**JUnit Testing Exercises**

**Exercise 1: Setting Up JUnit**

Scenario: You need to set up JUnit in your Java project to start writing unit tests.

Step 1: Create a New Maven Project in Eclipse

1. Launch **Eclipse IDE**.
2. Navigate to **File → New → Project**.
3. Select **Maven → Maven Project**, then click **Next**.
4. In the wizard, check the box **"Create a simple project (skip archetype selection)"** and click **Next**.
5. Provide the following Maven coordinates:  
   * **Group Id**: com.example
   * **Artifact Id**: JUnitSetupDemo
6. Click **Finish** to create the project.

Step 2: Add JUnit Dependency to pom.xml

1. In the **Project Explorer**, expand your newly created project and open the pom.xml file.
2. Inside the <project> tag, add the following dependency to include JUnit 4:

xml

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

1. Save the file and then right-click on the project > **Maven → Update Project** to download and configure the dependencies.

Step 3: Create the Java Class to Be Tested

1. Inside the directory src/main/java, right-click on the package com.example (or create one if it doesn’t exist).
2. Select **New → Class**, and name it **Calculator**.
3. Add the following code:

java

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

Step 4: Create the JUnit Test Class

1. Navigate to the src/test/java folder.
2. Right-click on the com.example package (create it if necessary) → **New → Class**.
3. Name the class **CalculatorTest**.
4. Add the following test code using JUnit 4:

java

package com.example;

import static org.junit.Assert.\*;

import org.junit.Test;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

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

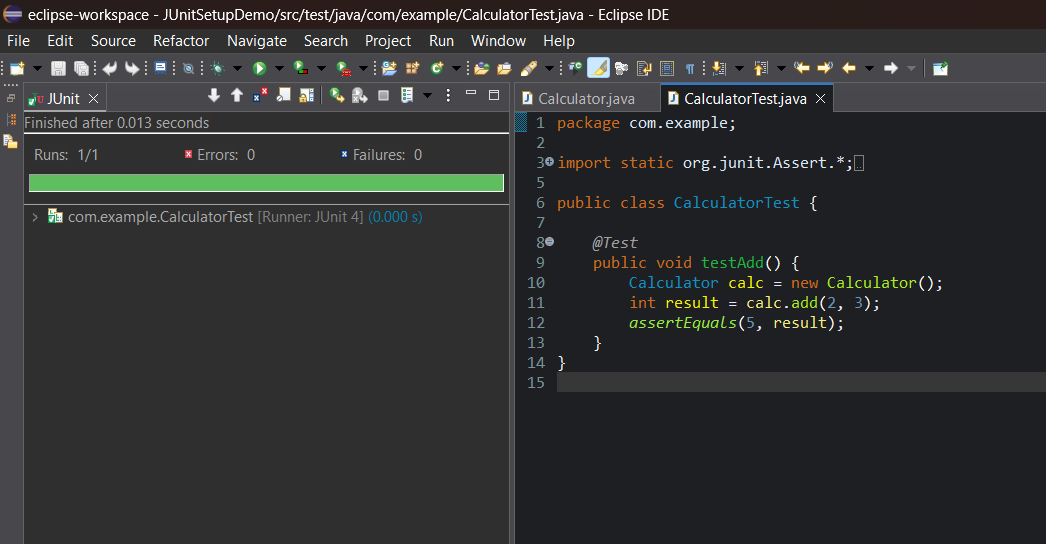
assertEquals(5, result);

}

}

Step 5: Run the JUnit Test

1. Right-click on the file CalculatorTest.java.
2. Select **Run As → JUnit Test**.
3. The **JUnit panel** will appear, showing the result of the test:  
   * **Runs: 1/1**
   * **Errors: 0**
   * **Failures: 0**
4. A **green bar** indicates that the test case has passed successfully.



**Exercise 2: Writing Basic JUnit Tests**

Scenario: You need to write basic JUnit tests for a simple Java class.

### Step 1: Create a Java Class with Methods to Test

1. In Eclipse, navigate to the folder:  
    src/main/java
2. Right-click on the package com.example  
    *(Or create it if not present: Right-click on src/main/java → New → Package → name it com.example)*
3. Select: **New → Class**
4. Name the class: MathUtils
5. Paste the following code:

java

package com.example;

public class MathUtils {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

public int multiply(int a, int b) {

return a \* b;

}

public int divide(int a, int b) {

if (b == 0) {

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

}

return a / b;

}

}

### Step 2: Create the JUnit Test Class

1. Navigate to the folder:  
    src/test/java
2. Right-click on the package com.example  
    *(Create it if it doesn’t exist: Right-click on src/test/java → New → Package → name it com.example)*
3. Select: **New → Class**
4. Name the class: MathUtilsTest
5. Paste the following JUnit test code:

java

package com.example;

import static org.junit.Assert.\*;

import org.junit.Test;

public class MathUtilsTest {

MathUtils math = new MathUtils();

@Test

public void testAdd() {

assertEquals(7, math.add(3, 4));

}

@Test

public void testSubtract() {

assertEquals(2, math.subtract(5, 3));

}

@Test

public void testMultiply() {

assertEquals(15, math.multiply(3, 5));

}

@Test

public void testDivide() {

assertEquals(5, math.divide(10, 2));

}

@Test(expected = IllegalArgumentException.class)

public void testDivideByZero() {

math.divide(10, 0);

}

}

Step 3: Run the JUnit Tests

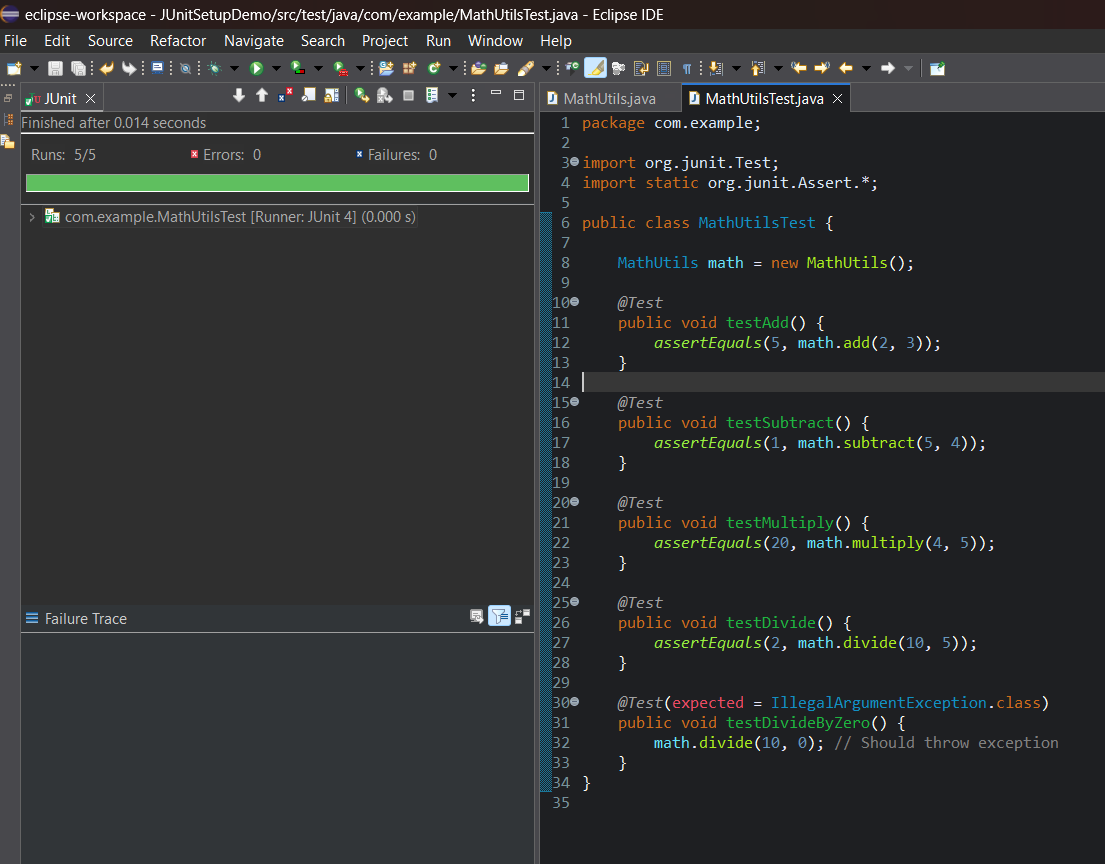
1. Right-click on the file MathUtilsTest.java
2. Select: **Run As → JUnit Test**
3. Eclipse will open the **JUnit panel** and display the result:

Runs: 5/5

Errors: 0

Failures: 0

A green bar means **all tests passed**.



**Exercise 3: Assertions in JUnit**

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

Steps:

### Step 1: Create a Test Class for Assertions

1. In **Eclipse**, navigate to the folder:  
    src/test/java
2. Right-click on the package com.example  
    *(Create it if it does not exist: Right-click on src/test/java → New → Package → name it com.example)*
3. Select: **New → Class**
4. Name the class: AssertionsTest
5. Paste the following code into AssertionsTest.java:

java

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

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

Object obj1 = null;

assertNull(obj1);

// Assert not null

Object obj2 = new Object();

assertNotNull(obj2);

}

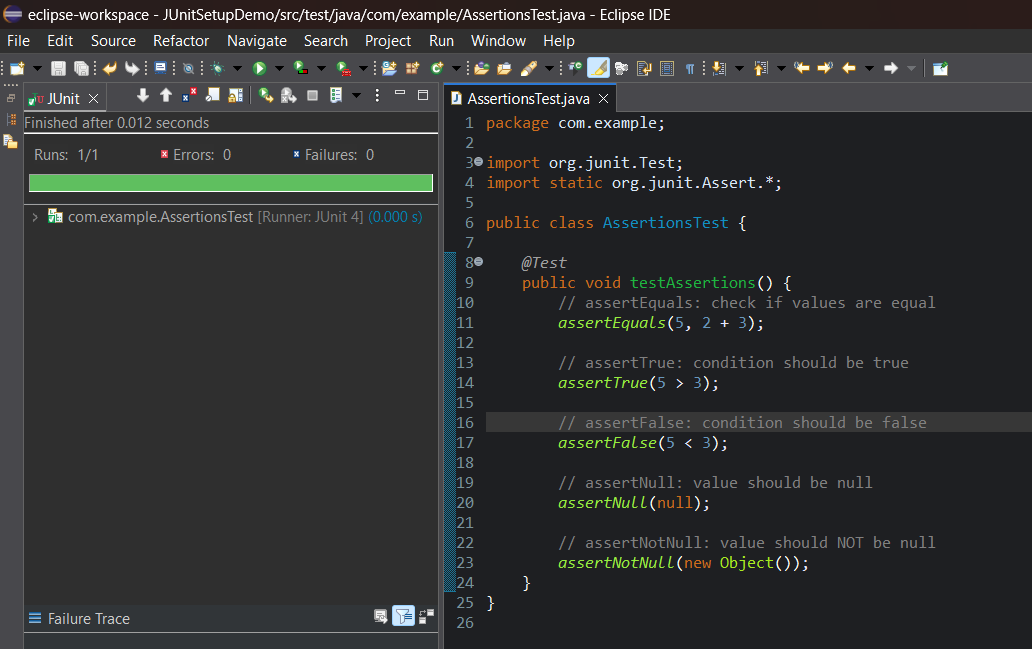
}

### Step 2: Run the JUnit Test Class

1. Right-click on the file AssertionsTest.java
2. Select: **Run As → JUnit Test**

### Expected Output:

* The **JUnit panel** will open and display the following result:  
  + **Runs**: 1/1
  + **Errors**: 0
  + **Failures**: 0
* A **green bar** indicates that all assertions passed successfully.



**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:

### Step 1: Create the Main Java Class

In Eclipse,

1. navigate to:src/main/java
2. Right-click > **New → Package** Name it: com.example
3. Right-click on the com.example package → **New → Class** Name it: MathUtils
4. Paste the following code into MathUtils.java:

java

package com.example;

public class MathUtils {

public int add(int a, int b) { return a + b; }

public int subtract(int a, int b) { return a - b; }

public int multiply(int a, int b) { return a \* b; }

public int divide(int a, int b) {

if (b == 0) throw new IllegalArgumentException("Division by zero");

return a / b;

}

}

### Step 2: Create the JUnit Test Class with Setup and Teardown

1. Navigate to:src/test/java
2. Right-click > **New → Package** Name it: com.example
3. Right-click on the com.example package → **New → Class** Name it: MathUtilsLifecycleTest
4. Paste the following code into MathUtilsLifecycleTest.java:

java

package com.example;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class MathUtilsLifecycleTest {

private MathUtils math;

