**WEEK 2 TDD Using Junit5 and Mockito,Logging framework**

**Question(1. JUnit\_Basic Testing Exercises):**

**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:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

1. Create a new test class in your project.

**Code :**

Message.java:

package com.example;

public class Message {

    public String greet(String name) {

        return "Hello, " + name;

    }

}

Messagetest.java:

package com.example;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class Messagetest {

    @Test

    public void testGreet() {

        Message message = new Message();

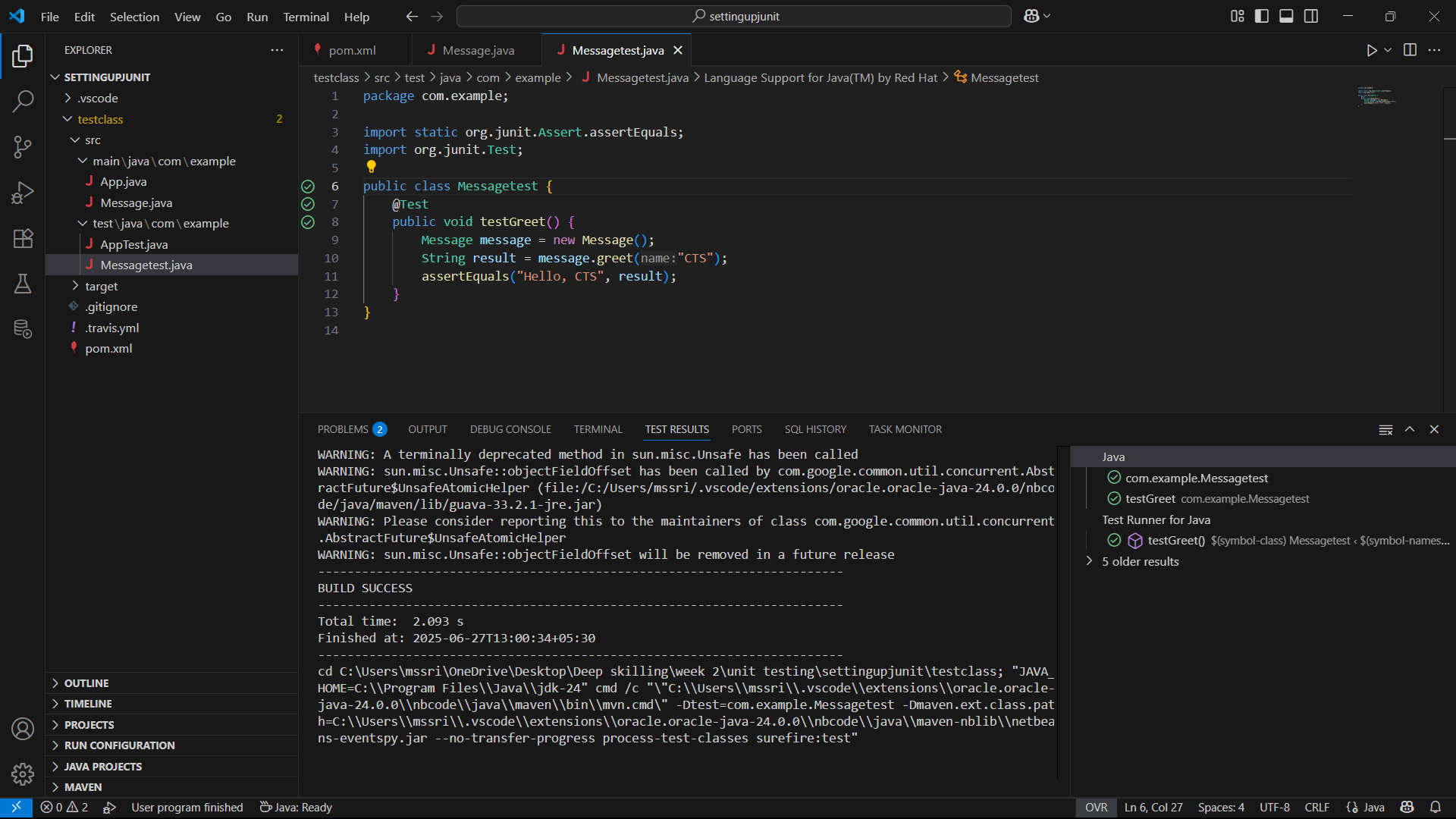
        String result = message.greet("CTS");

        assertEquals("Hello, CTS", result);

