**Cognizant Deep Skilling - Digital Nurture 4.0**

**JUnit Exercises:**

**Exercise 1: Setting up JUnit**

**Scenario:**

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

**Calculator.java**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java**

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

*@Test*

public void testAdd() {

Calculator calc = new Calculator();

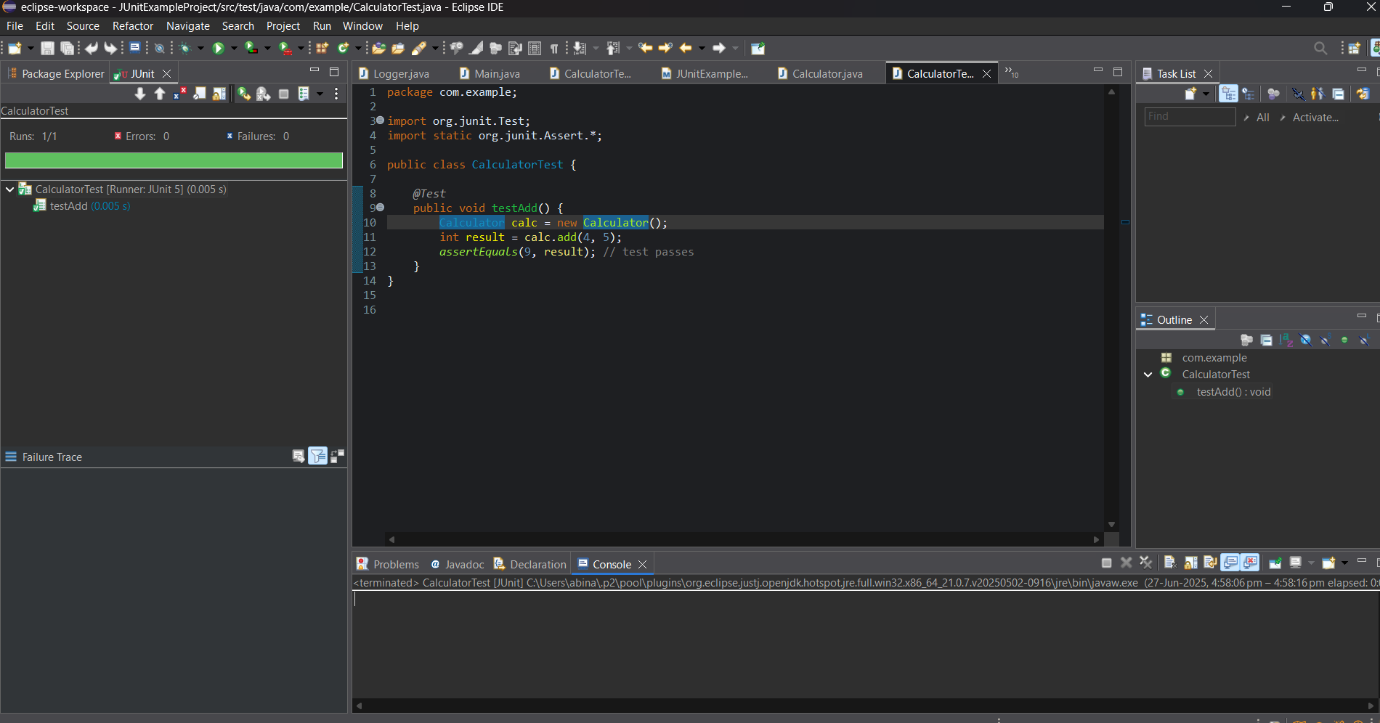
int result = calc.add(4, 5);

*assertEquals*(9, result); // test passes

}

}

**Output:**

****

**Exercise 3: Assertions in JUnit**

**Scenario:**

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

**AssertionsTest.java:**

package com.example;

import org.junit.jupiter.api.Test;

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

public class AssertionsTest {

*@Test*

void testAssertions() {

// Check if 2 + 3 = 5

*assertEquals*(5, 2 + 3);

// Check if 5 is greater than 3

*assertTrue*(5 > 3);

