JUnit Testing

By

Anand Kulkarni

Introduction

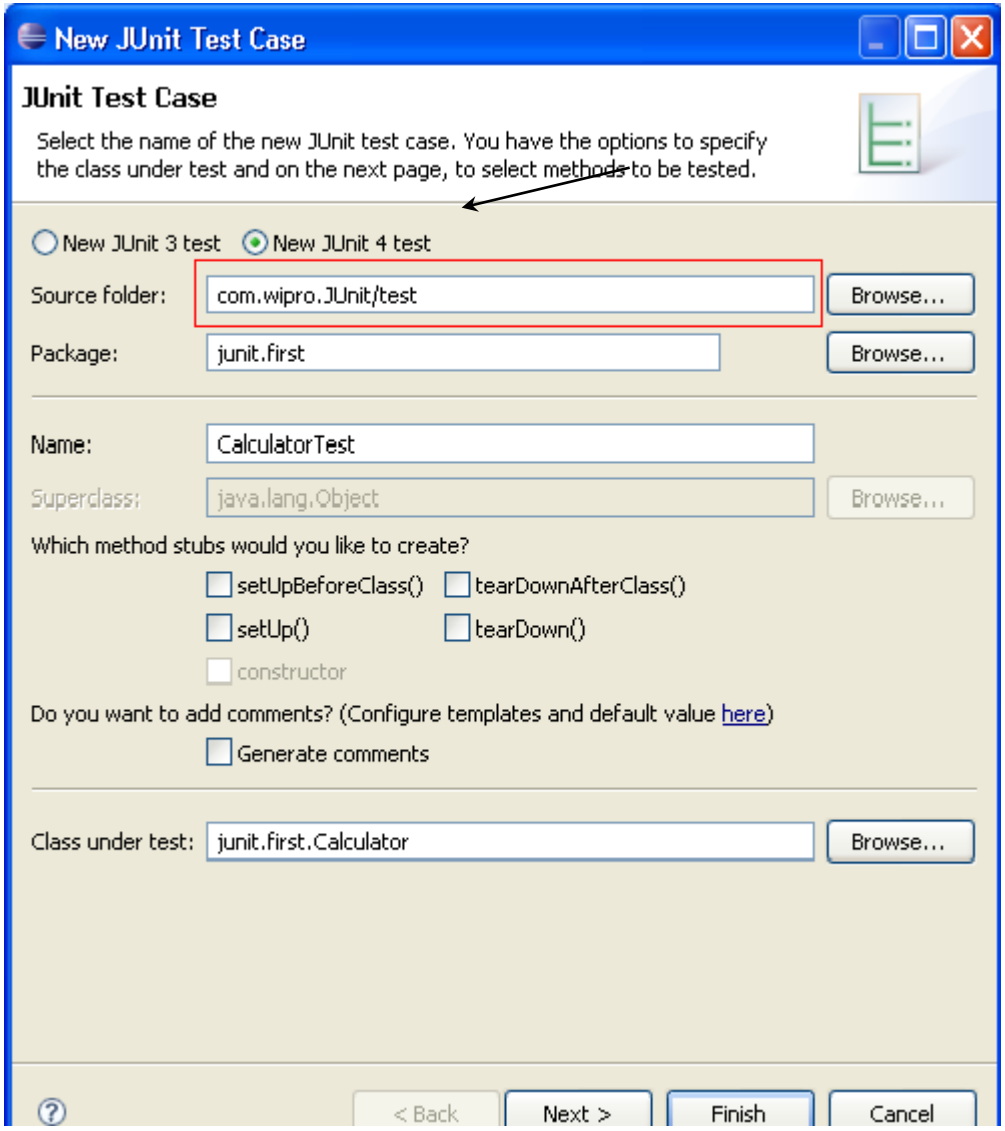
- JUnit is an open source testing framework for Java
- It is a simple framework for creating automated unit tests
- JUnit test cases are Java classes that contain one or more unit test methods
- These tests are grouped into test suites
- JUnit tests are pass/fail tests explicitly designed to run without human intervention
- JUnit can be integrated with several IDEs, including Eclipse
- The JUnit distribution can be downloaded as a single jar file from http://www.junit.org
- It has to be kept in the classpath of the application to be tested