    }

}

**Output:**



**Question(1. JUnit\_Basic Testing Exercises):**

**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 Code:

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());

}

}

**Code:**

AssertionsTest.java

package assertions;

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);

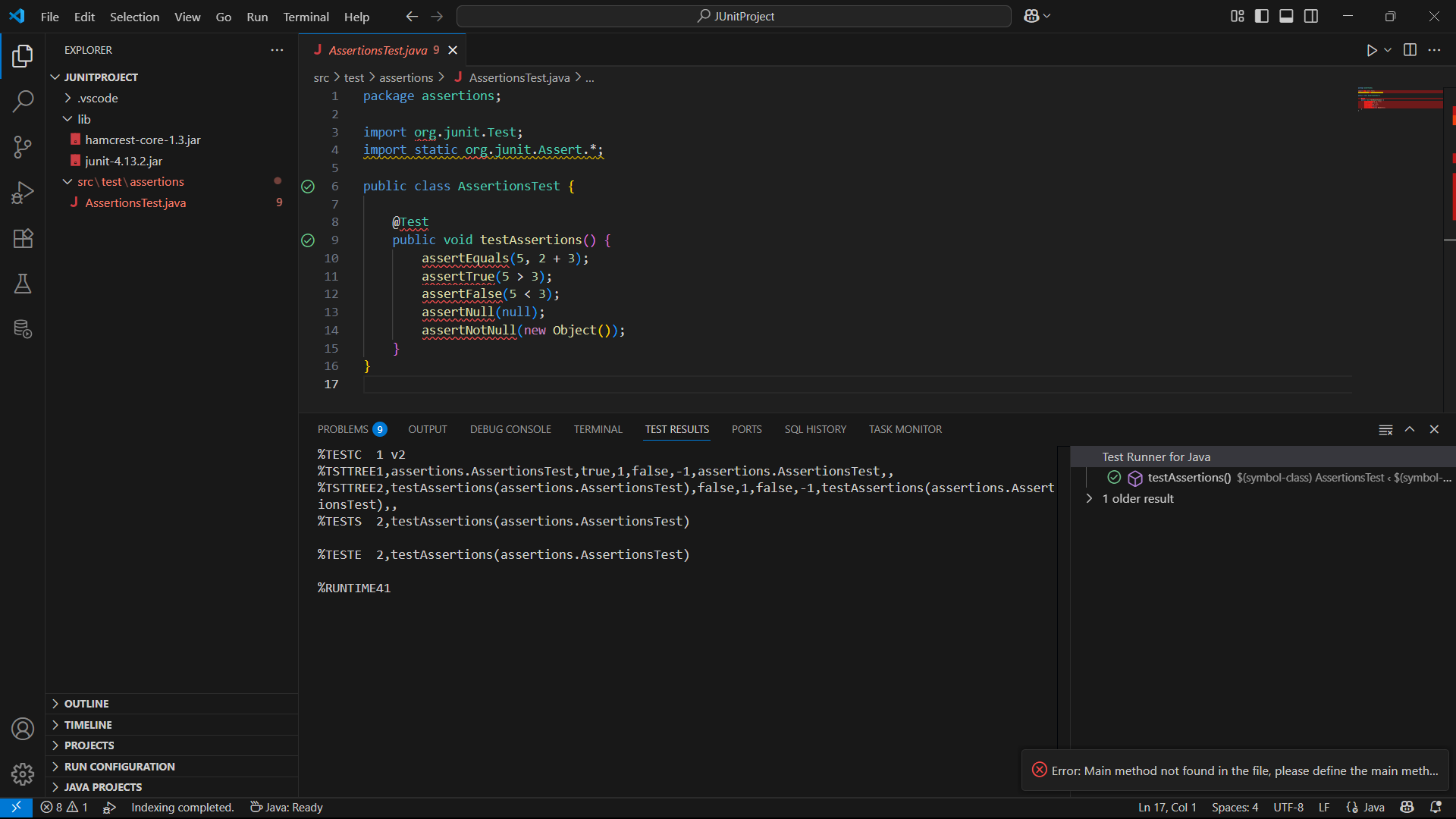
        assertNull(null);

