Orchard | Mindtree



**JUNIT TESTING**

By

Himaja Dhanyamraju (M1044292)

**What is JUnit?**

**JUnit** is an open source Unit Testing Framework for JAVA that offers standardized testing. It is useful for Java Developers to write and run repeatable tests. It is an instance of xUnit architecture. As the name implies, it is used for Unit Testing of a small chunk of code.

Developers who are following test-driven methodology must write and execute unit test first before any code.

Once you are done with code, you should execute all tests, and it should pass. Every time any code is added, you need to re-execute all test cases and makes sure nothing is broken.

**How to Create Junit Test Cases?**

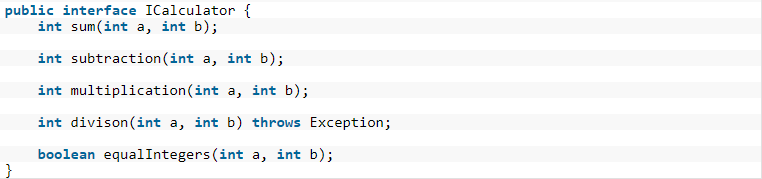
Before starting to do any unit testing using Junit Framework, we need to add some dependencies to our pom file so that it supports different annotations which will be used by us while writing test classes.

Firstly, we need to modify the version of the JUnit library. For this, we just need to modify our **pom.xml** file and change the version from **3.8.11** to **4.11**

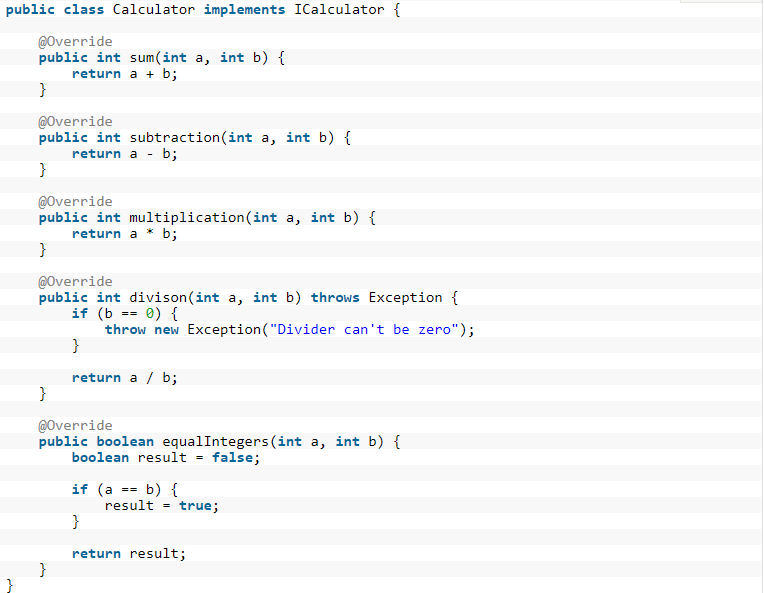
In versions prior to version 4, you needed to extend the junit.framework.TestCase class. JUnit 4 added annotations to the framework and eliminated the need to extend TestCase. You can direct both the lifecycle events and other aspects of the test execution with the provided annotations.

**Example of Junit using Maven:**

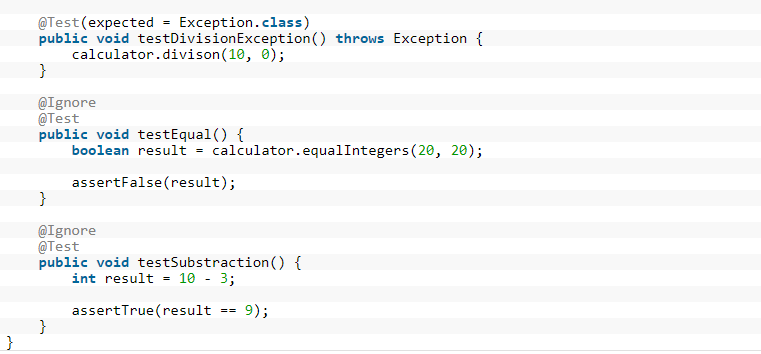
ICalculator.java:



Calculator.java:

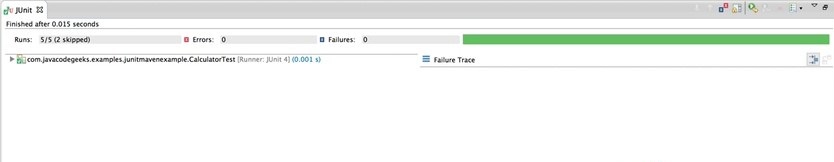


CalculatorTestSuccessful.java:



To execute our tests, we can right click on the name of our class (CalculatorTestSuccessful.java), then click on **Run As** and finally click on **JUnit Test** (notice that our source code, is located inside the folder **test/src/java**)

If we run the above code, we will get the following results:



Output in console:

This is executed before each Test

This is executed after each Test

This is executed before each Test

This is executed after each Test

This is executed before each Test

This is executed after each Test

**Junit Assert Methods**

Assert is a method useful in determining Pass or Fail status of a test case, the assert methods are provided by the class org.junit.Assert which extends java.lang.Object class.

There are various types of assertions like Boolean, Null, Identical etc.

Junit provides a class named Assert, which provides a bunch of assertion methods useful in writing test cases and to detect test failure. They all are static methods defined in org.junit.Assert.\* class. So we can import them in our class and use them directly.

