**Mandatory hands-on**

**Exercise 1: Control Structures**

**Scenario 1:**

**CODE:**

CREATE TABLE Customers (

CustomerID INTEGER PRIMARY KEY,

Name TEXT,

DOB TEXT, -- YYYY-MM-DD format

Balance REAL,

LastModified TEXT

);

CREATE TABLE Loans (

LoanID INTEGER PRIMARY KEY,

CustomerID INTEGER,

LoanAmount REAL,

InterestRate REAL,

StartDate TEXT,

EndDate TEXT,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES

(1, 'John Doe', '1950-05-15', 12000, DATE('now')),

(2, 'Jane Smith', '1990-07-20', 1500, DATE('now'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES

(1, 1, 5000, 5.00, DATE('now'), DATE('now', '+12 months')),

(2, 2, 3000, 4.50, DATE('now'), DATE('now', '+12 months'));

UPDATE Loans

SET InterestRate = InterestRate - (InterestRate \* 0.01)

WHERE CustomerID IN (

SELECT CustomerID

FROM Customers

WHERE (CAST(strftime('%Y', 'now') AS INTEGER) - CAST(strftime('%Y', DOB) AS INTEGER)) > 60

);

SELECT

l.LoanID,

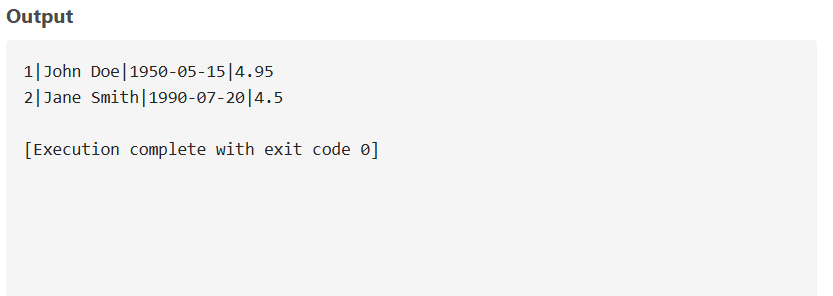
c.Name AS CustomerName,

c.DOB,

l.InterestRate AS UpdatedInterestRate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID;

**OUTPUT:**

**Scenario 2:**

**CODE:**

DROP TABLE IF EXISTS Customers;

CREATE TABLE Customers (

CustomerID INTEGER PRIMARY KEY,

Name TEXT,

DOB DATE,

Balance REAL,

LastModified DATE,

IsVIP TEXT DEFAULT 'FALSE'

);

INSERT INTO Customers VALUES

(1, 'John Doe', '1985-05-15', 8000, DATE('now'), 'FALSE'),

(2, 'Jane Smith', '1990-07-20', 15000, DATE('now'), 'FALSE'),

(3, 'Michael Ray', '1970-02-12', 20000, DATE('now'), 'FALSE');

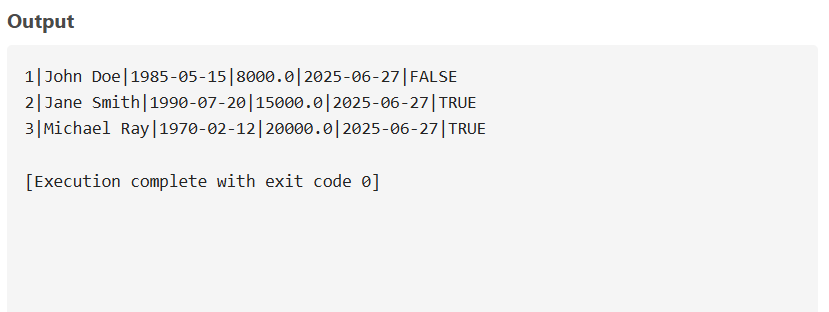
UPDATE Customers

SET IsVIP = 'TRUE'

WHERE Balance > 10000;

SELECT \* FROM Customers;

**OUTPUT:**



**Scenario 3:**

**CODE:**

DROP TABLE IF EXISTS Customers;

DROP TABLE IF EXISTS Loans;

CREATE TABLE Customers (

CustomerID INTEGER PRIMARY KEY,

Name TEXT,

DOB DATE

);

CREATE TABLE Loans (

LoanID INTEGER PRIMARY KEY,

CustomerID INTEGER,

LoanAmount REAL,

InterestRate REAL,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES

(1, 'John Doe', '1985-05-15'),

(2, 'Jane Smith', '1990-07-20'),

(3, 'Michael Ray', '1970-02-12');