// Check if 5 is not less than 3

*assertFalse*(5 < 3);

// Check if null is null

*assertNull*(null);

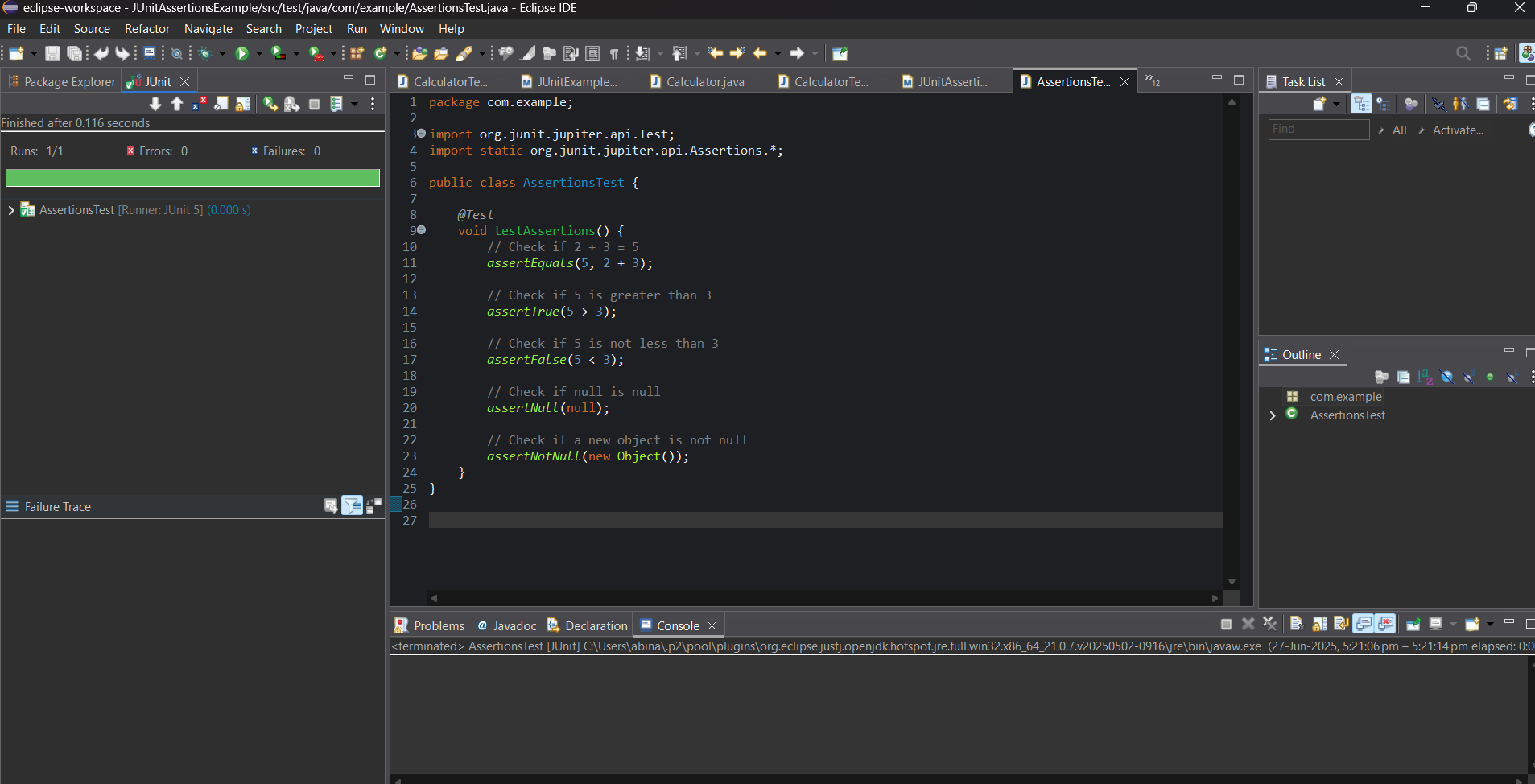
// Check if a new object is not null

*assertNotNull*(new Object());

