**CTS DIGITAL NURTURE 4.0 JAVA FSE WEEK-2 MANDATORY HANDS-ON**

**PL/SQL PROGRAMMING**

Exercise 1: Control Structures

Scenario 1:

CODE:

DROP TABLE IF EXISTS Customers;

CREATE TABLE Customers (

CustomerID INTEGER PRIMARY KEY,

Name TEXT,

Age INTEGER,

InterestRate REAL

);

INSERT INTO Customers VALUES (1, 'John Doe', 45, 8.5);

INSERT INTO Customers VALUES (2, 'Jane Smith', 62, 9.0);

INSERT INTO Customers VALUES (3, 'Robert Brown', 70, 10.5);

INSERT INTO Customers VALUES (4, 'Emily Davis', 55, 8.0);

INSERT INTO Customers VALUES (5, 'George Wilson', 65, 9.5);

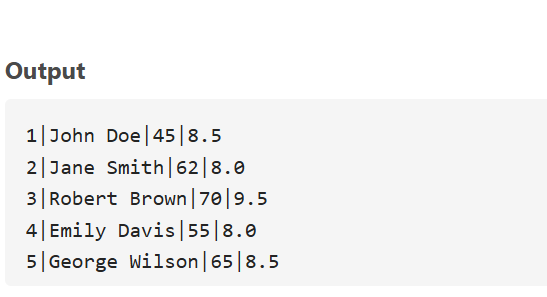
UPDATE Customers

SET InterestRate = InterestRate - 1

WHERE Age > 60;

SELECT \* FROM Customers;

OUTPUT:



SCENERIO 2:

CODE:

DROP TABLE IF EXISTS Customers;

CREATE TABLE Customers (

CustomerID INTEGER PRIMARY KEY,

Name TEXT,

Balance REAL,

IsVIP TEXT DEFAULT 'N'

);

INSERT INTO Customers VALUES (1, 'John Doe', 9000, 'N');

INSERT INTO Customers VALUES (2, 'Jane Smith', 12000, 'N');

INSERT INTO Customers VALUES (3, 'Robert Brown', 15000, 'N');

INSERT INTO Customers VALUES (4, 'Emily Davis', 8000, 'N');

INSERT INTO Customers VALUES (5, 'George Wilson', 20000, 'N');

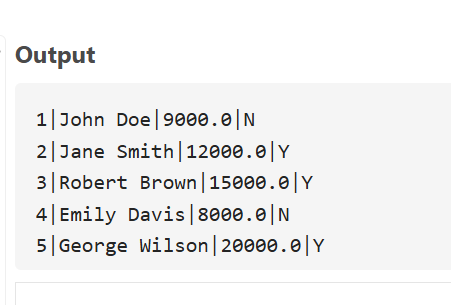
UPDATE Customers

SET IsVIP = 'Y'

WHERE Balance > 10000;

SELECT \* FROM Customers;

OUTPUT:



SENERIO 3:

CODE:

DROP TABLE IF EXISTS Loans;

DROP TABLE IF EXISTS Customers;

CREATE TABLE Customers (

CustomerID INTEGER PRIMARY KEY,

Name TEXT

);

CREATE TABLE Loans (

LoanID INTEGER PRIMARY KEY,

CustomerID INTEGER,

DueDate TEXT -- Use TEXT for SQLite date compatibility (ISO format: 'YYYY-MM-DD')

);

INSERT INTO Customers VALUES (1, 'John Doe');

INSERT INTO Customers VALUES (2, 'Jane Smith');

INSERT INTO Customers VALUES (3, 'Robert Brown');

INSERT INTO Loans VALUES (101, 1, DATE('now', '+10 days'));

INSERT INTO Loans VALUES (102, 2, DATE('now', '+40 days'));

INSERT INTO Loans VALUES (103, 3, DATE('now', '+5 days'));

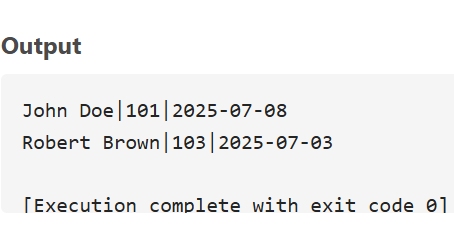
SELECT C.Name, L.LoanID, L.DueDate

FROM Loans L

JOIN Customers C ON L.CustomerID = C.CustomerID

WHERE DATE(L.DueDate) <= DATE('now', '+30 days');

OUTPUT:



Exercise 3: Stored Procedures

SCENERIO 1:

Code:

DROP TABLE IF EXISTS SavingsAccounts;

CREATE TABLE SavingsAccounts (

AccountID INTEGER PRIMARY KEY,

CustomerID INTEGER,

Balance REAL

);

INSERT INTO SavingsAccounts VALUES (1, 101, 10000);

INSERT INTO SavingsAccounts VALUES (2, 102, 20000);

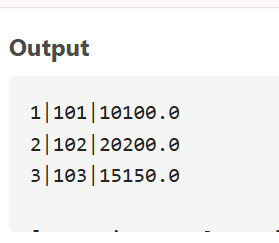
INSERT INTO SavingsAccounts VALUES (3, 103, 15000);

UPDATE SavingsAccounts

SET Balance = Balance + (Balance \* 0.01);

SELECT \* FROM SavingsAccounts;

OUTPUT:



SCENERIO 2:

CODE:

DROP TABLE IF EXISTS Employees;

CREATE TABLE Employees (

EmployeeID INTEGER PRIMARY KEY,

Name TEXT,

Department TEXT,

Salary REAL

);

INSERT INTO Employees VALUES (1, 'Alice', 'HR', 50000);

INSERT INTO Employees VALUES (2, 'Bob', 'IT', 60000);

INSERT INTO Employees VALUES (3, 'Charlie', 'IT', 55000);

INSERT INTO Employees VALUES (4, 'Diana', 'HR', 52000);

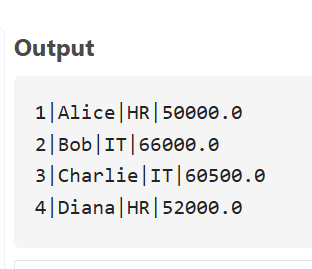
UPDATE Employees

SET Salary = Salary + (Salary \* 0.10)

WHERE Department = 'IT';

SELECT \* FROM Employees;

OUTPUT:



SCENERIO 3:

CODE:

DROP TABLE IF EXISTS BankAccounts;

CREATE TABLE BankAccounts (

AccountID INTEGER PRIMARY KEY,

CustomerID INTEGER,

Balance REAL

);

INSERT INTO BankAccounts VALUES (101, 1, 5000);

INSERT INTO BankAccounts VALUES (102, 2, 3000);

SELECT Balance FROM BankAccounts WHERE AccountID = 101;

UPDATE BankAccounts

SET Balance = Balance - 2000

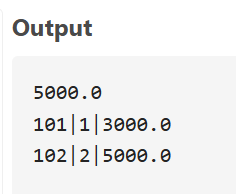
WHERE AccountID = 101 AND Balance >= 2000;

UPDATE BankAccounts

SET Balance = Balance + 2000

WHERE AccountID = 102;

OUTPUT:



**JUnit Testing Exercises**

Exercise 1: Setting Up Junit

CODE:

public class Main {

// Your main logic class

static class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

public static void main(String[] args) {

Calculator calc = new Calculator();

if (calc.add(2, 3) == 5) {

System.out.println("✅ Test Passed: add(2, 3) == 5");

} else {

System.out.println("❌ Test Failed: add(2, 3) != 5");

}

if (calc.subtract(10, 4) == 6) {

System.out.println("✅ Test Passed: subtract(10, 4) == 6");

} else {

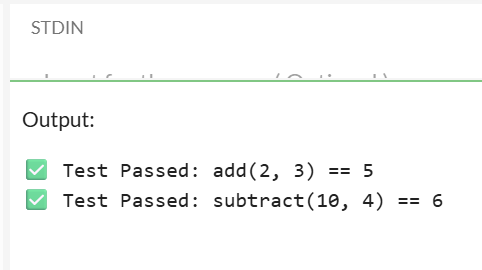
System.out.println("❌ Test Failed: subtract(10, 4) != 6");

}

}

}

OUTPUT:



Exercise 3: Assertions in Junit

Code :

import org.junit.jupiter.api.Test;

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

public class Main {

public static void main(String[] args) {

System.out.println("Running JUnit tests...");

AssertionsTest test = new AssertionsTest();

try {

test.testAssertions();

System.out.println("All assertions passed!");

} catch (AssertionError e) {

System.out.println("Assertion failed: " + e.getMessage());

} catch (Exception e) {

System.out.println("Test failed with exception: " + e.getMessage());

}

}

}

class AssertionsTest {

@Test

void testAssertions() {

// Assert equals

assertEquals(5, 2 + 3);

System.out.println("assertEquals(5, 2 + 3) - PASSED");