        assertNotNull(new Object());

    }

}

**Output:**



**Question(1. JUnit\_Basic Testing Exercises):**

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.

**Code:**

**AAAcalculatortest.java**

**package com.example;**

**import org.junit.After;**

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

**import org.junit.Before;**

**import org.junit.Test;**

**public class AAACalculatorTest {**

**private Calculator calc;**

**@Before**

**public void setUp() {**

**calc = new Calculator(); // Setup before each test**

**System.out.println("Setup called");**

**}**

**@After**

**public void tearDown() {**

**System.out.println("Teardown called"); // Cleanup**

**}**

**@Test**

**public void testAddition() {**

**int a = 3, b = 2;**

**int result = calc.add(a, b);**

**assertEquals(5, result);**

**}**

**@Test**

**public void testSubtraction() {**

**int a = 7, b = 4;**

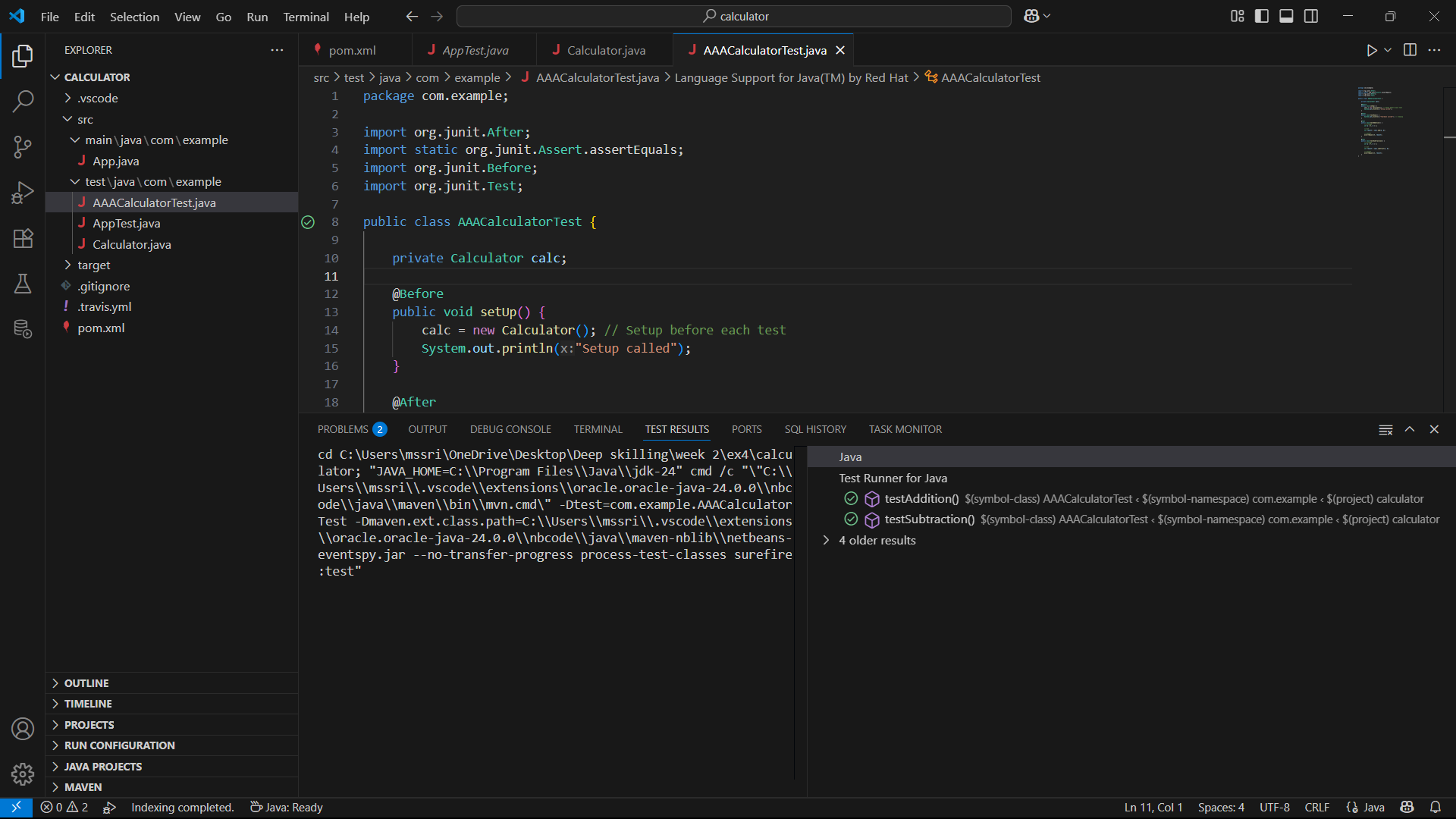