}

}

**Output:**

****

**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.

**Calculator.java**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public void reset() {

// Placeholder for future logic (simulate cleanup)

}

}

**CalculatorTest.java**

package com.example;

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

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

public class CalculatorTest {

private Calculator calculator;

*@BeforeEach*

void setUp() {

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

calculator = new Calculator(); // Setup

}

*@AfterEach*

void tearDown() {

System.***out***.println("Cleaning up...");

calculator.reset(); // Simulated cleanup

calculator = null;

}

*@Test*

void testAddition() {

// Arrange

int a = 4;

int b = 5;

// Act

int result = calculator.add(a, b);

// Assert

*assertEquals*(9, result);

}

*@Test*

void testAdditionWithNegative() {

// Arrange

int a = -2;

int b = 5;

// Act

int result = calculator.add(a, b);

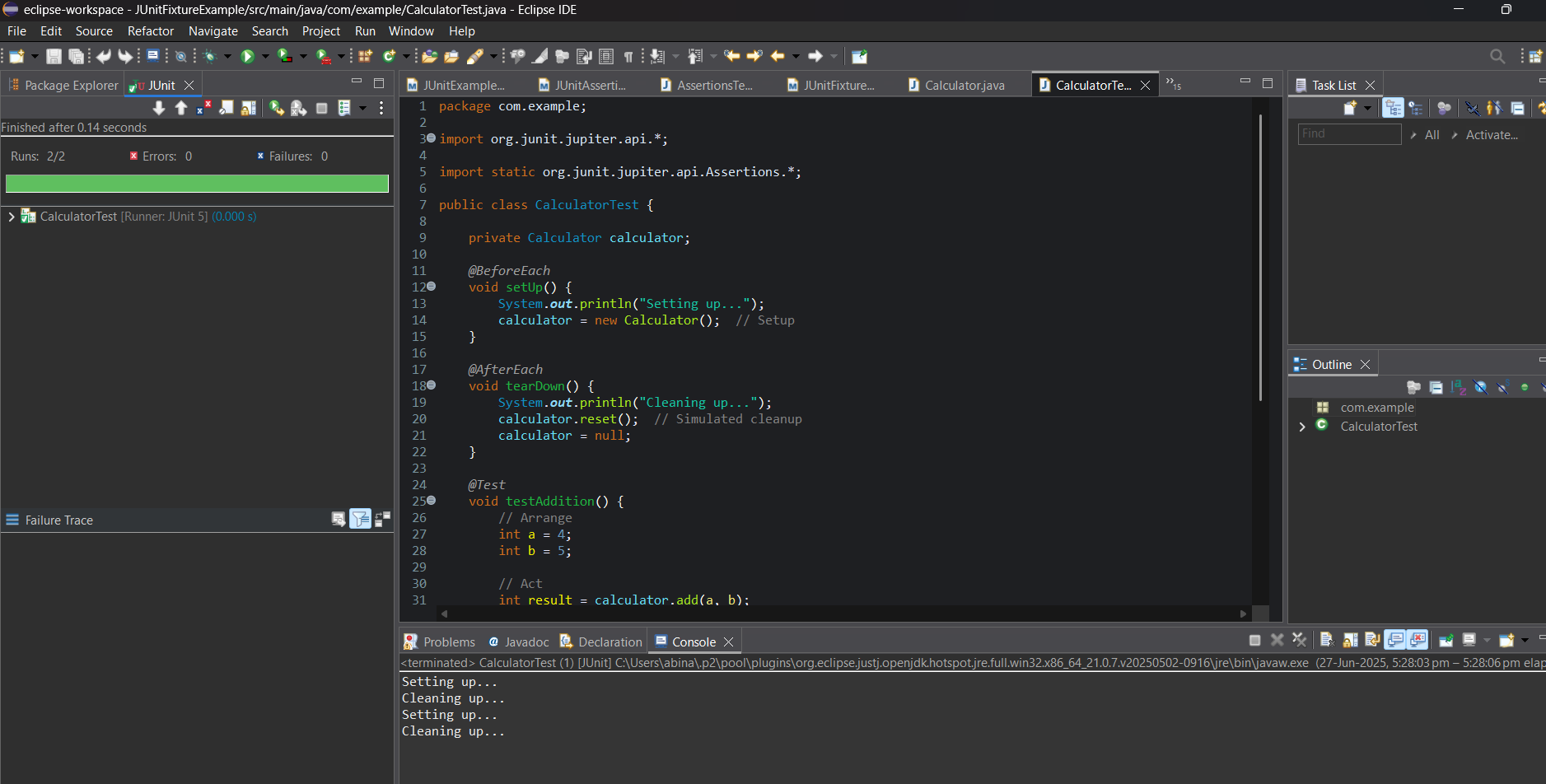
// Assert

*assertEquals*(3, result);

}

}

**Output:**



**Mockito Exercises:**

**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

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

System.out.println("Data fetcing from services");

return api.getData();

}

}

**MyServiceTest.java**

package com.example;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

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();

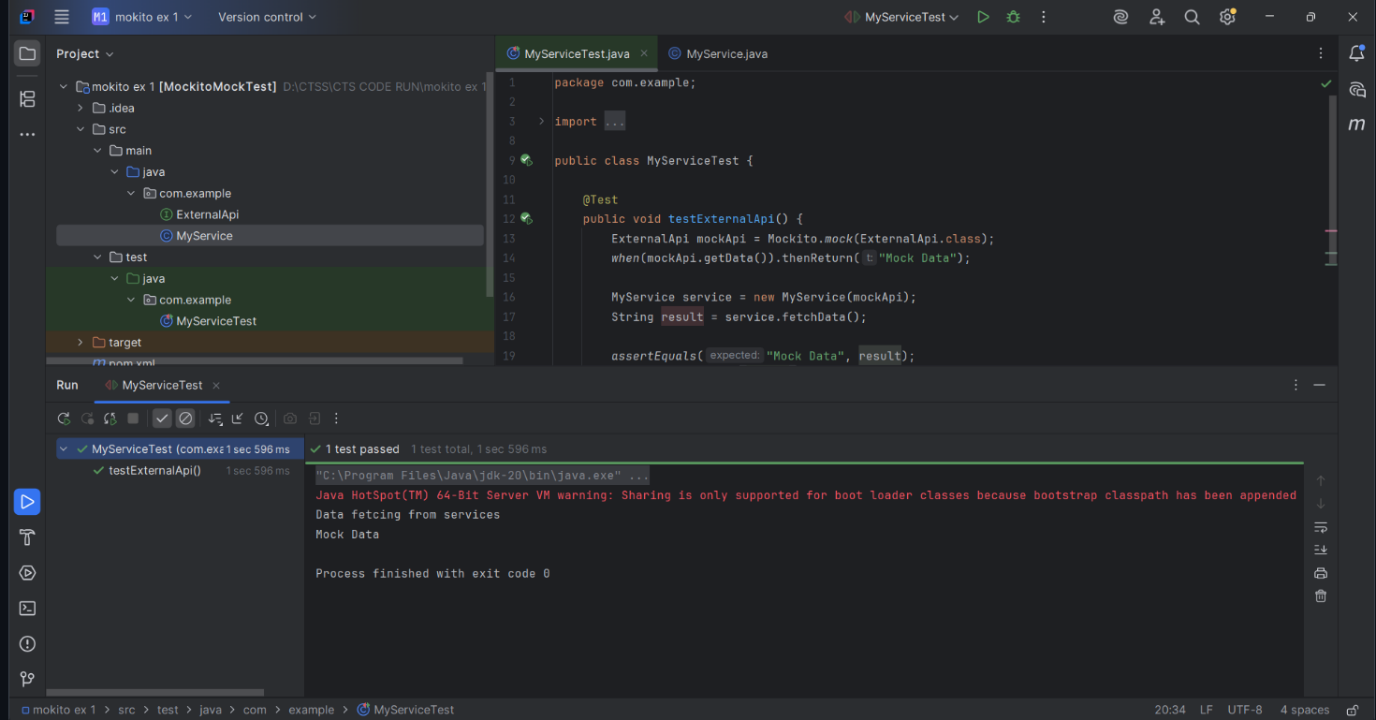
assertEquals("Mock Data", result);

