**Candidate Name**:- SHARANYA K

**Superset ID**:-6419887

**WEEK – 2 HANDS ON EXERCISE (JAVA FSE DEEPSKILLING)**

**Module 4 – Test driven development and Logging framework**

**(JUnit Testing)**

**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>

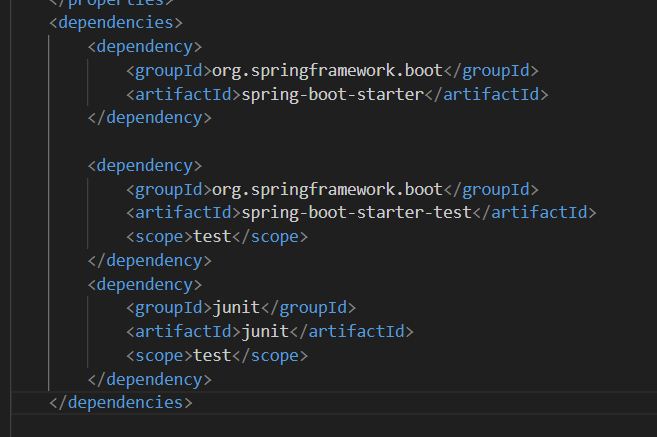
<version>4.13.2</version>

<scope>test</scope>

</dependency>

3. Create a new test class in your project.

**Dependency added in pom.xml**

****

**Code part**

package com.example.junitdemo;

public class Calculation {

    public int multiply(int x, int y) {

        return x \* y;

    }

    public int divide(int a, int b) {

        if (b == 0) {

            throw new IllegalArgumentException("Cannot divide by zero");

        }

        return a / b;

    }

}

**Testing For the code part**

package com.example.junitdemo;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class CalculationTest {

    @Test

    public void testMultiplication() {

        Calculation calc = new Calculation();

        int result = calc.multiply(4, 7);

        System.out.println("Multiplication result: " + result);

        assertEquals(28, result);

    }

    @Test

    public void testDivision() {

        Calculation calc = new Calculation();

        int result = calc.divide(20, 4);

        System.out.println("Division result: " + result);

        assertEquals(5, result);

    }

    @Test(expected = IllegalArgumentException.class)

    public void testDivideByZero() {

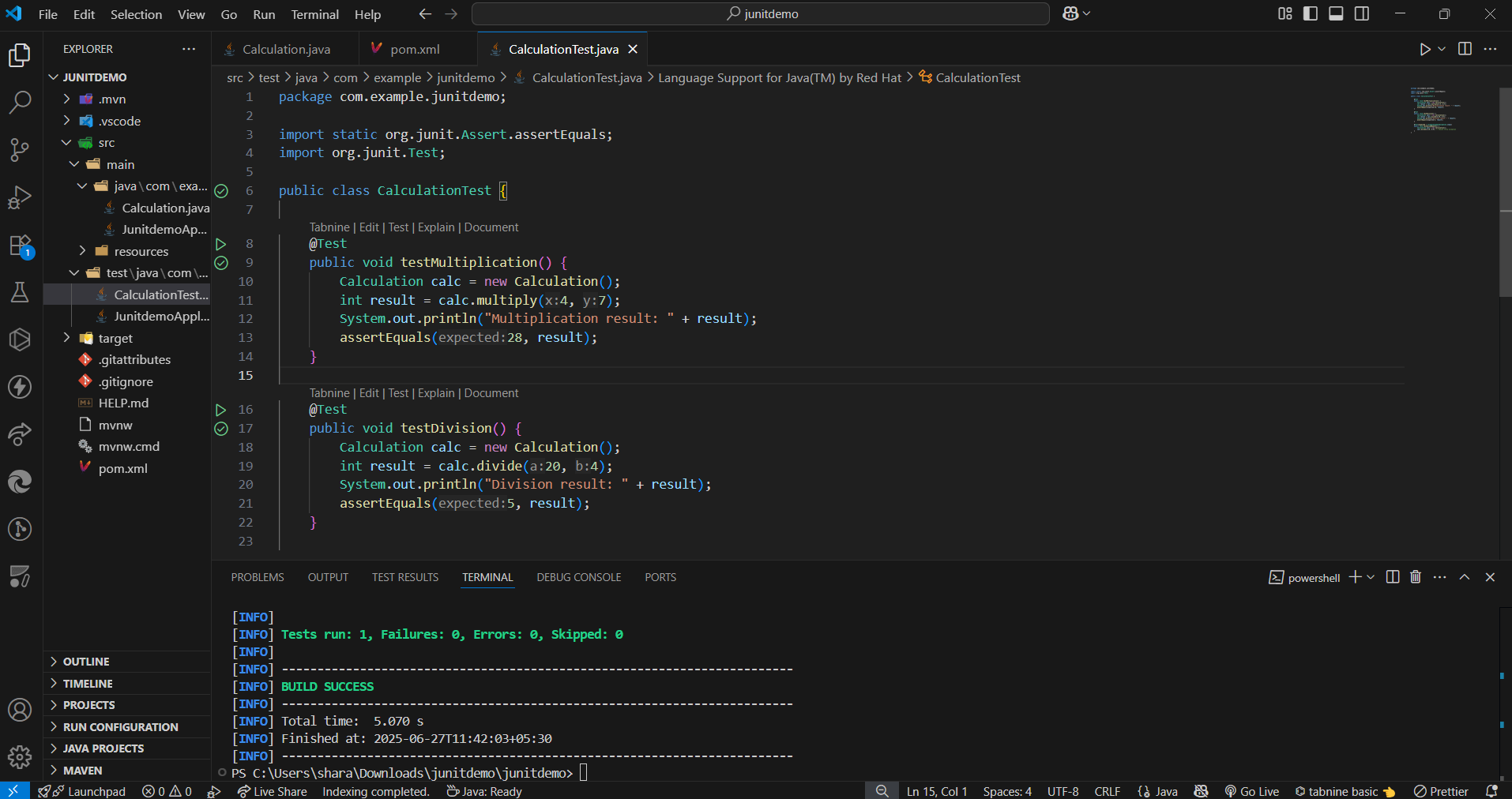
        Calculation calc = new Calculation();