// Setup: runs before each test

@Before

public void setUp() {

System.out.println("Setting up...");

math = new MathUtils(); // Arrange

}

// Teardown: runs after each test

@After

public void tearDown() {

System.out.println("Cleaning up...");

math = null; // Cleanup

}

@Test

public void testAdd() {

// Act

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

// Assert

assertEquals(5, result);

}

@Test

public void testMultiply() {

// Act

int result = math.multiply(4, 5);

// Assert

assertEquals(20, result);

}

@Test(expected = IllegalArgumentException.class)

public void testDivideByZero() {

// Act & Assert

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

}

}

### **Step 3: Run the JUnit Test Class**

* Right-click on the file MathUtilsLifecycleTest.java
* Select: **Run As → JUnit Test**

### **Expected Output:**

The JUnit panel will display:

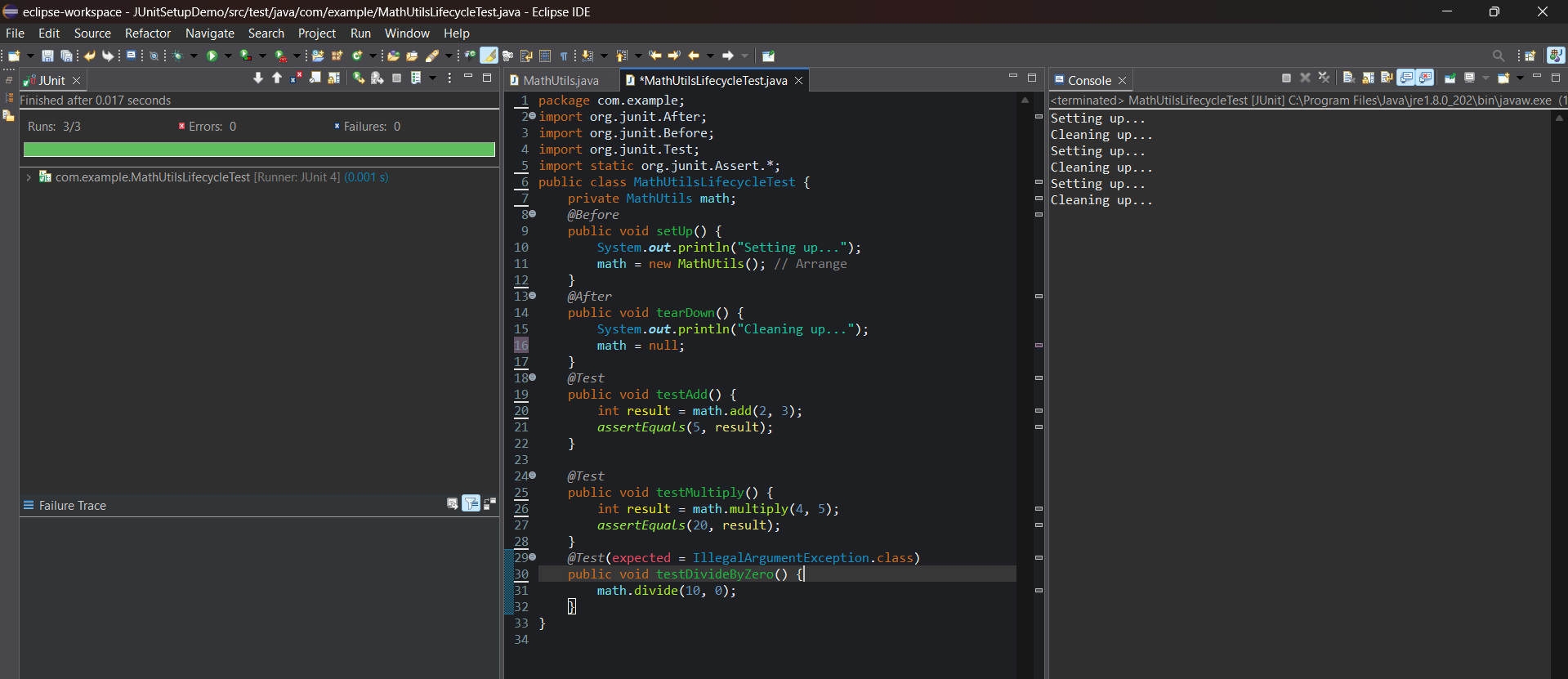
Runs: 3/3

Errors: 0

Failures: 0

Console output will include:  
Setting up...