System.out.println(result);

}

}

**Output:**



**Exercise 2: Verifying Interactions**

**Scenario:**

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

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

System.out.println("MyService: fetchData() called");

return api.getData();

}

}

**MyserviceTest.java**

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

System.out.println("Test started");

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

MyService service = new MyService(mockApi);

System.out.println("Calling fetchData()...");

service.fetchData();

System.out.println("Verifying getData() was called");

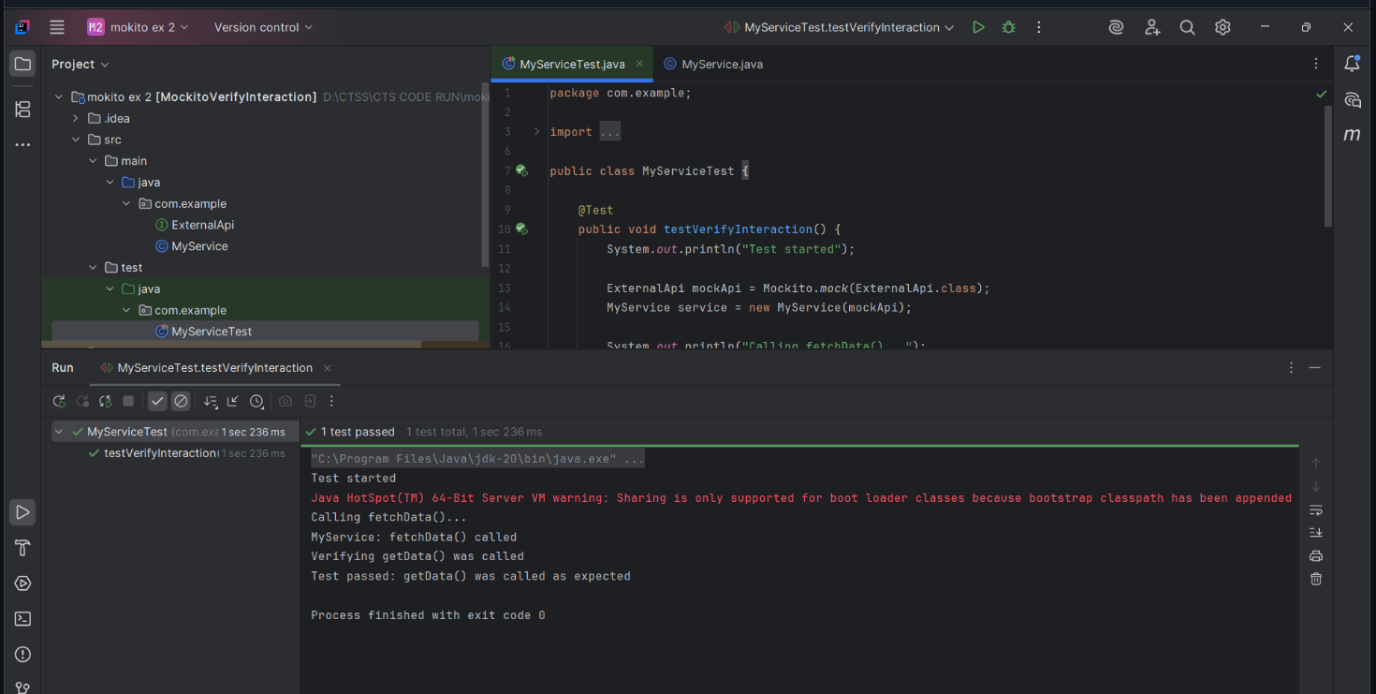
verify(mockApi).getData();

System.out.println("Test passed: getData() was called as expected");

}

}

**Output:**

****