**EXERCISE 1: SETTING UP JUnit**

**SCENARIO:**

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

CODE:

import org.junit.Test;

import static org.junit.Assert.\*;

public class Test {

@Test

public void testAddition() {

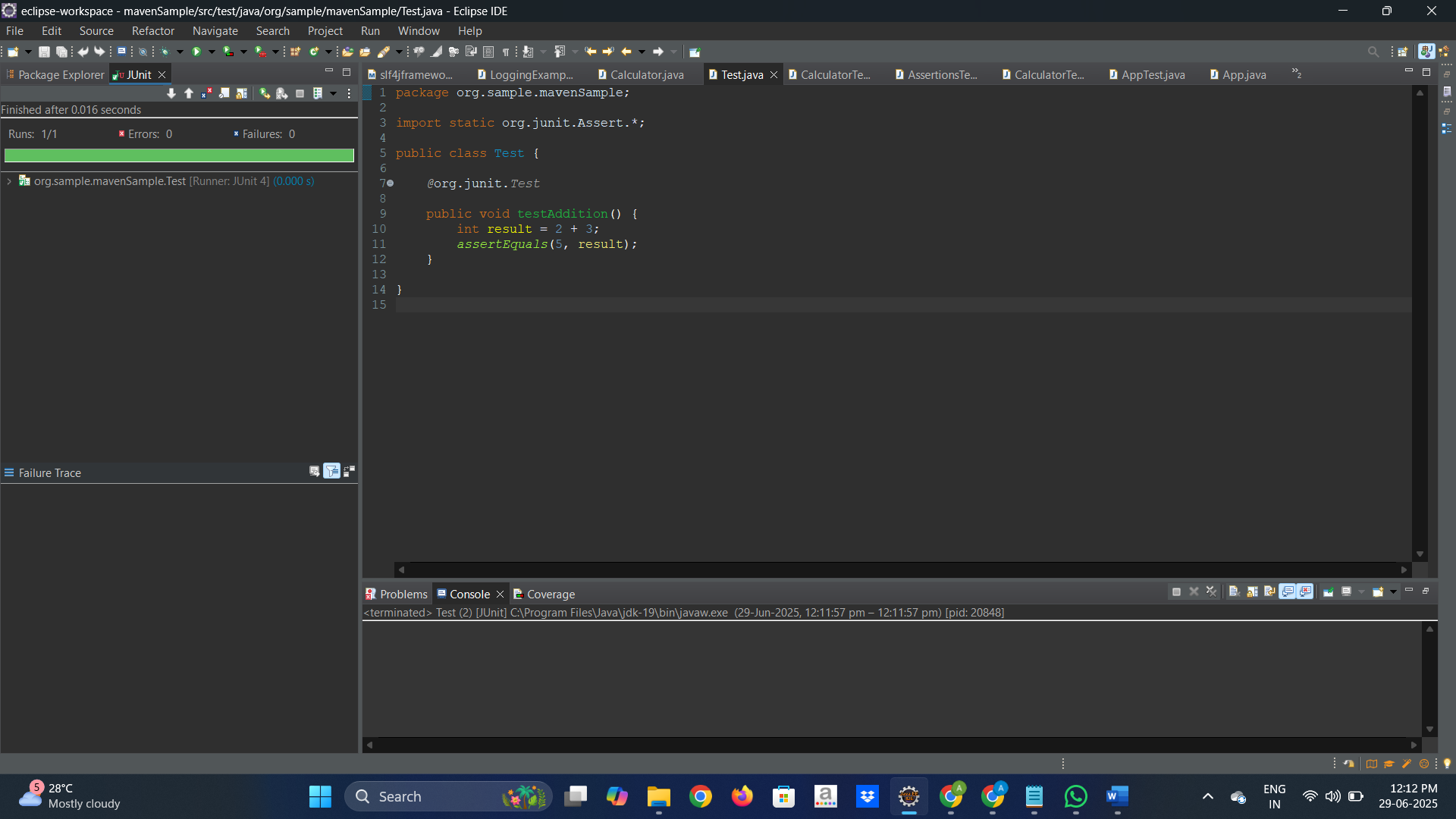
int result = 2 + 3;

assertEquals(5, result);

}

}

OUTPUT:



**EXERCISE 3: ASSERTIONS IN Junit**

**SCENARIO:**

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

**CODING:**

AssertionsTest.java

package com.ahamed.calculator;

import static org.junit.Assert.\*;

import org.junit.Test;

public class AssertionsTest {

@Test

public void testAssertions()

{

assertEquals(5, 2 + 3);

assertTrue(5!=2);

assertFalse(5>10);

Object ob=null;

assertNull(ob);

