**JUnit Testing Exercises**

**Exercise 1: Setting Up JUnit**

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

**Code:**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

import static org.junit.Assert.\*;

import org.junit.Test;

public class CalculatorTest {

Calculator calc = new Calculator();

@Test

public void testAdd() {

assertEquals(5, calc.add(2, 3));

}

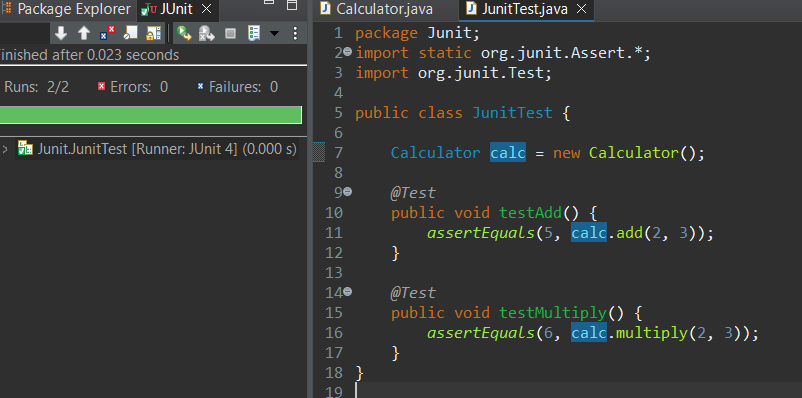
@Test

public void testMultiply() {

assertEquals(6, calc.multiply(2, 3));

}}

**Output:**



**Exercise 3: Assertions in Junit**

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

**Code:**

import static org.junit.Assert.\*;

import org.junit.Test;

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);

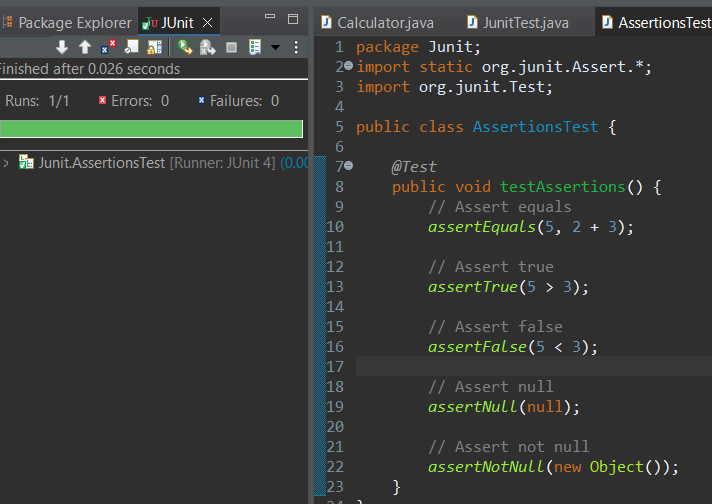
// Assert not null

assertNotNull(new Object());

}

}

**Output:**

****

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

**Scenario: You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.**

**Code:**

public class BankAccount {

private int balance;

public BankAccount(int initialBalance) {

this.balance = initialBalance;

}

public void deposit(int amount) {

balance += amount;

}

public void withdraw(int amount) {

balance -= amount;

}

public int getBalance() {

return balance;

}

}

import static org.junit.Assert.\*;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

public class BankAccountTest {

private BankAccount account;

@Before

public void setUp() {

// Arrange: Create a new BankAccount before each test

account = new BankAccount(100);

}

@After

public void tearDown() {

// Clean up after each test

account = null;

}

@Test

public void testDeposit() {

// Act

account.deposit(50);

// Assert

assertEquals(150, account.getBalance());

}

@Test

public void testWithdraw() {

// Act

account.withdraw(30);

