**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(2, 3);

assertEquals(5, result);

}

}

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>JUnitDemo</artifactId>

<packaging>jar</packaging>

<version>1.0-SNAPSHOT</version>

<name>JUnitDemo</name>

<url>http://maven.apache.org</url>

<properties>

<maven.compiler.source>23</maven.compiler.source>

<maven.compiler.target>23</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.11.0</version>

<configuration>

<source>23</source>

<target>23</target>

</configuration>

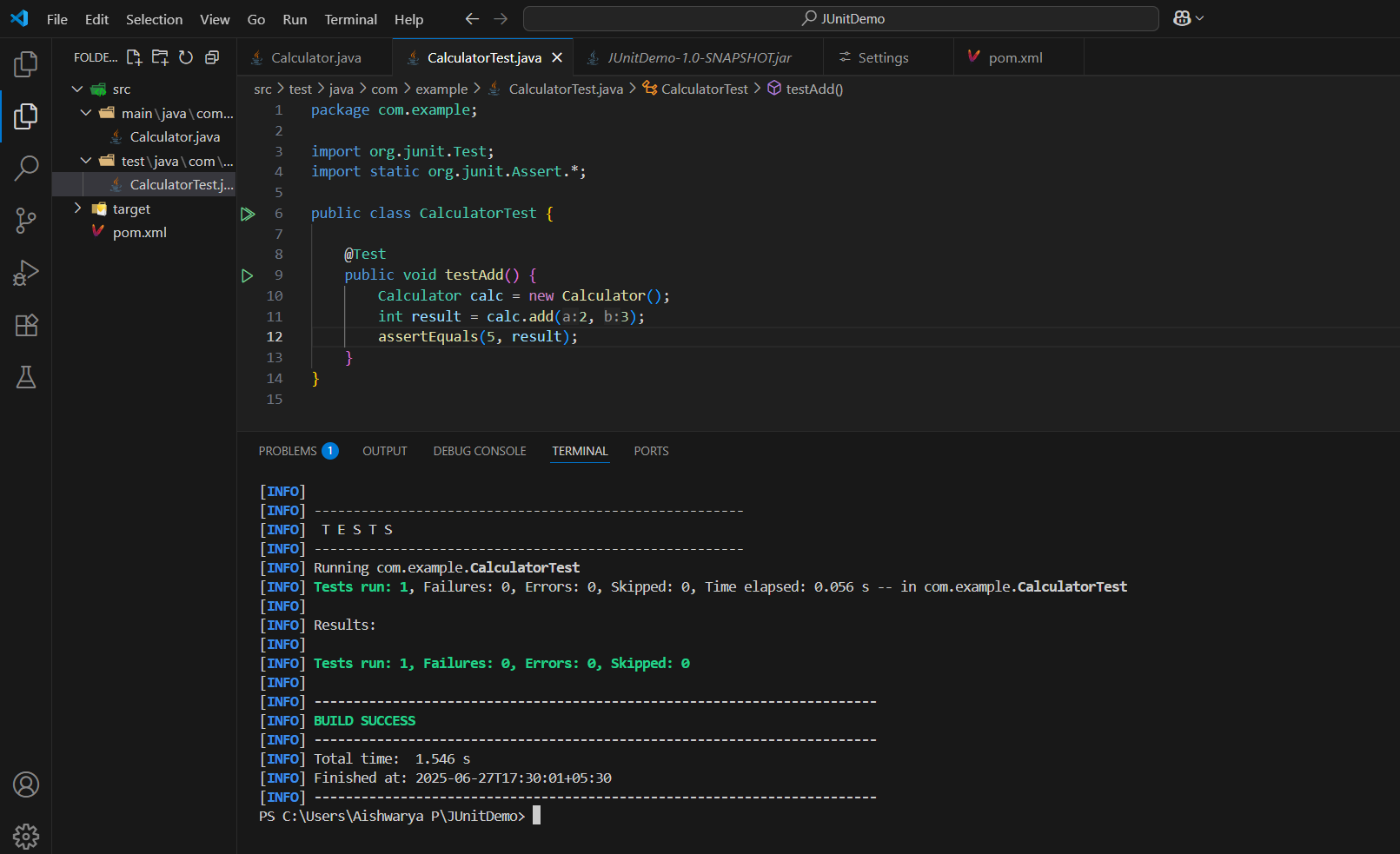
</plugin>

</plugins>

</build>

</project>

**Output:**



**Exercise 3: Assertions in JUnit**

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

**AssertionsTest.java**

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());