Cleaning up...

* A **green bar** will appear, indicating all test methods passed successfully.  
  

| **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.

**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** **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);

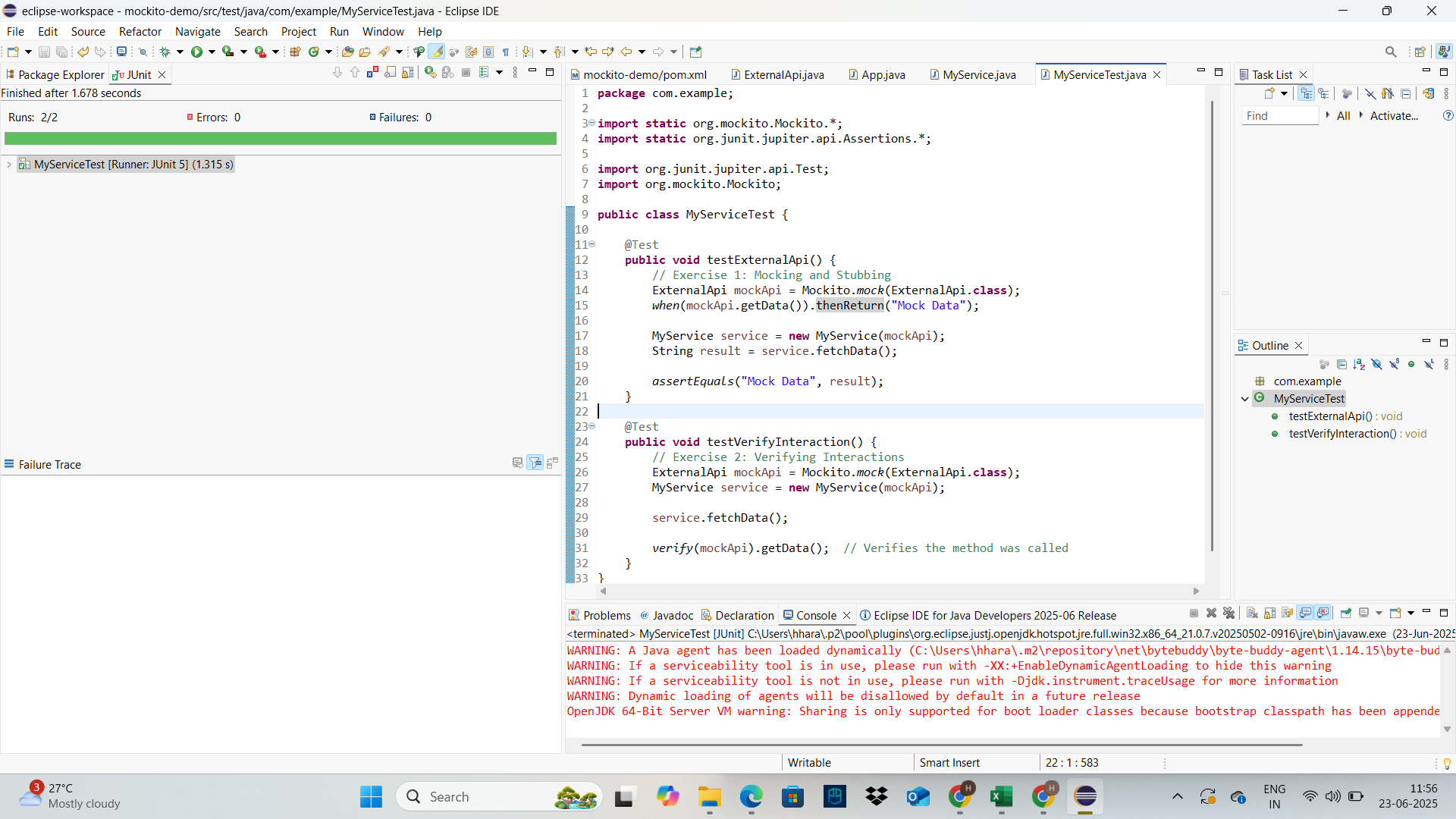
String result = service.fetchData();

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

}

}

**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.

**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** testVerifyInteraction() {

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

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

service.fetchData();

*verify*(mockApi).getData(); // Verifies the method was called

}

}

**OUTPUT:**

