Exercise 1: Setting Up JUnit Scenario:

import org.junit.Test;

import static org.junit.Assert.\*;

// ✅ Class to be tested

class Calculator {

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("Cannot divide by zero");

return a / b;

}

}

// ✅ JUnit test class

public class CalculatorTestExample {

@Test

public void testAdd() {

Calculator calc = new Calculator();

int result = calc.add(10, 5);

System.out.println("Add Result: " + result); // This will be printed in output

assertEquals(15, result);

}

@Test

public void testSubtract() {

Calculator calc = new Calculator();

int result = calc.subtract(10, 3);

System.out.println("Subtract Result: " + result);

assertEquals(7, result);

}

@Test

public void testMultiply() {

Calculator calc = new Calculator();

int result = calc.multiply(4, 3);

System.out.println("Multiply Result: " + result);

assertEquals(12, result);

}

@Test

public void testDivide() {

Calculator calc = new Calculator();

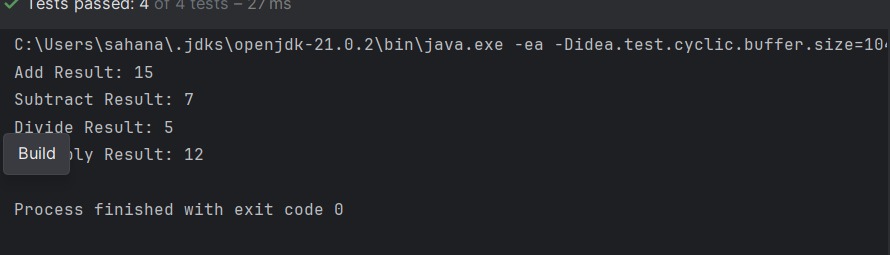
int result = calc.divide(20, 4);

System.out.println("Divide Result: " + result);

assertEquals(5, result);

}

}



Exercise 3: Assertions in Junit

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

// 🔢 Inputs

int input1 = 2;

int input2 = 3;

int sum = input1 + input2;

boolean isGreater = input1 + input2 > 4;

boolean isLess = input1 + input2 < 4;

Object nullObj = null;

Object nonNullObj = new Object();

// 🧪 Assertions (Expected Results)

assertEquals("2 + 3 should be 5", 5, sum);

assertTrue("Sum should be greater than 4", isGreater);

assertFalse("Sum should not be less than 4", isLess);

assertNull("Object should be null", nullObj);

assertNotNull("Object should not be null", nonNullObj);

// 🖨 Output-like behavior using prints

System.out.println("Inputs: " + input1 + " + " + input2);

System.out.println("Sum: " + sum);

System.out.println("Is Greater Than 4: " + isGreater);

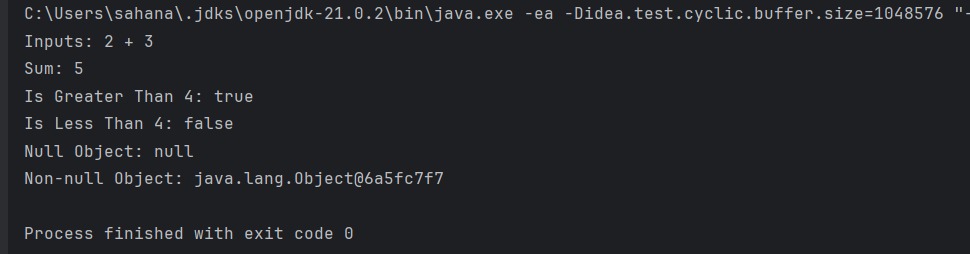
System.out.println("Is Less Than 4: " + isLess);

System.out.println("Null Object: " + nullObj);

System.out.println("Non-null Object: " + nonNullObj);

}

}



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

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

// Business logic class

class BankAccount {

private int balance;

public BankAccount(int initialBalance) {

this.balance = initialBalance;

}

public void deposit(int amount) {

balance += amount;

}

public void withdraw(int amount) {

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

balance -= amount;

}

public int getBalance() {

return balance;

}

}

// JUnit test class

public class BankAccountTest {

private BankAccount account;

// Setup: runs before every @Test

@Before

public void setUp() {

System.out.println("Setting up account with initial balance of 100");

account = new BankAccount(100); // Arrange

}

// Teardown: runs after every @Test

@After

public void tearDown() {

System.out.println("Tearing down account after test");

account = null;

}

@Test

public void testDeposit() {

// Act

account.deposit(50);

// Assert

System.out.println("Testing deposit...");

assertEquals(150, account.getBalance());

}

@Test

public void testWithdraw() {

// Act

account.withdraw(40);

// Assert

System.out.println("Testing withdraw...");

assertEquals(60, account.getBalance());

}

@Test(expected = IllegalArgumentException.class)

public void testWithdrawInsufficientFunds() {

// Act

System.out.println("Testing withdraw with insufficient funds...");

account.withdraw(200); // Should throw

}

}