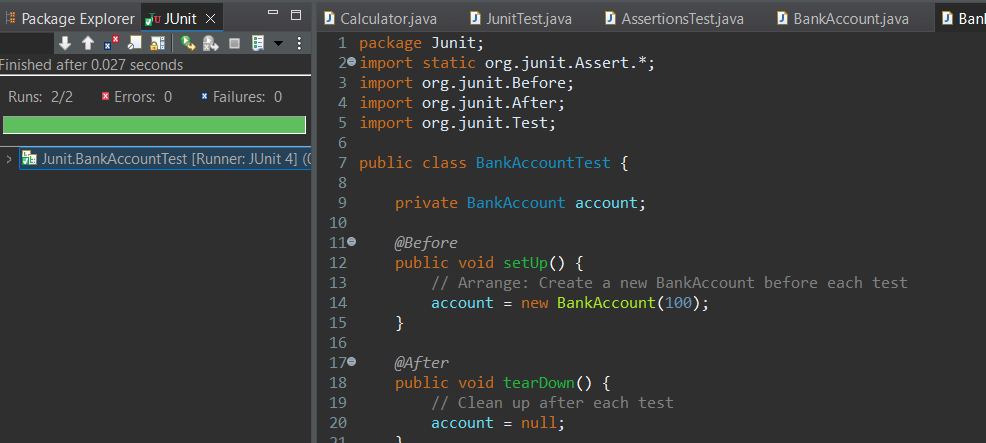
// Assert

assertEquals(70, account.getBalance());

}

}

**Output:**

****

**Mockito Hands-On 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:**

package com.example;

public interface ExternalApi {

String getData();

}

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

package com.example;

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

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testExternalApi() {

// Create mock

ExternalApi mockApi = mock(ExternalApi.class);

// Stub getData() to return "Mock Data"

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

// Inject mock into service

MyService service = new MyService(mockApi);

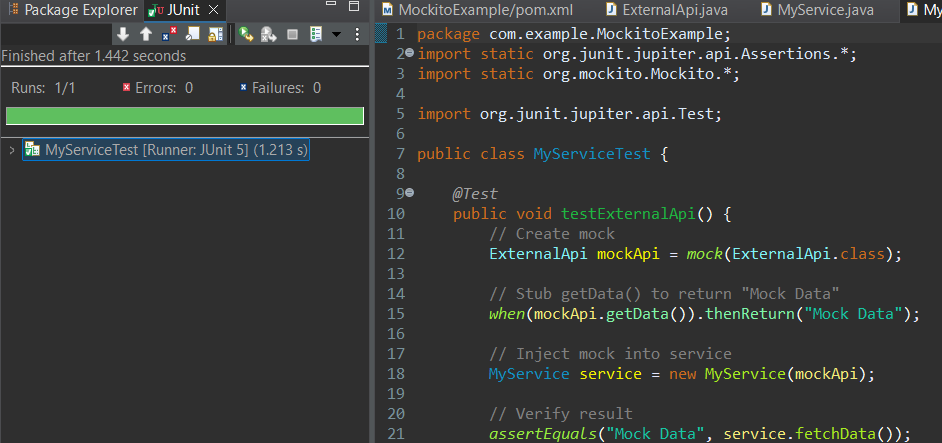
// Verify result

assertEquals("Mock Data", service.fetchData());

}

}

**Output:**

****

**Exercise 2: Verifying Interactions**

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

**Code:**

package com.example;

public interface ExternalApi {

String getData();

}

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData(); // This interaction should be verified

}

}

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

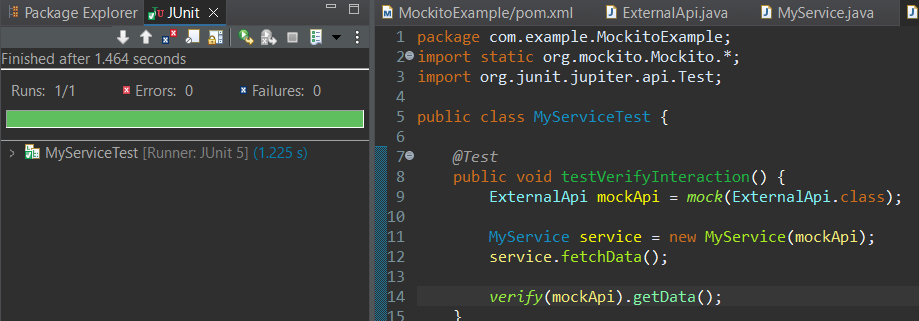
service.fetchData();

verify(mockApi).getData();

}

}

**Output:**

****