**Week2:**

**JUnit Basic Testing Exercises**

Exercise 1: Setting Up JUnit

We need to set up a basic Java project using Maven in Visual Studio Code and integrate JUnit for unit testing. The goal is to write and successfully run a sample test case.

Creation of maven project:

archetype: maven-archetype-quickstart

Group ID: com.example

Artifact ID: demo

Added JUnit dependency:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

App.test.java

Code:

package com.example;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class AppTest

{

    /\*\*

     \* Rigorous Test :-)

     \*/

    @Test

    public void testAddition() {

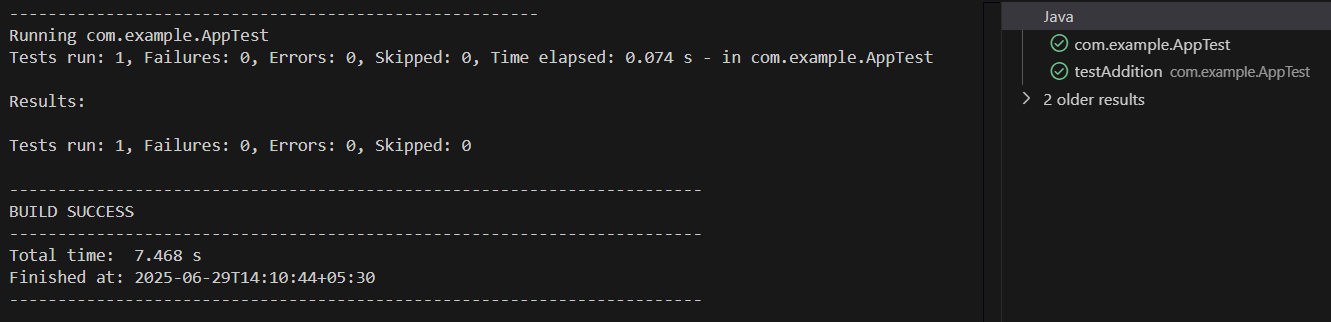
        assertEquals(4, 2 + 2);

    }

}

Running the Test:

Output:



**Exercise 3:** Assertions in JUnit

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

AssertionsTest.java

Code:

package com.example;

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() {

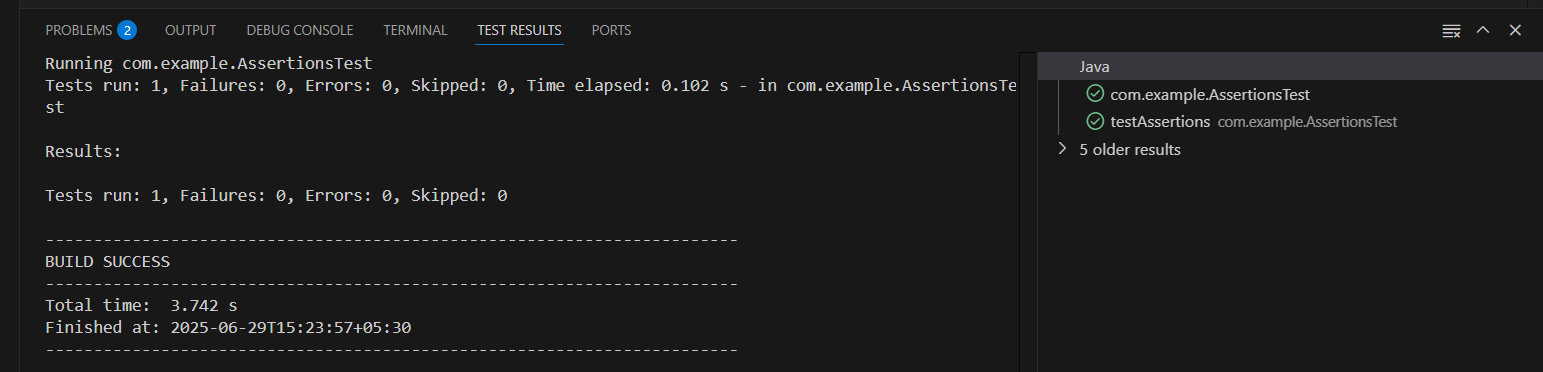
        assertEquals(5, 2 + 3);

        assertTrue(5 > 3);

        assertFalse(5 < 3);

        assertNull(null);

        assertNotNull(new Object());}

    }

**Exercise 4:**

Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit.  
I have used CalculatorTest program to test for it.