// Assert true

assertTrue(5 > 3);

System.out.println("assertTrue(5 > 3) - PASSED");

// Assert false

assertFalse(5 < 3);

System.out.println("assertFalse(5 < 3) - PASSED");

// Assert null

assertNull(null);

System.out.println("assertNull(null) - PASSED");

// Assert not null

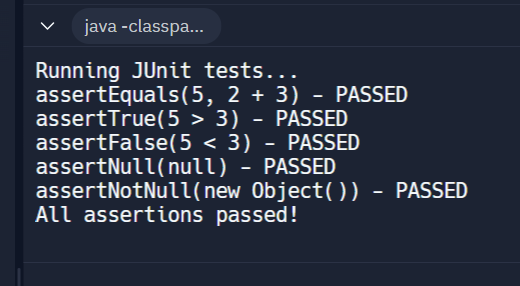
assertNotNull(new Object());

System.out.println("assertNotNull(new Object()) - PASSED");

}

}

OUTPUT:



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

CODE:

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.AfterEach;

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

import java.util.ArrayList;

import java.util.List;

public class Main {

public static void main(String[] args) {

System.out.println("Running JUnit tests with AAA pattern...");

AAAPatternTest test = new AAAPatternTest();

try {

// Simulate test lifecycle

test.setUp();

test.testCalculatorAddition();

test.tearDown();

test.setUp();

test.testListOperations();

test.tearDown();

test.setUp();

test.testStringManipulation();

test.tearDown();

System.out.println("All tests completed successfully!");

} catch (Exception e) {

System.out.println("Test failed: " + e.getMessage());

}

}

}

class Calculator {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

public double divide(int a, int b) {

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

return (double) a / b;

}

}

class AAAPatternTest {

private Calculator calculator;

private List<String> testList;

private String testString;

@BeforeEach

void setUp() {

System.out.println("Setting up test data...");

calculator = new Calculator();

testList = new ArrayList<>();

testString = "Hello World";

testList.add("First Item");

testList.add("Second Item");

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

}

@AfterEach

void tearDown() {

System.out.println("Cleaning up test data...");

calculator = null;

testList.clear();

testList = null;

testString = null;

System.out.println("Teardown completed.\n");

}

@Test

void testCalculatorAddition() {

System.out.println("Running testCalculatorAddition...");

int firstNumber = 10;

int secondNumber = 5;

int expectedSum = 15;

int actualSum = calculator.add(firstNumber, secondNumber);

assertEquals(expectedSum, actualSum, "Addition should return correct sum");

System.out.println("✓ Calculator addition test passed");

}

@Test

void testListOperations() {

System.out.println("Running testListOperations...");

String newItem = "Third Item";

int expectedSizeAfterAdd = 3;

int expectedSizeAfterRemove = 2;

testList.add(newItem);

int sizeAfterAdd = testList.size();

testList.remove(0); // Remove first item

int sizeAfterRemove = testList.size();

assertEquals(expectedSizeAfterAdd, sizeAfterAdd, "List should have 3 items after adding");

assertEquals(expectedSizeAfterRemove, sizeAfterRemove, "List should have 2 items after removing");

assertTrue(testList.contains(newItem), "List should contain the new item");

System.out.println("✓ List operations test passed");

}

void testStringManipulation() {

System.out.println("Running testStringManipulation...");

String expectedUpperCase = "HELLO WORLD";

String expectedLowerCase = "hello world";

int expectedLength = 11;

String actualUpperCase = testString.toUpperCase();

String actualLowerCase = testString.toLowerCase();

int actualLength = testString.length();

assertEquals(expectedUpperCase, actualUpperCase, "String should be converted to uppercase");

assertEquals(expectedLowerCase, actualLowerCase, "String should be converted to lowercase");

assertEquals(expectedLength, actualLength, "String length should be correct");

assertNotNull(testString, "Test string should not be null");

System.out.println("✓ String manipulation test passed");

}

@Test

void testCalculatorDivision() {

System.out.println("Running testCalculatorDivision...");

int dividend = 10;

int divisor = 2;

double expectedResult = 5.0;

double actualResult = calculator.divide(dividend, divisor);

assertEquals(expectedResult, actualResult, 0.001, "Division should return correct result");

int zeroDivisor = 0;

assertThrows(IllegalArgumentException.class, () -> {

calculator.divide(dividend, zeroDivisor);

