**HANDS ON – WEEK 2**

**MODULE – PL-SQL**

**Exercise 1: Control Structures**

**CODE:**

-- DROP TABLES IF EXISTS

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- 1. Create Customers Table

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

Age INT,

LoanInterestRate DECIMAL(5, 2),

Balance DECIMAL(10,2),

IsVIP VARCHAR2(5)

);

-- 2. Create Loans Table

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

LoanAmount DECIMAL(10, 2),

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- 3. Insert Sample Data into Customers

INSERT INTO Customers (CustomerID, Name, Age, LoanInterestRate, Balance, IsVIP)

VALUES (1, 'John Doe', 65, 5.5, 12000, 'FALSE');

INSERT INTO Customers (CustomerID, Name, Age, LoanInterestRate, Balance, IsVIP)

VALUES (2, 'Jane Smith', 55, 6.0, 8000, 'FALSE');

INSERT INTO Customers (CustomerID, Name, Age, LoanInterestRate, Balance, IsVIP)

VALUES (3, 'Michael Brown', 70, 4.8, 10500, 'FALSE');

-- 4. Insert Sample Data into Loans

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, DueDate)

VALUES (1, 1, 5000.00, TO\_DATE('2025-07-15', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, DueDate)

VALUES (2, 2, 10000.00, TO\_DATE('2025-06-30', 'YYYY-MM-DD'));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, DueDate)

VALUES (3, 3, 15000.00, TO\_DATE('2025-07-10', 'YYYY-MM-DD'));

COMMIT;

-- Scenario 1: Apply 1% Discount to Customers Aged > 60

BEGIN

FOR rec IN (SELECT CustomerID, LoanInterestRate FROM Customers WHERE Age > 60) LOOP

UPDATE Customers

SET LoanInterestRate = LoanInterestRate \* 0.99

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Updated interest rate for CustomerID ' || rec.CustomerID);

END LOOP;

END;

/

-- Setcenario 2: Set IsVIP = 'TRUE' for Balance > 10000

BEGIN

FOR rec IN (SELECT CustomerID FROM Customers WHERE Balance > 10000) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('CustomerID ' || rec.CustomerID || ' promoted to VIP');

END LOOP;

END;

/

-- Scenario 3: Loan Reminders for Loans Due in Next 30 Days

BEGIN

FOR loan\_rec IN (

SELECT l.LoanID, c.Name, l.DueDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

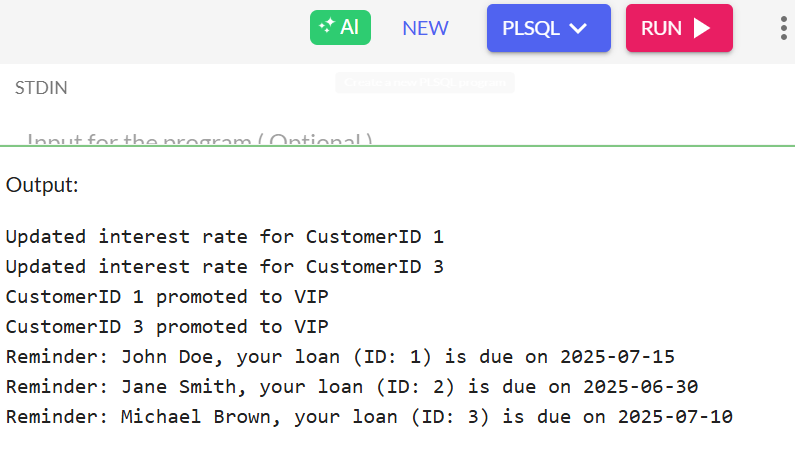
DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || loan\_rec.Name || ', your loan (ID: ' || loan\_rec.LoanID || ') is due on ' || TO\_CHAR(loan\_rec.DueDate, 'YYYY-MM-DD'));

END LOOP;

END;

/

**OUTPUT:**

  
**Exercise 3: Stored Procedures**

**CODE:**

-- DROP OLD TABLES IF NEEDED

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- TABLES

-- 1. Accounts Table

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

CustomerName VARCHAR2(100),

AccountType VARCHAR2(20), -- e.g., 'Savings', 'Checking'

Balance DECIMAL(10,2)

);

-- 2. Employees Table

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(50),

Salary DECIMAL(10,2)

);

-- SAMPLE DATA Accounts

INSERT INTO Accounts VALUES (1, 'Alice', 'Savings', 10000);

INSERT INTO Accounts VALUES (2, 'Bob', 'Checking', 15000);

INSERT INTO Accounts VALUES (3, 'Charlie', 'Savings', 20000);

INSERT INTO Accounts VALUES (4, 'David', 'Savings', 5000);

-- Employees

INSERT INTO Employees VALUES (1, 'John', 'IT', 50000);

INSERT INTO Employees VALUES (2, 'Sara', 'HR', 40000);

INSERT INTO Employees VALUES (3, 'Mike', 'IT', 60000);

COMMIT;

-- SCENARIO 1: Monthly Interest for Savings Accounts

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR acc IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings') LOOP