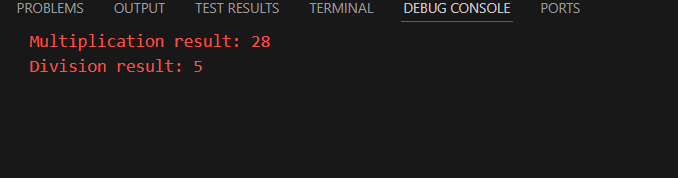
        calc.divide(10, 0); // Should throw exception

    }

}

**Terminal Output For Test Pass Confirmation**

****



**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());

}

}

**Solution code part**

**AssertionDemoTest.java**

package com.example.junitdemo;

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertFalse;

import static org.junit.Assert.assertNotNull;

import static org.junit.Assert.assertNull;

import static org.junit.Assert.assertTrue;

import org.junit.Test;

public class AssertionsTest {

    @Test

    public void testAssertions() {

        System.err.println("Starting testAssertions...");

        assertEquals("Check if 2 + 3 equals 5", 5, 2 + 3);

        System.err.println("Passed: 2 + 3 equals 5");

        assertTrue("Check if 5 is greater than 3", 5 > 3);

        System.err.println("Passed: 5 is greater than 3");

        assertFalse("Check if 5 is not less than 3", 5 < 3);

        System.err.println("Passed: 5 is not less than 3");

        String sample = null;

        assertNull("Expected sample to be null", sample);

        System.err.println("Passed: sample is null");

        String message = "JUnit is running";

        assertNotNull("Expected message to be not null", message);

        System.err.println("Passed: message is not null");

