JUnit

JUnit

Introduction

- **JUnit** was created as an open source framework for writing automated, self-verifying tests in Java. It is an instance of the xUnit architecture for unit testing frameworks.

- JUnit may be used in software development following TDD approach or in the test phase of developed software systems.

JUnit Terminology

- **Test Cases.** Test Cases are parts of code, which ensures that another part of code (method) works as expected. To achieve the desired results quickly, a test framework, as JUnit, is required.

- 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.

- **Test Methods.** They are defined in test cases. Inside a test method, a variation of the assert() method (e.g., **`assertTrue()`**, **`assertFalse()`**, **`assertEquals()`**) to compare the expected and actual results is used.

JUnit Terminology

- **Test Fixture.** It is a fixed state of a set of objects used as a baseline for running tests. Its purpose is to ensure that there is a well-known and fixed environment in which tests are run so that results are repeatable.

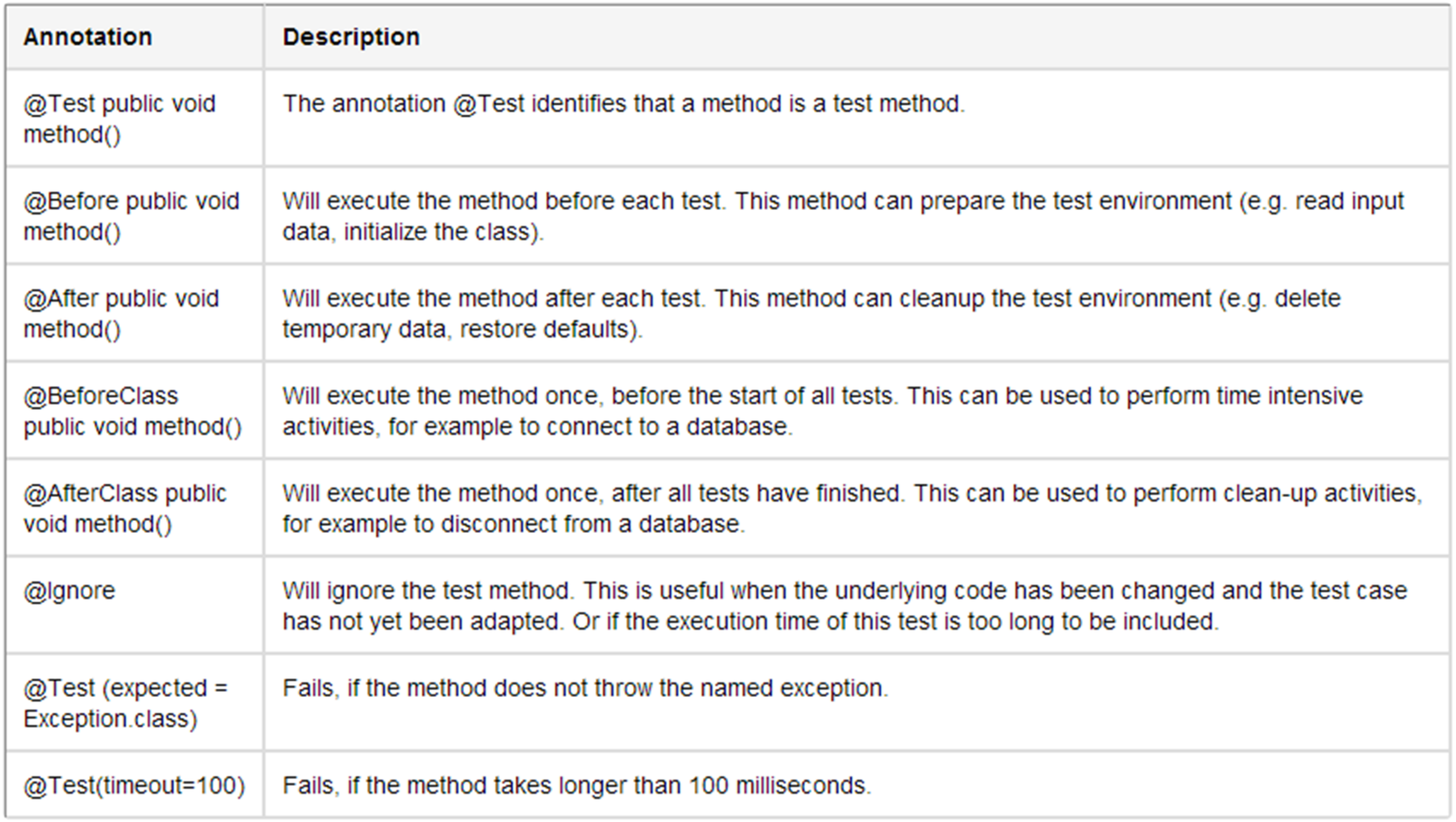
- **Test Suite.** JUnit provides a natural grouping mechanism for related Test Cases (e.g., written by different person) into a Test Suite, and run them at once. Test Suite can also be composed of other Test Suites.

- **Test Runner.** JUnit provides test runners for running Test Cases or Test Suites. The test runner reports on the test that fail, and if none fail, it simply says “OK”.

Steps for Defining a Test Case

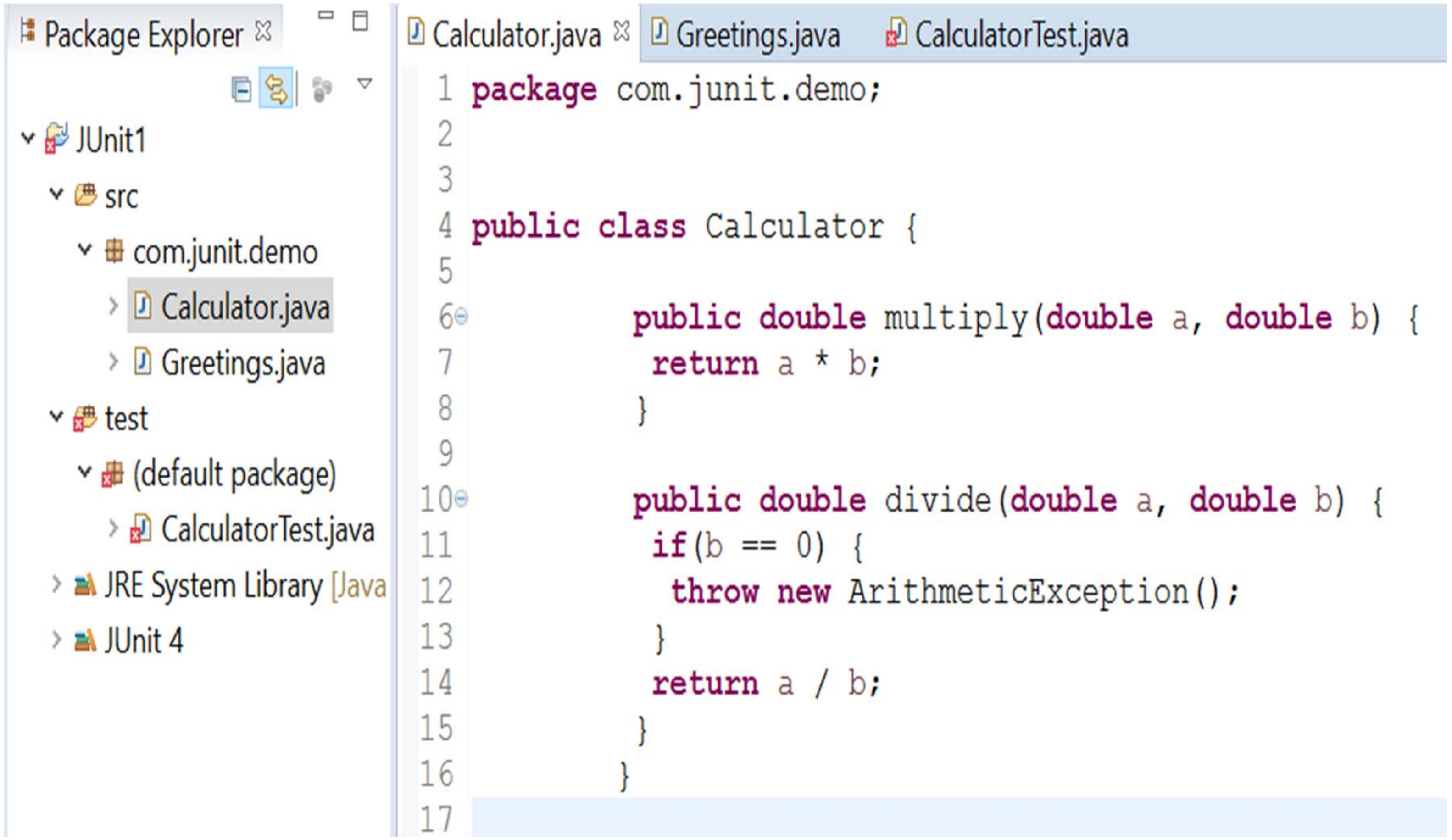
- Define a class that will be a Test Case.
- Define one or more public test methods.
- Define a method (usually called **`setUp()`**) to initialize objects under test.
- Optionally, define a method (usually called **`tearDown()`**) to release object under test.

Steps for Defining a Test Case: JUnit Annotations



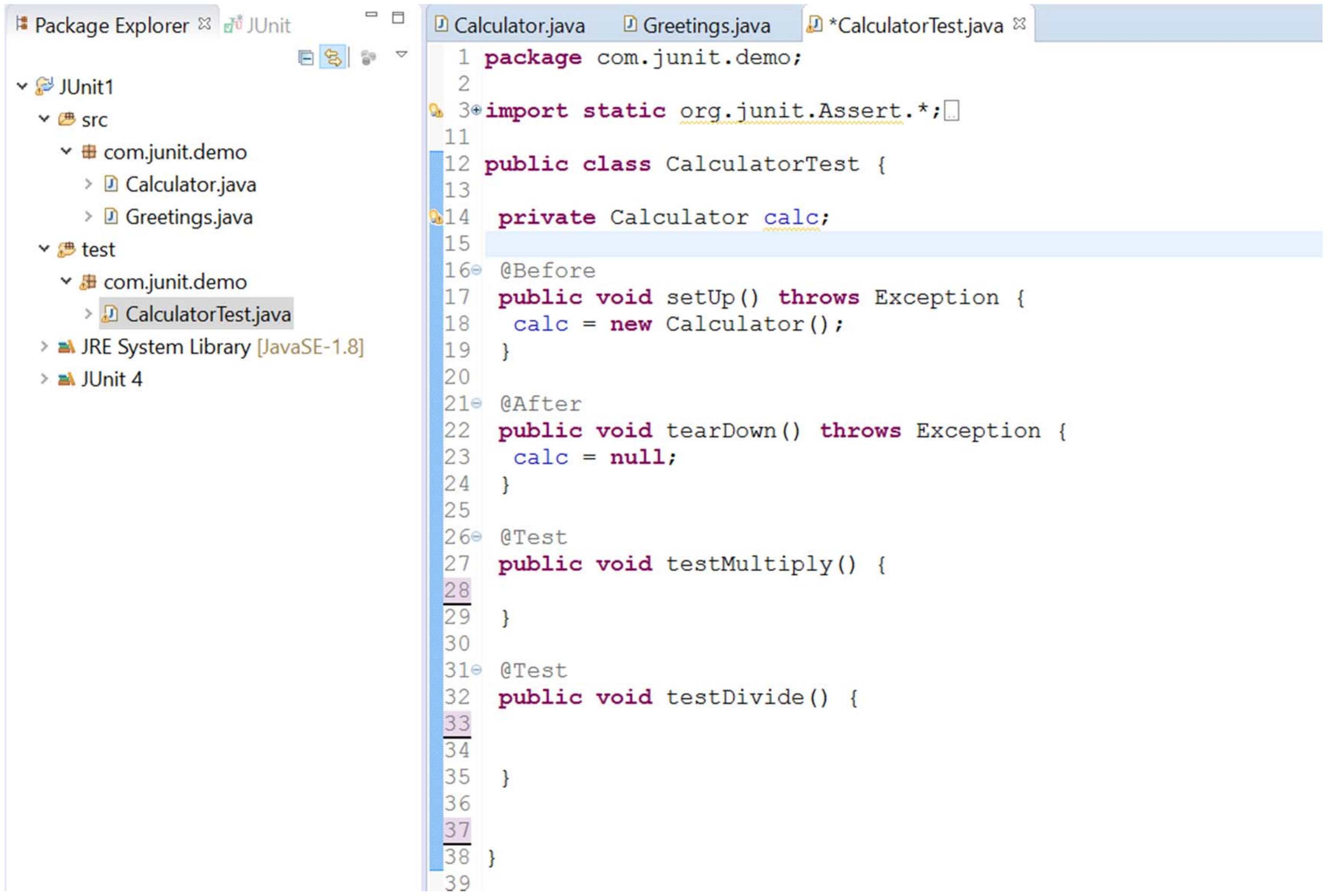
Annotation	Description
@Test public void method()	The annotation @Test identifies that a method is a test method.
@Before public void method()	Will execute the method before each test. This method can prepare the test environment (e.g. read input data, initialize the class).
@After public void method()	Will execute the method after each test. This method can cleanup the test environment (e.g. delete temporary data, restore defaults).
@BeforeClass public void method()	Will execute the method once, before the start of all tests. This can be used to perform time intensive activities, for example to connect to a database.
@AfterClass public void method()	Will execute the method once, after all tests have finished. This can be used to perform clean-up activities, for example to disconnect from a database.
@Ignore	Will ignore the test method. This is useful when the underlying code has been changed and the test case has not yet been adapted. Or if the execution time of this test is too long to be included.
@Test (expected = Exception.class)	Fails, if the method does not throw the named exception.
@Test(timeout=100)	Fails, if the method takes longer than 100 milliseconds.

Steps for Defining a Test Case: Example



```java
package com.junit.demo;


public class Calculator {

        public double multiply(double a, double b) {
         return a * b;
        }

        public double divide(double a, double b) {
         if(b == 0) {
          throw new ArithmeticException();
         }
         return a / b;
        }
       }

```

Steps for Defining a Test Case: Example

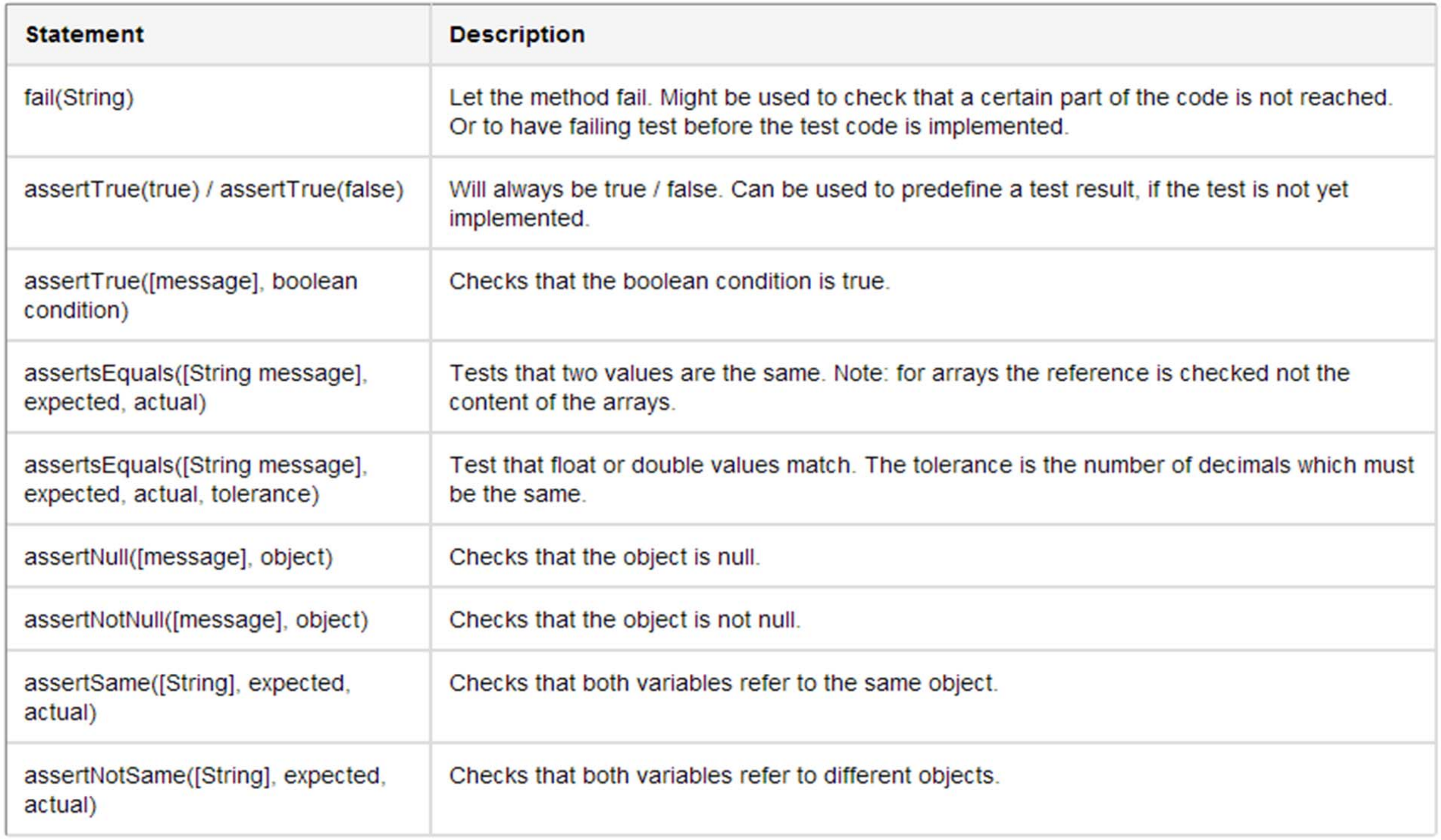


```java
package com.junit.demo;

import static org.junit.Assert.*;

public class CalculatorTest {

 private Calculator calc;

 @Before
 public void setUp() throws Exception {
  calc = new Calculator();
 }

 @After
 public void tearDown() throws Exception {
  calc = null;
 }

 @Test
 public void testMultiply() {

 }

 @Test
 public void testDivide() {


 }


}

```

Steps for Defining Test Method

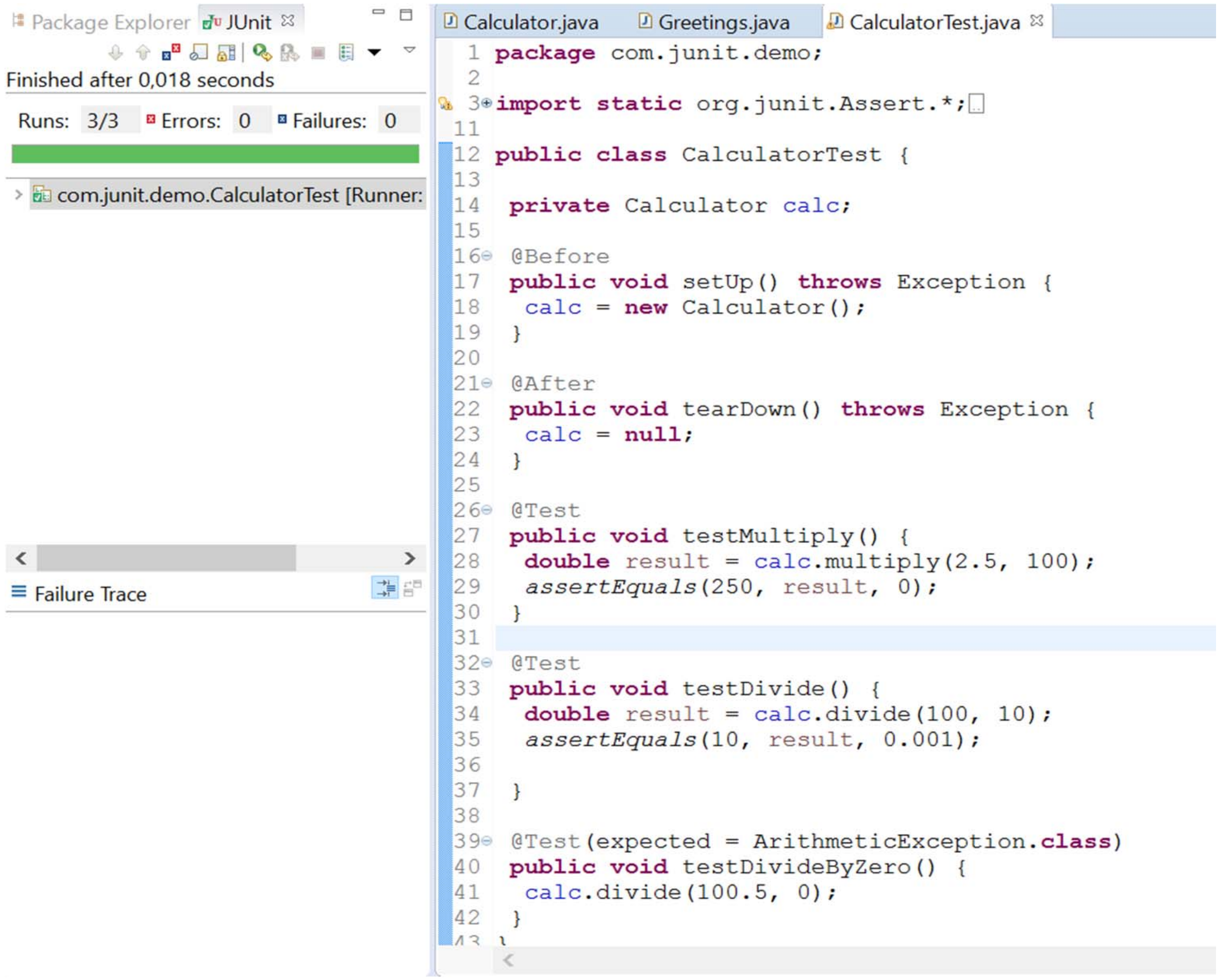
- Use a meaningful name for the test method.
- Call the method that is going to be tested.
- Use a variation of the assert() method to compare the expected and actual results.

Steps for Defining a Test Method: JUnit Assertions



Statement	Description
fail(String)	Let the method fail. Might be used to check that a certain part of the code is not reached. Or to have failing test before the test code is implemented.
assertTrue(true) / assertTrue(false)	Will always be true / false. Can be used to predefine a test result, if the test is not yet implemented.
assertTrue([message], boolean condition)	Checks that the boolean condition is true.
assertsEquals([String message], expected, actual)	Tests that two values are the same. Note: for arrays the reference is checked not the content of the arrays.
assertsEquals([String message], expected, actual, tolerance)	Test that float or double values match. The tolerance is the number of decimals which must be the same.
assertNull([message], object)	Checks that the object is null.
assertNotNull([message], object)	Checks that the object is not null.
assertSame([String], expected, actual)	Checks that both variables refer to the same object.
assertNotSame([String], expected, actual)	Checks that both variables refer to different objects.

Steps for Defining a Test Method: Example



Finished after 0,018 seconds

Runs: 3/3 Errors: 0 Failures: 0

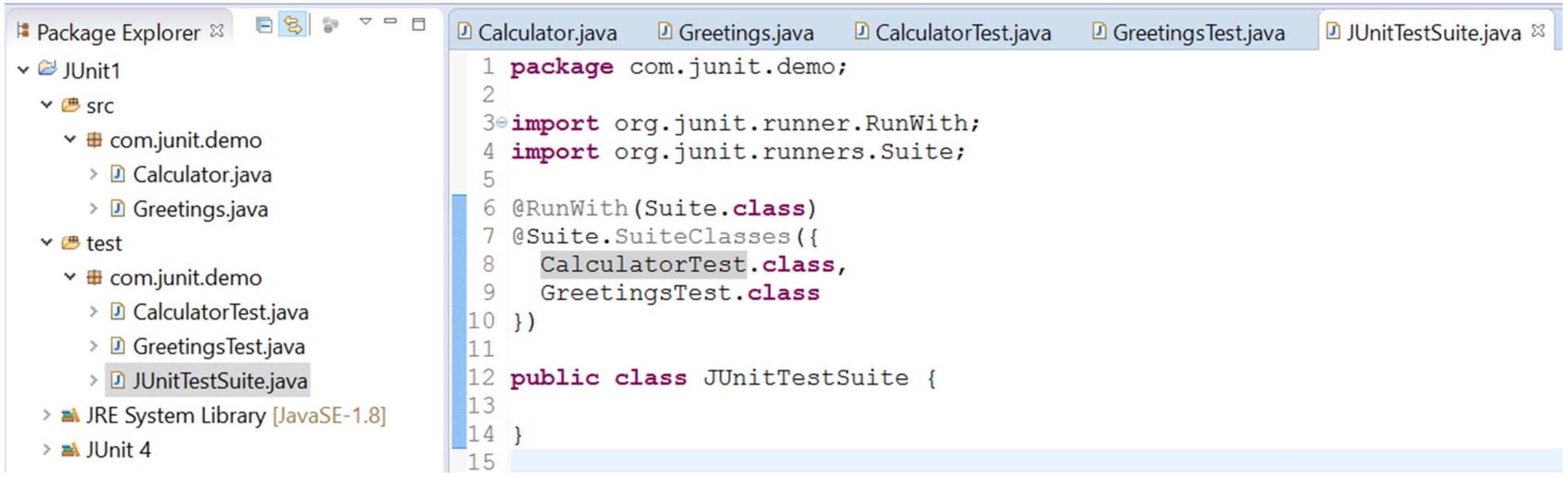
com.junit.demo.CalculatorTest [Runner:

```java
package com.junit.demo;

import static org.junit.Assert.*;

public class CalculatorTest {

 private Calculator calc;

 @Before
 public void setUp() throws Exception {
  calc = new Calculator();
 }

 @After
 public void tearDown() throws Exception {
  calc = null;
 }

 @Test
 public void testMultiply() {
  double result = calc.multiply(2.5, 100);
  assertEquals(250, result, 0);
 }

 @Test
 public void testDivide() {
  double result = calc.divide(100, 10);
  assertEquals(10, result, 0.001);

 }

 @Test(expected = ArithmeticException.class)
 public void testDivideByZero() {
  calc.divide(100.5, 0);
 }
}
```

Steps for Defining a Test Suite

- Create a class that defines the Test Suite.
- Attach @RunWith(Suite.class) Annotation with class.
- Add reference to Junit test classes using @Suite.SuiteClasses annotation

Steps for Defining a Test Suite: Example



```java
package com.junit.demo;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({
  CalculatorTest.class,
  GreetingsTest.class
})

public class JUnitTestSuite {

}
```

References

- *xUnit Test Patterns. Refactoring Test Code*
  Gerard Meszaros
  Addison-Wesley

- *Junit Recipes. Practical Methods for Programmer Testing*
  J.B Rainsberger
  Manning Publications

- *https://www.tutorialspoint.com/junit/index.htm*