**Junit Testing Exercises**

**Exercise 1: Setting Up JUnit**

Scenario:

You need to set up JUnit in your Java project to start writing unit tests.

Steps:

1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).
2. Add JUnit dependency to your project. If you are using Maven, add the following to your pom.xml:

<dependency>

<groupId>junit</groupId>

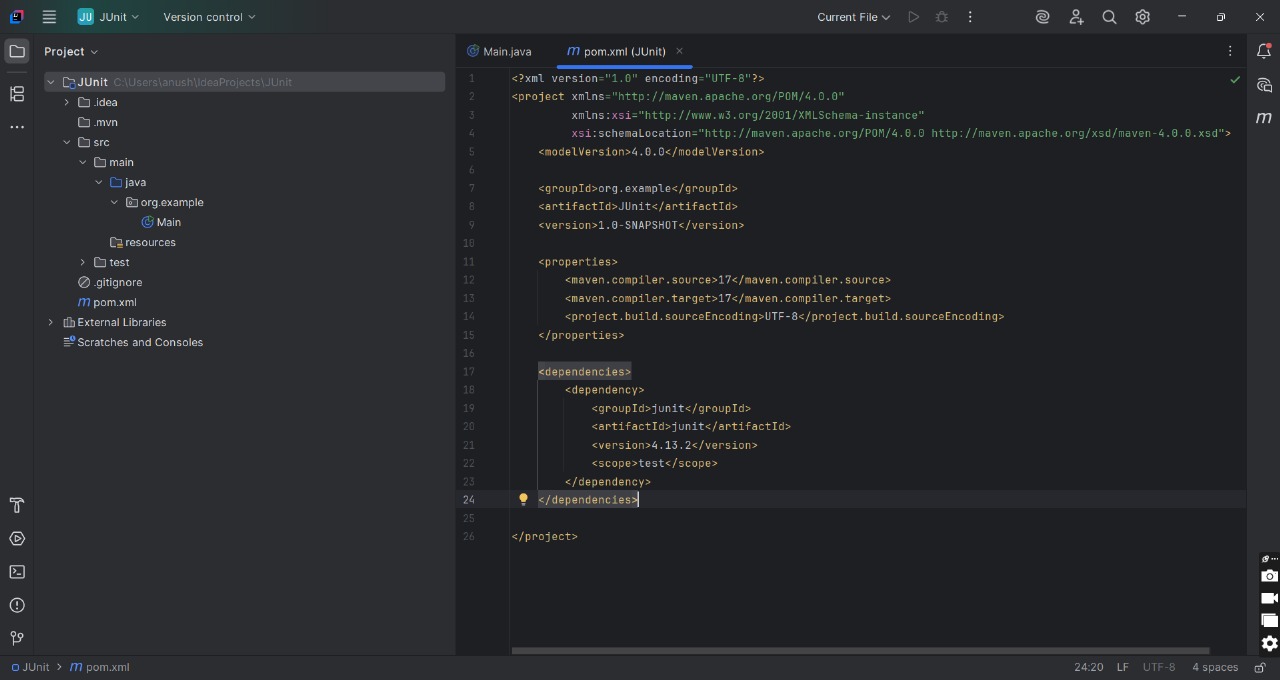
<artifactId>junit</artifactId>

<version>4.13.2</version>

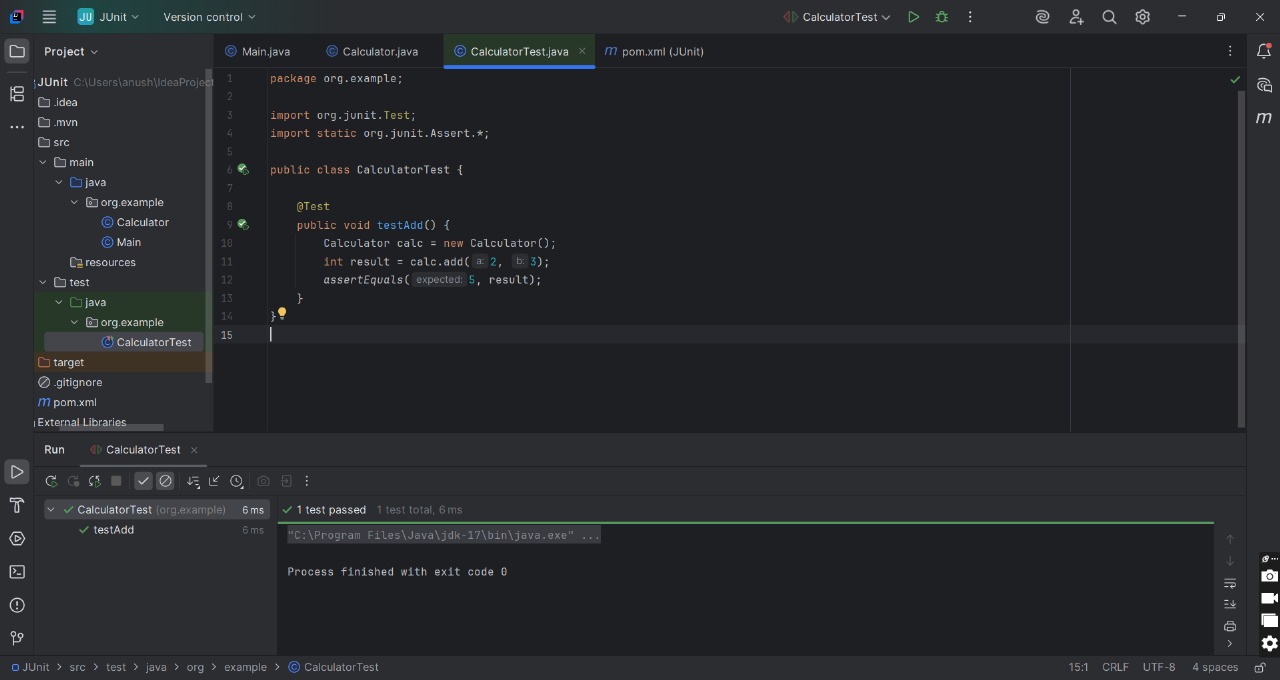
<scope>test</scope>

</dependency>

**OUTPUT:**



1. Create a new test class in your project.



**Exercise 2: Writing Basic JUnit Tests**

Scenario:

You need to write basic JUnit tests for a simple Java class.

Steps:

1. Create a new Java class with some methods to test.
2. Write JUnit tests for these methods.

**CODE:**

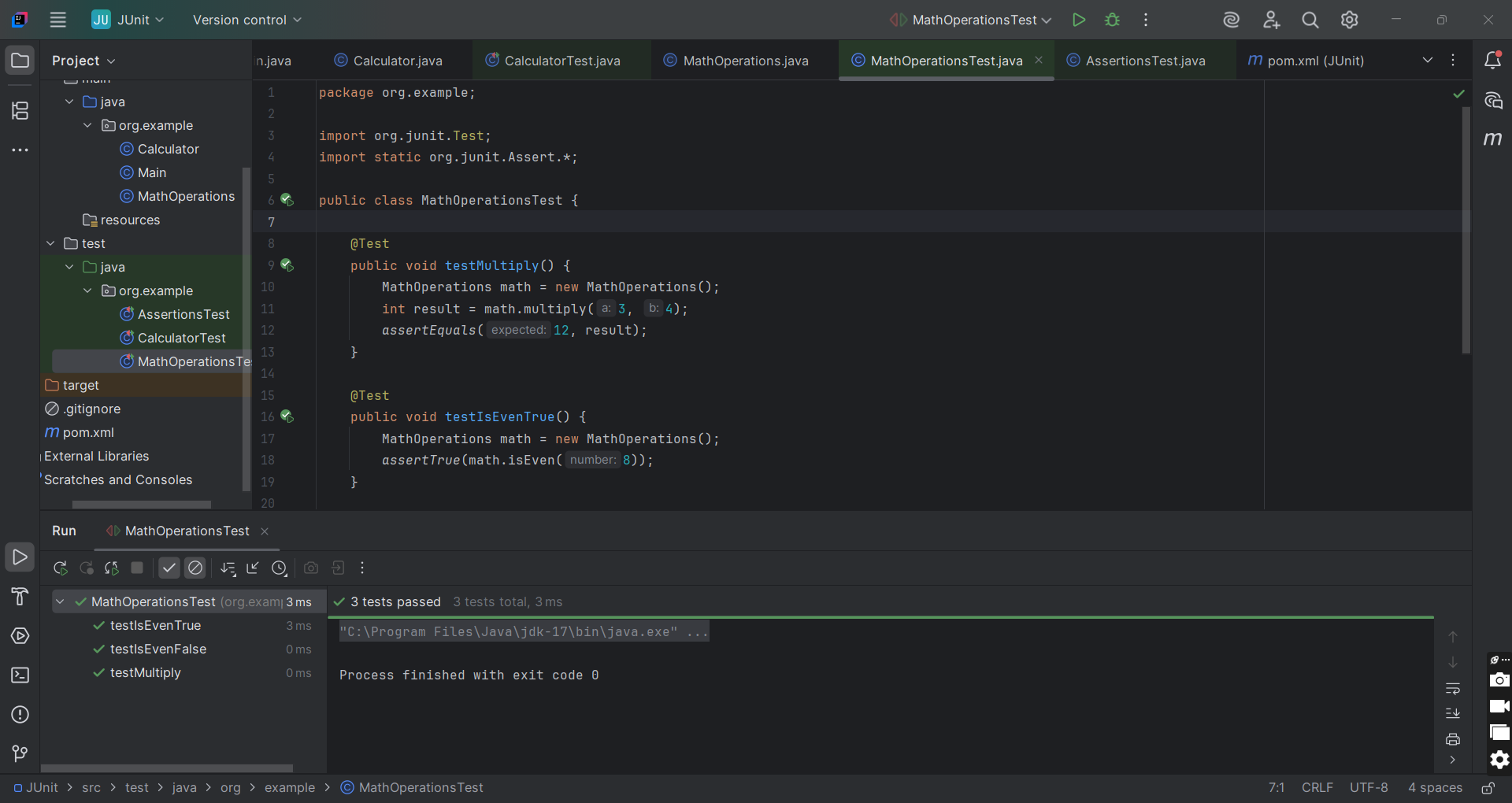
**1.MathOperations.java**

package org.example;  
  
public class MathOperations {  
  
 public int multiply(int a, int b) {  
 return a \* b;  
 }  
  
 public boolean isEven(int number) {  
 return number % 2 == 0;  
 }  
}

**2.MathOperationsTest.java**

package org.example;  
  
import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class MathOperationsTest {  
  
 @Test  
 public void testMultiply() {  
 MathOperations math = new MathOperations();  
 int result = math.multiply(3, 4);  
 *assertEquals*(12, result);  
 }  
  
 @Test  
 public void testIsEvenTrue() {  
 MathOperations math = new MathOperations();  
 *assertTrue*(math.isEven(8));  
 }  
  
 @Test  
 public void testIsEvenFalse() {  
 MathOperations math = new MathOperations();  
 *assertFalse*(math.isEven(7));  
 }  
}

**OUTPUT:**

****

**Exercise 3: Assertions in JUnit**

Scenario:

You need to use different assertions in JUnit to validate your test results.

Steps:

1. Write tests using various JUnit assertions.

Solution Code:

public class AssertionsTest {

@Test public void testAssertions() {

// Assert equals

assertEquals(5, 2 + 3);

// Assert true

assertTrue(5 > 3);

// Assert false

assertFalse(5 < 3);

// Assert null

assertNull(null);

// Assert not null

assertNotNull(new Object());

}

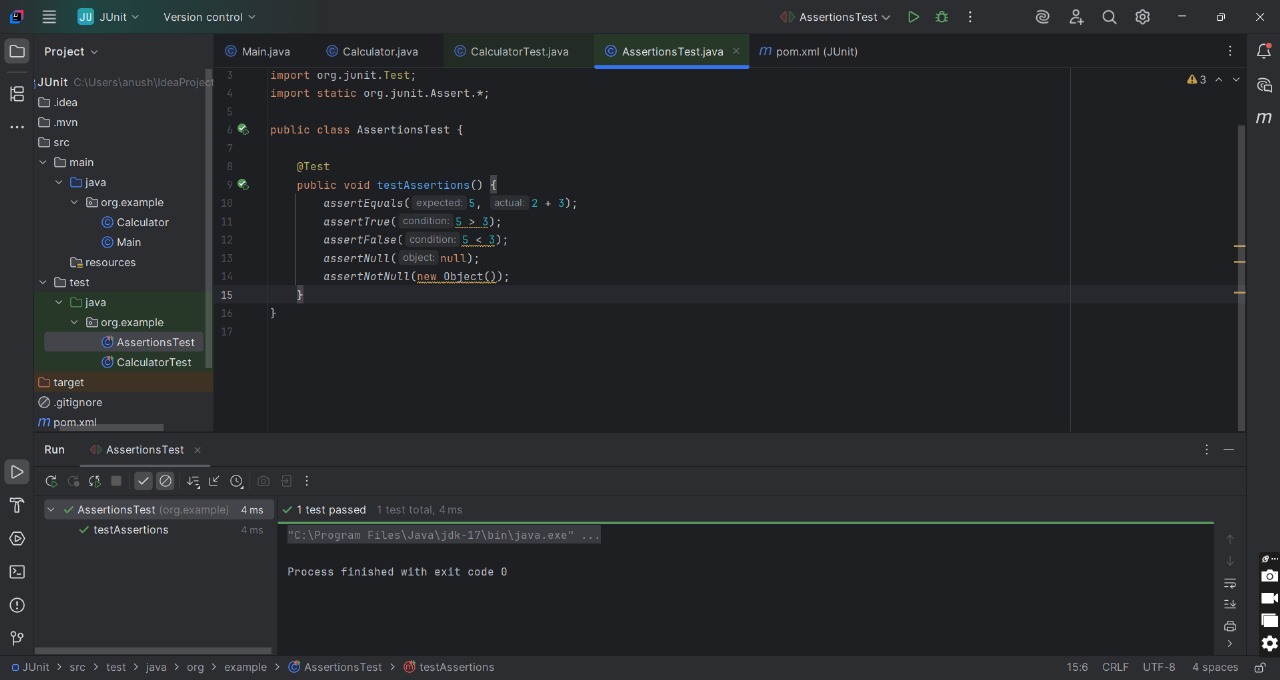
}

**CODE:**

**1.AssertionsTest.java**

package org.example;  
  
import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class AssertionsTest {  
  
 @Test  
 public void testAssertions() {  
 *assertEquals*(5, 2 + 3);  
 *assertTrue*(5 > 3);  
 *assertFalse*(5 < 3);  
 *assertNull*(null);  
 *assertNotNull*(new Object());  
 }  
}

**OUTPUT:**



**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and**

**Teardown Methods in JUnit**

Scenario:

You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

Steps:

1. Write tests using the AAA pattern.
2. Use @Before and @After annotations for setup and teardown methods.

**CODE:**

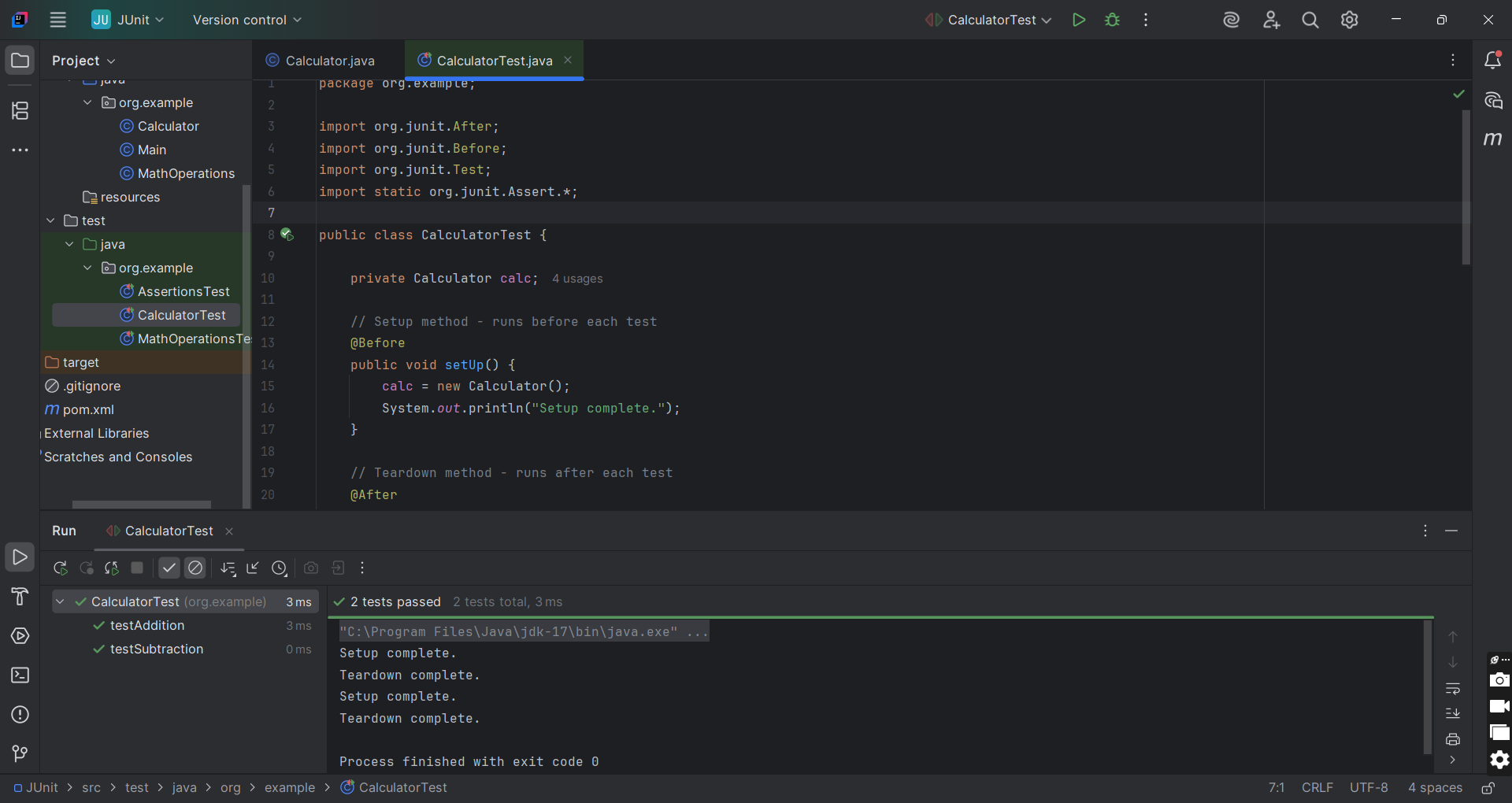
**1.Calculator.java**

package org.example;  
  
public class Calculator {  
  
 public int add(int a, int b) {  
 return a + b;  
 }  
  
 public int subtract(int a, int b) {  
 return a - b;  
 }  
}

**2.CalculatorTest.java**

package org.example;  
  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class CalculatorTest {  
  
 private Calculator calc;  
  
 // Setup method - runs before each test  
 @Before  
 public void setUp() {  
 calc = new Calculator();  
 System.*out*.println("Setup complete.");  
 }  
  
 // Teardown method - runs after each test  
 @After  
 public void tearDown() {  
 calc = null;  
 System.*out*.println("Teardown complete.");  
 }  
  
 @Test  
 public void testAddition() {  
 // Arrange - Done in setup  
  
 // Act  
 int result = calc.add(2, 3);  
  
 // Assert  
 *assertEquals*(5, result);  
 }  
  
 @Test  
 public void testSubtraction() {  
 // Act  
 int result = calc.subtract(5, 3);  
  
 // Assert  
 *assertEquals*(2, result);  
 }  
}

**OUTPUT:**

****