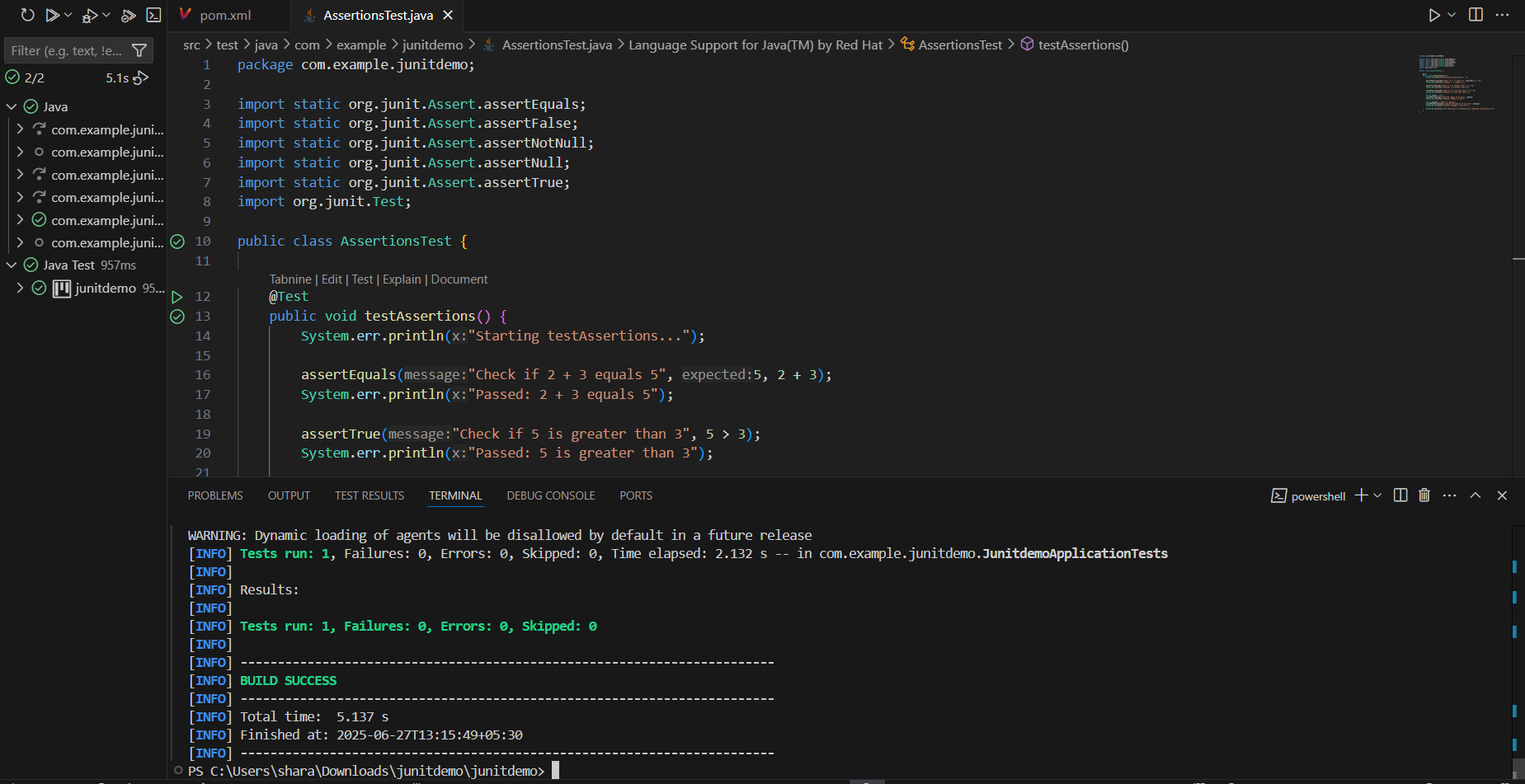
        System.err.println("All assertions in testAssertions completed successfully.\n");

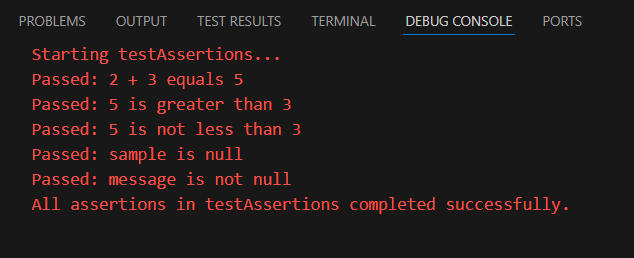
    }

}

| **Assertion** | **Purpose** |
| --- | --- |
| assertEquals | Validates arithmetic or logic matches expected |
| assertTrue | Confirms condition is true |
| assertFalse | Confirms condition is false |
| assertNull | Ensures an object is null |
| assertNotNull | Ensures an object is not null |

**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.

Steps: 1. Write tests using the AAA pattern.

