BSc (Hons) in Information Technology
Year 3

Lab Exercise - JUnit

SE3010 – SEPQM **Semester 1**

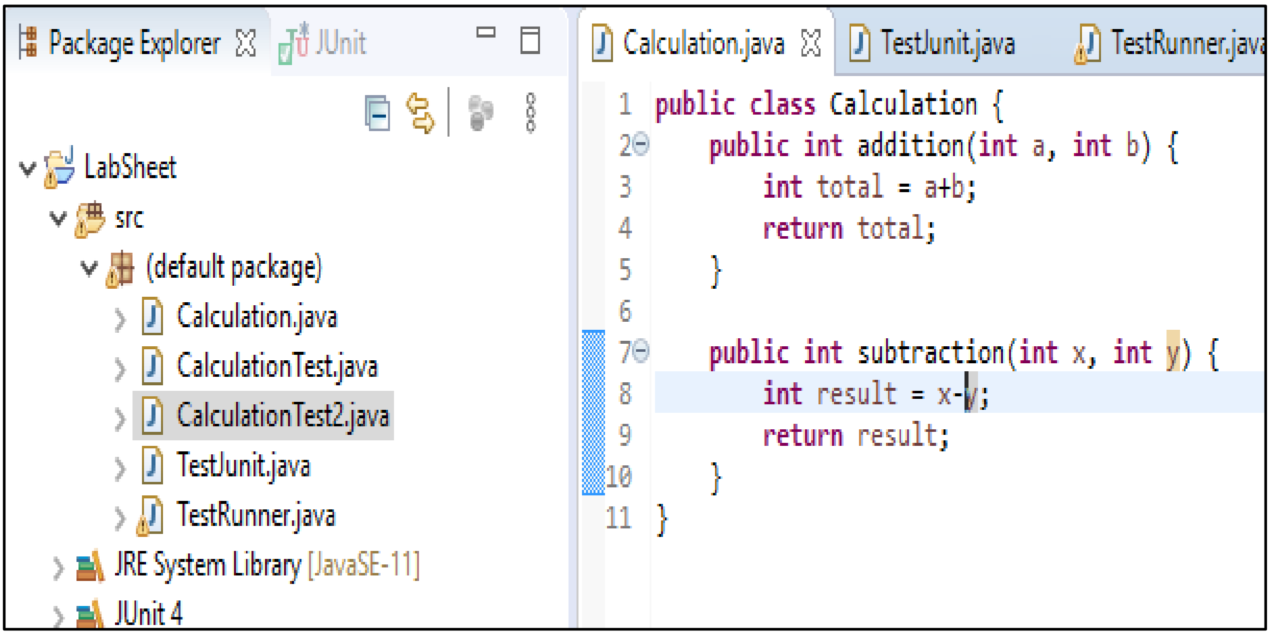
JUnit is an open source unit testing framework for JAVA. It is useful for java developers to write and run repeatable tests. Using JUnit, you can easily and incrementally build a test suite that will help you measure your progress, spot unintended side effects, and focus on your development efforts.

Key JUnit Notions

- Tested Class – The class that is being tested
- Tested Method – The method that is tested
- Test Case – The testing of a class's method against some specified conditions
- Test Case Class – A class performing the test cases
- Test Case Method – A Test Case Class's method implementing a test case
- Test Suite – A collection of test cases that can be tested in a single batch

Step 1:

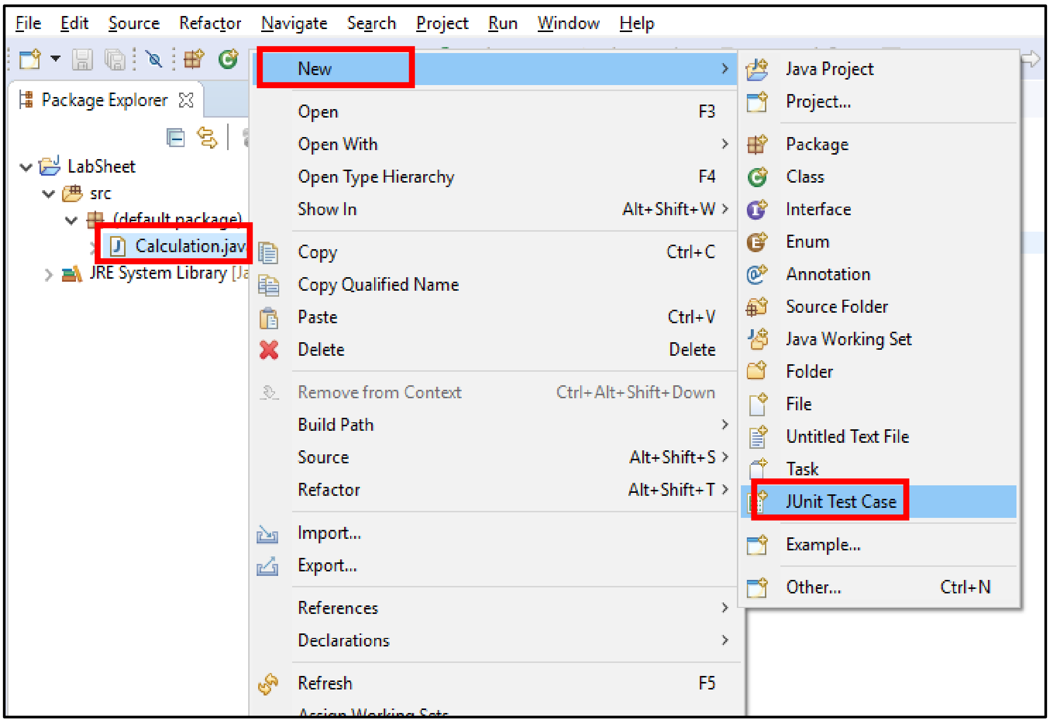
There should be a class to test. Thus, first create a java project and create a class called "Calculation.java" to add two numbers.



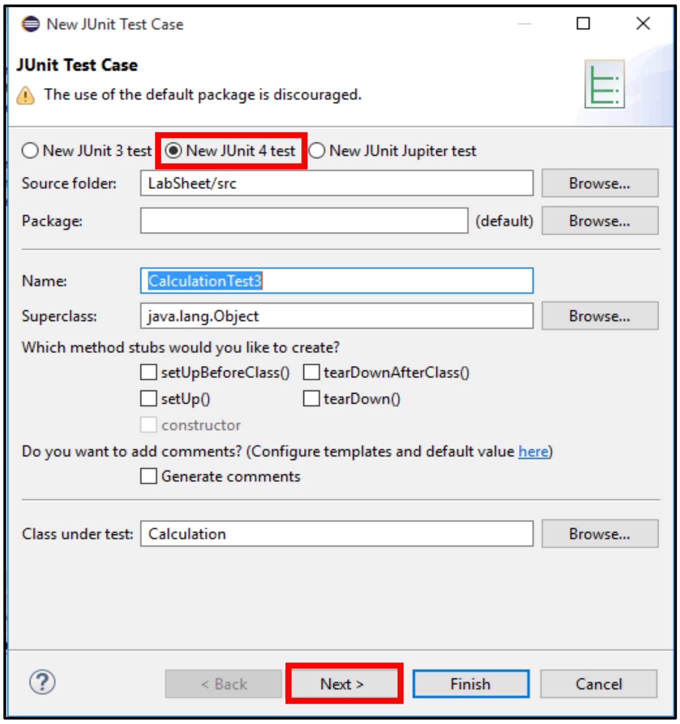
```
public class Calculation {
    public int addition(int a, int b) {
        int total = a+b;
        return total;
    }

    public int subtraction(int x, int y) {
        int result = x-y;
        return result;
    }
}
```

Step 2: Adding a test case to the project

1. Right click on the created class ⟶ Select “New” ⟶ Click “JUnit Test Case”



2. Then you would be navigated to the following window. Select the radio button “New JUnit 4 test” and then click on the “next” button.


New JUnit Test Case

JUnit Test Case

The use of the default package is discouraged.

○ New JUnit 3 test ● New JUnit 4 test ○ New JUnit Jupiter test

Source folder: LabSheet/src Browse...

Package: (default) Browse...

Name: CalculationTest3

Superclass: java.lang.Object Browse...

Which method stubs would you like to create?