    }

}

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

         https://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>

    <artifactId>junit-demo</artifactId>

    <version>1.0-SNAPSHOT</version>

    <name>JUnit Demo</name>

    <properties>

    <maven.compiler.source>23</maven.compiler.source>

    <maven.compiler.target>23</maven.compiler.target>

</properties>

    <dependencies>

        <dependency>

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

            <artifactId>junit-jupiter</artifactId>

            <version>5.10.0</version>

            <scope>test</scope>

        </dependency>

    </dependencies>

    <build>

        <plugins>

            <plugin>

                <groupId>org.apache.maven.plugins</groupId>

                <artifactId>maven-surefire-plugin</artifactId>

                <version>3.0.0-M9</version>

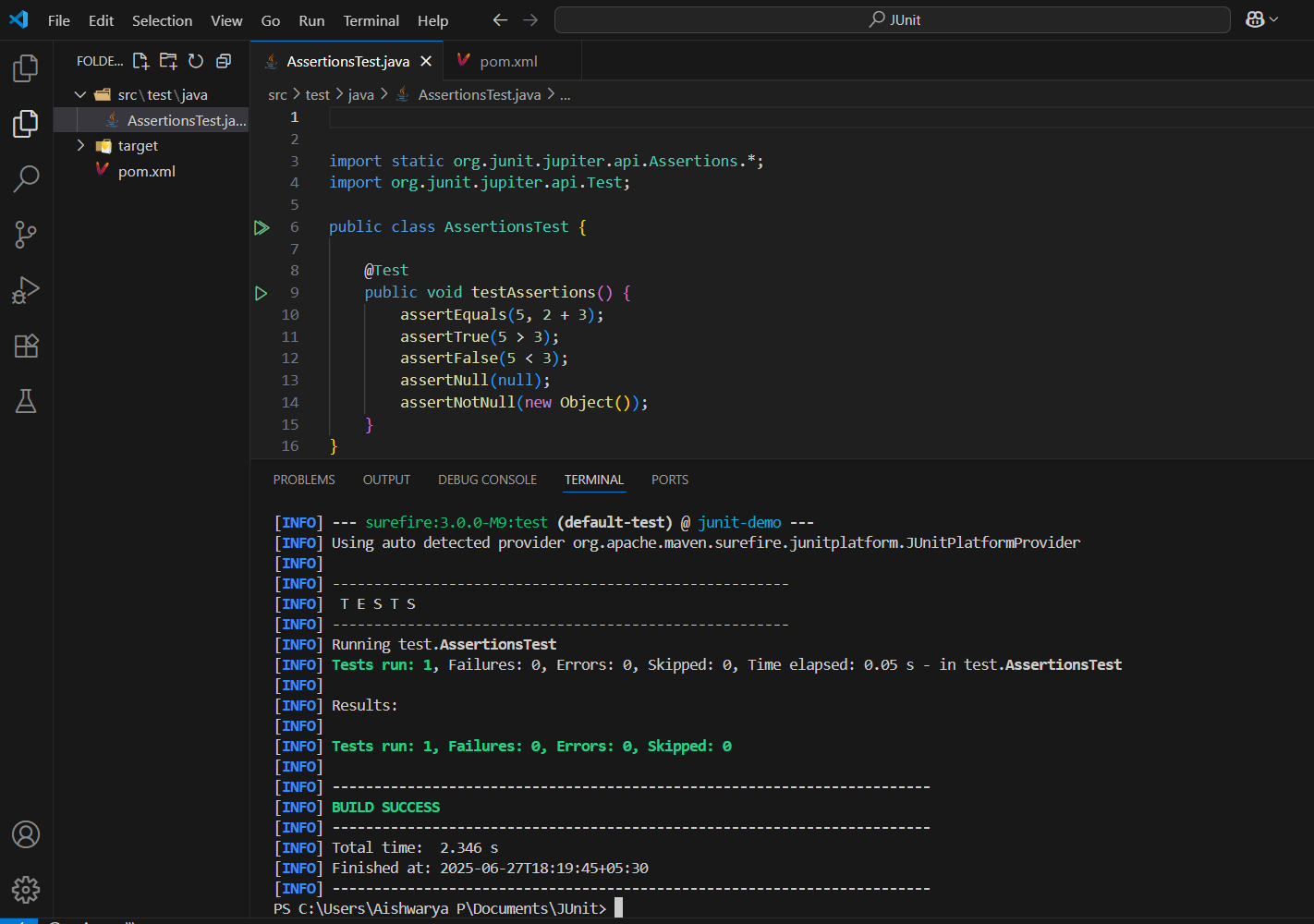
            </plugin>

        </plugins>

    </build>

</project>

**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**

public class Calculator {

    public int add(int a, int b) {

        return a + b;

    }

    public int subtract(int a, int b) {

        return a - b;

    }

}

**CalculatorTest.java**

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

    private Calculator calculator;

    @Before

    public void setUp() {

        calculator = new Calculator();

        System.out.println("Setup complete.");

    }

    @After

    public void tearDown() {

        calculator = null;

        System.out.println("Teardown complete.");

    }

    @Test

    public void testAddition() {

        int a = 5, b = 3;

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

        assertEquals(8, result);

    }

    @Test

    public void testSubtraction() {

        int a = 10, b = 4;

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

        assertEquals(6, result);