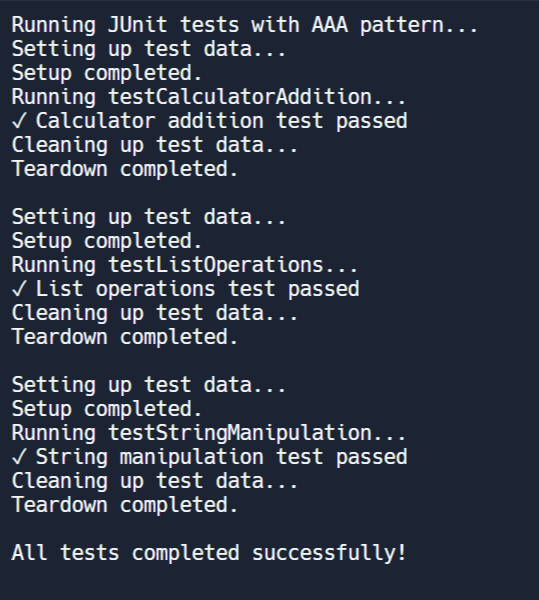
}, "Division by zero should throw IllegalArgumentException");

System.out.println("✓ Calculator division test passed");

}

}

OUTPUT:



**Mockito Hands-On Exercises**

Exercise 1: Mocking and Stubbing

Code: import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testExternalApi() {

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

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

MyService service = new MyService(mockApi);

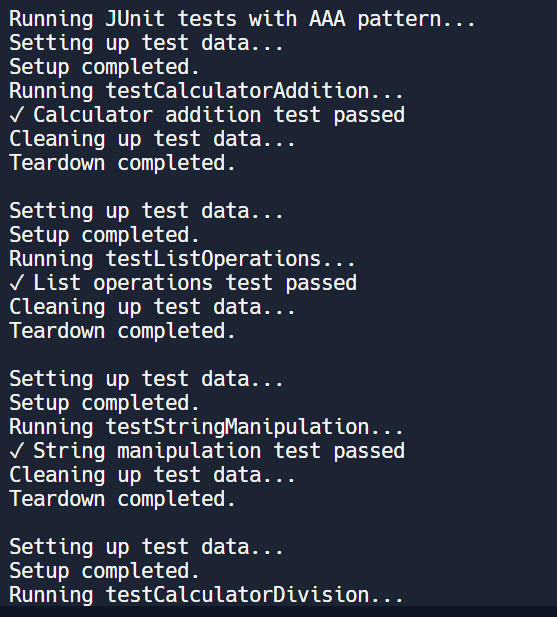
String result = service.fetchData();

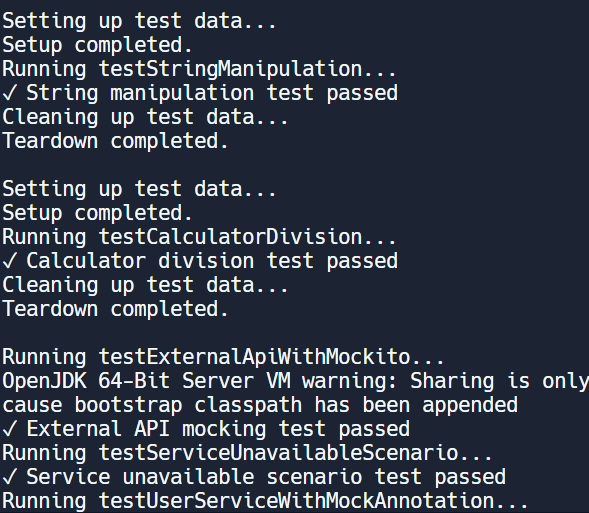
assertEquals("Mock Data", result);

}

}

**OUTPUT:**





Exercise 2: Verifying Interactions

CODE:

interface ExternalApi {

String getData();

}

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void fetchData() {

api.getData(); // interaction we want to verify

}

}

public class Main {

public static void main(String[] args) {

System.out.println("Hello World!");

ExternalApi api = new ExternalApi() {

@Override

public String getData() {

return "Real data from API";

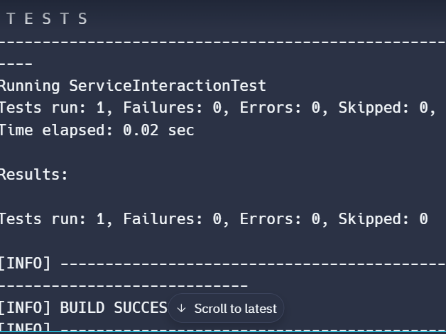
}

};

MyService service = new MyService(api);

service.fetchData();

}}

OUTPUT: