**Exercise 1: JUnit Setup and Basic Test Execution**

**Scenario 1: Setting Up JUnit 5 in Eclipse IDE 2025-03**

**Objective:**  
To configure a Maven-based Java project in Eclipse with **JUnit 5** and to create and run a basic unit test for a simple class.

**Steps:**

1. **Created a Maven Project:**
   * Group Id: com.example
   * Artifact Id: junit-demo
2. **Added Dependencies in pom.xml:**

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

1. **Created Sample Class for Testing:**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

1. **Created a JUnit Test Class:**

package com.example;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

class CalculatorTest {

*@Test*

public void testAdd() {

Calculator calc = new Calculator();

int result = calc.add(2, 3);

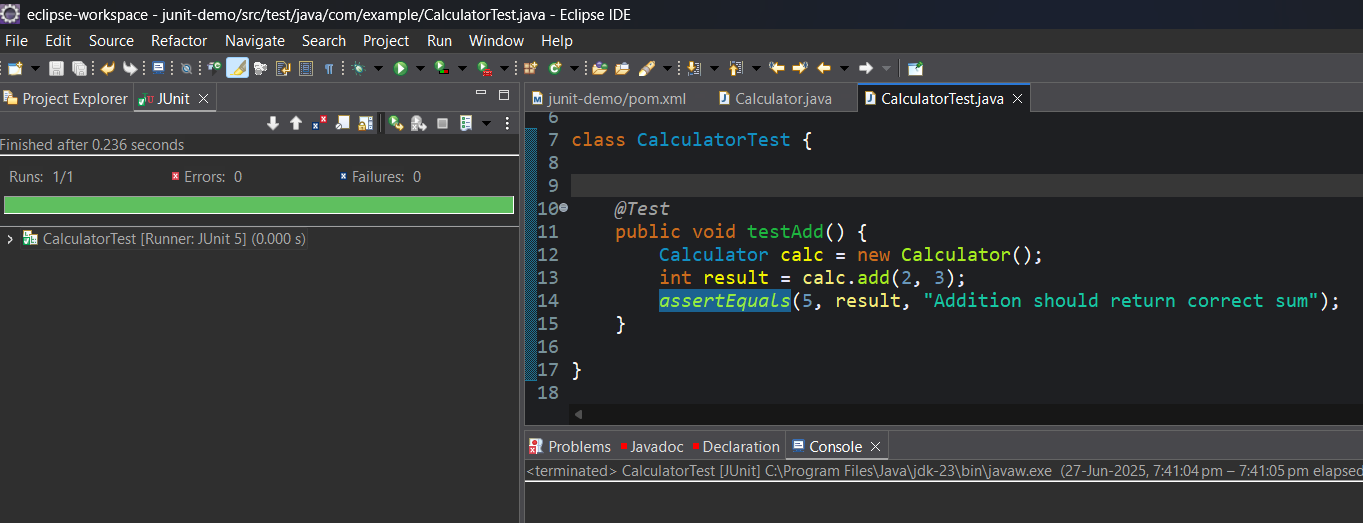
*assertEquals*(5, result, "Addition should return correct sum");

}

}

1. **Executed JUnit Test:**
   * Used **Run As → JUnit Test**
   * Test passed successfully with green bar confirmation.

**Output:**



**Exercise 3: Assertions in JUnit**

**Scenario:**  
To write unit tests using various **JUnit assertions** that validate different conditions like equality, nullability, and boolean expressions.

**Objective:**  
To demonstrate how to use different assertion methods provided by **JUnit 5** to verify test logic effectively.