INSERT INTO Loans VALUES

(1, 1, 5000, 5, '2022-01-01', DATE('now', '+10 days')), -- Due soon

(2, 2, 10000, 4.5, '2023-01-01', DATE('now', '+40 days')), -- Not due yet

(3, 3, 15000, 6, '2021-01-01', DATE('now', '+5 days')); -- Due very soon

SELECT

c.Name AS CustomerName,

l.LoanID,

l.LoanAmount,

l.EndDate,

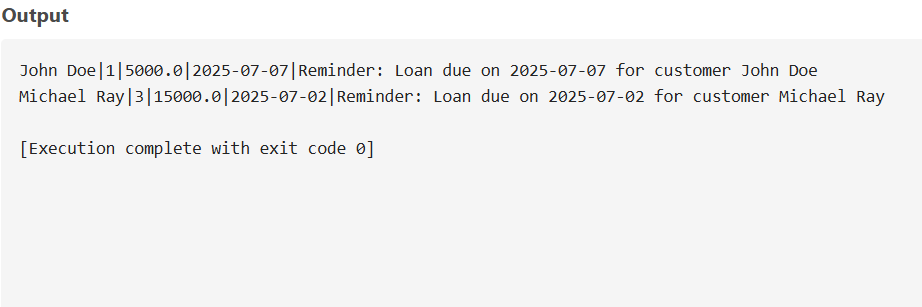
'Reminder: Loan due on ' || l.EndDate || ' for customer ' || c.Name AS ReminderMessage

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN DATE('now') AND DATE('now', '+30 days');

**OUTPUT:**



**Exercise 2: Stored Procedures**

**Scenario 1:**

**CODE:**

DROP TABLE IF EXISTS Accounts

CREATE TABLE Accounts (

AccountID INTEGER PRIMARY KEY,

CustomerID INTEGER,

AccountType TEXT,

Balance REAL,

LastModified DATE

);

INSERT INTO Accounts VALUES

(1, 101, 'Savings', 1000.00, DATE('now')),

(2, 102, 'Savings', 2000.00, DATE('now')),

(3, 103, 'Checking', 1500.00, DATE('now'));

SELECT 'Before interest' AS Note, AccountID, AccountType, Balance FROM Accounts;

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

SELECT 'After interest' AS Note, AccountID, AccountType, Balance FROM Accounts;

**OUTPUT:**



**Scenario 2:**

**CODE:**

DROP TABLE IF EXISTS Employees;

CREATE TABLE Employees (

EmployeeID INTEGER PRIMARY KEY,

Name TEXT,

Position TEXT,

Salary REAL,

Department TEXT,

HireDate DATE

);

INSERT INTO Employees VALUES

(1, 'Alice Johnson', 'Manager', 70000, 'HR', '2015-06-15'),

(2, 'Bob Brown', 'Developer', 60000, 'IT', '2017-03-20'),

(3, 'Charlie Lee', 'Developer', 62000, 'IT', '2018-08-10');

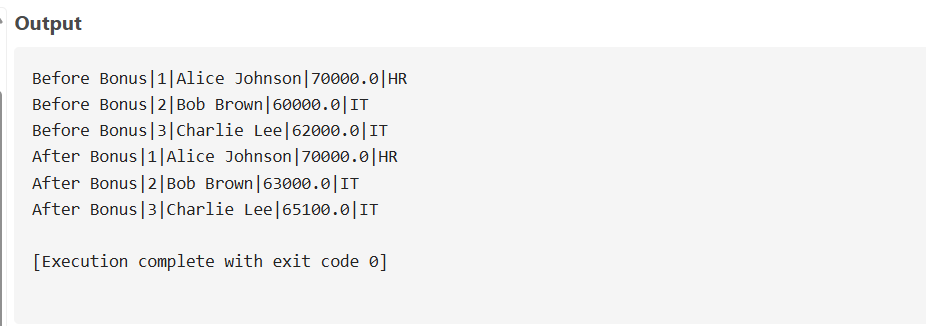
SELECT 'Before Bonus' AS Note, EmployeeID, Name, Salary, Department FROM Employees;

UPDATE Employees

SET Salary = Salary + (Salary \* 5.0 / 100)

WHERE Department = 'IT';

SELECT 'After Bonus' AS Note, EmployeeID, Name, Salary, Department FROM Employees;

**OUTPUT:**

**Scenario 3:**

**CODE:**

DROP TABLE IF EXISTS Accounts;

CREATE TABLE Accounts (

AccountID INTEGER PRIMARY KEY,

CustomerID INTEGER,

AccountType TEXT,

Balance REAL,

LastModified DATE

);

INSERT INTO Accounts VALUES

(1, 101, 'Savings', 1000.00, DATE('now')),

(2, 101, 'Checking', 500.00, DATE('now'));

SELECT 'Before Transfer' AS Note, AccountID, Balance FROM Accounts;

BEGIN TRANSACTION;

SELECT Balance FROM Accounts WHERE AccountID = 1;

UPDATE Accounts

SET Balance = Balance – 300

WHERE AccountID = 1 AND Balance >= 300;

UPDATE Accounts

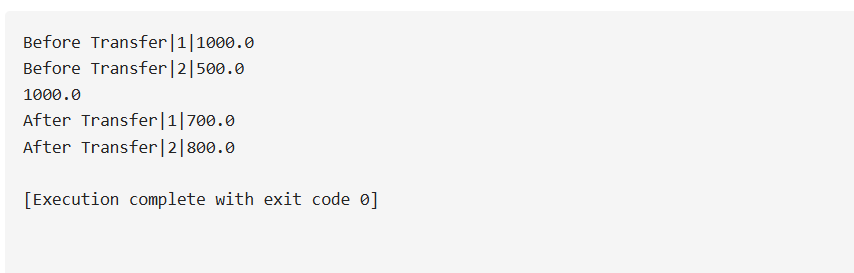
SET Balance = Balance + 300

WHERE AccountID = 2;

COMMIT;

SELECT 'After Transfer' AS Note, AccountID, Balance FROM Accounts;

**OUTPUT:**



**JUnit Testing Exercises Exercise 1:**

**Setting Up Junit**

**Code:**

**Calculator.java** :

public class Calculator {

public int add(int a, int b) {

return a + b;

}}

**Main.java** :

public class Main {

public static void main(String[] args) {

Calculator calc = new Calculator();

System.out.println("Calculator Demo");

System.out.println("2 + 3 = " + calc.add(2, 3));

System.out.println("10 + 15 = " + calc.add(10, 15));

}

}

**CalculatorTest.java** :

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

public void testAdd() {

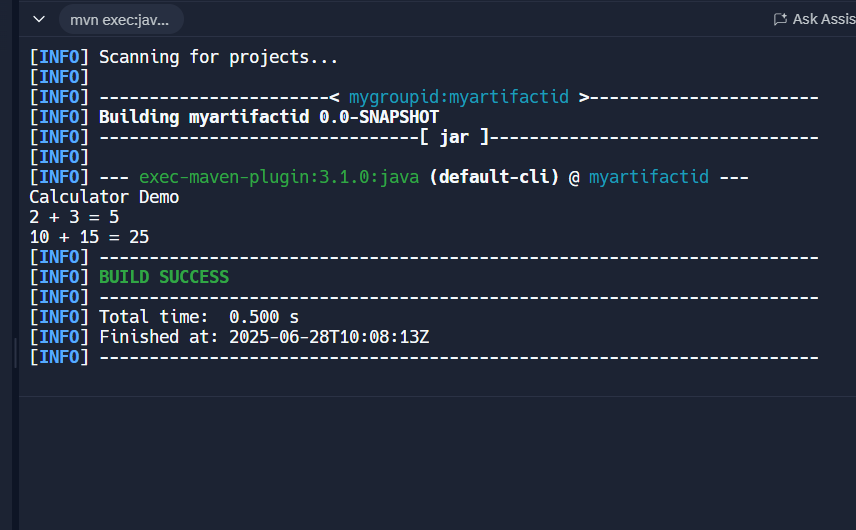
Calculator calc = new Calculator();

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

}

}

**OUTPUT:**



**Exercise 3: Assertions in Junit**

**Code:**

**Main.java** :

public class Main {

public static void main(String[] args) {

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

// Example usage of Calculator

Calculator calc = new Calculator();

System.out.println("2 + 3 = " + calc.add(2, 3));

System.out.println("10 - 4 = " + calc.subtract(10, 4));

System.out.println("5 \* 6 = " + calc.multiply(5, 6));

System.out.println("20 / 4 = " + calc.divide(20, 4));

}

}

**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;

}

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");

