**Exercise 1: Setting Up JUnit**

Scenario:

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

Step 1: Add JUnit dependency in pom.xml

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

Step 2: Create test class AppTest.java

package org.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AppTest {

@Test

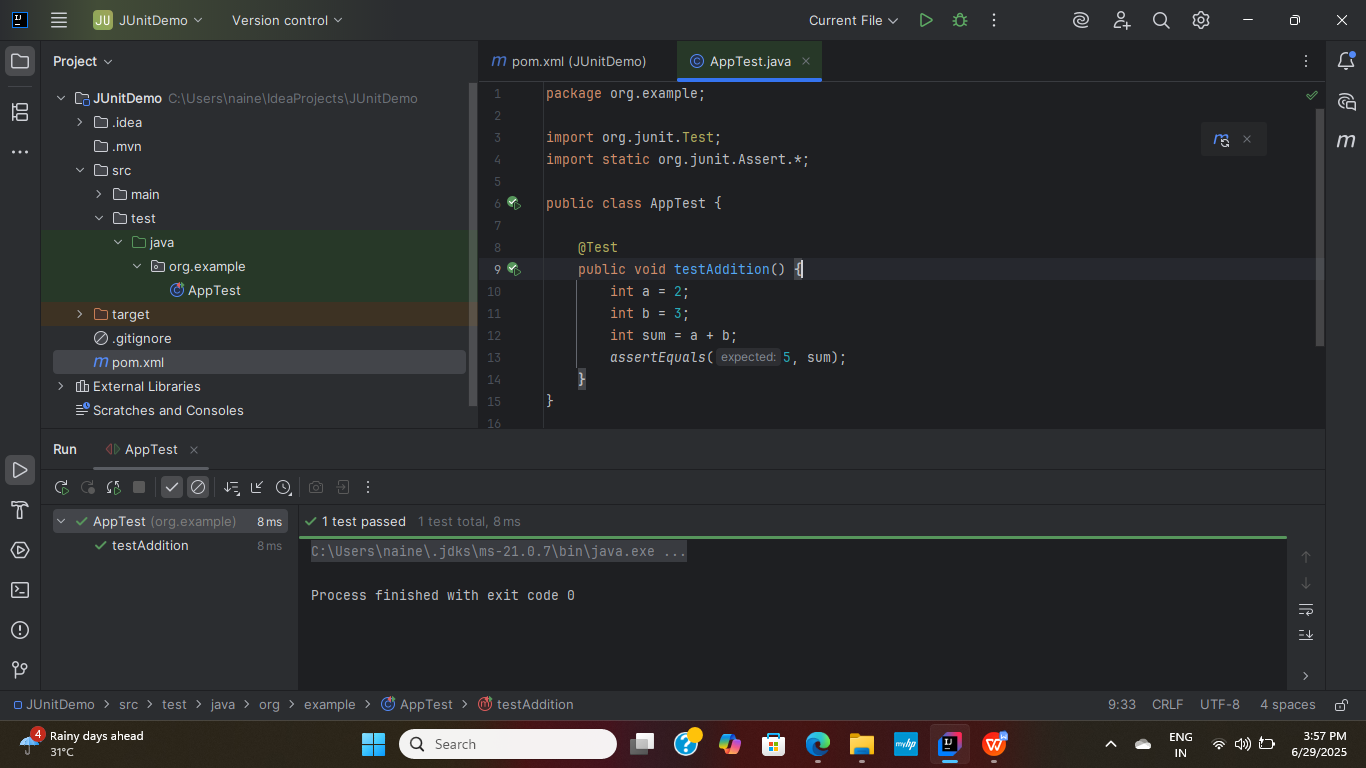
