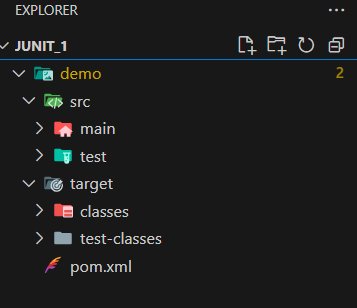
**Exercise 1: Setting Up JUnit Scenario: You need to set up JUnit in your Java project to start writing unit tests. Steps: 1. Create a new Java project in your IDE:-**

**Answer:-**

Step 1: Create a New Java Project in

1. Open an IDE eg. vscode.
2. Press Create Java Project from the options or press ctrl+shift+p and type java:create java project.
3. Select Maven as the project type.
4. Choose a project location, then enter a groupId (e.g., com.example) and artifactId (e.g.Demo).
5. The Maven project will be created in the destination directory.

**Output:-**

****

**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>**

**Answer:-**

In the root folder of your project, open the pom.xml file.

Code:-inside dependencies tag

<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.**

Inside the src/test/java/com/example, create a new Java file called Testclass.java.

Code:-

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class Testclass

{

@Test

public void testAddition()

{

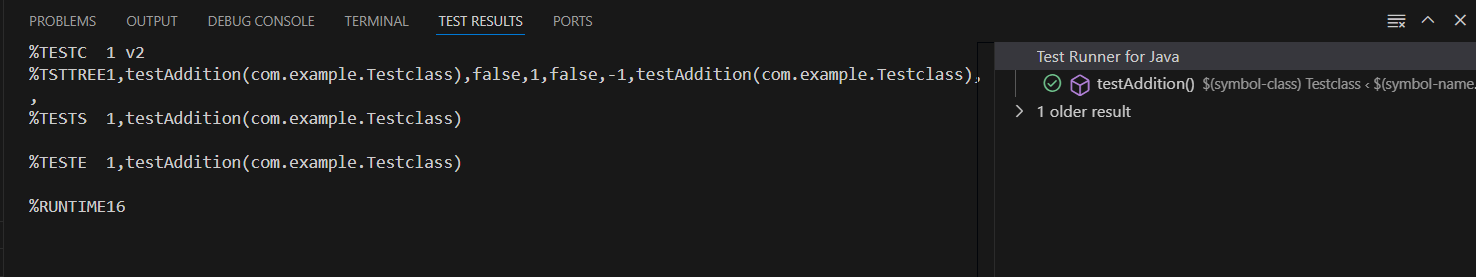
int result = 2 + 3;

assertEquals(5, result);

}

}

**Output:-**

****

**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()); } }**

**ANSWER:-**

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class Testclass

{

    @Test

    public void testAssertions()

{

        assertEquals(5, 2 + 3);

        assertTrue(5 > 3);

        assertFalse(5 < 3);

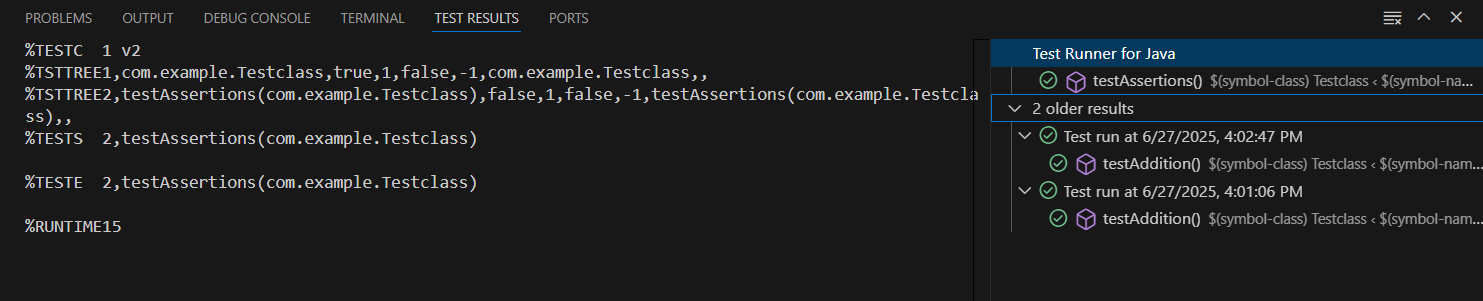
        assertNull(null);

        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. Steps: 1. Write tests using the AAA pattern. 2. Use @Before and @After annotations for setup and teardown methods.**

**ANSWER:-**

**File: Calculator.java (in src/main/java/com/example/)**

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;

    }

}

**File: Testclass.java (in src/test/java/com/example/)**

package com.example;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class Testclass

{

    private Calculator calculator;

    @Before

    public void setUp()

    {

        calculator = new Calculator();

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

    }

    @After

    public void tearDown()

    {

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

    }

    @Test

    public void testAddition()

    {

        int result = calculator.add(5, 3);

        assertEquals(8, result);

    }

    @Test

    public void testSubtraction()