JUnit with Eclipse

- Create a new Project JUnitProj
- Add JUnit.jar to the Classpath
- Right click the Project and create a new Source folder called ‘test’
- Create a new Java class called Calculator in a package junit.first
- Add 2 methods add and sub to the Calculator class which does addition and subtraction of 2 numbers respectively

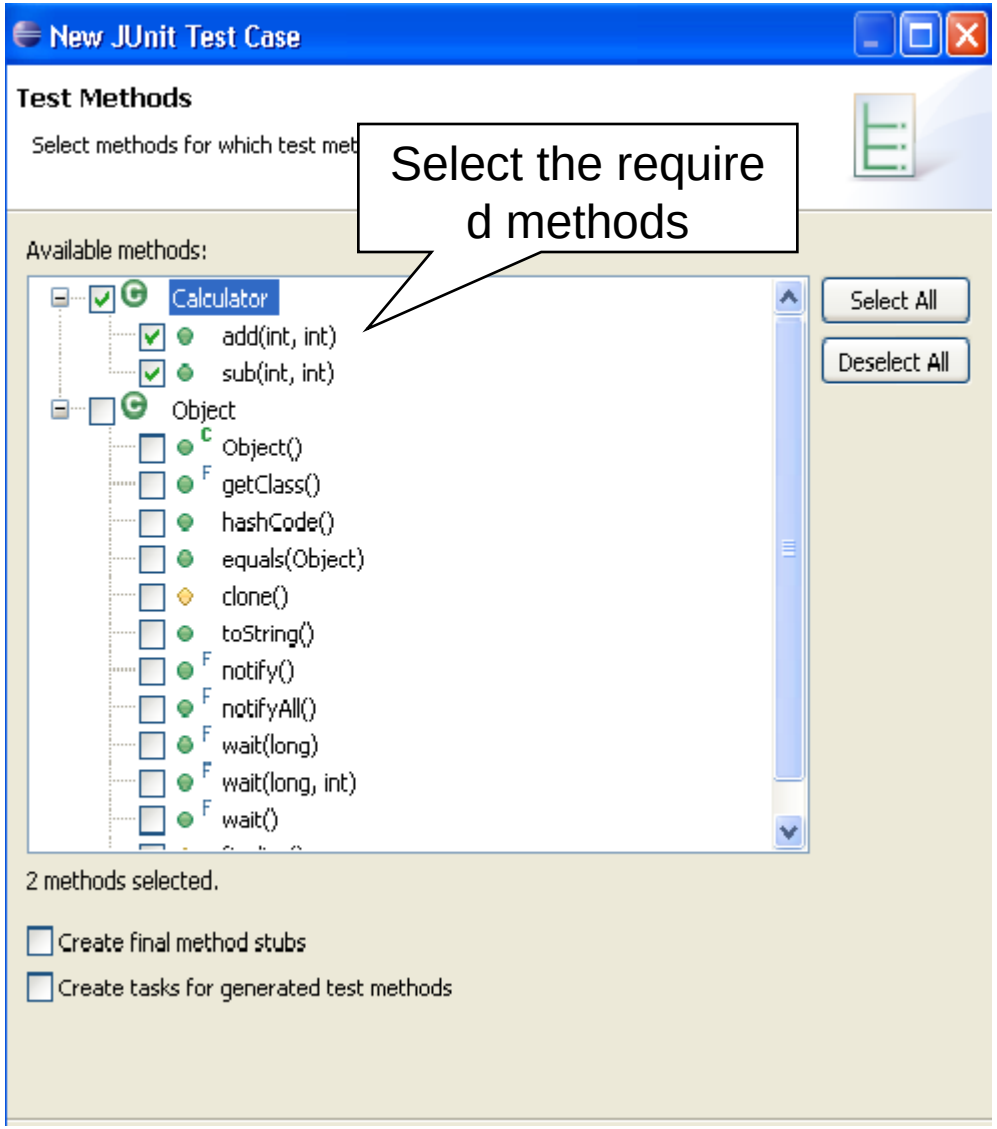
```
package junit.first;
public class Calculator {
public int add(int x,int y)
{ return x+y; }
public int sub(int x,int y)
{ return x-y; } }
```

JUnit with Eclipse (Contd.).