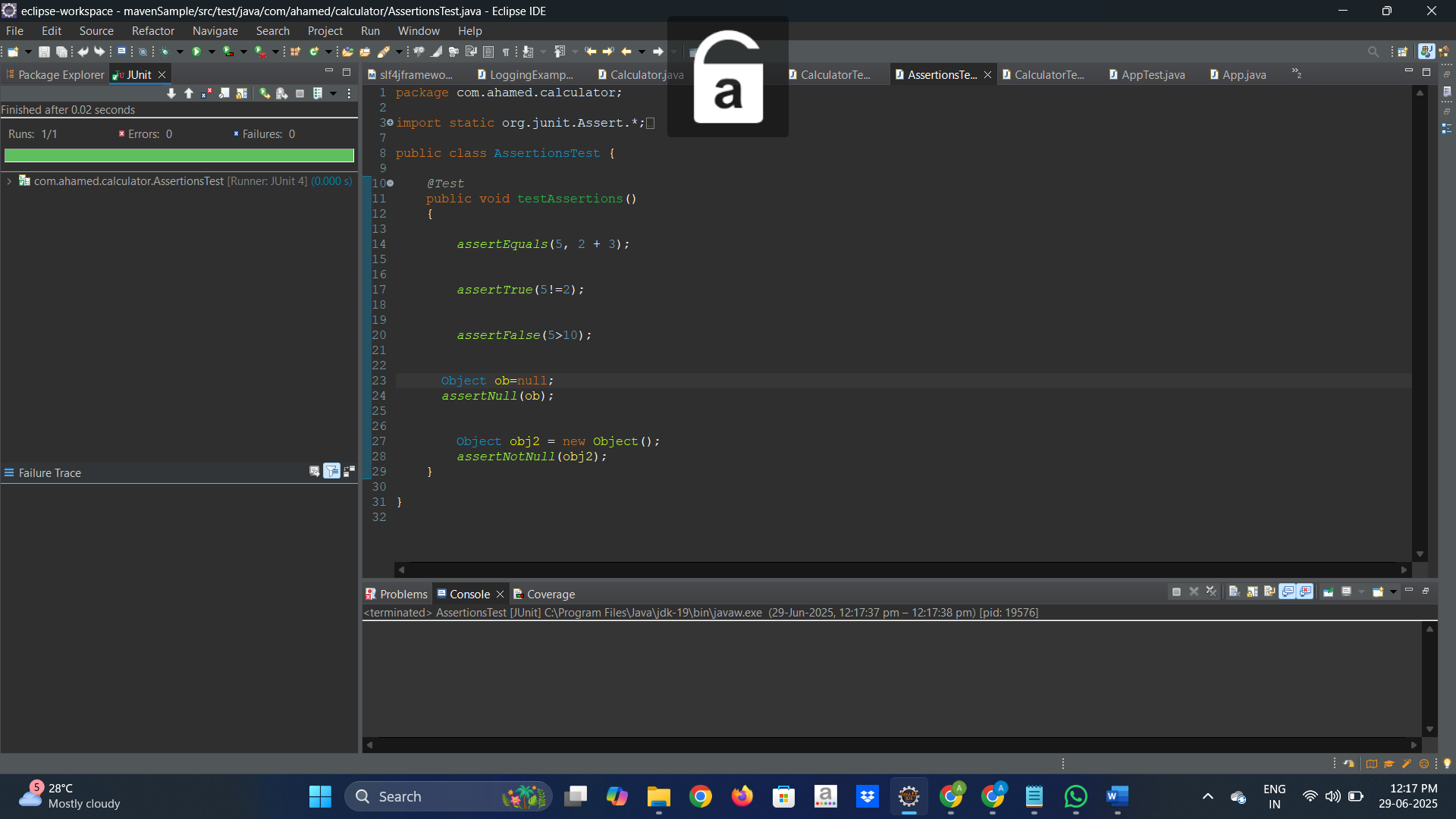
Object obj2 = new Object();

assertNotNull(obj2);

}

}

OUTPUT:



EXERCISE 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setupand

Teardown Methods in JUnit

**SCENARIO:**

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

**CODING:**

Calculator.java

package com.ahamed.calculator;

public class Calculator {

public int add(int a,int b)

{

return a+b;

}

public int sub(int a,int b)

{

return a-b;

}

public int mul(int a,int b)

{

return a\*b;

}

public int div(int a,int b)

{

if(a==0 || b==0)

{

System.***out***.println("Cannot divide a number by 0 - division error");

}

return a/b;

