JUnit Tutorial | Testing Framework for Java

JUnit tutorial provides basic and advanced concepts of **unit testing in java** with examples. Our junit tutorial is designed for beginners and professionals.

It is an *open-source testing framework* for java programmers. The java programmer can create test cases and test his/her own code.

It is one of the unit testing framework. Current version is junit 4.

To perform unit testing, we need to create test cases. The **unit test case** is a code which ensures that the program logic works as expected.

The **org.junit** package contains many interfaces and classes for junit testing such as Assert, Test, Before, After etc.

Types of unit testing

There are two ways to perform unit testing: 1) manual testing 2) automated testing.

1) Manual Testing

If you execute the test cases manually without any tool support, it is known as manual testing. It is time consuming and less reliable.

2) Automated Testing

If you execute the test cases by tool support, it is known as automated testing. It is fast and more reliable.

Annotations for Junit testing

The Junit 4.x framework is annotation based, so let's see the annotations that can be used while writing the test cases.

**@Test** annotation specifies that method is the test method.

**@Test(timeout=1000)** annotation specifies that method will be failed if it takes longer than 1000 milliseconds (1 second).

**@BeforeClass** annotation specifies that method will be invoked only once, before starting all the tests.

**@Before** annotation specifies that method will be invoked before each test.

**@After** annotation specifies that method will be invoked after each test.

**@AfterClass** annotation specifies that method will be invoked only once, after finishing all the tests.

Assert class

The org.junit.Assert class provides methods to assert the program logic.

**Methods of Assert class**

The common methods of Assert class are as follows:

1. **void assertEquals(boolean expected,boolean actual)**: checks that two primitives/objects are equal. It is overloaded.
2. **void assertTrue(boolean condition)**: checks that a condition is true.
3. **void assertFalse(boolean condition)**: checks that a condition is false.
4. **void assertNull(Object obj)**: checks that object is null.
5. **void assertNotNull(Object obj)**: checks that object is not null.

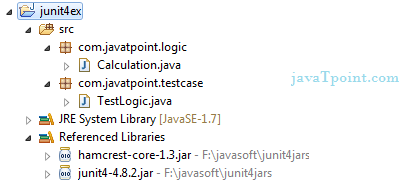
Required jar files

You need to load **junit4.jar** and **hamcrest-core.jar** files.

[download the junit jar files](https://www.javatpoint.com/src/junit/junit4jars.zip)

Simple JUnit example in eclipse IDE

Let's see the directory structure of this example.



Write the program logic

Let's write the logic to find the maximum number for an array.

1. **package** com.javatpoint.logic;
2. **public** **class** Calculation {
4. **public** **static** **int** findMax(**int** arr[]){
5. **int** max=0;
6. **for**(**int** i=1;i<arr.length;i++){
7. **if**(max<arr[i])
8. max=arr[i];
9. }
10. **return** max;
11. }
12. }

Write the test case

Here, we are using JUnit 4, so there is no need to inherit TestCase class. The main testing code is written in the testFindMax() method. But we can also perform some task before and after each test, as you can see in the given program.

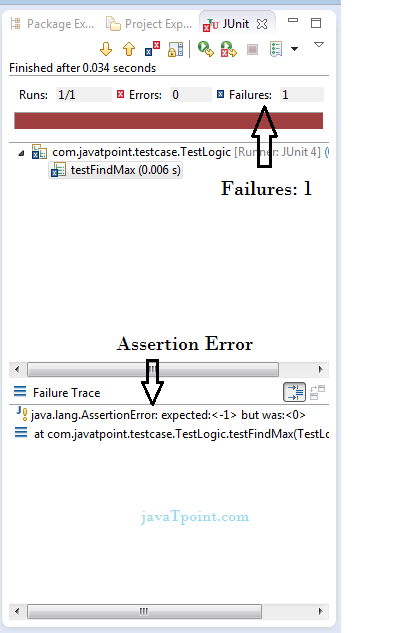
1. **package** com.javatpoint.testcase;
3. **import** **static** org.junit.Assert.\*;
4. **import** com.javatpoint.logic.\*;
5. **import** org.junit.Test;
7. **public** **class** TestLogic {
9. @Test
10. **public** **void** testFindMax(){
11. assertEquals(4,Calculation.findMax(**new** **int**[]{1,3,4,2}));
12. assertEquals(-1,Calculation.findMax(**new** **int**[]{-12,-1,-3,-4,-2}));
13. }
14. }

[download this example](https://www.javatpoint.com/src/junit/junittesting.zip)

To run this example, **right click on TestLogic class -> Run As -> 1Junit Test**.

