**JUnit Testing Exercises**

**Exercise 1: Setting Up JUnit**

Add JUnit Dependency (for Maven)

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

Calculator.java:

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

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(2, 3);

assertEquals(5, result);

}

}

**Exercise 3: Assertions in Junit**

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 nonNullObj = new Object();

assertNotNull(nonNullObj);

}

}

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

#### Calculator.java

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

#### CalculatorTest.java

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

// Setup: Runs before each test

@Before

public void setUp() {

calculator = new Calculator();

System.out.println("Setup: Calculator instance created");

}

// Teardown: Runs after each test

@After

public void tearDown() {

calculator = null;

System.out.println("Teardown: Calculator instance cleared");

}

@Test

public void testAdd() {

// Arrange

int a = 5;

int b = 3;

// Act

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

// Assert

assertEquals(8, result);

}

@Test

public void testSubtract() {

// Arrange

int a = 10;

int b = 4;

// Act

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

// Assert

assertEquals(6, result);

}

}

**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

// External API to be mocked

interface ExternalApi {

String getData();

}

// Service that depends on ExternalApi

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

// Test Class

public class MyServiceTest {

@Test

public void testExternalApi() {

// Step 1: Create mock

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

// Step 2: Stub method

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

// Step 3: Inject mock into service and test

MyService service = new MyService(mockApi);

String result = service.fetchData();

// Assert

assertEquals("Mock Data", result);

}

}

**Exercise 2: Verifying Interactions**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

// External dependency interface

interface ExternalApi {

String getData();

}

// Service that uses the external API

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

// Test class

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Step 1: Create a mock

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

// Stub getData to return some dummy data

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

// Step 2: Create service and call method

MyService service = new MyService(mockApi);

service.fetchData();

// Step 3: Verify that getData() was called exactly once

verify(mockApi, times(1)).getData(); // Optional: times(1) is the default

}

}

**Logging using SLF4J**

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

### **Add SLF4J and Logback to** pom.xml **(for Maven)**

<dependencies>

<!-- SLF4J API -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<!-- Logback Implementation -->

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

Create the Logging Class

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

// Create the logger instance

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

public static void main(String[] args) {

// Log error and warning messages

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

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

}

}

Customize logback.xml

<configuration>

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<root level="debug">

<appender-ref ref="STDOUT" />

</root>

</configuration>