**JUNIT BASIC TESTING.**

**Exercise 1: Setting Up JUnit**

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

**SOLUTION:**

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

**CLASS CALCULATOR:**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**JUNIT TEST CLASS:**

package com.example;

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 2: Writing Basic JUnit Tests**

Scenario: You need to write basic JUnit tests for a simple Java class.

**SOLUTION:**

**CLASS MATHUTILS:**

package com.example;

public class MathUtils {

public int multiply(int a, int b) {

return a \* b;

}

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**JUNIT TEST:**

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class MathUtilsTest {

@Test

public void testMultiply() {

MathUtils math = new MathUtils();

assertEquals(20, math.multiply(4, 5));

}

@Test

public void testIsEven() {

MathUtils math = new MathUtils();

assertTrue(math.isEven(10));

assertFalse(math.isEven(7));

}

}

**Exercise 3: Assertions in JUnit**

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

**SOLUTION:**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**TESTING:**

package com.example;

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());

}

}

**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.

**SOLUTION:**

package com.example;

public class Calculator {

public int subtract(int a, int b) {

return a - b;

}

public int divide(int a, int b) {

if (b == 0) throw new IllegalArgumentException("Cannot divide by zero");

return a / b;

}

}

**TESTING:**

package com.example;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

@Before

public void setUp() {

calculator = new Calculator();

System.out.println("Before each test");

}

@After

public void tearDown() {

System.out.println("After each test");

}

@Test

public void testSubtract() {

// Arrange

int a = 10;

int b = 4;

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

assertEquals(6, result);

}

@Test(expected = IllegalArgumentException.class)

public void testDivideByZero() {

// Act & Assert (Exception expected)

calculator.divide(5, 0);

}

@Test

public void testDivide() {

// Arrange

int a = 20;

int b = 4;

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

assertEquals(5, result);

}

}