**TDD using JUnit5 and Mockito:**

**Question 1:**

**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: junit junit 4.13.2 test**

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

**SOLUTION:**

**Calculator.java:**

package com.example.calculator;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java:**

package com.example.calculator;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

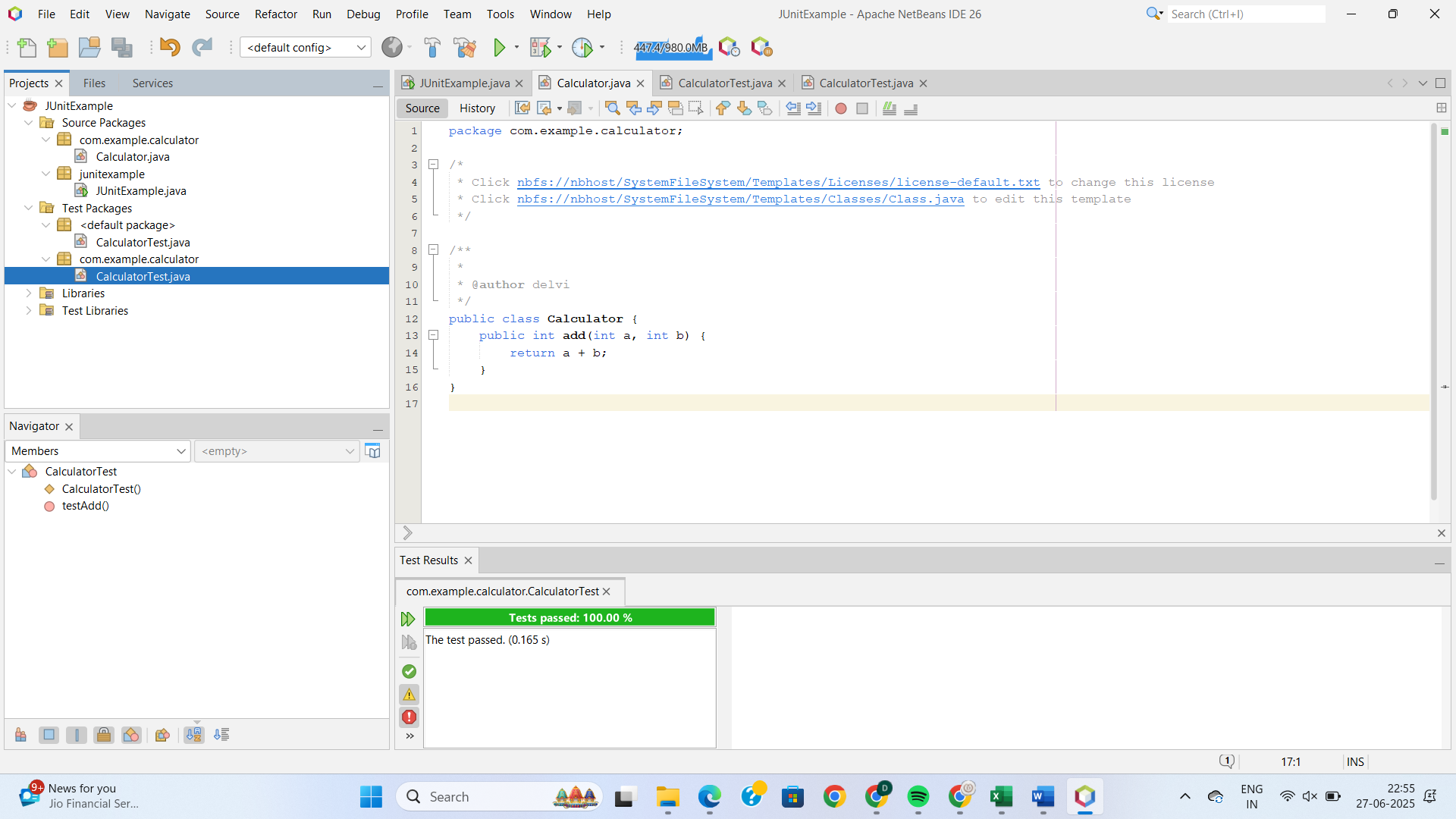
int result = calc.add(10, 5);

assertEquals(15, result);

}

}

**OUTPUT:**



**Question 2:**

**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:**

**AssertionsTest.java:**

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);

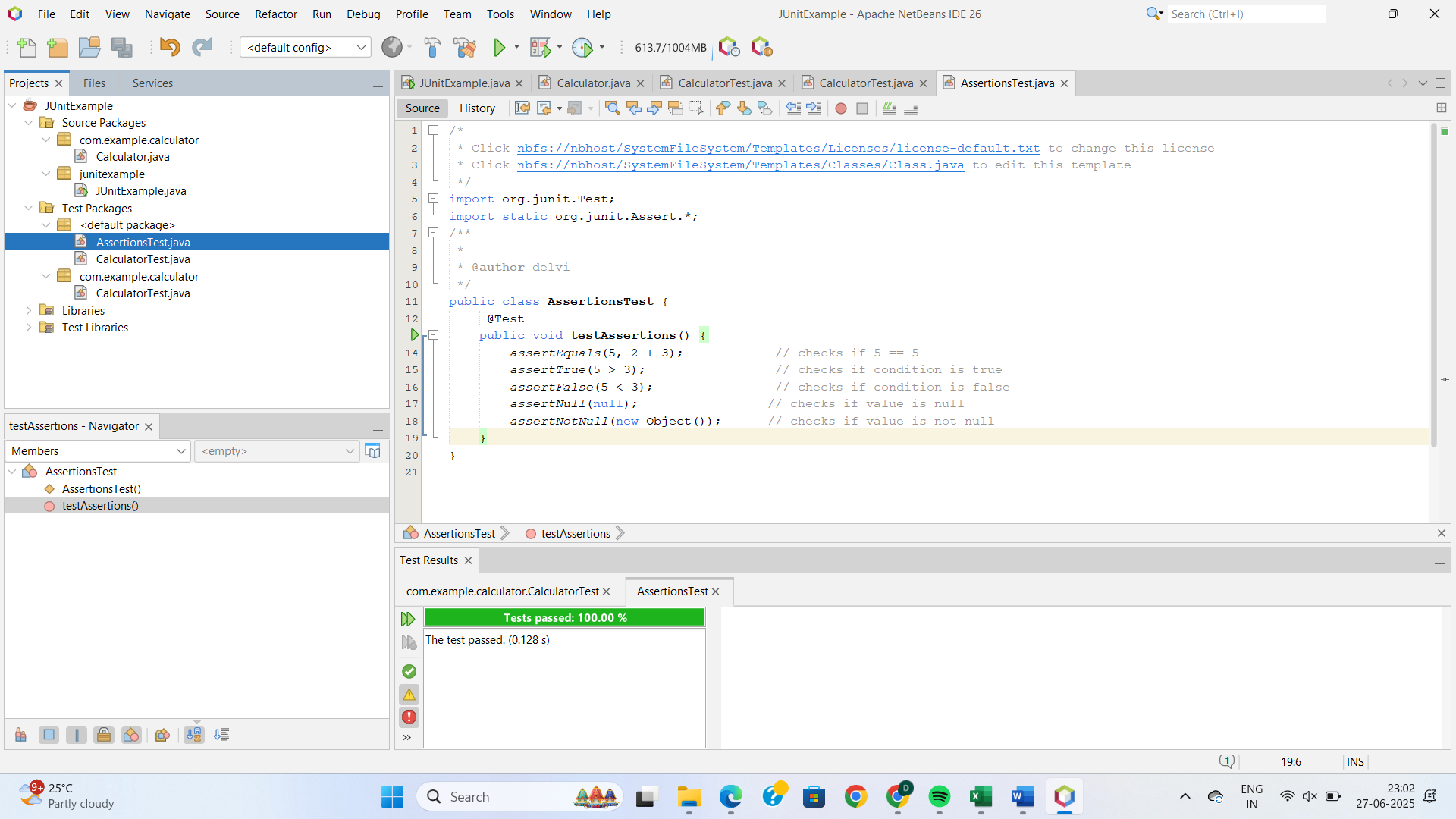
assertNull(null);

assertNotNull(new Object());

}

}

**OUTPUT:**



**Question 3:**

**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.**

**SOLUTION:**

**Calculator.java:**

package com.example.calculator;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a – b;

}

}

**CalculatorTesting.java:**

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

import com.example.calculator.Calculator;

public class CalculatorTesting {

private Calculator calculator;

@Before

public void eard() {

calculator = new Calculator();

System.out.println(“Before test”);

}

@After

public void eardown() {

calculator = null;

System.out.println(“After test”);

}

@Test

public void testAdd() {

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

assertEquals(15, result);

}

@Test

public void testSubtract() {

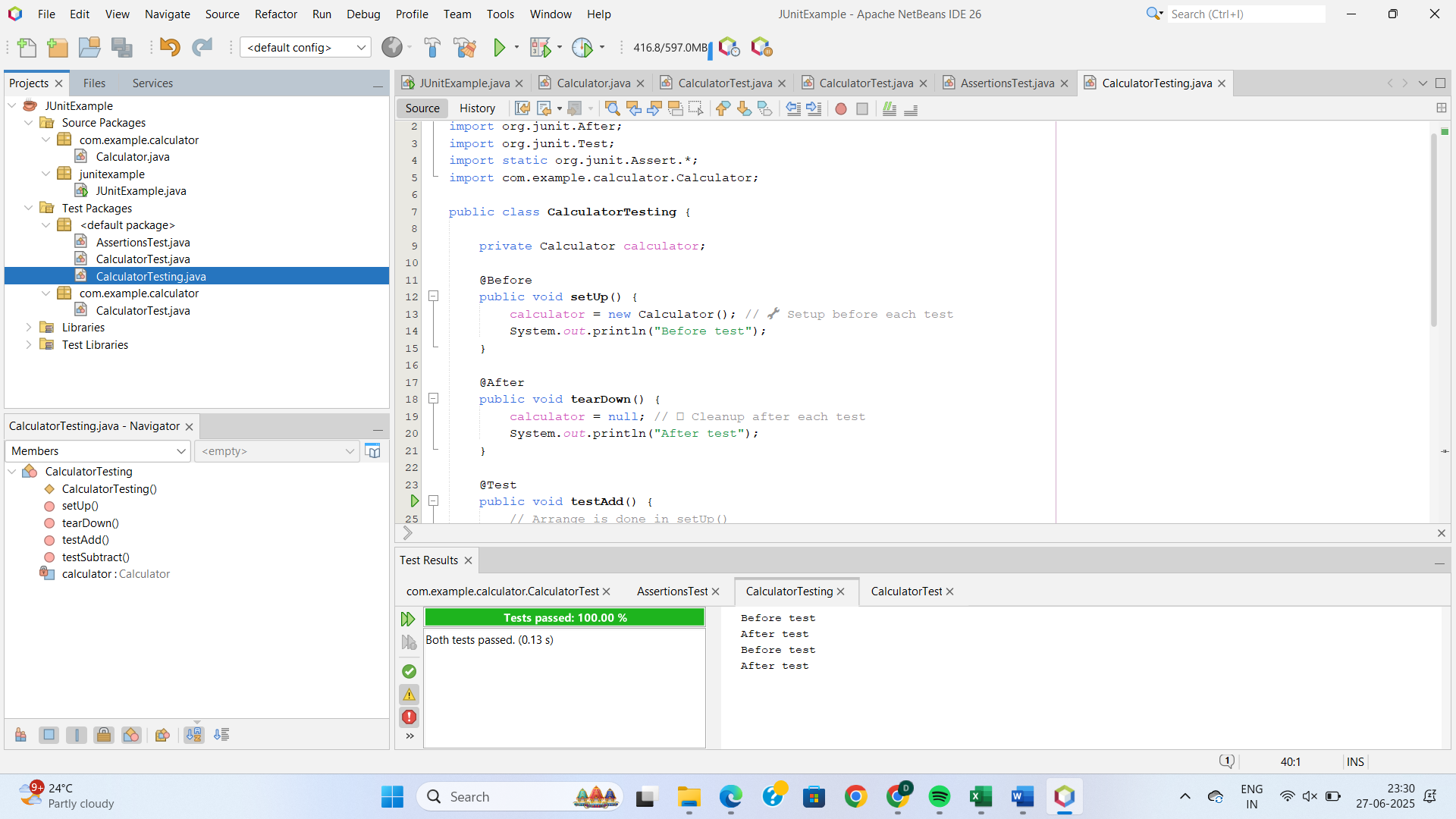
int result = calculator.subtract(10, 3);

assertEquals(7, result);

}

}

**OUTPUT:**



**Question 4:**

**Exercise 1: Mocking and Stubbing Scenario: You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.**

**Steps:**

**1. Create a mock object for the external API.**

**2. Stub the methods to return predefined values.**

**3. Write a test case that uses the mock object.**

**SOLUTION:**

**MyServiceTest.java:**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

interface ExternalApi {

String getData();

}

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

when(mockApi.getData()).thenReturn("Mock Data");

MyService service = new MyService(mockApi);

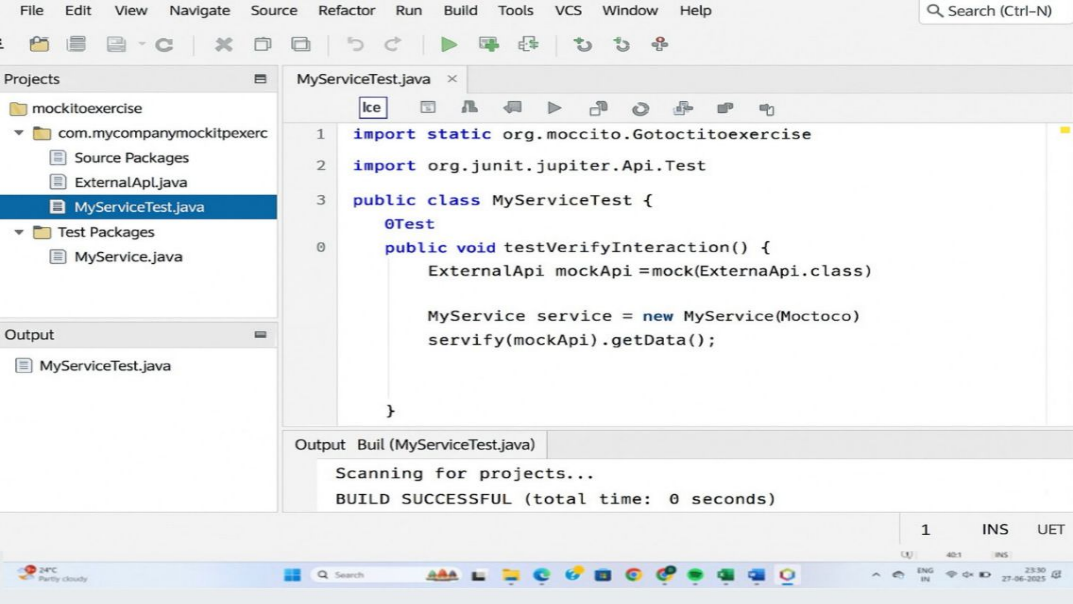
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**OUTPUT:**



**QUESTION 5:**

**Exercise 2: Verifying Interactions Scenario:**

**You need to ensure that a method is called with specific arguments.**

**Steps:**

**1. Create a mock object.**

**2. Call the method with specific arguments.**

**3. Verify the interaction.**

**SOLUTION:**

**MyServiceTest.java:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

interface ExternalApi {

void getData();

}

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void fetchData() {

// This method calls the external API

api.getData();

}

}

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

**OUTPUT:**