}

}

**CalculatorTestWithFixture.java**

**package com.ahamed.calculator;**

**import org.junit.Before;**

**import org.junit.Test;**

**import static org.junit.Assert.\*;**

**import org.junit.After;**

**public class CalculatorTestWithFixture {**

**private Calculator calculator;**

**@Before**

**public void setUp()**

**{**

**calculator=new Calculator();**

**System.out.println("Setting up the Test");**

**}**

**@Test**

**public void testAddition()**

**{**

**int res=calculator.add(5,3);**

**assertEquals(8,res);**

**}**

**@Test**

**public void testSubtraction()**

**{**

**int ans=calculator.sub(10,5);**

**assertEquals(5,ans);**

**}**

**@Test**

**public void testMultiplication()**

**{**

**int ans=calculator.mul(10,5);**

**assertEquals(50,ans);**

**}**

**@Test**

**public void testDivision()**

**{**

**int ans=calculator.div(10,5);**

**assertEquals(2,ans);**

**}**

**@After**

**public void tearDown()**

**{**

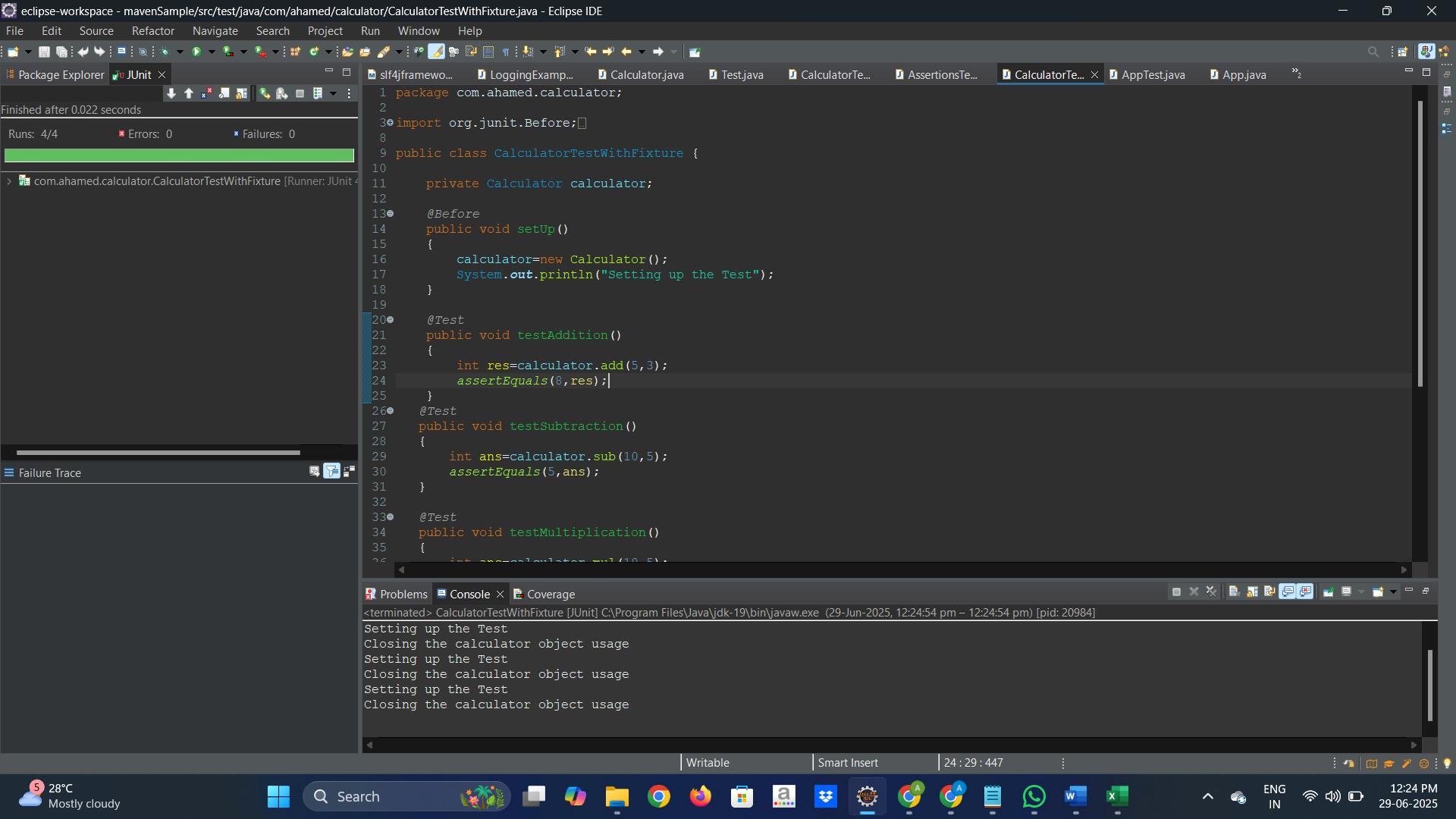
**System.out.println("Closing the calculator object usage");**

