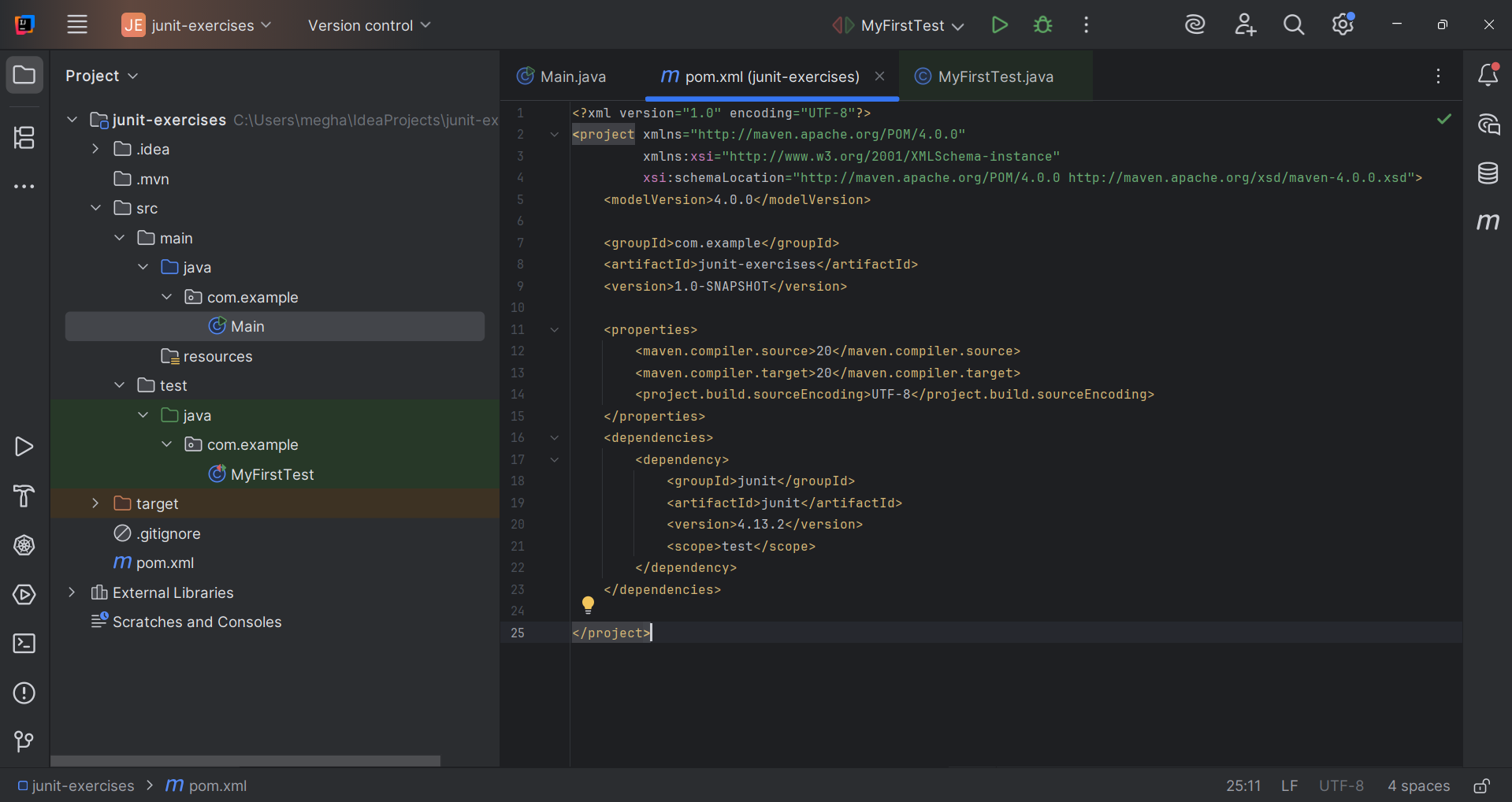
**JUnit Testing Exercises:**

**Exercise 1: Setting Up JUnit Scenario: You need to set up JUnit in your Java project to start writing unit tests.**

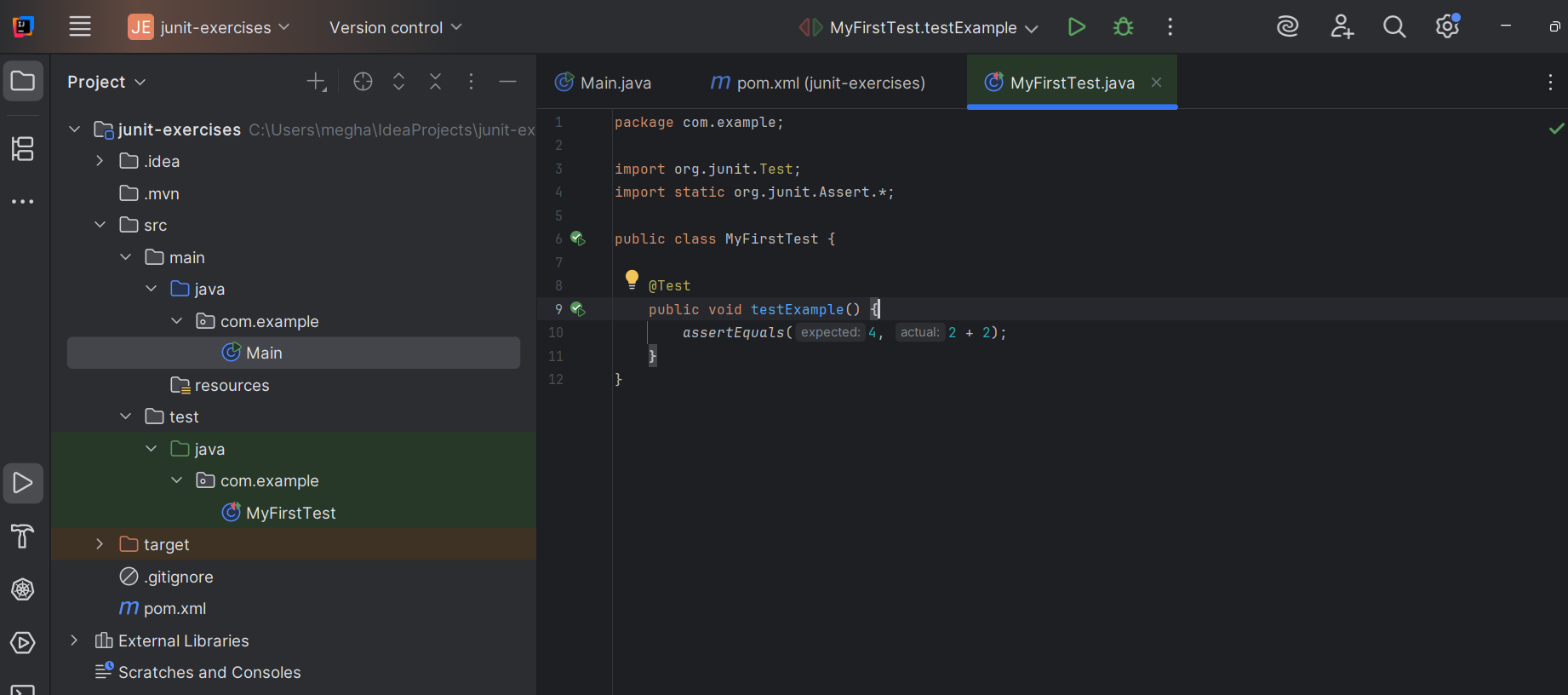
**Steps:**

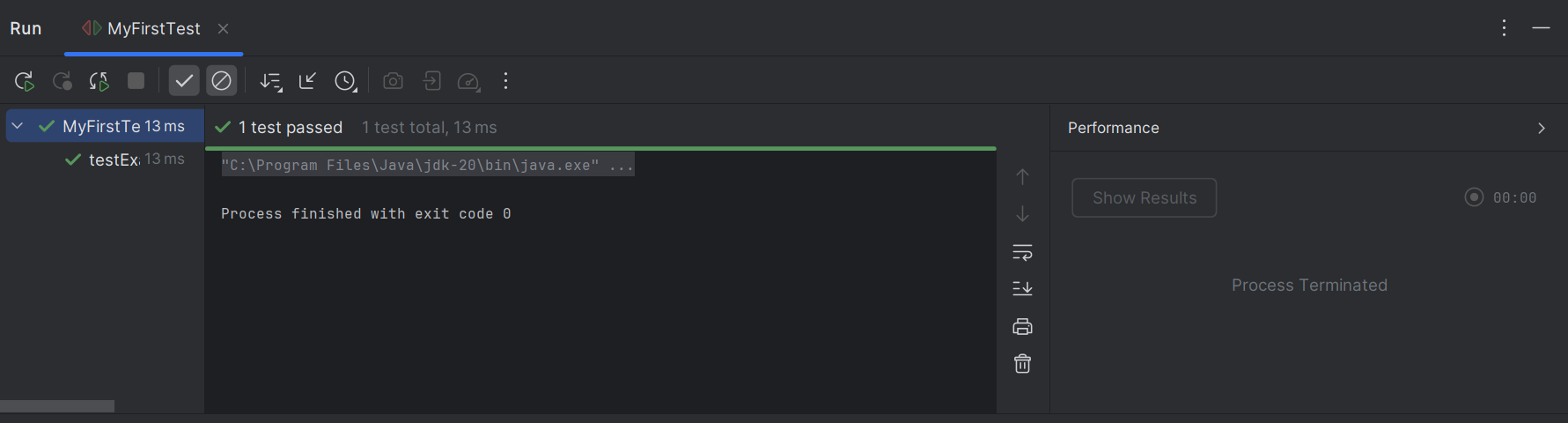
**1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).**

**2. Add JUnit dependency to your project. If you are using Maven, add the following to your pom.xml**



**3. Create a new test class in your project.**



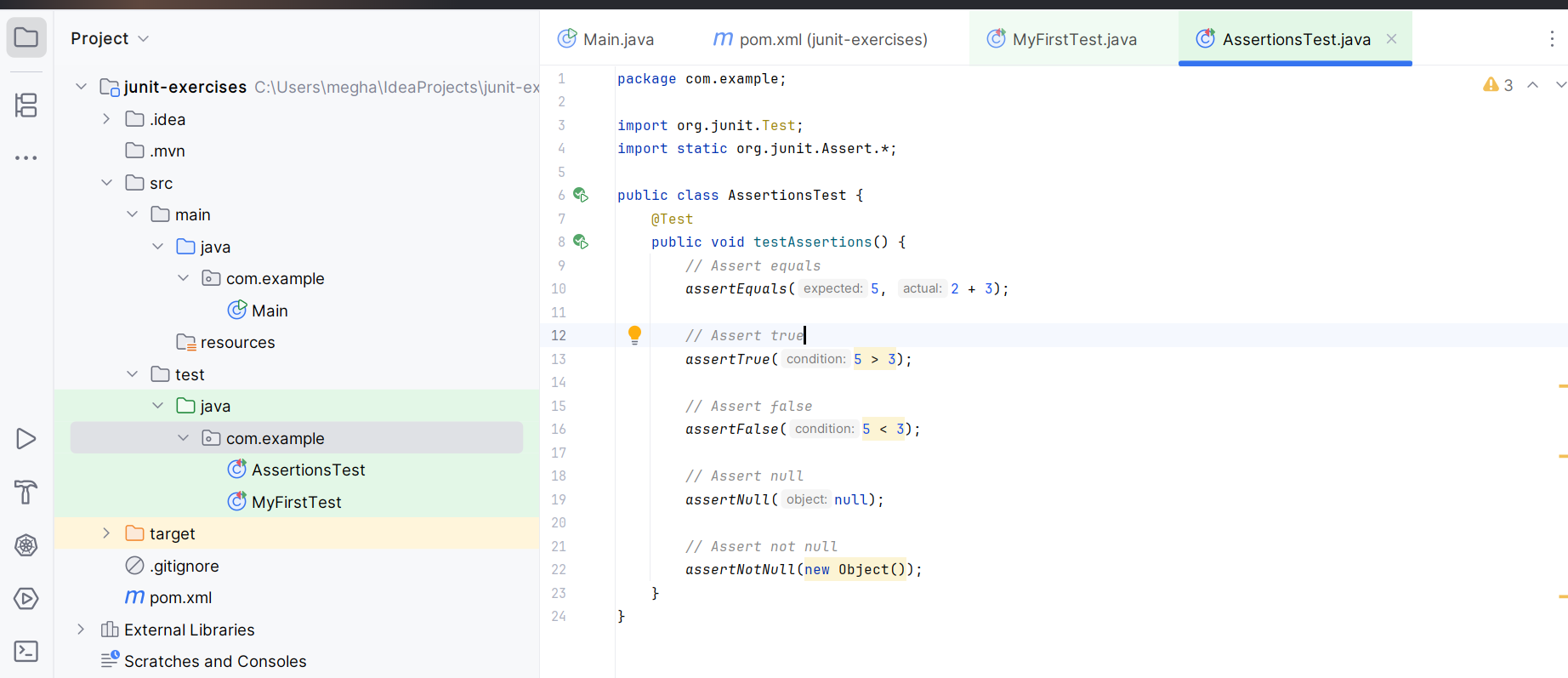


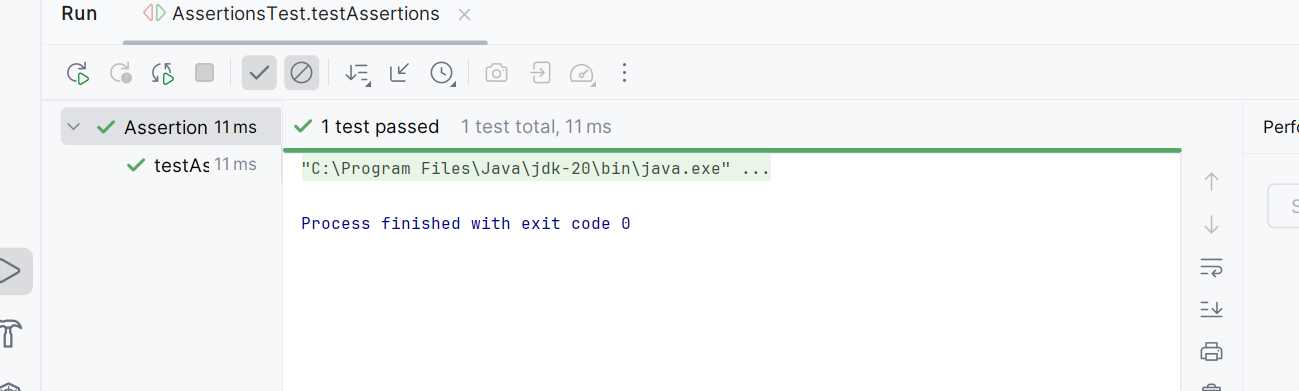
--------------------------------------------------------------------------------------------------------------------------------------

--------------------------------------------------------------------------------------------------------------------------------------

**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.**





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:

package com.example;  
  
import org.junit.Before;  
import org.junit.After;  
import org.junit.Test;  
import org.junit.BeforeClass;  
import org.junit.AfterClass;  
import static org.junit.Assert.\*;  
  
public class CalculatorTest {  
  
 *// Test fixture - shared test data* private Calculator calculator;  
 private static int *testCounter* = 0;  
  
 *// Setup method - runs before each test method* @Before  
 public void setUp() {  
 *// Arrange phase setup - create fresh instance for each test* calculator = new Calculator();  
 *testCounter*++;  
 System.*out*.println("Setting up test #" + *testCounter* + " - Calculator initialized");  
 }  
  
 *// Teardown method - runs after each test method* @After  
 public void tearDown() {  
 *// Clean up after each test* calculator.clear();  
 System.*out*.println("Cleaning up after test #" + *testCounter* + " - Calculator cleared");  
 }  
  
 *// Setup method - runs once before all tests in the class* @BeforeClass  
 public static void setUpClass() {  
 System.*out*.println("Starting Calculator Test Suite");  
 }  
  
 *// Teardown method - runs once after all tests in the class* @AfterClass  
 public static void tearDownClass() {  
 System.*out*.println("Calculator Test Suite Completed ");  
 System.*out*.println("Total tests executed: " + *testCounter*);  
 }  
  
 *// Test using AAA pattern - Addition* @Test  
 public void testAddition() {  
 *// ARRANGE - Set up test data and expected results* double firstNumber = 10.0;  
 double secondNumber = 5.0;  
 double expectedResult = 15.0;  
  
 *// ACT - Execute the method being tested* double actualResult = calculator.add(firstNumber, secondNumber);  
  
 *// ASSERT - Verify the results  
 assertEquals*("Addition should return correct sum", expectedResult, actualResult, 0.001);  
 *assertEquals*("Last result should be updated", expectedResult, calculator.getLastResult(), 0.001);  
 }  
  
 *// Test using AAA pattern - Subtraction* @Test  
 public void testSubtraction() {  
 *// ARRANGE* double minuend = 20.0;  
 double subtrahend = 8.0;  
 double expectedResult = 12.0;  
  
 *// ACT* double actualResult = calculator.subtract(minuend, subtrahend);  
  
 *// ASSERT  
 assertEquals*("Subtraction should return correct difference", expectedResult, actualResult, 0.001);  
 *assertEquals*("Last result should be updated", expectedResult, calculator.getLastResult(), 0.001);  
 }  
  
 *// Test using AAA pattern - Multiplication* @Test  
 public void testMultiplication() {  
 *// ARRANGE* double multiplicand = 6.0;  
 double multiplier = 7.0;  
 double expectedResult = 42.0;  
  
 *// ACT* double actualResult = calculator.multiply(multiplicand, multiplier);  
  
 *// ASSERT  
 assertEquals*("Multiplication should return correct product", expectedResult, actualResult, 0.001);  
 *assertEquals*("Last result should be updated", expectedResult, calculator.getLastResult(), 0.001);  
 }  
  
 *// Test using AAA pattern - Division* @Test  
 public void testDivision() {  
 *// ARRANGE* double dividend = 15.0;  
 double divisor = 3.0;  
 double expectedResult = 5.0;  
  
 *// ACT* double actualResult = calculator.divide(dividend, divisor);  
  
 *// ASSERT  
 assertEquals*("Division should return correct quotient", expectedResult, actualResult, 0.001);  
 *assertEquals*("Last result should be updated", expectedResult, calculator.getLastResult(), 0.001);  
 }  
  
 *// Test using AAA pattern - Exception handling* @Test(expected = IllegalArgumentException.class)  
 public void testDivisionByZero() {  
 *// ARRANGE* double dividend = 10.0;  
 double divisor = 0.0;  
  
 *// ACT & ASSERT (exception expected)* calculator.divide(dividend, divisor);  
  
 *// This line should not be reached due to exception  
 fail*("Division by zero should throw IllegalArgumentException");  
 }  
  
 *// Test using AAA pattern - Testing state changes* @Test  
 public void testCalculatorStateManagement() {  
 *// ARRANGE* double firstOperation = 10.0;  
 double secondOperation = 5.0;  
 double expectedFirstResult = 15.0;  
 double expectedSecondResult = 75.0; *// 15 \* 5  
  
 // ACT - Multiple operations* double firstResult = calculator.add(firstOperation, secondOperation);  
 double secondResult = calculator.multiply(firstResult, secondOperation);  
  
 *// ASSERT  
 assertEquals*("First operation should be correct", expectedFirstResult, firstResult, 0.001);  
 *assertEquals*("Second operation should be correct", expectedSecondResult, secondResult, 0.001);  
 *assertEquals*("Final state should reflect last operation", expectedSecondResult, calculator.getLastResult(), 0.001);  
 }  
  
 *// Test using AAA pattern - Testing clear functionality* @Test  
 public void testClearFunctionality() {  
 *// ARRANGE* calculator.add(10, 5); *// Perform an operation first* double expectedResultAfterClear = 0.0;  
  
 *// ACT* calculator.clear();  
 double actualResult = calculator.getLastResult();  
  
 *// ASSERT  
 assertEquals*("Clear should reset last result to zero", expectedResultAfterClear, actualResult, 0.001);  
 }  
}

**Output:**