**int result = calc.subtract(a, b);**

**assertEquals(3, result);**

**}**

**}**

**Output:**



**Question(3. 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.**

**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 Code:**

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

**}**

**}**

**Code:**

**Externalapi.java**

**package com.example;**

**public interface ExternalApi {**

**String getData();**

**}**

**MyService.java**

**public class MyService {**

**private ExternalApi api;**

**public MyService(ExternalApi api) {**

**this.api = api;**

**}**

**public String fetchData() {**

**return api.getData();**

**}**

**}**

**MyServiceTest.java**

**package com.example;**

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

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

**import org.mockito.Mockito;**

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

**public class MyServiceTest {**

**@Test**

**public void testExternalApi() {**

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

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

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

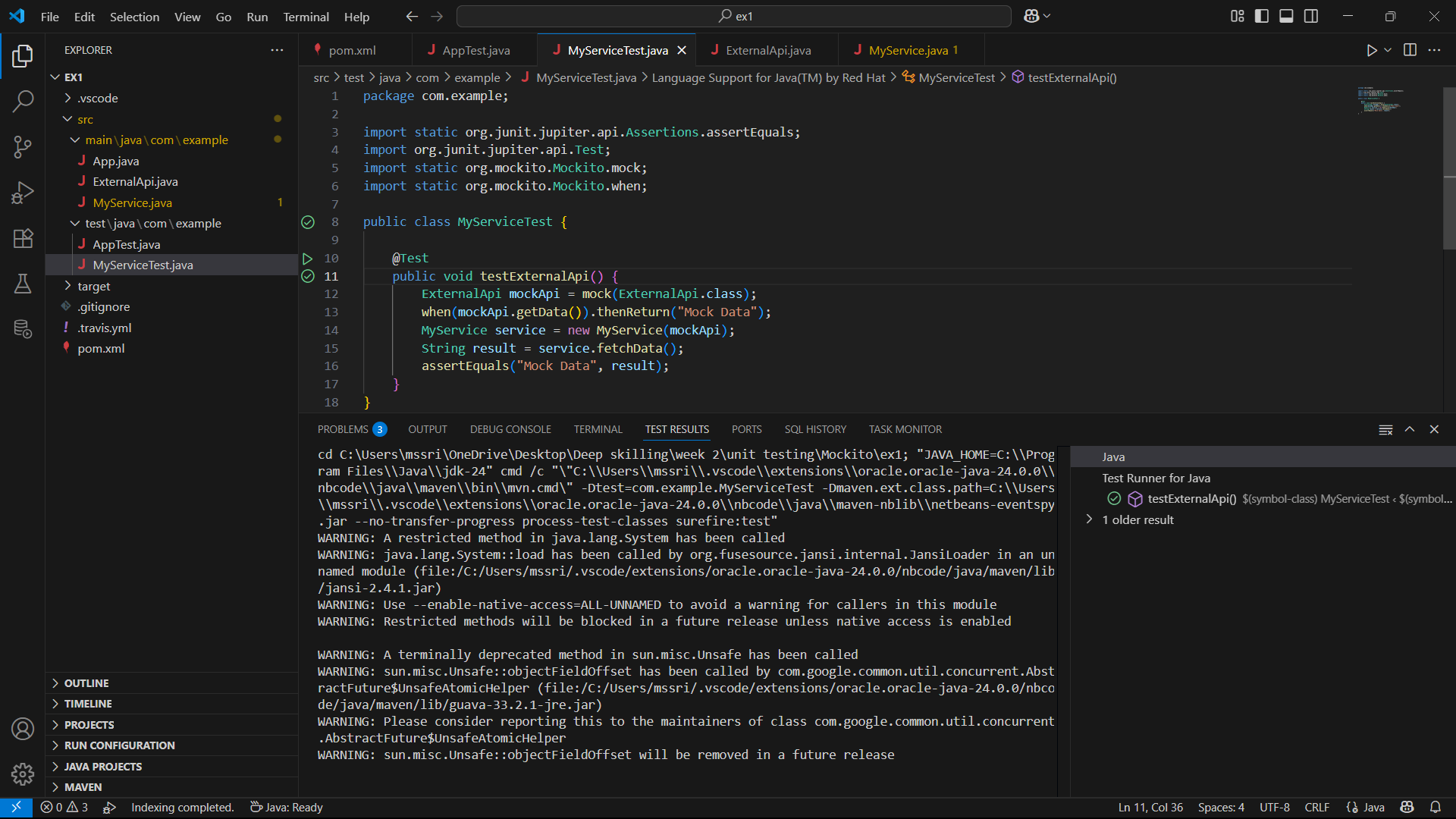
**String result = service.fetchData();**

**assertEquals("Mock Data", result);**

**}**

**}**

**Output:**



**Question(3. Mockito exercises):**

**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 Code:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test; import org.mockito.Mockito;

public class MyServiceTest { @Test

public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class); MyService service = new MyService(mockApi); service.fetchData();

verify(mockApi).getData();

}

}

**Code:**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

    String getData();

}

**MyService.java**

package com.example;

public class MyService {

    private ExternalApi api;

    public MyService(ExternalApi api) {

        this.api = api;

    }

    public String fetchData() {

        return api.getData();

    }

}

**MyServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

    @Test

    public void testVerifyInteraction() {

        ExternalApi mockApi = mock(ExternalApi.class);

        MyService service = new MyService(mockApi);

