**JUnit Testing Exercises**

**Exercise 1: Setting Up Junit**

**Code :-**

**// File: Calculator.java**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**// File: CalculatorTest.java**

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

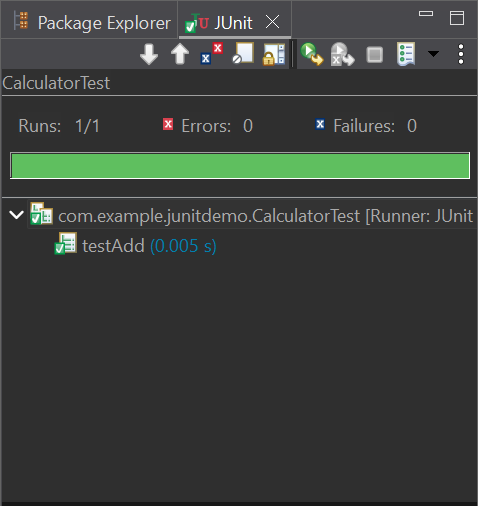
int result = calc.add(5, 3);

assertEquals(8, result); // Passes if result == 8

}

}

**Output :-**

****

**Exercise 3: Assertions in JUnit**

**Code :-**

**// File: AssertionsTest.java**

package com.example.junitdemo;

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

assertNull(null);

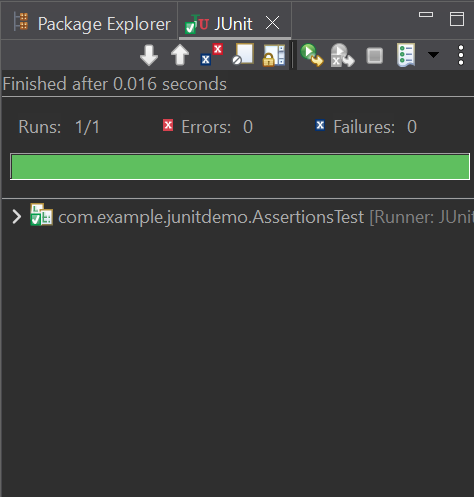
// Assert not null

assertNotNull(new Object());

}

}

**Output :-**

****

**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in Junit**

**Code :-**

**// File: Calculator.java**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**// File: CalculatorTest2.java**

package com.example.junitdemo;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest2 {

private Calculator calculator;

@Before

public void setUp() {

calculator = new Calculator();

System.out.println("Setup for CalculatorTest2 complete");

}

@After

public void tearDown() {

System.out.println("Test in CalculatorTest2 finished");

}

@Test

public void testAdditionWithPositiveNumbers() {

// Arrange

int a = 10;

int b = 20;

// Act

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

// Assert

assertEquals(30, result);

}

@Test

public void testAdditionWithZero() {

// Arrange

int a = 0;

int b = 15;

// Act

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

// Assert

assertEquals(15, result);

}

@Test

public void testAdditionWithNegativeNumber() {

// Arrange

int a = -5;

int b = 10;

// Act

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

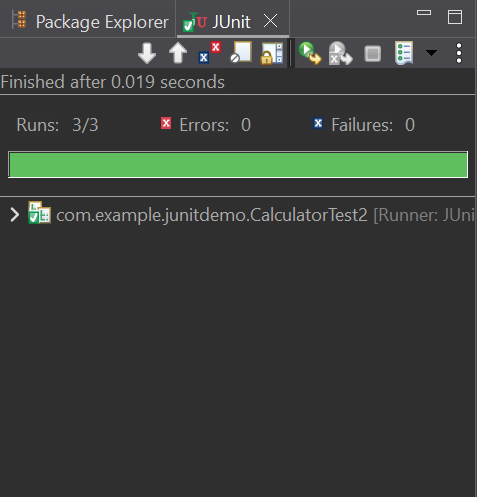
// Assert

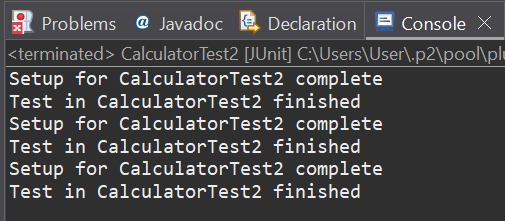
assertEquals(5, result);

}

}

**Output :-**

****

****

**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

**Code :-**

**// File: ExternalApi.java**

package com.example.junitdemo;

public interface ExternalApi {

String getData();

}

**// File: MyService.java**

package com.example.junitdemo;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**// File: MyServiceTest.java**

package com.example.junitdemo;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

import static org.junit.jupiter.api.Assertions.\*;

public class MyServiceTest {

@Test

public void testExternalApi() {

// Step 1: Create a mock object

ExternalApi mockApi = mock(ExternalApi.class);

// Step 2: Stub the method

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

// Step 3: Inject mock and test

MyService service = new MyService(mockApi);

String result = service.fetchData();

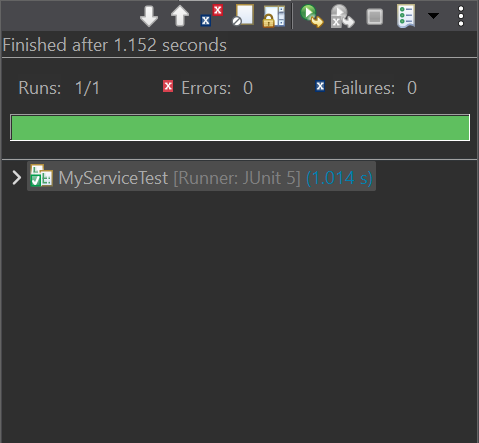
// Step 4: Assert

assertEquals("Mock Data", result);

}

}

**Output :-**

****

**Exercise 2: Verifying Interactions**

**Code :-**

**// File: ExternalApi.java**

package com.example.junitdemo;

public interface ExternalApi {

String getData();

}

**// File: MyService.java**

package com.example.junitdemo;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**// File: MyServiceTest.java**

package com.example.mockingdemo;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Step 1: Create a mock of ExternalApi

ExternalApi mockApi = mock(ExternalApi.class);

// Step 2: Use mock in MyService

MyService service = new MyService(mockApi);

service.fetchData(); // This should call mockApi.getData()

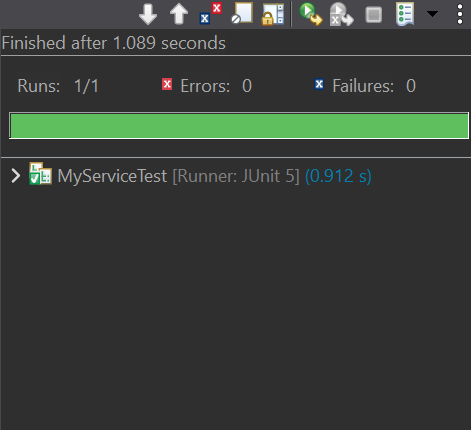
// Step 3: Verify interaction

verify(mockApi).getData(); //

}

}

**Output :-**

****