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest applied for AccountID ' || acc.AccountID);

END LOOP;

END;

/

-- EXECUTE Procedure

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Applying Monthly Interest ---');

ProcessMonthlyInterest;

END;

/

-- SCENARIO 2: Update Employee Bonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_name IN VARCHAR2,

bonus\_pct IN NUMBER

) AS

BEGIN

FOR emp IN (SELECT EmployeeID, Salary FROM Employees WHERE Department = dept\_name) LOOP

UPDATE Employees

SET Salary = Salary + (Salary \* bonus\_pct / 100)

WHERE EmployeeID = emp.EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to EmployeeID ' || emp.EmployeeID);

END LOOP;

END;

/

-- EXECUTE Procedure with 10% bonus to IT department

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Giving 10% Bonus to IT Department ---');

UpdateEmployeeBonus('IT', 10);

END;

/

-- SCENARIO 3: Fund Transfer Between Accounts

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_account IN INT,

to\_account IN INT,

amount IN DECIMAL

) AS

v\_balance DECIMAL(10,2);

BEGIN

-- Get balance of source account

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = from\_account FOR UPDATE;

IF v\_balance < amount THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient balance.');

ELSE

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = from\_account;

-- Add to target

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = to\_account;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amount || ' from AccountID ' || from\_account || ' to AccountID ' || to\_account);

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: One or both accounts not found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);

END;

/

-- EXECUTE Procedure to Transfer Funds

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Transferring Funds ---');

TransferFunds(1, 2, 2000); -- Successful

TransferFunds(4, 3, 6000); -- Fails: Insufficient funds

END;

/

-- FINAL OUTPUTS: VIEW TABLES

-- View Accounts

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Final Accounts ---');

FOR rec IN (SELECT \* FROM Accounts) LOOP

DBMS\_OUTPUT.PUT\_LINE('ID: ' || rec.AccountID || ', Name: ' || rec.CustomerName || ', Type: ' || rec.AccountType || ', Balance: ' || rec.Balance);

END LOOP;

END;

/

-- View Employees

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Final Employees ---');

FOR rec IN (SELECT \* FROM Employees) LOOP

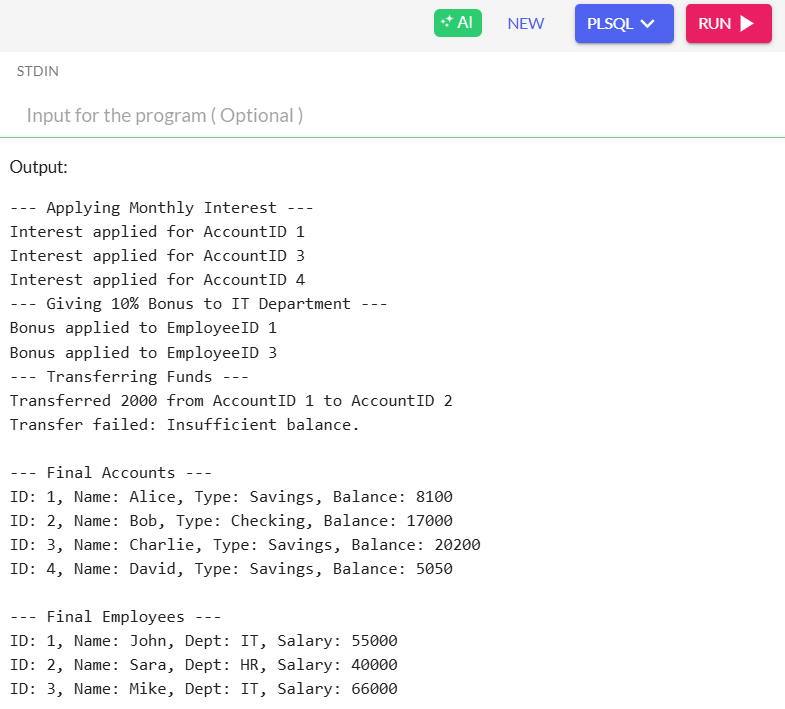
DBMS\_OUTPUT.PUT\_LINE('ID: ' || rec.EmployeeID || ', Name: ' || rec.Name || ', Dept: ' || rec.Department || ', Salary: ' || rec.Salary);

END LOOP;

END;