public void testAddition() {

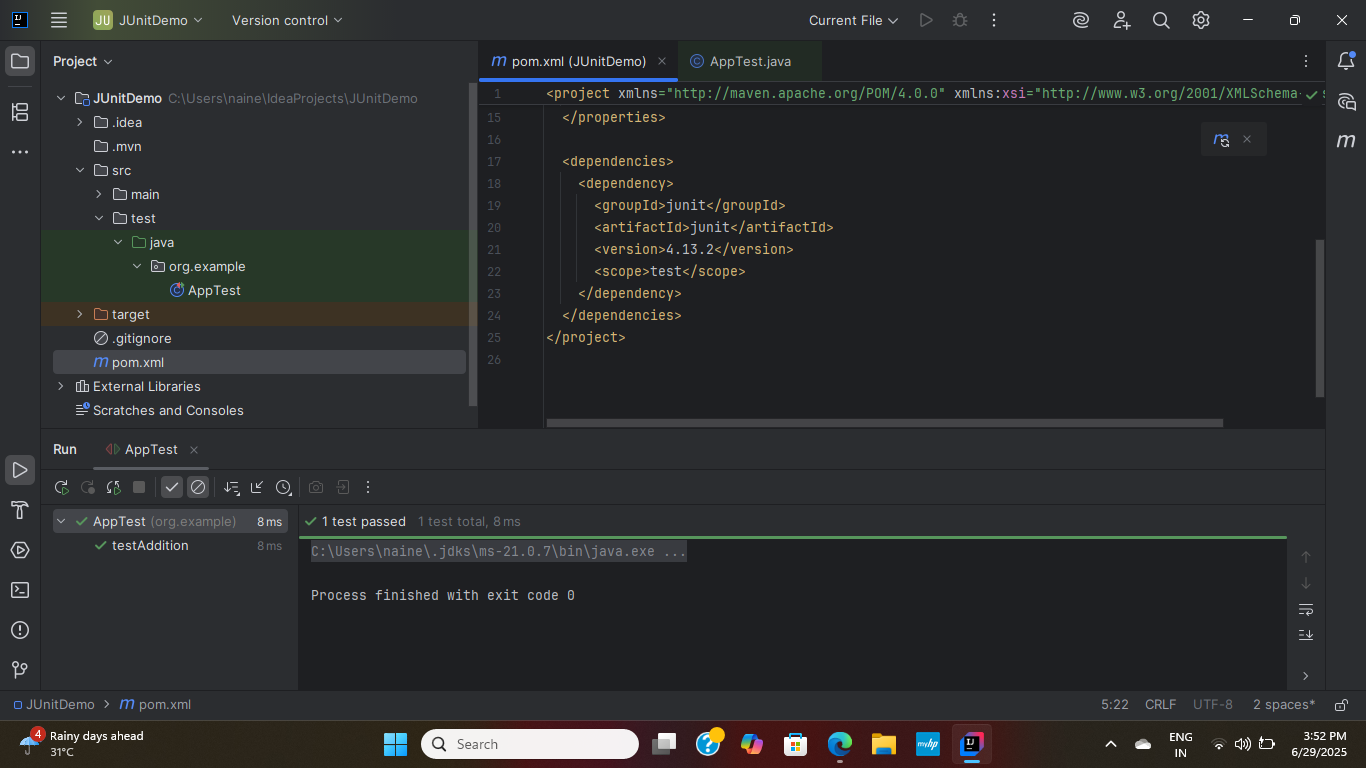
assertEquals(5, 2 + 3);

}

}

**OUTPUT:**

****

****

**3)Exercise 3: Assertions in JUnit**

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

**Solution Code:**

Java

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

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);

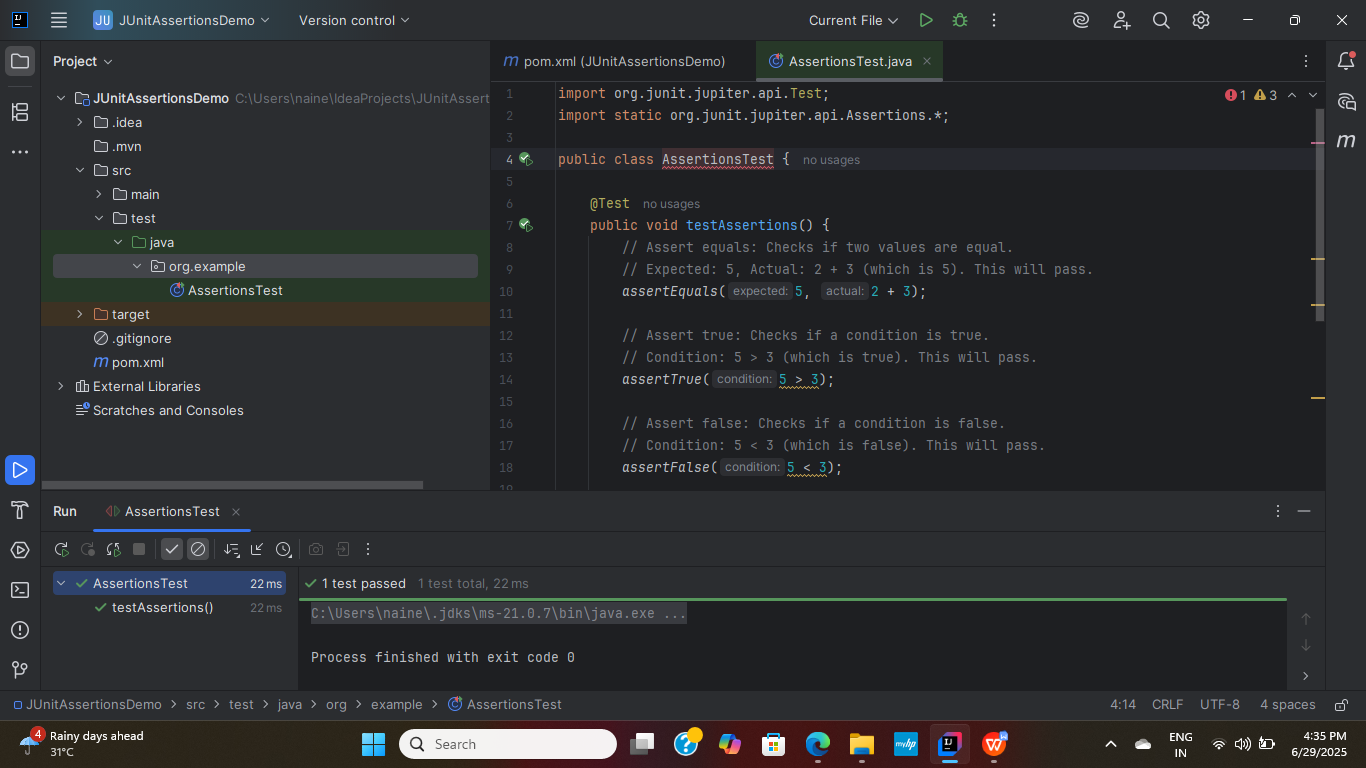
// Assert not null

assertNotNull(new Object());

}

}

**OUTPUT:**

****

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

**STEP1:** The class to be tested (Calculator.java)

Code:

package org.example; // Adjust package name as per your project structure

/\*\*

\* A simple Calculator class to demonstrate unit testing concepts.

\*/

public class Calculator {

/\*\*

\* Adds two integers.

\* @param a The first integer.

\* @param b The second integer.

\* @return The sum of a and b.

\*/

public int add(int a, int b) {

return a + b;

}

/\*\*

\* Subtracts the second integer from the first.

\* @param a The first integer.

\* @param b The second integer.

\* @return The result of a - b.

\*/

public int subtract(int a, int b) {

return a - b;

}

// You can add more methods here if needed for further tests

}

**Solution Code which demonstrates AAA pattern:**

package org.example; // Adjust package name as per your project structure

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

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

/\*\*

\* Test class for the Calculator, demonstrating Arrange-Act-Assert (AAA) pattern

\* and JUnit 5 setup/teardown methods (@BeforeEach, @AfterEach).

\*/

public class CalculatorTest {

private Calculator calculator; // This is our test fixture

/\*\*

\* Setup method: Executed before EACH test method.

\* Used to initialize common objects or set up the test environment.

\* This ensures each test runs with a fresh, isolated instance of Calculator.

\*/

@BeforeEach

void setUp() {

System.out.println("Setting up Calculator instance...");

calculator = new Calculator(); // Initialize the test fixture

}

/\*\*

\* Teardown method: Executed after EACH test method.

\* Used to clean up resources, reset state, or perform any necessary cleanup.

\* For simple objects like Calculator, it might just be a log or setting to null.

\*/

@AfterEach

void tearDown() {

System.out.println("Tearing down Calculator instance...");

calculator = null; // Dereference the object (helps with garbage collection)

}

/\*\*

\* Test method for the add operation, demonstrating the AAA pattern.

\* AAA stands for:

\* 1. Arrange: Set up the test data, objects, and preconditions.

\* 2. Act: Perform the action or call the method being tested.

\* 3. Assert: Verify the outcome of the action using assertions.

\*/

@Test

void testAdd() {

// 1. Arrange: Prepare the test data and environment.

// The 'calculator' instance is already set up by @BeforeEach.

int numberA = 5;

int numberB = 3;

int expectedSum = 8;

// 2. Act: Execute the method under test.

int actualSum = calculator.add(numberA, numberB);

// 3. Assert: Verify the result.

assertEquals(expectedSum, actualSum, "The add method should correctly sum two numbers.");

System.out.println("testAdd executed.");

}

/\*\*

\* Another test method for the subtract operation, also using AAA.

\*/

@Test

void testSubtract() {

// 1. Arrange

int numberA = 10;

int numberB = 4;

int expectedDifference = 6;

// 2. Act

int actualDifference = calculator.subtract(numberA, numberB);

// 3. Assert

assertEquals(expectedDifference, actualDifference, "The subtract method should correctly find the difference.");

System.out.println("testSubtract executed.");

}

/\*\*

\* Example of a test that might fail to demonstrate failure output.

\* (Uncomment and run to see a failing test)

\*/

// @Test

// void testAddFailingExample() {

// // Arrange

// int num1 = 10;

// int num2 = 5;

// int expected = 16; // Incorrect expected value

//

// // Act

// int actual = calculator.add(num1, num2);

//

// // Assert

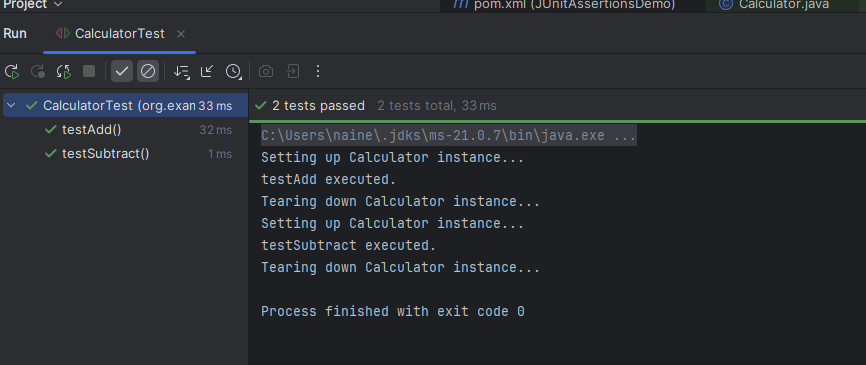
// assertEquals(expected, actual, "This assertion is designed to fail!");

// System.out.println("testAddFailingExample executed.");

// }

}

**OUTPUT:**

****