**JUnit Testing Exercises**

**Exercise 1: Setting Up Junit**

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

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 2: Writing Basic JUnit Tests**

public class MathUtils {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

public int multiply(int a, int b) {

return a \* b;

}

public int divide(int a, int b) {

if (b == 0) throw new IllegalArgumentException("Division by zero is not allowed.");

return a / b;

}

}

import org.junit.Test;

import static org.junit.Assert.\*;

public class MathUtilsTest {

MathUtils mathUtils = new MathUtils();

@Test

public void testAdd() {

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

}

@Test

public void testSubtract() {

assertEquals(4, mathUtils.subtract(10, 6));

}

@Test

public void testMultiply() {

assertEquals(15, mathUtils.multiply(3, 5));

}

@Test

public void testDivide() {

assertEquals(2, mathUtils.divide(10, 5));

}

@Test(expected = IllegalArgumentException.class)

public void testDivideByZero() {

mathUtils.divide(10, 0);

}

}

Output:-



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

public class StringUtils {

public String reverse(String input) {

if (input == null) return null;

return new StringBuilder(input).reverse().toString();

}

public boolean isPalindrome(String input) {

if (input == null) return false;

String reversed = reverse(input);

return input.equalsIgnoreCase(reversed);

}

}

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class StringUtilsTest {

private StringUtils stringUtils;

@Before

public void setUp() {

// Setup: Create a new instance before each test

stringUtils = new StringUtils();

System.out.println("Setup done");

}

@After

public void tearDown() {

// Teardown: Clean up after each test (if needed)

stringUtils = null;

System.out.println("Teardown done");

}

@Test

public void testReverse() {

// Arrange

String input = "hello";

// Act

String result = stringUtils.reverse(input);

// Assert

assertEquals("olleh", result);

}

@Test

public void testIsPalindrome\_true() {

// Arrange

String input = "madam";

// Act

boolean result = stringUtils.isPalindrome(input);

// Assert

assertTrue(result);

}

@Test

public void testIsPalindrome\_false() {

// Arrange

String input = "java";

// Act

boolean result = stringUtils.isPalindrome(input);

// Assert

assertFalse(result);

}

@Test

public void testIsPalindrome\_null() {

// Arrange

String input = null;

// Act

boolean result = stringUtils.isPalindrome(input);

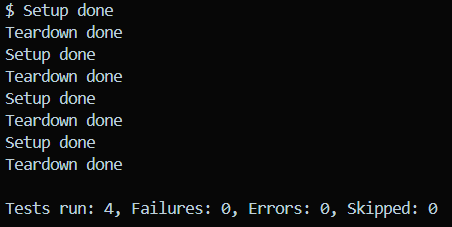
// Assert

assertFalse(result);

}

}

Output:-



**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

public interface ExternalApi {

String getData();

}

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

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() {

// Step 1: Create a mock of ExternalApi

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

// Step 2: Stub getData() to return predefined value

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

// Step 3: Inject mock into MyService and test

MyService service = new MyService(mockApi);

String result = service.fetchData();

// Step 4: Assert the result

assertEquals("Mock Data", result);

}

}

Output:-



**Exercise 2: Verifying Interactions**

public interface ExternalApi {

String getData();

}

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData(); // should be verified

}

}

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Step 1: Create mock

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

// Step 2: Call the method through MyService

MyService service = new MyService(mockApi);

service.fetchData();

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

verify(mockApi).getData(); // or verify(mockApi, times(1)).getData();

}

}

Output:-



**Logging using SLF4J**

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

<dependencies>

<!-- SLF4J API -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<!-- Logback Classic (Backend for SLF4J) -->

<dependency>

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

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

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");

}

}

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");

}

}

<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="warn">

<appender-ref ref="STDOUT"/>

</root>

</configuration>

Output:-