Different types of Assert methods which are mostly used:

**fail()** : fails the test

**assertNotNull(obj) / assertNull(obj)** : assert if object is (not) null

**assertFalse(b) / assertTrue(b)** : asserts if b is false / true

**assertEquals(a,b)** : compares two objects using equals() method: a.equals(b)

**assertSame(a,b)**: compares two objects using == operator: a==b

**assertNotSame(obj1, obj2):**The assertNotSame() method tests if two object references do not point to the same object

**assertArrayEquals(a,b)** : compares two arrays a and b

**assertThat(o, condition)**: asserts that o satisfies the condition specified by an instance of **org.hamcrest.Matcher**

**Hamcrest Matcher Framework:**

*Hamcrest* is a framework for software tests. Hamcrest allows checking for conditions in your code via existing matchers classes. It also allows you to define your custom matcher implementations.

To use Hamcrest matchers in JUnit you use the assertThat statement followed by one or several matchers which uses Matchers and will provide a more readable testcode and better failure messages.

To use this there are some core matchers included in junit. You can start with these for basic tests.

If you want to use more matchers you can write them by yourself or use the hamcrest lib.

The following snippets compare pure JUnit 4 assert statements with Hamcrest matchers.

*// JUnit 4 for equals check*

assertEquals(expected, actual);

*// Hamcrest for equals check*

assertThat(actual, is(equalTo(expected)));

*// JUnit 4 for not equals check*

assertNotEquals(expected, actual)

*// Hamcrest for not equals check*

assertThat(actual, is(not(equalTo(expected))));

To use the library for a Maven based project, add the following dependency to your pom file.

<dependency>

<groupId>org.hamcrest</groupId>

<artifactId>hamcrest-library</artifactId>

<version>1.3</version>

<scope>test</scope>

</dependency>

**Different Annotations:**

@BeforeClass will be executed before any tests and only once for all tests in one test case. Next, for each test, methods will be executed in the following order:

Methods annotated with ‘@Before’ 🡺 Methods annotated with ‘@Test’ 🡺 Methods annotated with ‘@After’

**@Test**: This annotation is a replacement of org.junit.TestCase which indicates that public void method to which it is attached can be executed as a test Case.

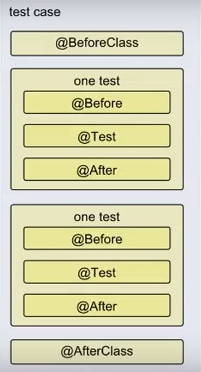
**@Before** annotation is used on a method containing Java code to run before each test case. i.e., it runs before each test execution.

**@After** annotation is used on a method containing java code to run after each test case. These methods will run even if any exceptions are thrown in the test case or in the case of assertion failures.

It is possible to run a method only once for the entire test class before any of the tests are executed, and prior to any **@Before** method(s). "Once only setup" are useful for starting servers, opening communications, etc. It's time-consuming to close and re-open resources for each test. This can be done using the **@BeforeClass** annotation.

Similar to once only setup, a once-only cleanup method is also available. It runs after all test case methods and **@After**annotations have been executed. It is useful for stopping servers, closing communication links, etc. This can be done using the **@AfterClass** annotation

**@Ignores**: This annotation can be used if you want to ignore some statements during test execution for e.g. disabling some test cases during test execution.



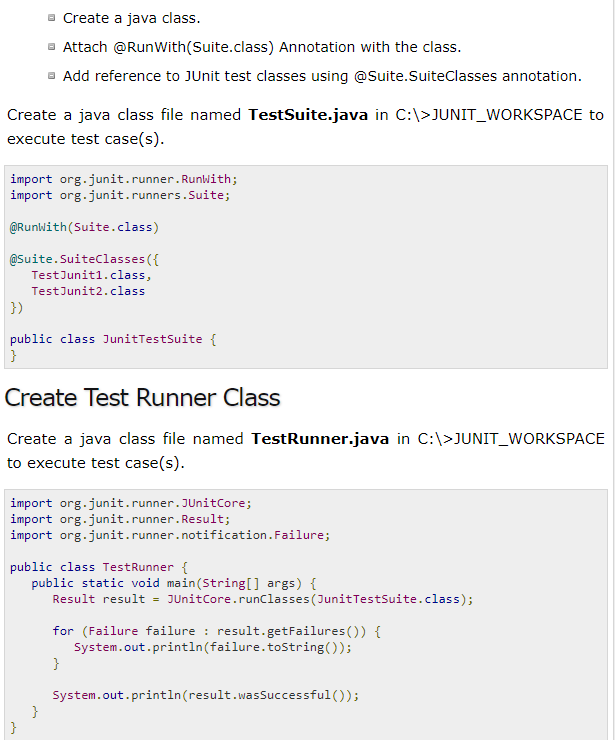
**Test suite** is used to bundle a few unit test cases and run them together. In JUnit, both **@RunWith** and **@Suite** annotations are used to run the suite tests. Here is an example having two test classes, **TestJunit1** & **TestJunit2** that run together using Test Suite:



**Junit Test Runner**

JUnit provides a tool for execution of your test cases.

* **JUnitCore** class is used to execute these tests.
* A method called **runClasses** provided by **org.junit.runner.JUnitCore,** is used to run one or several test classes.
* Return type of this method is the **Result** object (**org.junit.runner.Result**), which is used to access information about the tests. See following code example for more clarity



**Steps in Test Runner Class:**

* Declaring the main method of the class test which will run our JUnit test.
* Executing test cases using JunitCore.runclasses which takes the testclass name as a parameter (In the example above, you are using JunitTestSuite.class).
* Processing the result using for loop and printing out failed result.
* Printing out the successful result.

Now run the Test Runner, which will run the test case defined in the provided Test Case class and verify the output

Output:

Inside testPrintMessage()

Robert

Inside testSalutationMessage()

Hi Robert

true