***WEEK 2***

***Mandatory hands on***

***1. JUnit Testing Exercises***

**Exercise 1: Setting Up Junit**

**Calculator.java**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java**

package com.example;

import static org.junit.Assert.\*;

import org.junit.Test;

public class CalculatorTest {

*@Test*

public void testAdd() {

Calculator calc = new Calculator();

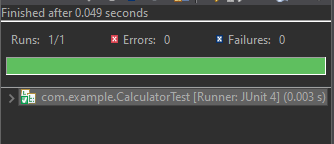
int result = calc.add(5, 3);

*assertEquals*(8, result);

}

}

**OUTPUT**

****

**Exercise 3: Assertions in Junit**

**AssertionsTest.java**

package com.example;

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

Object obj1 = null;

*assertNull*(obj1);

// Assert not null

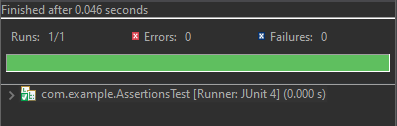
Object obj2 = new Object();

*assertNotNull*(obj2);

}

}

**OUTPUT**

****

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

**BankAccount.java**

package com.example;

public class BankAccount {

private String owner;

private double balance;

public BankAccount(String owner, double balance) {

this.owner = owner;

this.balance = balance;

}

public void deposit(double amount) {

if (amount <= 0) throw new IllegalArgumentException("Deposit must be positive");

balance += amount;

}

public void withdraw(double amount) {

if (amount > balance) throw new IllegalArgumentException("Insufficient balance");

balance -= amount;

}

public double getBalance() {

return balance;

}

public String getOwner() {

return owner;

}

}

**BankAccountTest.java**

package com.example;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

public class BankAccountTest {

private BankAccount account;

// Setup method - runs before each test

*@Before*

public void setUp() {

account = new BankAccount("Indhumathi", 1000.0);

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

}

// Teardown method - runs after each test

*@After*

public void tearDown() {

account = null;

System.***out***.println("Teardown complete");

}

*@Test*

public void testDeposit() {

// Arrange - Already done in setUp()

// Act

account.deposit(500.0);

// Assert

*assertEquals*(1500.0, account.getBalance(), 0.001);

}

*@Test*

public void testWithdraw() {

// Arrange

// Act

account.withdraw(400.0);

// Assert

*assertEquals*(600.0, account.getBalance(), 0.001);

}

*@Test*(expected = IllegalArgumentException.class)

public void testWithdrawInsufficientFunds() {

// Act

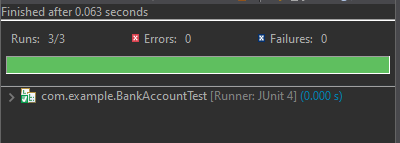
account.withdraw(1500.0); // Should throw exception

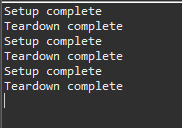
// Assert is handled by expected exception

}

}

**OUTPUT**

****

****

**Exercise 1: Mocking and Stubbing**

**MyService.java**

package com.example;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.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() {

// 1. Create mock object

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

// 2. Stub the method

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

// 3. Inject mock into service

MyService service = new MyService(mockApi);

// 4. Call method and assert

String result = service.fetchData();

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

}

}

**ExternalApi.java**

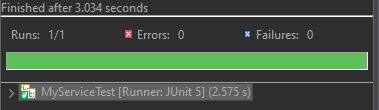
package com.example;

public interface ExternalApi {

String getData();

}

**OUTPUT**



**Exercise 2: Verifying Interactions**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**InteractionTest.java**

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class InteractionTest {

*@Test*

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApi);

// Act

service.fetchData();

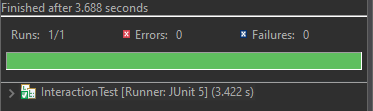
// Assert

*verify*(mockApi).getData(); // verifies that getData() was called once

}

}

**OUTPUT**

****

**Exercise 1: Logging Error Messages and Warning Levels**

**LoggingExample.java**

package com.example;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

private static final Logger ***logger*** = LoggerFactory.*getLogger*(LoggingExample.class);

public static void main(String[] args) {

***logger***.error("This is an error message");

***logger***.warn("This is a warning message");

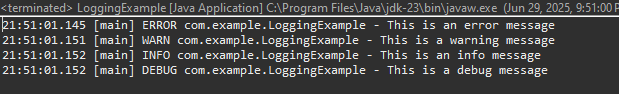
***logger***.info("This is an info message");

***logger***.debug("This is a debug message");

}

}

**OUTPUT**

****