    }

}

**Pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

         http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>

  <artifactId>junit-demo</artifactId>

  <version>1.0-SNAPSHOT</version>

  <dependencies>

    <dependency>

      <groupId>junit</groupId>

      <artifactId>junit</artifactId>

      <version>4.13.2</version>

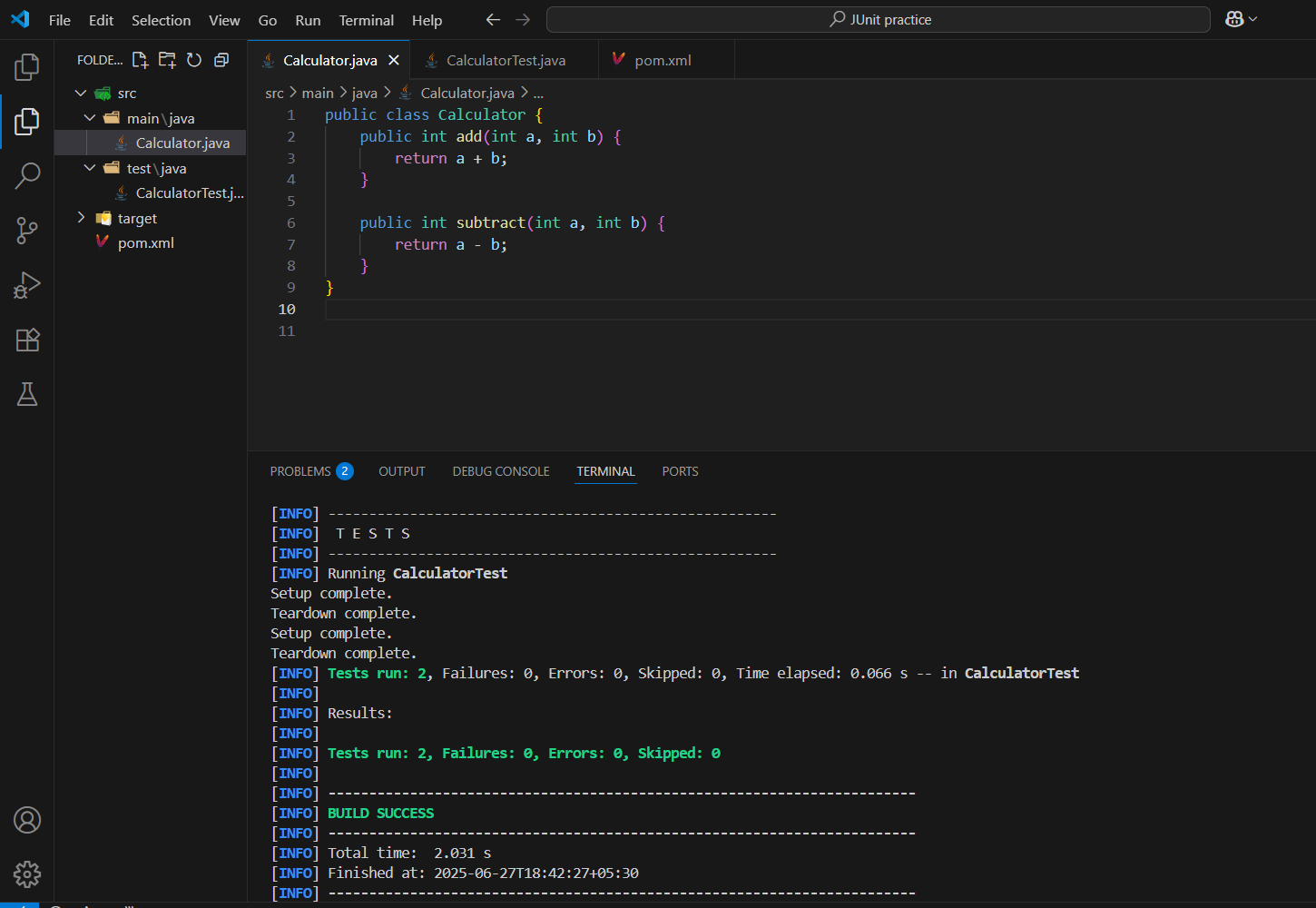
      <scope>test</scope>

    </dependency>

  </dependencies>

</project>

**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();

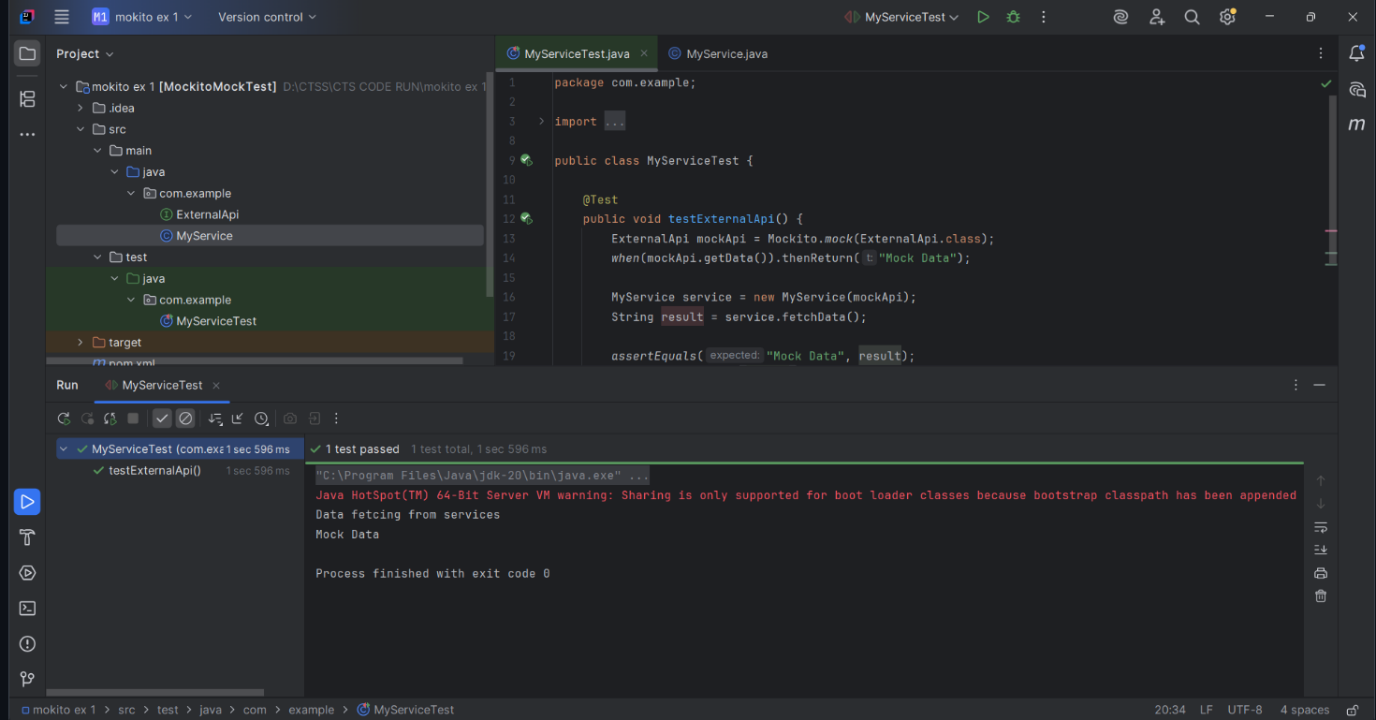
assertEquals("Mock Data", result);

