1. JUnit\_Basic Testing Exercises

# Exercise 1: Setting Up JUnit

**Step 1: Create a New Java Project**

**In Eclipse**:

* Go to **File > New > Maven Project**.
* Check **Create a simple project (skip archetype selection)**.
* Click **Next**.
* Enter **Group Id**, **Artifact Id**, etc.
* Finish the setup.

**Step 2: Add JUnit Dependency**

Edit pom.xml and add this under <dependencies>:

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

**Step 3: Create a Test Class**

Create a class named Calculator

package io.dn.genc;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

Create a test class named CalculatorTest in src/test/java

package io.dn.genc;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

*@Test*

public void testAdd() {

Calculator calc = new Calculator();

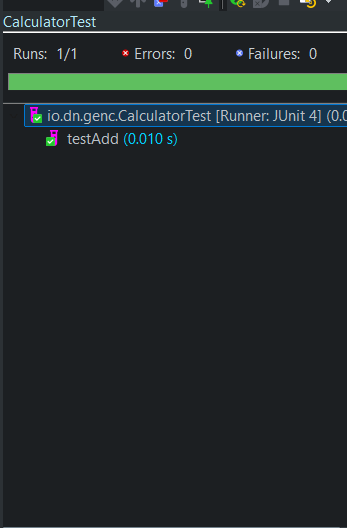
int result = calc.add(2, 3);

*assertEquals*(5, result);

}

}

**Output:**



# **Exercise 3: Assertions in JUnit**

**Goal: Practice various JUnit assertions**

package io.dn.genc;

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 obj = null;

*assertNull*(obj);

// Assert not null

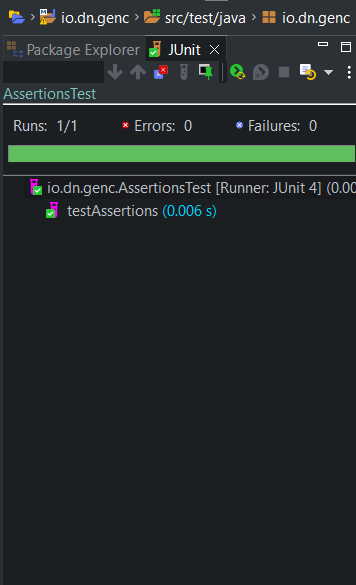
Object anotherObj = new Object();

*assertNotNull*(anotherObj);

}

}

**Output:**



# Exercise 4: Arrange-Act-Assert (AAA) Pattern, Setup and Teardown

**Goal: Use @Before and @After, and follow AAA structure**

package io.dn.genc;

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();

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() {

// Arrange

int a = 5, b = 7;

// Act

int result = calculator.add(a, b);

// Assert

*assertEquals*(12, result);

}

*@Test*

public void testSubtraction() {

// Arrange

int a = 10, b = 4;

// Act

int result = calculator.subtract(a, b);

// Assert

*assertEquals*(6, result);

}

}

**Sample Calculator class:**

package io.dn.genc;

