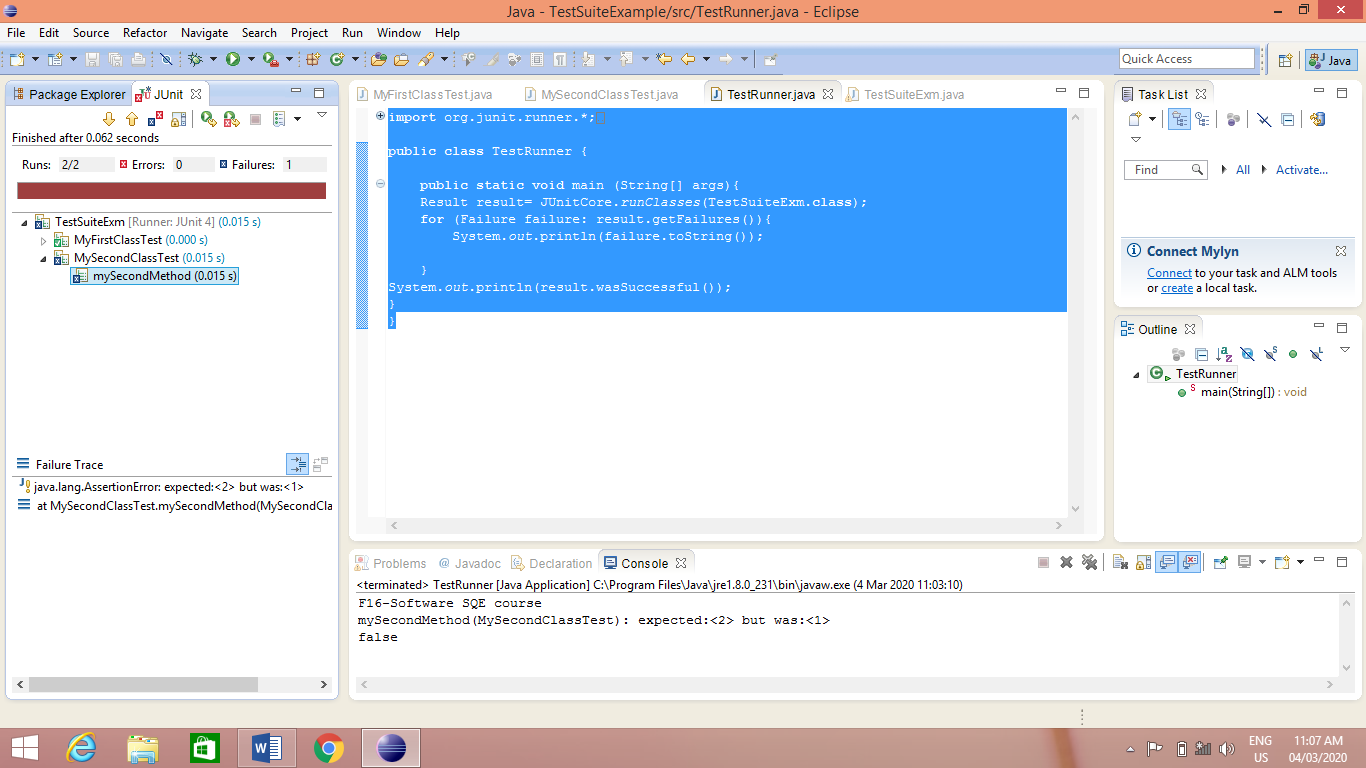
**Code Explanation**

[](https://www.guru99.com/images/junit/052416_0732_CreateJUnit4.png)

* **Code Line 8:**Declaring the main method of the class test which will run our JUnit test.
* **Code Line 9:**Executing test cases using JunitCore.runclasses which takes the testclass name as a parameter (In the example above, you are using TestSuiteExample.class shown in step 3).
* **Code Line 11:**Processing the result using for loop and printing out failed result.
* **Code Line 13:**Printing out the successful result.

**Output:**Here is the output which shows successful test with no failure trace as given below:





|  |
| --- |
| Lab Tasks |

1. Rewrite the above code and observe the results.
2. A bank account class may have

* Methods to open the account, deposit funds, withdraw funds and close the account.
* The states of the account include open with positive balance, open with negative or zero balance and closed.
* How the methods behave depends on the state of the account.
  + An account with a zero or negative balance will not allow the customer to withdraw funds. If positive it might allow customer to go to overdraft once.
  + You could introduce a new method to determine if the account is open or closed and if balance is positive.

Test bank account using test suite.

|  |  |
| --- | --- |
| Lab Tasks Assessment/Rubrics along with Score/Marks | |
| *Rubric Description* | ***Rubric Marks*** |
| 1. Test coverage | 0.5 |
| 1. Test correctness | 0.25 |
| 1. Test completeness | 0.25 |