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");

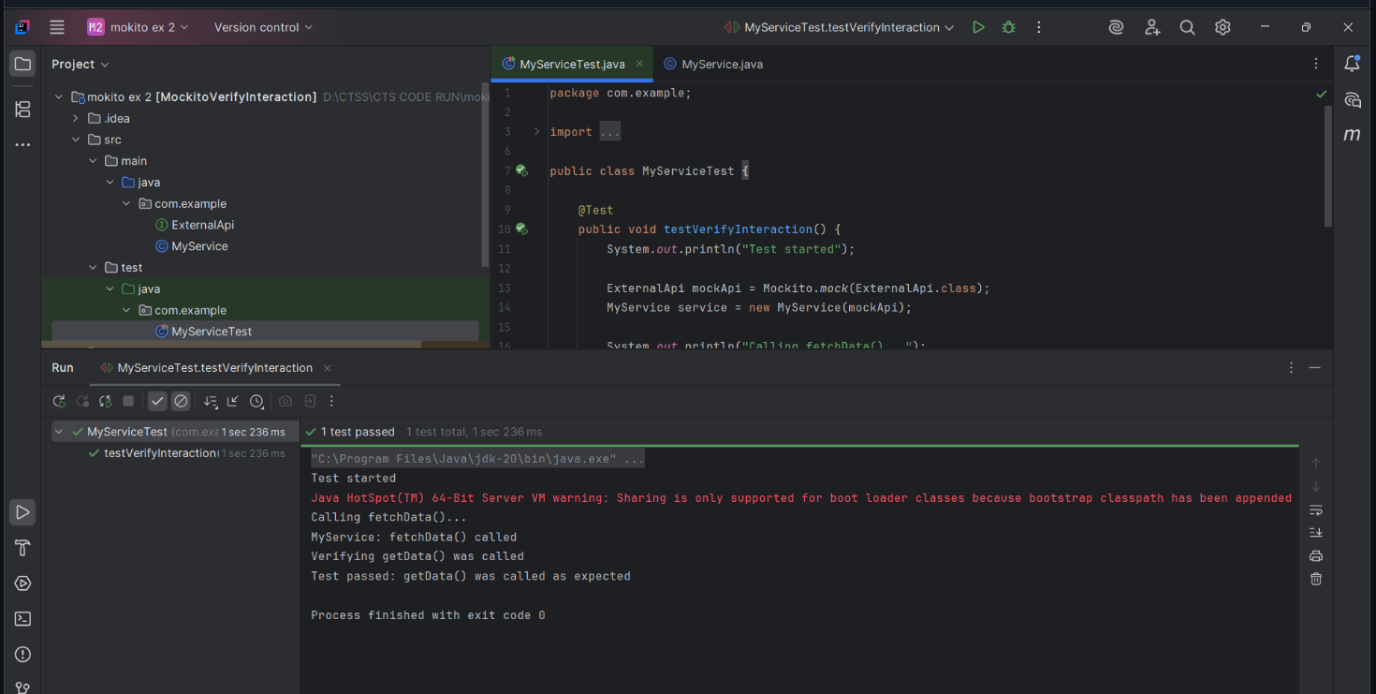
verify(mockApi).getData();

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

}

}

**Output:**

****