2. Use @Before and @After annotations for setup and teardown methods.

**Code Part**

**SimpleCalculation.java**

package com.example.junitdemo;

public class SimpleCalculation {

    public int addValues(int a, int b) {

        return a + b;

    }

    public int subtractValues(int a, int b) {

        return a - b;

    }

}

**SimpleCalculationTest.java**

package com.example.junitdemo;

import org.junit.After;

import static org.junit.Assert.assertEquals;

import org.junit.Before;

import org.junit.Test;

public class SimpleCalculationTest {

    private SimpleCalculation calc;

    @Before

    public void setUp() {

        System.err.println("Setting up SimpleCalculation object...");

        calc = new SimpleCalculation();  // Arrange

    }

    @After

    public void tearDown() {

        System.err.println("Cleaning up after test...\n");

        calc = null;

    }

    @Test

    public void testAddition() {

        // Act

        int result = calc.addValues(3, 2);

        // Assert

        assertEquals("Expected 3 + 2 to equal 5", 5, result);

        System.err.println("Addition test passed: 3 + 2 = " + result);

    }

    @Test

    public void testSubtraction() {

        // Act

        int result = calc.subtractValues(10, 4);

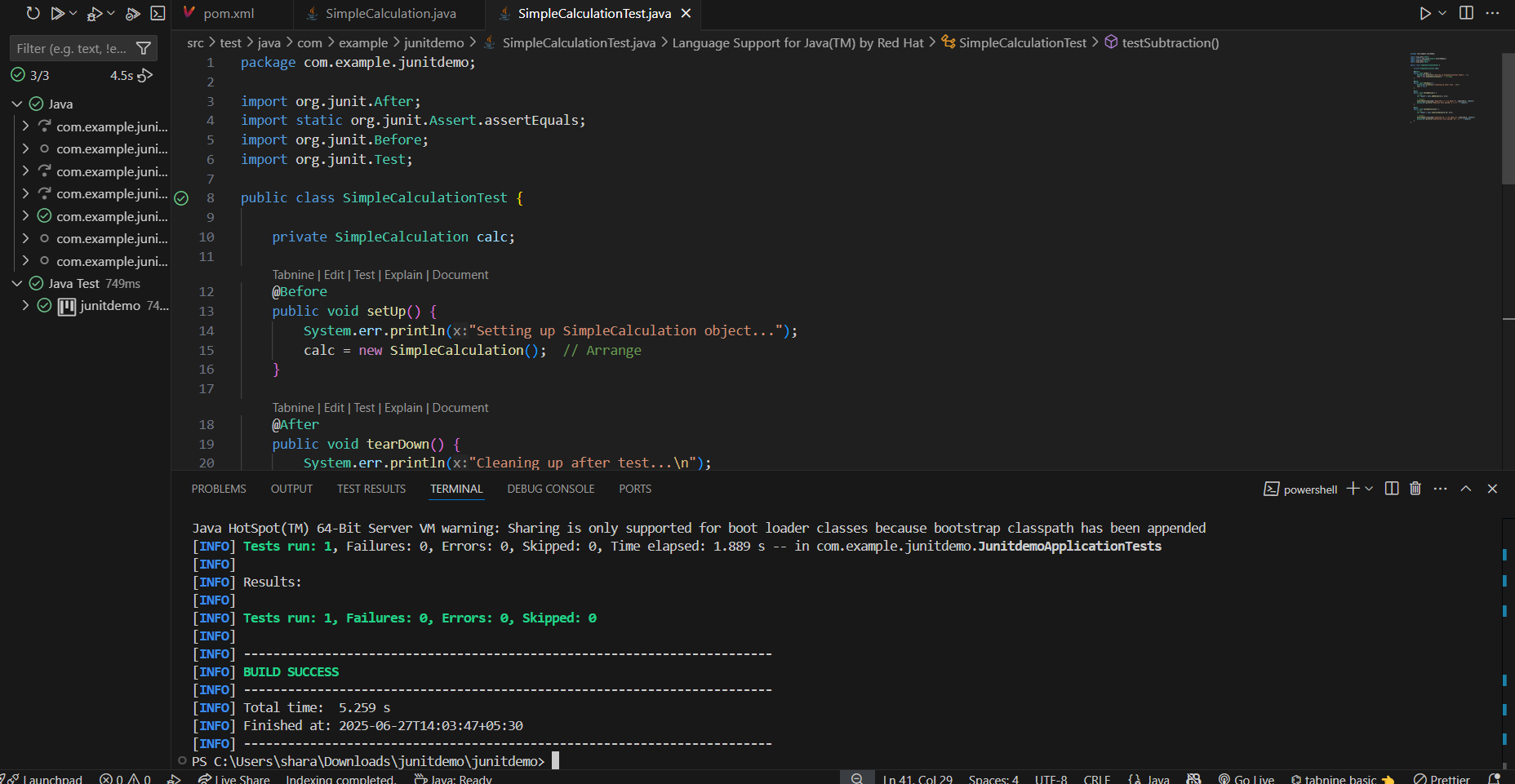
        // Assert

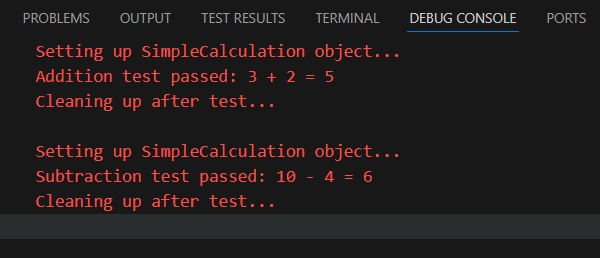
        assertEquals("Expected 10 - 4 to equal 6", 6, result);

        System.err.println("Subtraction test passed: 10 - 4 = " + result);

    }

}





**(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);

}

}

**Solution Code:**

**RemoteService.java**

package com.example.junitdemo;

public interface RemoteService {

    String fetchInfo();

}

**ClientHandler.java**

package com.example.junitdemo;

public class ClientHandler {

    private final RemoteService remote;

    public ClientHandler(RemoteService remote) {

        this.remote = remote;

    }

    public String processRequest() {

        return remote.fetchInfo();

    }

}

**ClientHandlerTest.java**

package com.example.junitdemo;

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

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.when;

public class ClientHandlerTest {

    @Test

    public void shouldReturnStubbedInfoFromRemoteService() {

        RemoteService mockRemote = mock(RemoteService.class);

        when(mockRemote.fetchInfo()).thenReturn("Simulated Response");

        ClientHandler handler = new ClientHandler(mockRemote);

        String output = handler.processRequest();

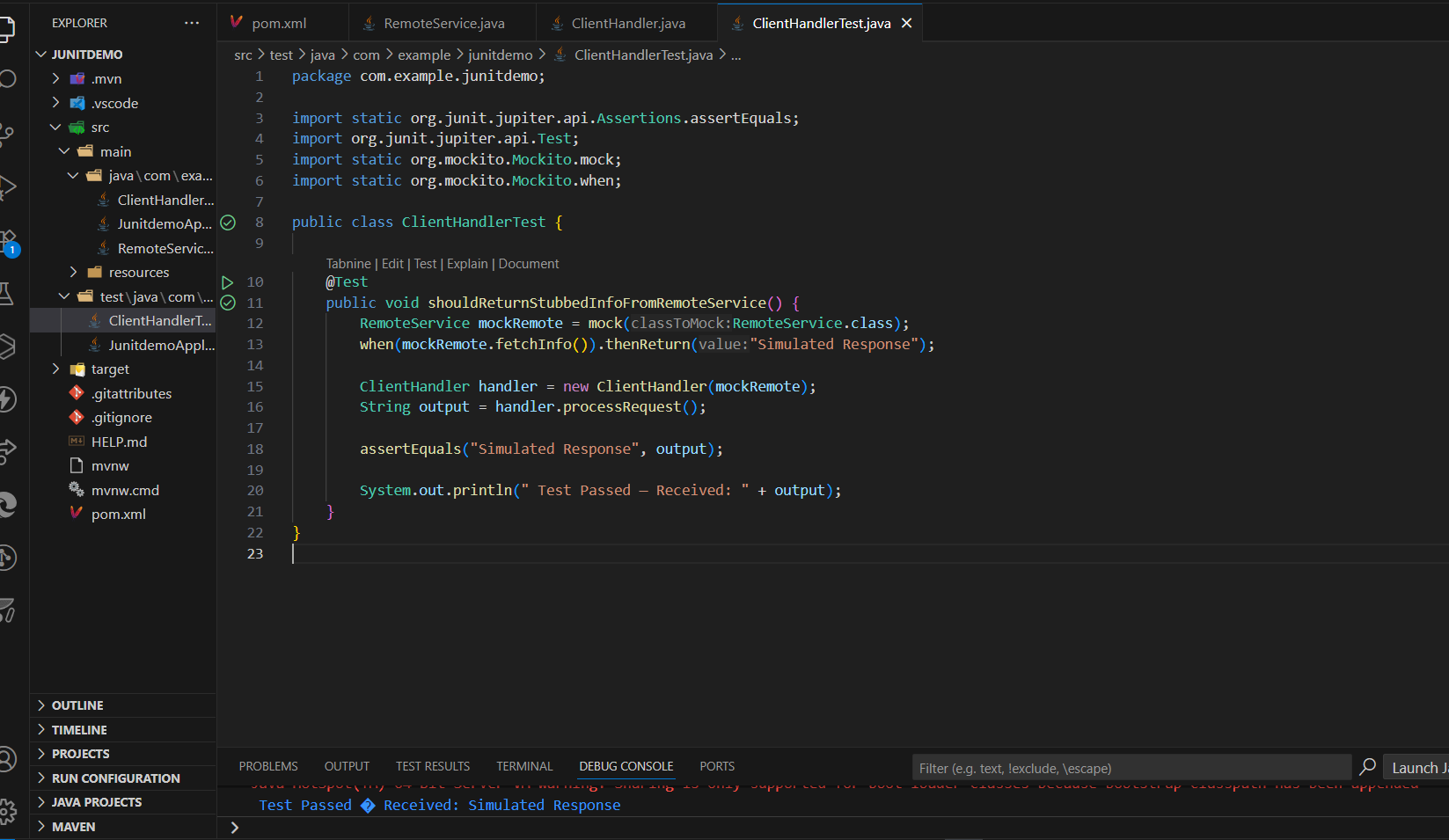
        assertEquals("Simulated Response", output);

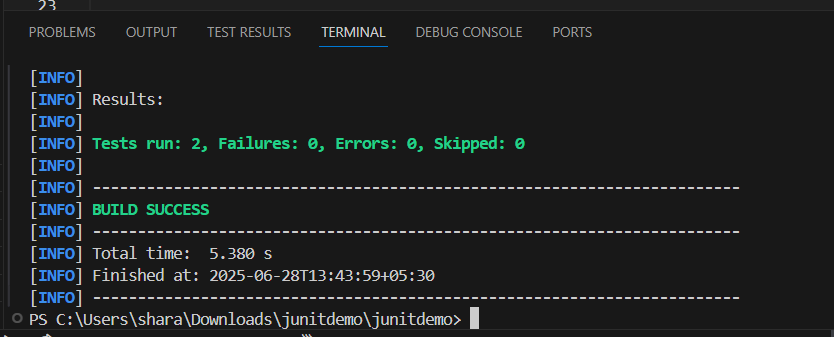
        System.out.println(" Test Passed — Received: " + output);

    }

}

**Output**

****



**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();

}

}

**Solution Code:**

**ApiClient.java**

package com.example.junitdemo;

public interface ApiClient {

    void fetchUserDetails();

}

**UserProcessor.java**

package com.example.junitdemo;

public class UserProcessor {

    private final ApiClient apiClient;

    public UserProcessor(ApiClient apiClient) {

        this.apiClient = apiClient;

    }

    public void processUser() {

        apiClient.fetchUserDetails();

    }

}

**UserProcessorTest.java**

package com.example.junitdemo;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.verify;

public class UserProcessorTest {

    @Test

    public void testInteractionWithApi() {

        // Arrange

        ApiClient mockApi = mock(ApiClient.class);

        UserProcessor processor = new UserProcessor(mockApi);

        // Act

        processor.processUser();

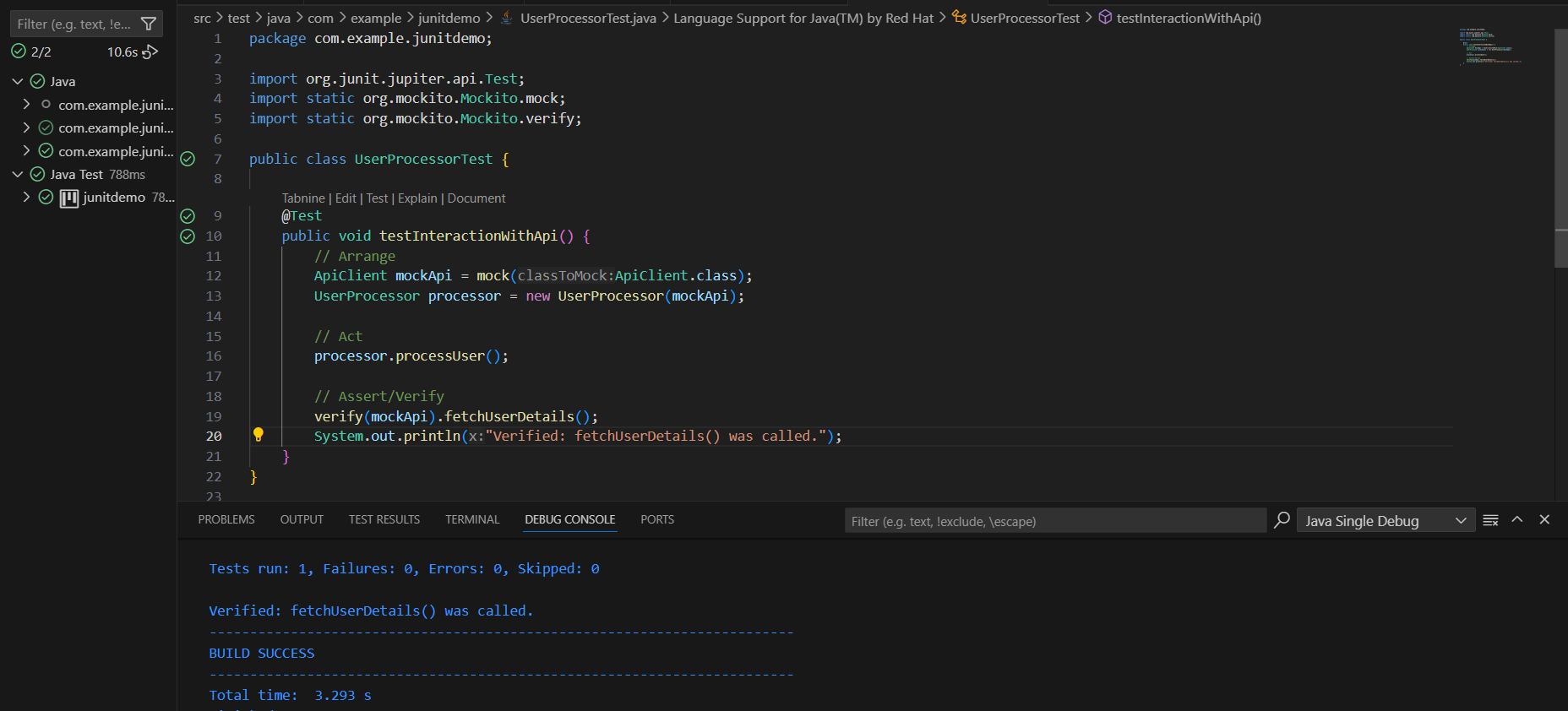
        // Assert/Verify

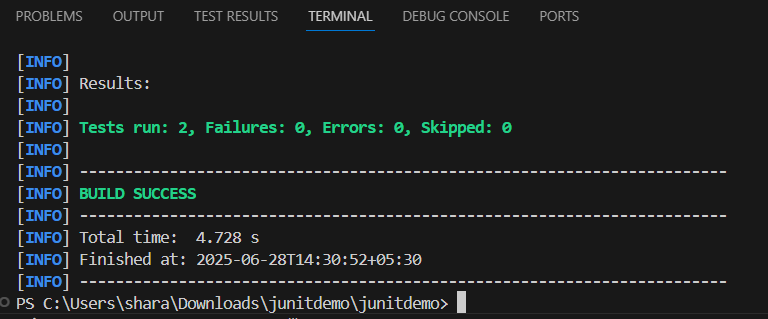
        verify(mockApi).fetchUserDetails();

        System.out.println("Verified: fetchUserDetails() was called.");

    }

}





**(SL4J Logging exercises)**

**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>

2. Create a Java class that uses SLF4J for logging:

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

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

public static void main(String[] args) {

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

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

}

}

**Solution Code:**

**AppLogger.java**

package com.example.junitdemo;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class AppLogger {

    private static final Logger log = LoggerFactory.getLogger(AppLogger.class);

    public static void main(String[] args) {

        log.error("Error: Database connection failed.");

        log.warn("Warning: Memory usage is high.");

    }

}