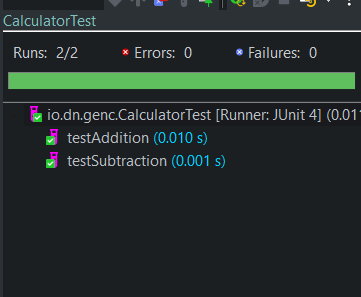
public class Calculator {

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

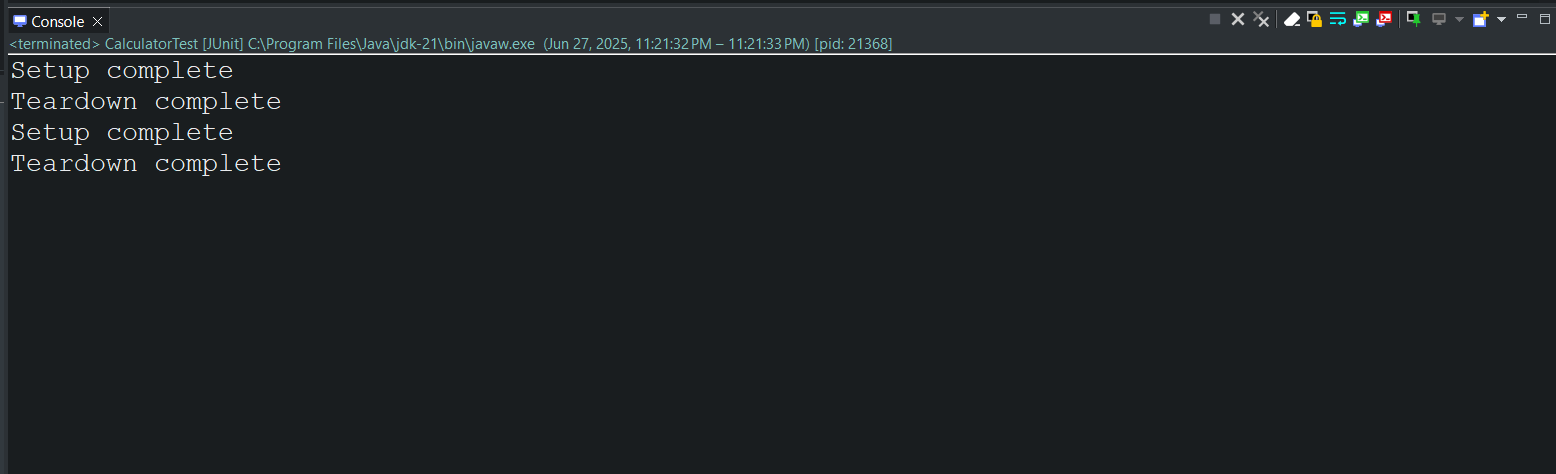
public int subtract(int a, int b) { return a - b; }

}

**Output:**

****

**Console:**

****

3. Mockito exercises

# Exercise 1: Mocking and Stubbing, And Exercise 2: Verifying Interactions

**Goal**: The goal is to test a service class (MyService) that relies on an external dependency (ExternalApi) by using Mockito to create a mock object. This includes stubbing the external API’s method to return predefined data (Exercise 1), and verifying that the service interacts with the mock correctly by calling the expected method with the correct arguments (Exercise 2).

ExternalApi.java

package io.dn.genc.ritika;

public interface ExternalApi {

String getData();

}

MyService.java

package io.dn.genc.ritika;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

MyServiceTest.java

public class MyServiceTest {

// Exercise 1: Mocking and Stubbing

*@Test*

public void testExternalApi() {

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

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

System.***out***.println("Exercise 1 Output:");

System.***out***.println("Fetched Data: " + result);

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

}

// Exercise 2: Verifying Interaction

*@Test*

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApi);

service.fetchData();

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

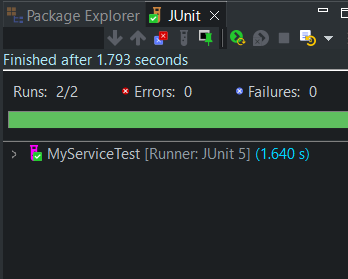
System.***out***.println("Exercise 2 Output:");

System.***out***.println("Verified that getData() was called on the mock.");

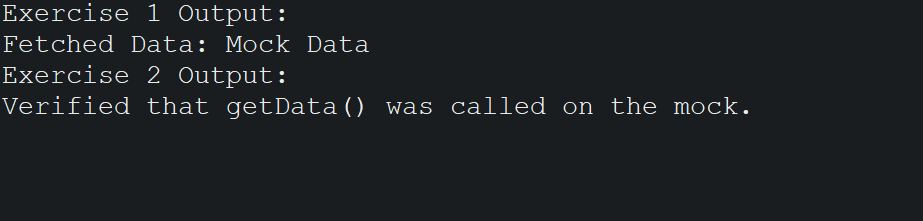
}

}

**Output:**

****

**Console:**

****