CalculatorTest.java

Code:

package com.example;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

// Setup method: runs before each test

@Before

public void setUp() {

calculator = new Calculator(); // Arrange

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

}

// Teardown method: runs after each test

@After

public void tearDown() {

calculator = null;

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

}

@Test

public void testAddition() {

// Act

int result = calculator.add(2, 3);

// Assert

assertEquals(5, result);

}

@Test

public void testSubtraction() {

int result = calculator.subtract(10, 4);

assertEquals(6, result);

}

}

Calculator.java:

package com.example;

public class Calculator {

    public int add(int a, int b) {

        return a + b;

    }

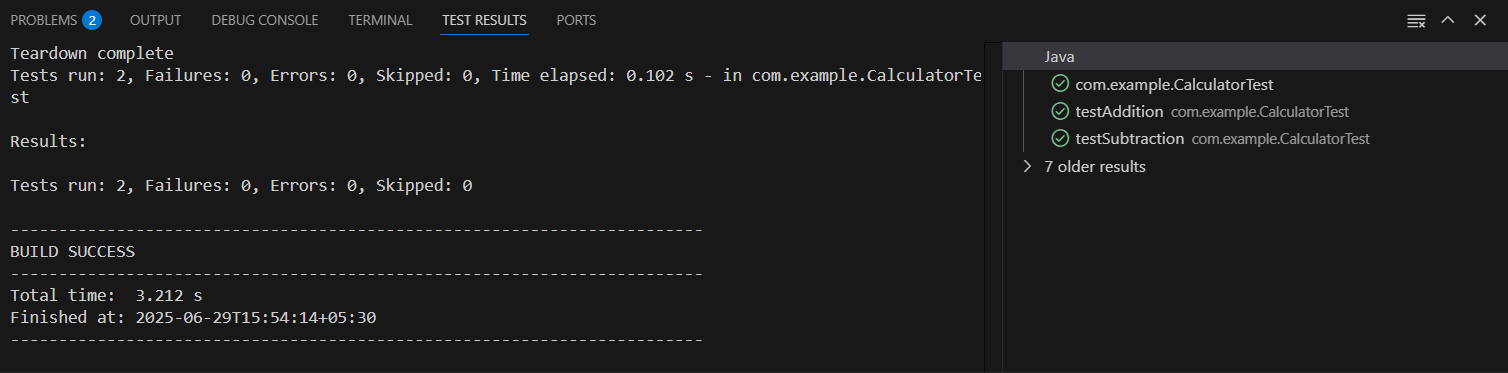
    public int subtract(int a, int b) {

        return a - b;

    }

}

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.

Code:

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 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 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);

    }

}

Added a Mockito dependency in pom.xml:

<dependency>

      <groupId>org.mockito</groupId>

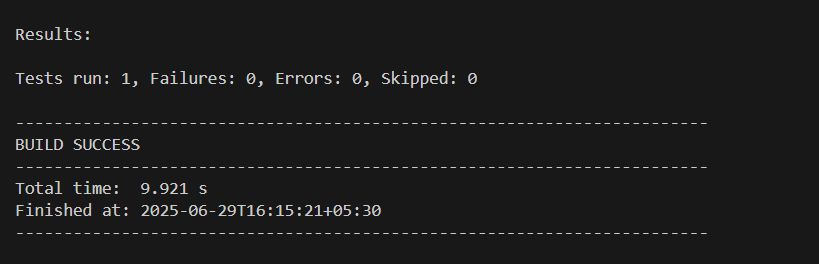
      <artifactId>mockito-core</artifactId>

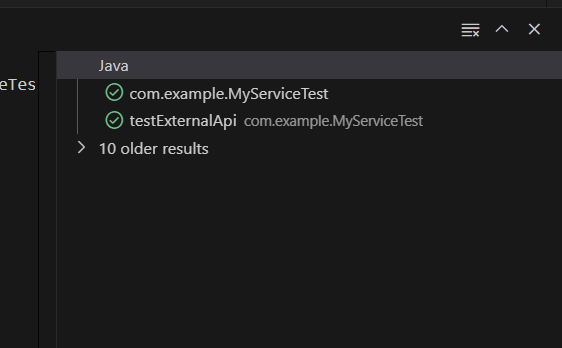
      <version>4.11.0</version>

      <scope>test</scope>

    </dependency>

**Output:**

****



**Exercise 2:**

Verifying Interactions

Scenario:

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

We need to Ensure that your code calls mockApi.getData() when service.fetchData() is executed.

Code:

**ExternalApi.java**:

( to define the interface)

package com.example;

public interface ExternalApi {

String getData();

}

**MyService.java:**

(the class using the external API)

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

**(verifying the interaction)**

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Step 1: Create mock object

ExternalApi mockApi = mock(ExternalApi.class);

// Step 2: Use mock in service

MyService service = new MyService(mockApi);

service.fetchData();

// Step 3: Verify method was called

verify(mockApi).getData();

}

}

Output:

