TESTING – JUNIT (JAVA UNIT)

Testing is the process of checking the functionality of an application to ensure it runs as per requirements.

Unit testing comes into picture at the developers’ level; it is the testing of single entity (class or method). Unit testing plays a critical role in helping a software company deliver quality products to its customers.

JUnit is a unit testing framework for Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks collectively known as xUnit, that originated with JUnit.

## **What is a Unit Test Case?**

A Unit Test Case is a part of code, which ensures that another part of code (method) works as expected. To achieve the desired results quickly, a test framework is required. JUnit is a perfect unit test framework for Java programming language.

A formal written unit test case is characterized by a known input and an expected output, which is worked out before the test is executed. The known input should test a precondition and the expected output should test a post-condition.

There must be at least two unit test cases for each requirement − one positive test and one negative test. If a requirement has sub-requirements, each sub-requirement must have at least two test cases as positive and negative.

|  |  |
| --- | --- |
| **Manual Testing** | **Automated Testing** |
| Executing a test cases manually without any tool support is known as manual testing. | Taking tool support and executing the test cases by using an automation tool is known as automation testing. |
| **Time-consuming and tedious** − Since test cases are executed by human resources, it is very slow and tedious. | **Fast** − Automation runs test cases significantly faster than human resources. |
| **Huge investment in human resources** − As test cases need to be executed manually, more testers are required in manual testing. | **Less investment in human resources** − Test cases are executed using automation tools, so less number of testers are required in automation testing. |
| **Less reliable** − Manual testing is less reliable, as it has to account for human errors. | **More reliable** − Automation tests are precise and reliable. |
| **Non-programmable** − No programming can be done to write sophisticated tests to fetch hidden information. | **Programmable** − Testers can program sophisticated tests to bring out hidden information. |

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

Steps to create Junit App

1. Create an application with maven using eclipse
2. Now add Junit Dependency in POM.xml

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.0</version>

<scope>test</scope>

</dependency>

1. Now create a class with collection of methods in “src/main/java”. (Create an application)
   1. Note : **First create package then create a class**
2. Now create a class in “src/test/java”
   1. Note : **First create package then create a class**
3. Now a create a method in that class for testing(usually called as test case). There are few rules to define a method(test case)
   1. Method should be always public
   2. Method return type must be “void”
   3. Method should not have any arguments.
   4. For method name, it is better have java method preceded with “test”
4. Since Junit is annotation based framework, the methods creating in test class should starts with @Test annotation.

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

@Test

public void methodname()

{

Testing functionality

}

**Annotations In JUNIT**

**@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.

Note :- @BeforeClass and @AfterClass methods should be “static”

**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. **assertArrayEquals**
5. **void assertNull(Object obj)**: checks that object is null.
6. **void assertNotNull(Object obj)**: checks that object is not null.

**public** **class** Calculation {

**public** **static** **int** findMax(**int** arr[]){

**int** max=0;

**for**(**int** i=1;i<arr.length;i++){

**if**(max<arr[i])

                max=arr[i];

        }

**return** max;

    }

}

**public** **class** TestLogic {

  @Test

**public** **void** testFindMax(){

        assertEquals(4,Calculation.findMax(**new** **int**[]{1,3,4,2}));

        assertEquals(-1,Calculation.findMax(**new** **int**[]{-12,-1,-3,-4,-2}));

    }

}

Ex:- TestLogic

1. **package** com.javatpoint.logic;
2. **public** **class** Calculation {
3. //method that returns maximum number

**public** **static** **int** findMax(**int** arr[]){

**int** max=0;

**for**(**int** i=1;i<arr.length;i++){

**if**(max<arr[i])

                max=arr[i];

        }

**return** max;

    }

1. //method that returns cube of the given number
2. **public** **static** **int** cube(**int** n){
3. **return** n\*n\*n;
4. }
5. //method that returns reverse words

**public** **static** String reverseWord(String str){

        StringBuilder result=**new** StringBuilder();

        StringTokenizer tokenizer=**new** StringTokenizer(str," ");

**while**(tokenizer.hasMoreTokens()){

        StringBuilder sb=**new** StringBuilder();

        sb.append(tokenizer.nextToken());

        sb.reverse();

        result.append(sb);

        result.append(" ");

        }

**return** result.toString();

1. }
2. }

Ex:- TestCase

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. }
47. }

assertTrue("Both are not same",dm.compareStrings("venu", "VENU"));

public void Performance()

{

for(int i=1;i<=1000000;i++) {}

System.out.println("Loop Done");

}

@Test(timeout=1000) // 1000 milli seconds = 1second

public void PerformanceTest()

{

d1.Performance();

}

public void exception(int n)

{

int div = 10/n;

System.out.println("Result " + div);

}

@Test(expected= ArithmeticException.class)

public void TestException()

{

//Demo dm = new Demo();

dm.exception(0);

}