□ setUpBeforeClass() □ tearDownAfterClass()

□ setUp() □ tearDown()

□ constructor

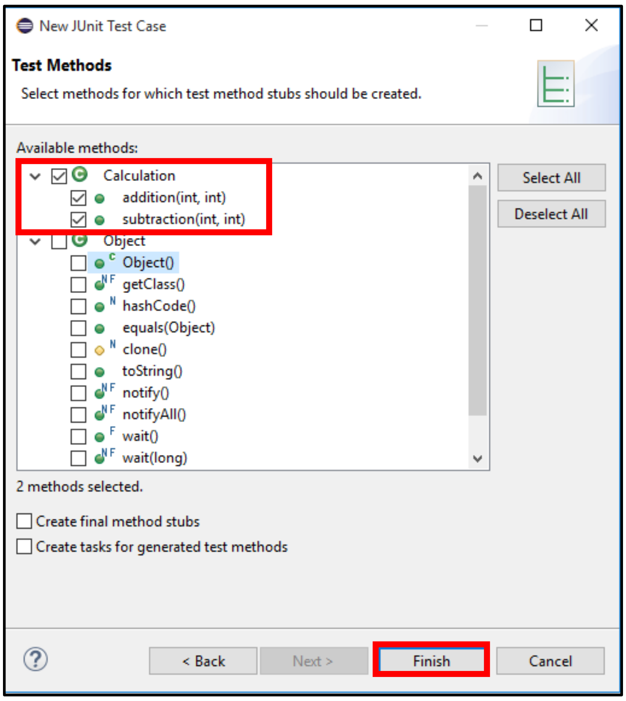
Do you want to add comments? (Configure templates and default value here)

□ Generate comments

Class under test: Calculation Browse...

< Back Next > Finish Cancel

3. Now you would get following window. Select the units (methods) you want to test and click finish.



New JUnit Test Case

Test Methods

Select methods for which test method stubs should be created.

Available methods:

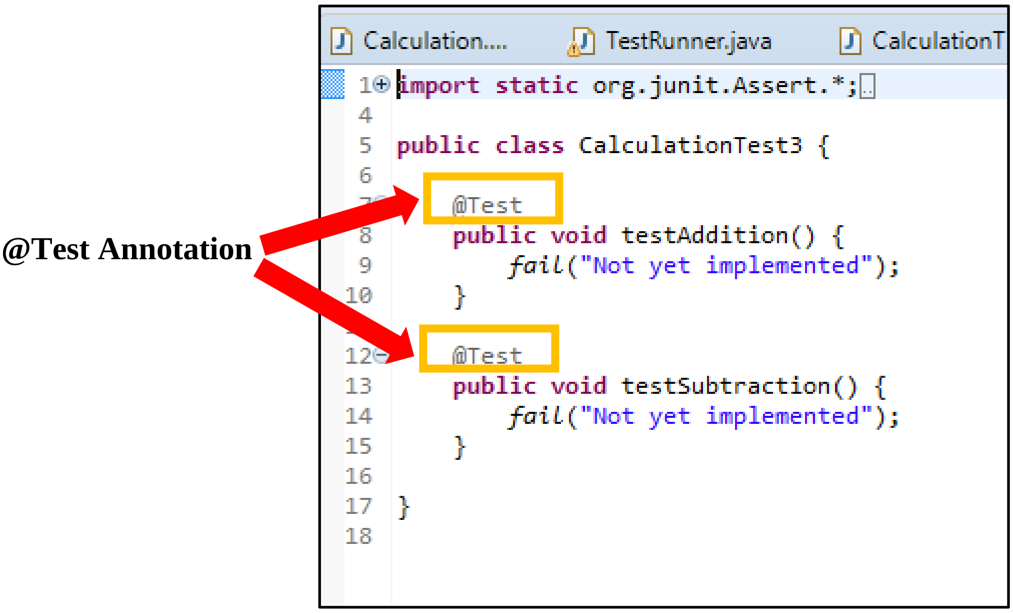
- ☑ Calculation
  - ☑ addition(int, int)
  - ☑ subtraction(int, int)
- □ Object
  - □ Object()
  - □ getClass()
  - □ hashCode()
  - □ equals(Object)
  - □ clone()
  - □ toString()
  - □ notify()
  - □ notifyAll()
  - □ wait()
  - □ wait(long)

2 methods selected.