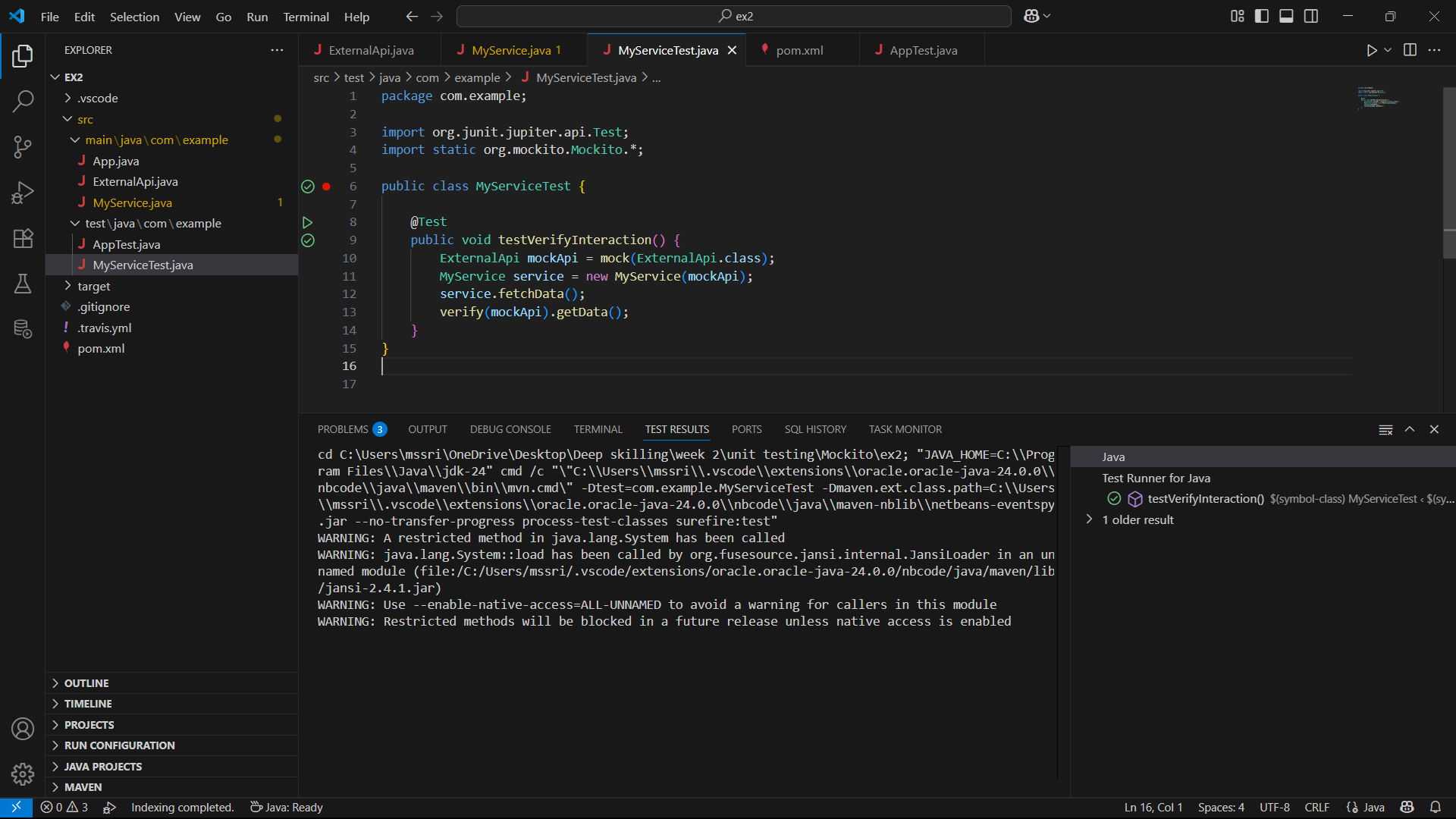
        service.fetchData();

        verify(mockApi).getData();

    }

}

**Output:**



**Question(SLF4J logging framework):**

**Exercise 1: Logging Error Messages and Warning Levels**

**Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J.**

**Step-by-Step Solution:**

1. **Add SLF4J and Logback dependencies to your `pom.xml` file:**

**<dependency>**

**<groupId>org.slf4j</groupId>**

**<artifactId>slf4j-api</artifactId>**

**<version>1.7.30</version>**

**</dependency>**

**<dependency>**

**<groupId>ch.qos.logback</groupId>**

**<artifactId>logback-classic</artifactId>**

**<version>1.2.3</version>**

**</dependency>**

**Code:**

**Logging.java:**

**package com.example;**

**import org.slf4j.Logger;**

**import org.slf4j.LoggerFactory;**

**public class logging{**

**private static final Logger logger = LoggerFactory.getLogger(logging.class);**

**public static void main(String[] args) {**

**logger.error("This is an error message");**

**logger.warn("This is a warning message");**

**}**

**}**

**Output:**