    {

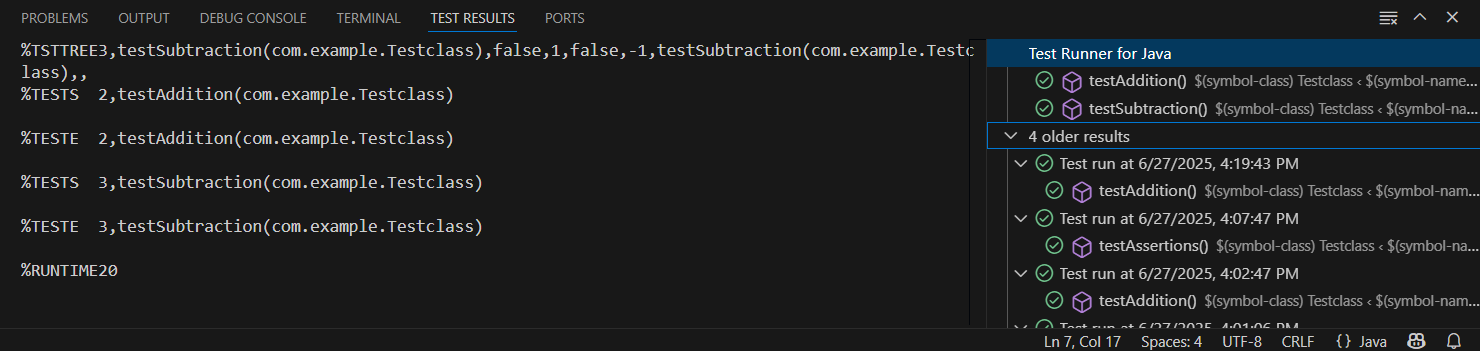
        int result = calculator.subtract(10, 4);

        assertEquals(6, result);

    }

}

**Output:-**

****

**File: Mockito excercices**

**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); } }**

**Answer:-**

**ExternalApi.java — in src/main/java/com/example/**

package com.example;

public interface ExternalApi

{

String getData();

}

**MyService.java — in src/main/java/com/example/**

package com.example;

public class Myservice

{

    private Externalapi externalApi;

    public Myservice(Externalapi externalApi)

    {

        this.externalApi = externalApi;

    }

    public String fetchData()

    {

        return externalApi.getData();

    }

}

**Testclass.java — in src/test/java/com/example/**

package com.example;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

public class Testclass {

    @Test

    public void testExternalApi()

    {

        Externalapi mockApi = mock(Externalapi.class);

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

        Myservice service = new Myservice(mockApi);

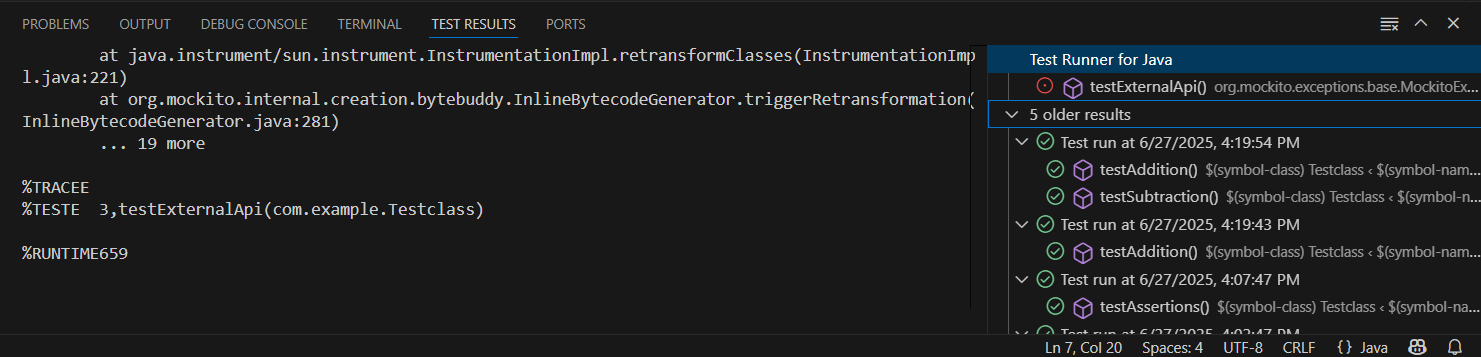
        String result = service.fetchData();

        assertEquals("Mock Data", result);

    }

}

**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(); } }**

**Answer:-**

**ExternalApi.java — in src/main/java/com/example/**

package com.example;

public interface ExternalApi

{

String getData();

}

**MyService.java — in src/main/java/com/example/**

package com.example;

public class MyService

{

private ExternalApi externalApi;

public MyService(ExternalApi externalApi)

{

this.externalApi = externalApi;

}

public String fetchData()

{

return externalApi.getData();

}

}

**Testclass.java — in src/test/java/com/example/**

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);

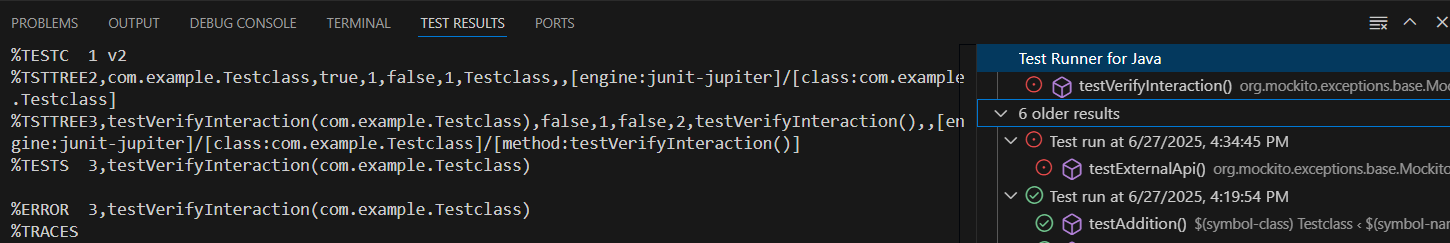
service.fetchData();

verify(mockApi).getData(); // ✅ Test will fail if not called

}

}

**Output:-**

****