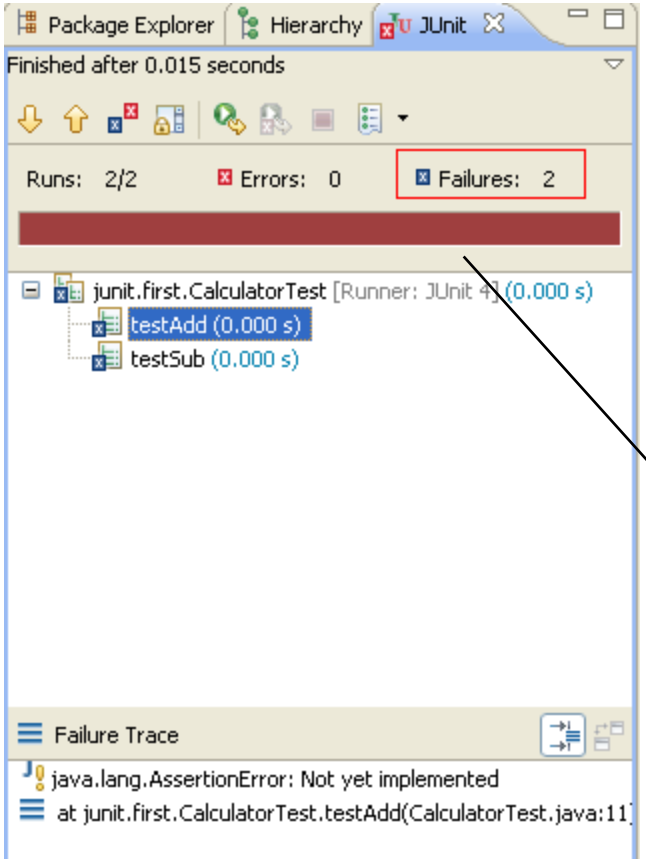
- Right click on the Calculator cl ass in the Package Explorer and
- select New->JUnitTestCase
- select "New JUnit4 test“
- set the source folder to "test“ – the test class gets created here

JUnit with Eclipse (Contd.).



- Press "Next" and select the methods you want to test

JUnit with Eclipse (Contd.).



- Right click on CalculatorTest class and sel ect
- Run-As → JUnit Test
- The results of the test will be displayed in JUnit view
- This is because the testAdd and testSub are not implemented correctly

Brown color indicates failure

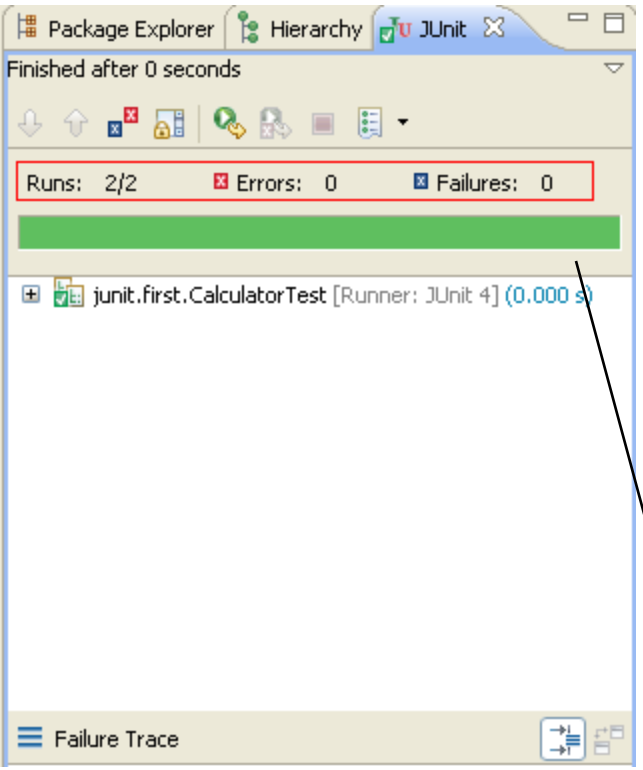
How to write a JUnit test method

- All the test methods should be marked with the Junit annotation - @org.junit.Test
- All the JUnit test methods should be "public" methods
- The return type of the JUnit test method must be “void"
- The test method need not start with the test keyword
- Here is a simple JUnit test method:

```
@Test
public void testAdd() {
        Calculator c=new Calculator();
        assertEquals("Result",5,c.add(2,3));
}
```

JUnit with Eclipse

- Now let's provide implementation to the code and run the test again




```
public class CalculatorTest {
@Test
public void testAdd() {

Calculator c=new Calculator();
assertEquals(5,c.add(2,3));
}
@Test
public void testSub() {

Calculator c=new Calculator();
assertEquals(20,c.sub(100,80));
} }
```

Green color indicates pass

Assert methods with JUnit

- **assertArrayEquals()**
  - Used to test if two arrays are equal to each other

```
int[] expectedArray = {100,200,300};
int[] resultArray =  myClass.getTheIntArray();
assertArrayEquals(expectedArray, resultArray);
```

- **assertEquals()**
  - It compares two objects for their equality

```
String result = myClass.concat("Hello", "World");
assertEquals("HelloWorld", result);
assertEquals(“Reason for failure”,”HelloWorld”,result);
```

Will get printed if the test will fail

Assert methods with JUnit (Contd.).

- **assertTrue() , assertFalse()**
  - Used to test whether a method returns true or false

    assertTrue (testClass.isSafe());

    assertFalse(testClass.isSafe());
- **assertNull(),assertNotNull()**
  - Used to test a variable to see if it is null or not null

    assertNull(testClass.getObject());

    assertNotNull(testClass.getObject());
- **assertSame() and assertNotSame()**
  - Used to test if two object references point to the same object or not

    String s1="Hello";

    String s2="Hello";

    assertSame(s1,s2); ->true

Annotations

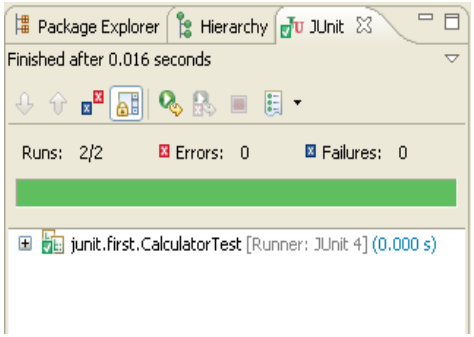
- Fixtures
  - The set of common resources or data that you need to run on e or more tests
- @Before
  - It is used to call the annotated function before running each of the tests
- @After
  - It is used to call the annotated function after each test metho d

```
public class CalculatorTest {
Calculator c=null;

@Before
public void before()        {
                System.out.println("Before Test");
                 c=new Calculator();
}
@After
public void after()     {
                System.out.println("After Test");
}
@Test
public void testAdd() {
System.out.println("Add function");
assertEquals("Result",5,c.add(2,3));
}
@Test
public void testSub() {
System.out.println("Sub function");
assertEquals("Result",20,c.sub(100,80));
} }
```

O/P :

```
Before Test
Add function
After Test
Before Test
Sub function
After Test
```



Annotations (Contd.).

- @BeforeClass
  - The annotated method will run before executing any of the test method
  - The method has to be static
- @AfterClass
  - The annotated method will run after executing all the test methods
  - The method has to be static

```
public class CalculatorTest {
static Calculator c=null;
@BeforeClass
public static void before() {
System.out.println("Before Test");
c=new Calculator();
}
@AfterClass
public  static void after() {
System.out.println("After Test");
}
@Test
public void testAdd() {
System.out.println("Add function");
assertEquals("Result",5,c.add(2,3));
}
@Test
public void testSub() {
System.out.println("Sub function");
assertEquals("Result",20,c.sub(100,80));
}
}
```

O/P :

Before Test
Add function
Sub function
After Test

Annotations (Contd.).

- **@Ignore**
  - Used for test cases you wanted to ignore
  - A String parameter can be added to define the reason for ignoring

```
@Ignore("Not Ready to Run")
@Test
public void testComuteTax() { }
```

- **@Test**
  - Used to identify that a method is a test method

Annotations (Contd.).

- **Timeout**
  - It defines a timeout period in miliseconds with “timeout” parameter
  - The test fails when the timeout period exceeds.

```
@Test (timeout = 1000)
 public void testinfinity() {
        while (true);
 }
```

Parameterised Tests

- New feature added in JUnit 4
- Used to test a method with varying number of Parameters
- Steps for testing a code with multiple parameters
  - The testing class should be annotated with @RunWith(Parameterized.class)
  - The class should have these 3 entities
    - A single constructor that stores the test data
      - Is expected to store each data set in the class fields
    - A static method that generates and returns test data
      - This should be annotated with @Parameters
      - It should return a Collection of Arrays
      - Each array represent the data to be used in a particular test run
      - Number of elements in an array should correspond to the numberof el ements in the constructor
      - Because each array element will be passed to the constructor for every run
    - A test method

Handling an Exception

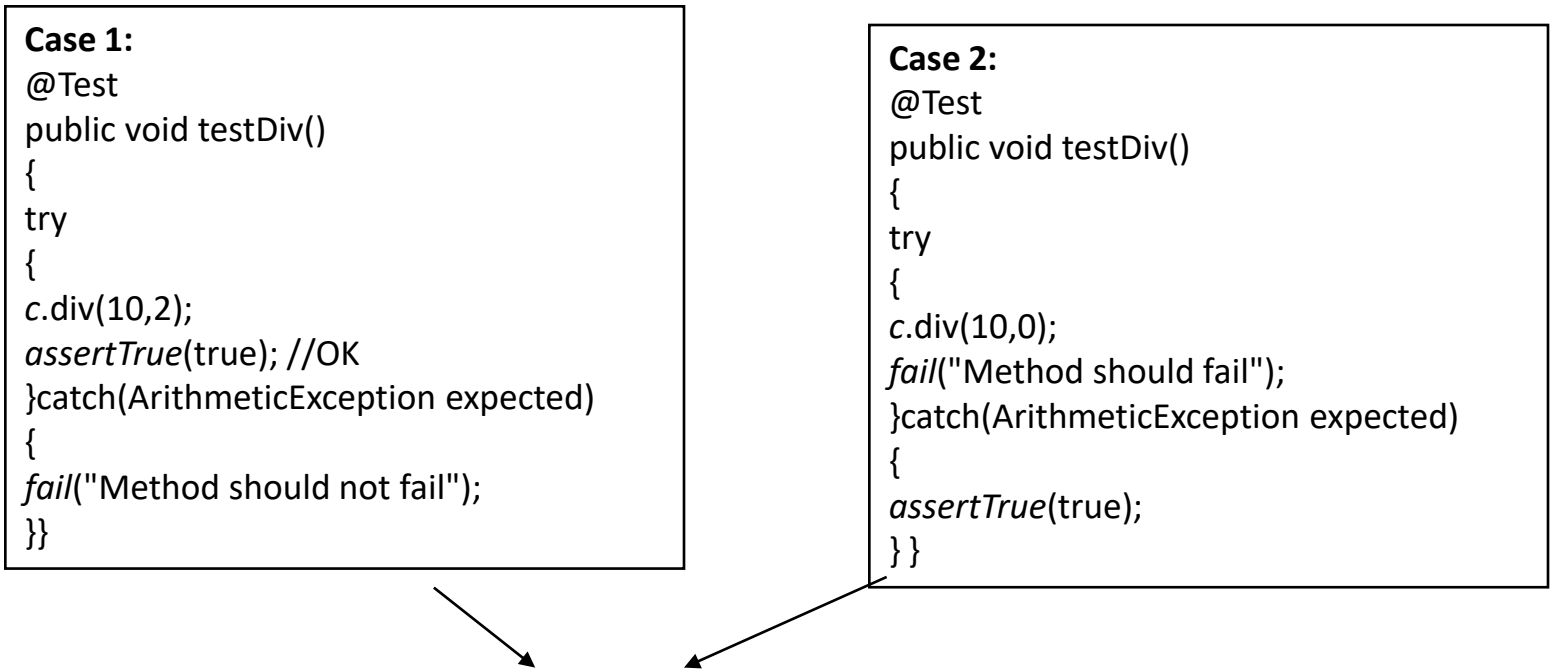
- Two cases are there:
  - Case 1 :We expect a normal behavior and then no exceptions
  - Case 2: We expect an anomalous behavior and then an exception

Case 1:

```
@Test
public void testDiv()
{
try
{
c.div(10,2);
assertTrue(true); //OK
}catch(ArithmeticException expected)
{
fail("Method should not fail");
}}
```

Case 2:

```
@Test
public void testDiv()
{
try
{
c.div(10,0);
fail("Method should fail");
}catch(ArithmeticException expected)
{
assertTrue(true);
} }
```

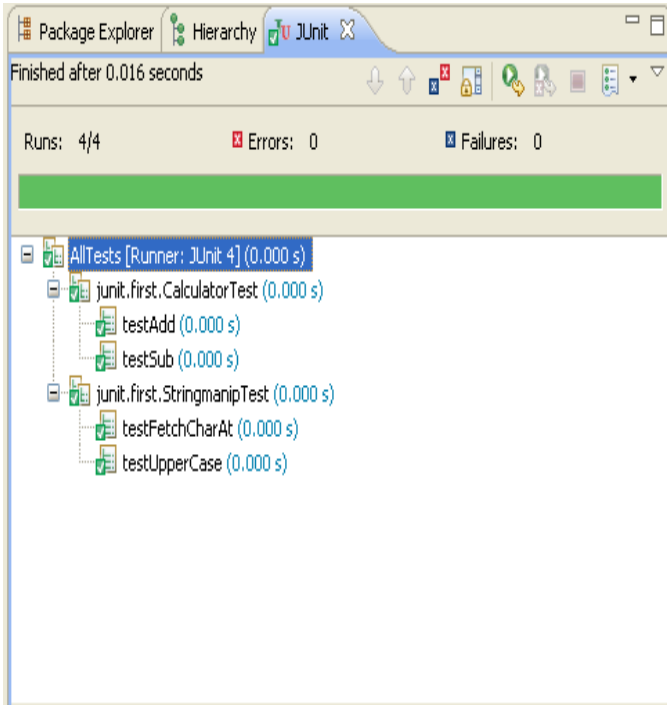


```
public void div(int a,int b) {
                int c=0;
                c=a/b;
                System.out.println(c);
}
```

Test Suite

- Convenient way to group together tests that are related
- Used to bundle a few unit test cases and run it together
- Annotations used for this
  - @RunWith
    - Used to invoke the class which is annotated to run the tests in that class
  - @Suite
    - Allows you to manually build a suite containing tests from many classes

```
@RunWith(Suite.class)
@Suite.SuiteClasses({
        CalculatorTest.class,
        StringmanipTest.class
    })
public class AllTests {
}
```



Quiz

1. Which of the following annotations has to be used before each of the test method?

a. @Before

b. @BeforeClass

c. @After

d. None of the above

None of the above

2. Which of the following are true?

a. All assert methods are static methods

b. The JUnit test methods can be private

c. The JUnit test methods should start with the test keyword

d. All of the above true

All assert methods are static methods