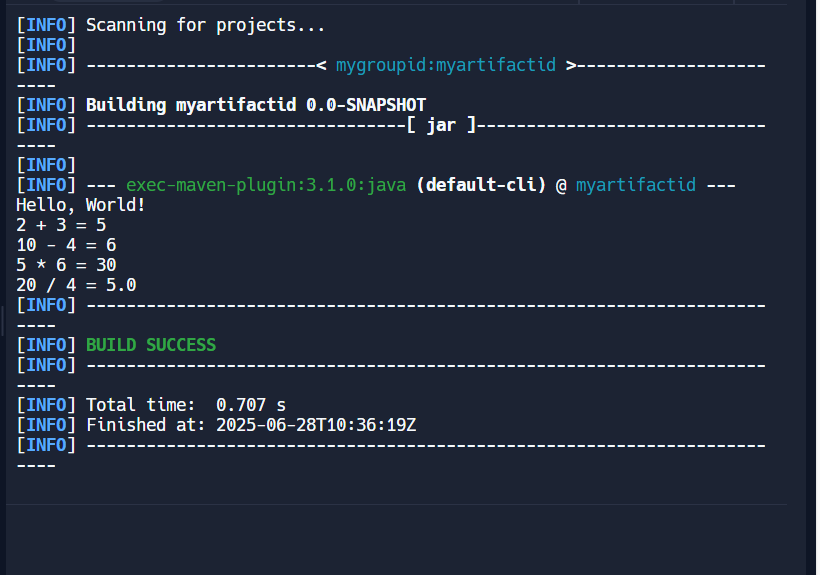
}

return (double) a / b;

}

}

**OUTPUT:**



**Exercise 4:**

**Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit**

**Code:**

MathServiceTest.java:

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class MathServiceTest {

private MathService math;

public void setUp() {

math = new MathService();

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

}

public void tearDown() {

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

}

public void testAdd() {

int a = 7;

int b = 2;

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

assertEquals(9, result);

}

public void testSubtract() {

int a = 10;

int b = 5;

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

assertEquals(5, result);

}

}

**MathService.java:**

public class MathService {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

**Output:**



**Mockito Hands-On Exercises Exercise 1:**

**Mocking and Stubbing**

**Code:**

**Main.java** :

public class Main {

public static void main(String[] args) {

System.out.println("Application started successfully!");

ExternalApi api = new ExternalApi() {

public String getData() {

return "Real API Data";

}

};

MyService service = new MyService(api);

String result = service.fetchData();

System.out.println("Fetched data: " + result);

}

}

**ExternalApi.java** :

public interface ExternalApi {

String getData();

}

**MyService.java** :

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java** :

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

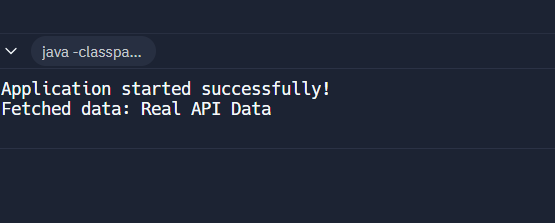
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**OUTPUT:**



**Exercise 2: Verifying Interactions**

**CODE:**

**ExternalApi.java** :

public interface ExternalApi {

String getData();

}

**MyService.java**:

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void fetchData() {

api.getData();

}

}

**MyServiceTest.java** :

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest {

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

**Main.java** :

public class Main {

public static void main(String[] args) {

System.out.println("Running Java Application with Mockito Testing");

MyServiceTest test = new MyServiceTest();

try {

test.testVerifyInteraction();

System.out.println(" Test passed: Mock interaction verified successfully!");

} catch (Exception e) {

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

}

System.out.println("\nDemonstrating service with real implementation:");

ExternalApi realApi = new ExternalApi() {

public String getData() {

return "Real data from external API";

}

};

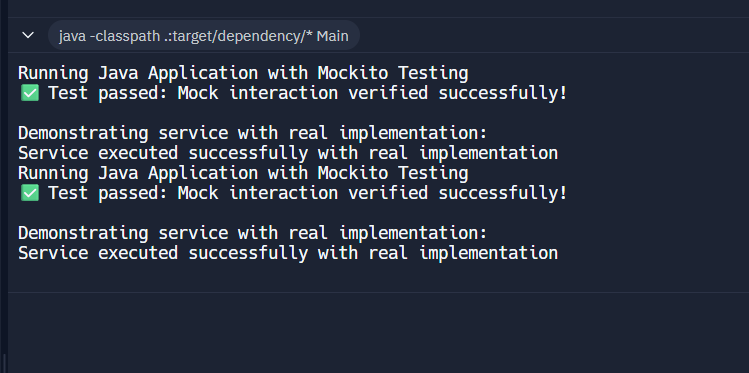
MyService service = new MyService(realApi);

service.fetchData();

System.out.println("Service executed successfully with real implementation");

}

}

**OUTPUT:**

**Logging using SLF4J Exercise 1:**

**Logging Error Messages and Warning Levels**

**Code:**

**Main.java** :

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class Main {

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

public static void main(String[] args) {

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

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

}

}

**Output:**