**calculator=null;**

**}**

**}**

**OUTPUT:**



**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.

CODING:

**ExternalApi.java**

package com.ahamed.mockitoexample;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.ahamed.mockitoexample;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

MyServiceTest.java

package com.ahamed.mockitoexample;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

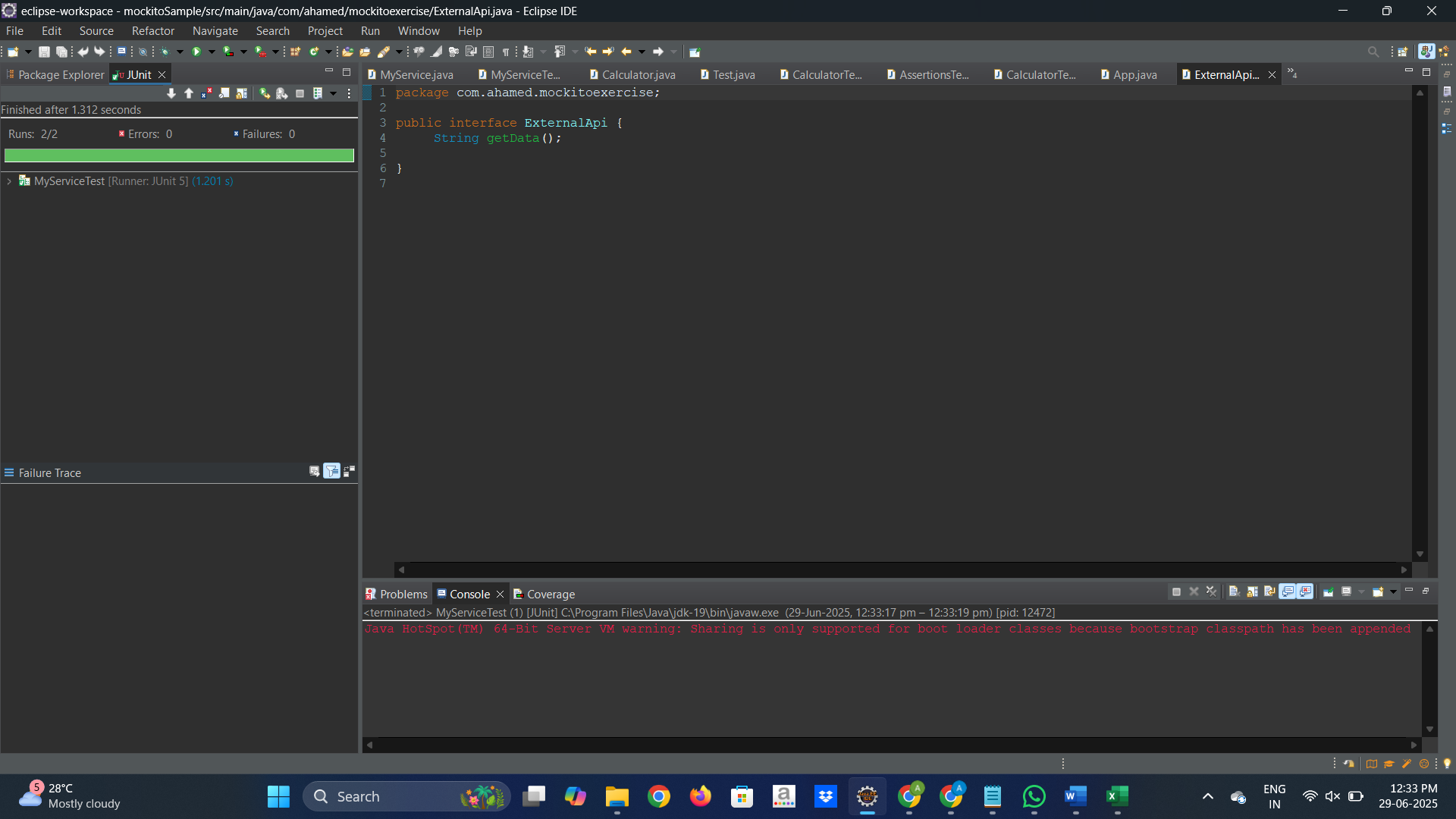
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

OUTPUT



**Exercise 2: Verifying Interactions**

**Scenario:**

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

**CODING:**

**MyServiceTest.java**

**package com.ahamed.mockitoexercise;**

**import org.junit.jupiter.api.Test;**

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

**import static org.mockito.Mockito.\*;**

**public class MyServiceTest {**

***@Test***

**public void testVerifyInteraction() {**

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

**MyService service = new MyService(mockApi);**

**service.fetchData();**

***verify*(mockApi).getData();**

**}**

**}}**

**OUTPUT:**