□ Create final method stubs

□ Create tasks for generated test methods

4. Now you would be able to see the “TestCase class”.



@Test Annotation

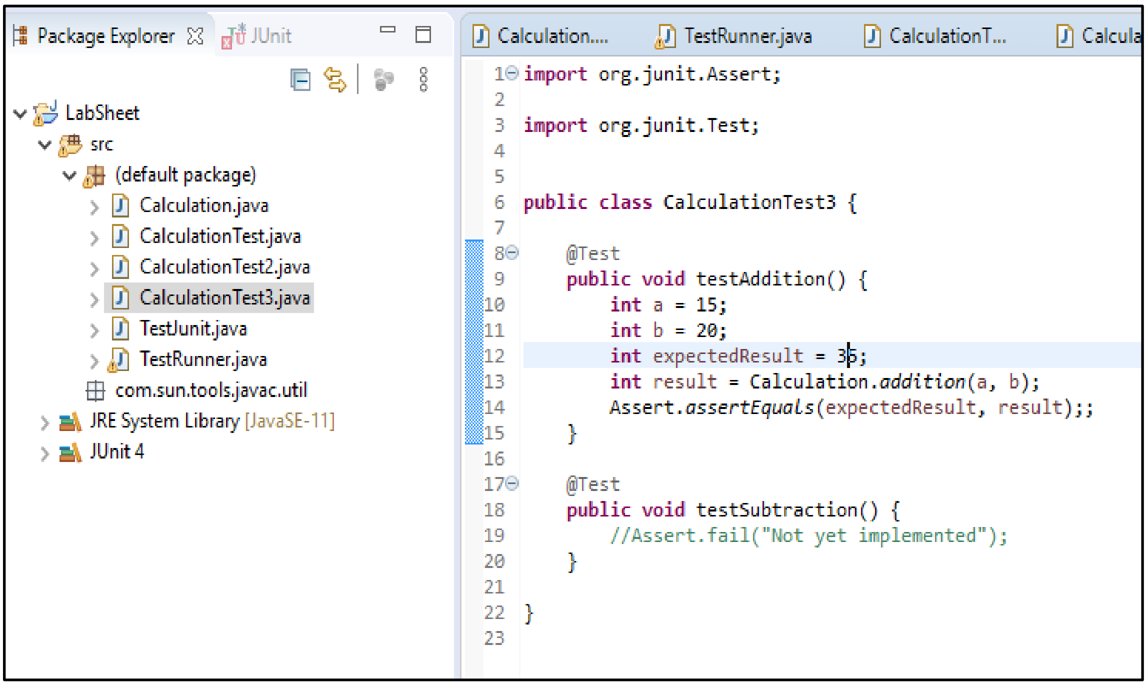
```
import static org.junit.Assert.*;

public class CalculationTest3 {

    @Test
    public void testAddition() {
        fail("Not yet implemented");
    }

    @Test
    public void testSubtraction() {
        fail("Not yet implemented");
    }

}
```

5. **Assertions**

The **org.junit.Assert** class provides a group of assertion methods which are useful for writing tests. Some important methods of the Assert class are follows:

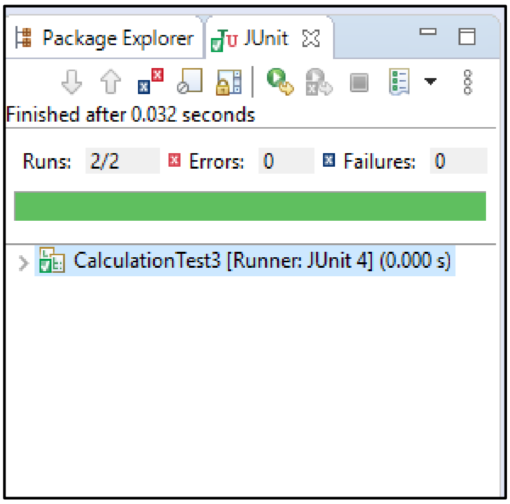
Method	Description
assertEquals(boolean expected, boolean actual)	Is used to test the equality of two primitives/objects.
assertArrayEquals(expectedArray, resultArray)	Is used to test whether two arrays are equal.
assertTrue(boolean condition)	Is used to test whether a condition is true.
assertFalse(boolean condition)	Is used to test whether a condition is false.
assertNull(Object object)	Is used to test whether an object is null.
assertNotNull(Object object)	Is used to test whether an object is not null.
assertSame(object1, object2)	Is used to test whether two object references are pointing to the same object
assertNotSame(object1, object2)	Is used to test whether two object references are not pointing to the same object

5.1. Include the appropriate assert method(s) and complete the code.



```java
import org.junit.Assert;

import org.junit.Test;


public class CalculationTest3 {

    @Test
    public void testAddition() {
        int a = 15;
        int b = 20;
        int expectedResult = 35;
        int result = Calculation.addition(a, b);
        Assert.assertEquals(expectedResult, result);;
    }

    @Test
    public void testSubtraction() {
        //Assert.fail("Not yet implemented");
    }

}
```

5.2. Next, right click on your test case class and run it as a JUnit test. Then, you would get a window such as the following:



Finished after 0.032 seconds

Runs: 2/2 Errors: 0 Failures: 0

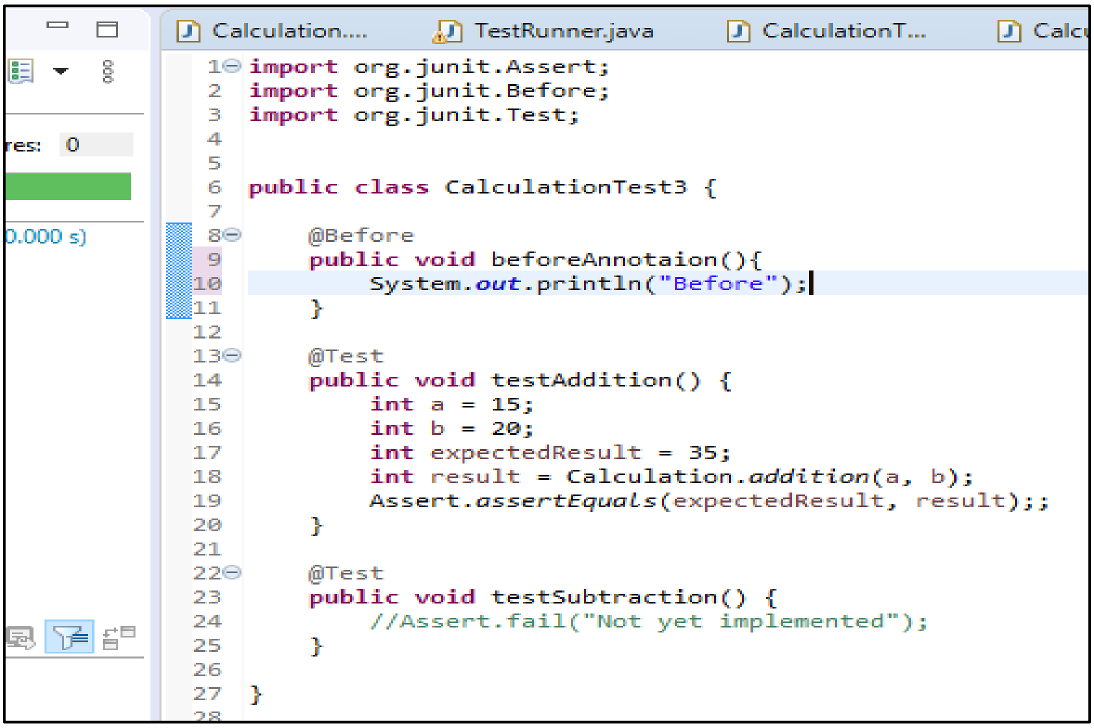
CalculationTest3 [Runner: JUnit 4] (0.000 s)

6. **Annotations:** Annotations were introduced in Junit4. They are like meta-tags which can be added to a java code for better readability. They provide the following information about test methods:
   - The methods which are going to run before and after test methods
   - The methods which run before and after all the methods
   - The methods or classes that will be ignored during the execution

   Following table provides a list of important and frequently used annotations:

Annotation	Description
@Test	This annotation indicates that public void method to which it is attached can be executed as a test case.
@Before	This annotation can be used to execute some statement(s) such as preconditions before each test case.
@BeforeClass	This annotation can be used to execute some statement(s) before all the test cases.
@After	This annotation can be used to execute some statement(s) after each test case.
@AfterClass	This annotation can be used to execute some statement(s) after all test cases.
@Ignores	This annotation can be used to ignore some statement(s) during test execution.
@Test(timeout=500)	This annotation can be used to specify a time limit for the execution of a test.

6.1. Add the @Before annotation to your code and run it.

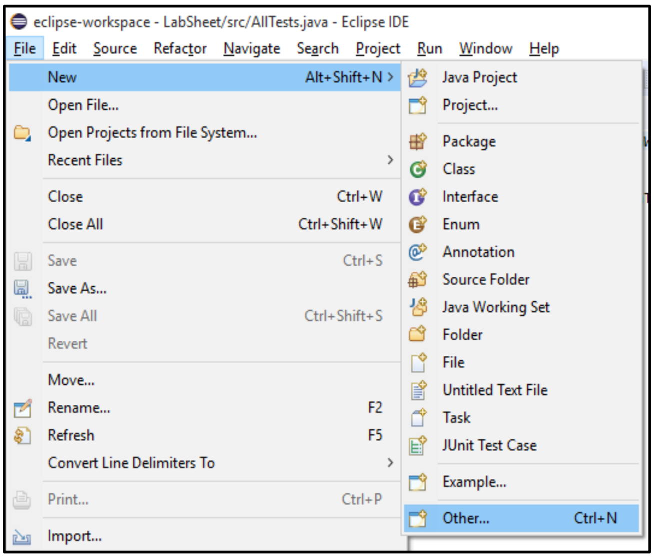


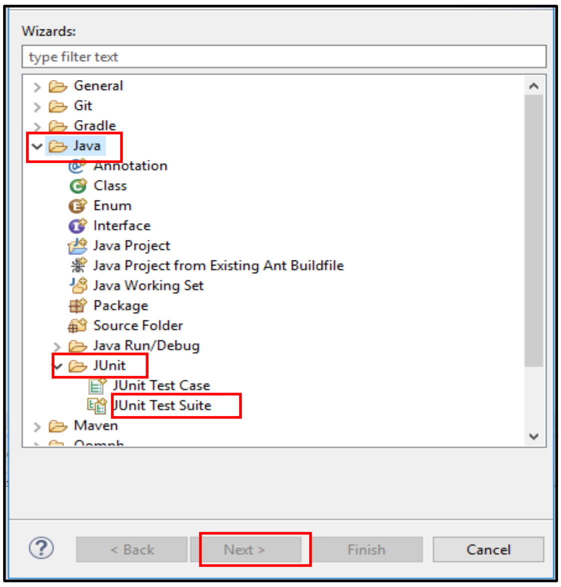
```
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;


public class CalculationTest3 {

    @Before
    public void beforeAnnotaion(){
        System.out.println("Before");
    }

    @Test
    public void testAddition() {
        int a = 15;
        int b = 20;
        int expectedResult = 35;
        int result = Calculation.addition(a, b);
        Assert.assertEquals(expectedResult, result);;
    }

    @Test
    public void testSubtraction() {
        //Assert.fail("Not yet implemented");
    }

}
```

Step 3: Test Suite

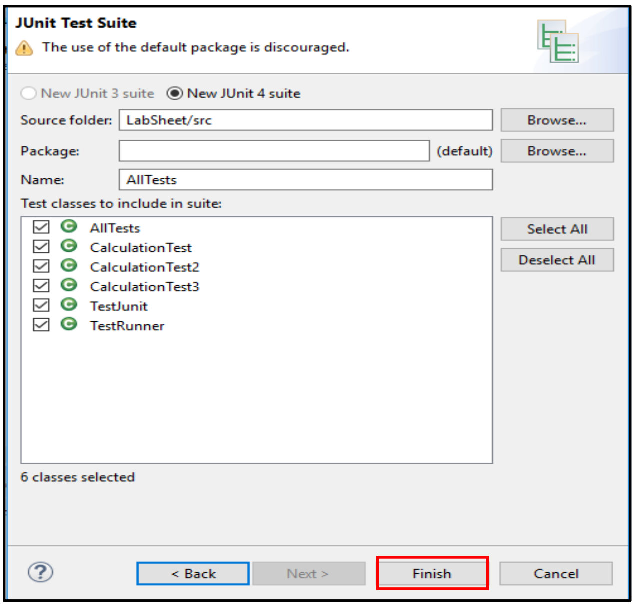
We have performed tests on only one class, i.e. we have tested methods under the consideration that they belong to the same class. However, in large projects there are many classes with methods that should be tested. For testing multiple classes, Eclipse and JUnit introduced the concept of Test Suit. A Test Suite is a simple way running a collection of test cases as a single batch. To open a Test Suite, follow the following steps:

1. Select File ⟶ New ⟶ Other... ⟶ Java ⟶ JUnit ⟶ JUnit Test Suite ⟶ Next





2. Next, select which test classes that you wish to test together and click finish.


JUnit Test Suite

The use of the default package is discouraged.

○ New JUnit 3 suite ● New JUnit 4 suite

Source folder: LabSheet/src

Package: (default)

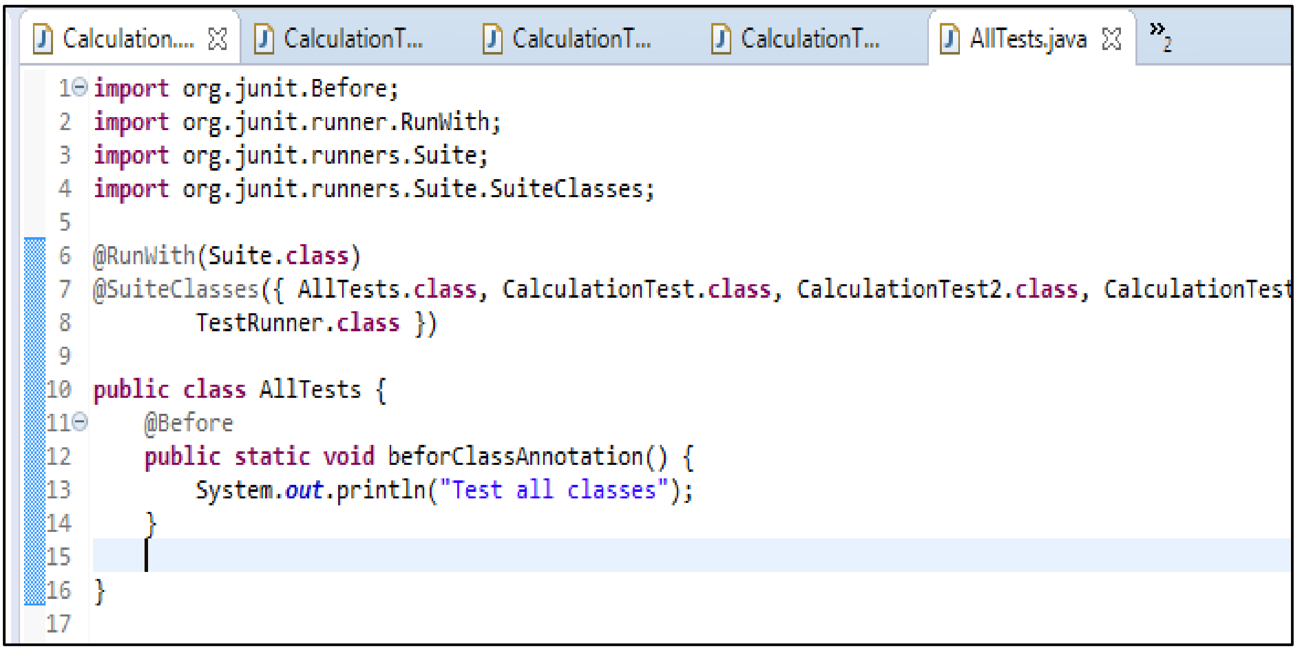
Name: AllTests

Test classes to include in suite:

☑ AllTests
☑ CalculationTest
☑ CalculationTest2
☑ CalculationTest3
☑ TestJunit
☑ TestRunner

6 classes selected

3. Then, you would get the code related to the test suite, which can be altered according to your needs.



```java
import org.junit.Before;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ AllTests.class, CalculationTest.class, CalculationTest2.class, CalculationTest
        TestRunner.class })

public class AllTests {
    @Before
    public static void beforClassAnnotation() {
        System.out.println("Test all classes");
    }
    
}

```