**Output:**Assertion Error

Let's see the output displayed in eclipse IDE.



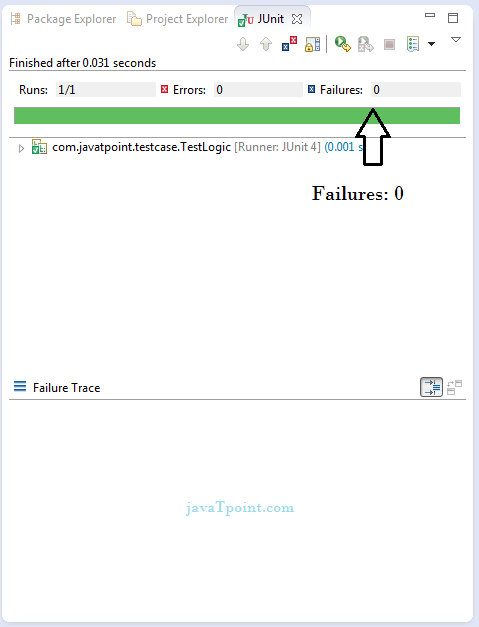
As you can see, when we pass the negative values, it throws AssertionError because second time findMax() method returns 0 instead of -1. It means our program logic is incorrect.

Correct program logic

As you can see, program logic to find the maximum number for the given array is not correct because it doesn't return -1 in case of negative values. The correct program logic is given below:

1. **package** com.javatpoint.logic;
2. **public** **class** Calculation {
4. **public** **static** **int** findMax(**int** arr[]){
5. **int** max=arr[0];//arr[0] instead of 0
6. **for**(**int** i=1;i<arr.length;i++){
7. **if**(max<arr[i])
8. max=arr[i];
9. }
10. **return** max;
11. }
12. }

If you run the junit program again, you will see the following output.



AD

Another example of Junit framework

Write the program code

1. **package** com.javatpoint.logic;
2. **public** **class** Calculation {
3. //method that returns maximum number
4. **public** **static** **int** findMax(**int** arr[]){
5. **int** max=0;
6. **for**(**int** i=1;i<arr.length;i++){
7. **if**(max<arr[i])
8. max=arr[i];
9. }
10. **return** max;
11. }
12. //method that returns cube of the given number
13. **public** **static** **int** cube(**int** n){
14. **return** n\*n\*n;
15. }
16. //method that returns reverse words
17. **public** **static** String reverseWord(String str){
19. StringBuilder result=**new** StringBuilder();
20. StringTokenizer tokenizer=**new** StringTokenizer(str," ");
22. **while**(tokenizer.hasMoreTokens()){
23. StringBuilder sb=**new** StringBuilder();
24. sb.append(tokenizer.nextToken());
25. sb.reverse();
27. result.append(sb);
28. result.append(" ");
29. }
30. **return** result.toString();
31. }
32. }

Write the test case

1. **package** com.javatpoint.testcase;
3. **import** **static** org.junit.Assert.assertEquals;
4. **import** org.junit.After;
5. **import** org.junit.AfterClass;
6. **import** org.junit.Before;
7. **import** org.junit.BeforeClass;
8. **import** org.junit.Test;
9. **import** com.javatpoint.logic.Calculation;
11. **public** **class** TestCase2 {
13. @BeforeClass
14. **public** **static** **void** setUpBeforeClass() **throws** Exception {
15. System.out.println("before class");
16. }
17. @Before
18. **public** **void** setUp() **throws** Exception {
19. System.out.println("before");
20. }
22. @Test
23. **public** **void** testFindMax(){
24. System.out.println("test case find max");
25. assertEquals(4,Calculation.findMax(**new** **int**[]{1,3,4,2}));
26. assertEquals(-2,Calculation.findMax(**new** **int**[]{-12,-3,-4,-2}));
27. }
28. @Test
29. **public** **void** testCube(){
30. System.out.println("test case cube");
31. assertEquals(27,Calculation.cube(3));
32. }
33. @Test
34. **public** **void** testReverseWord(){
35. System.out.println("test case reverse word");
36. assertEquals("ym eman si nahk",Calculation.reverseWord("my name is khan");
37. }
38. @After
39. **public** **void** tearDown() **throws** Exception {
40. System.out.println("after");
41. }
43. @AfterClass
44. **public** **static** **void** tearDownAfterClass() **throws** Exception {
45. System.out.println("after class");
46. }
48. }

**Output:**before class

before

test case find max

after

before

test case cube

after

before

test case reverse word

after

after class