**Steps:**

1. Created a new class named AssertionsTest under the test package.
2. Annotated the test method with @Test.
3. Used multiple assertion methods from JUnit:
   * assertEquals()
   * assertTrue()
   * assertFalse()
   * assertNull()
   * assertNotNull()

**Code:**

package com.example;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

public class AssertionsTest {

*@Test*

public void testAssertions() {

*assertEquals*(5, 2 + 3);

*assertTrue*(5 > 3);

*assertFalse*(5 < 3);

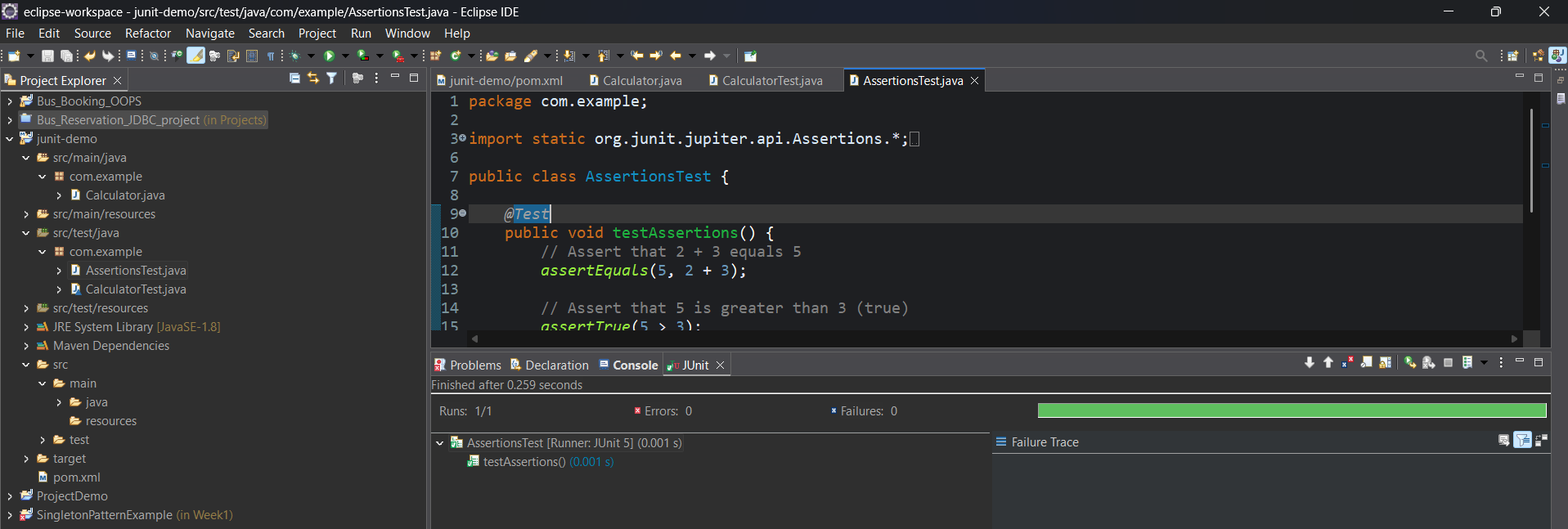
*assertNull*(null);

*assertNotNull*(new Object());

}

}

**Output:**



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

**Scenario:**  
You are required to write unit tests using the **Arrange-Act-Assert (AAA)** pattern and organize reusable setup and cleanup logic using JUnit's **test fixtures** with @BeforeEach and @AfterEach annotations.

**Objective:**  
To structure test methods for better readability and maintainability, and use **setup/teardown methods** to avoid repetitive code and manage test state.

**Steps:**

1. Created a class CalculatorTestWithFixtures for testing.
2. Applied the **AAA pattern**:
   * **Arrange**: Prepare test data and environment.
   * **Act**: Execute the method under test.
   * **Assert**: Verify the result.
3. Used:
   * @BeforeEach to run code before each test method.
   * @AfterEach to clean up after each test method.

**Code:**

package com.example;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.Test;

public class CalculatorTestWithFixtures {

private Calculator calculator;

*@BeforeEach*

public void setUp() {

// Arrange: Initialize Calculator before each test

calculator = new Calculator();

System.***out***.println("Setup before test");

}

*@AfterEach*

public void tearDown() {

// Cleanup logic after each test

calculator = null;

System.***out***.println("Teardown after test");

}

*@Test*

public void testAddition() {

// Act

int result = calculator.add(10, 5);

// Assert

*assertEquals*(15, result, "10 + 5 should equal 15");

}

*@Test*

public void testAdditionWithNegative() {

// Act

int result = calculator.add(-2, -3);

// Assert

*assertEquals*(-5, result, "-2 + -3 should equal -5");

}

}

**Output:**  