/

**OUTPUT:**

****

**Exercise 1: Setting Up Junit**

**CODE:**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

import static org.junit.Assert.\*;

import org.junit.Test;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

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

}

@Test

public void testSubtract() {

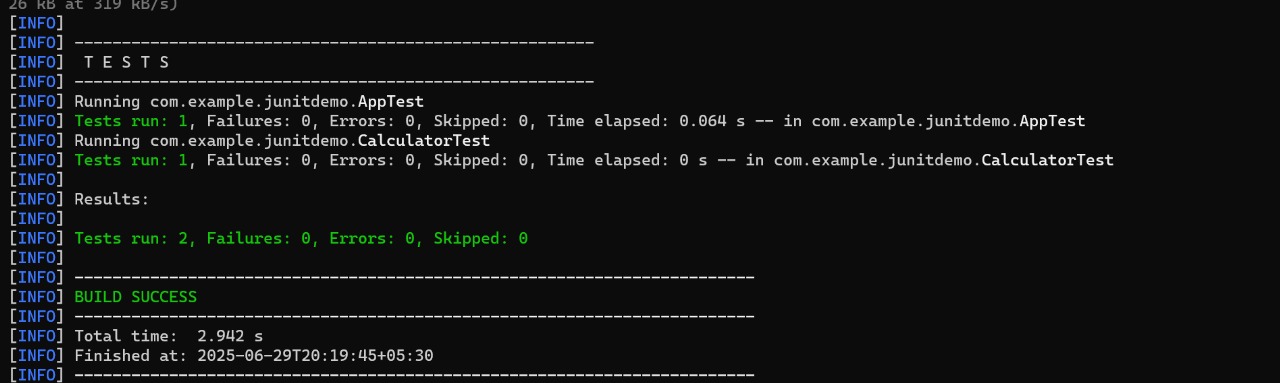
Calculator calc = new Calculator();

assertEquals(1, calc.subtract(3, 2));

}

}

**OUTPUT:**



**Exercise 3: Assertions in Junit**

**CODE:**

package com.example;

import org.junit.jupiter.api.Test;

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

public class AssertionsTest {

@Test

public void testAssertions() {

// Assert equals

assertEquals(5, 2 + 3, "Expected 2 + 3 to equal 5");

// Assert true

assertTrue(5 > 3, "Expected 5 > 3");

// Assert false

assertFalse(5 < 3, "Expected 5 < 3 to be false");

// Assert null

String str = null;

assertNull(str, "Expected string to be null");

// Assert not null

Object obj = new Object();

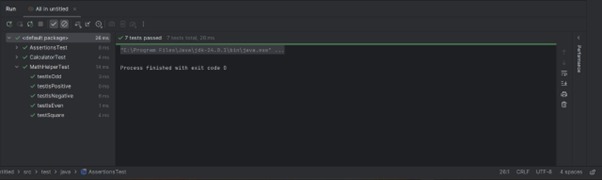
assertNotNull(obj, "Expected object to not be null");

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

}

}

**OUTPUT:**



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

**CODE:**

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 void clear() {

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

}

}

CalculatorTest.java

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

// Setup method runs before each test

@Before

public void setUp() {

System.out.println("Setting up Calculator...");

calculator = new Calculator(); // Arrange

}

// Teardown method runs after each test

@After

public void tearDown() {

calculator.clear(); // Teardown action

System.out.println("Tear down completed.\n");

}

@Test

public void testAddition() {

// Act

int result = calculator.add(10, 20);

// Assert

assertEquals("Addition failed", 30, result);

}

@Test

public void testSubtraction() {

// Act

int result = calculator.subtract(50, 15);

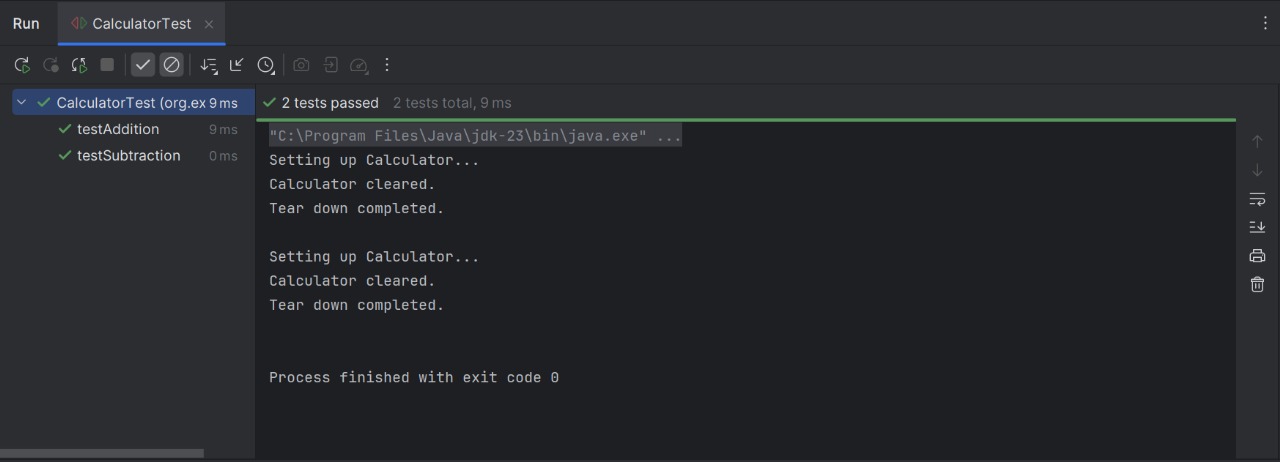
// Assert

assertEquals("Subtraction failed", 35, result);

}

}

**OUTPUT:**



**Exercise 1: Mocking and Stubbing**

**CODE:**

package com.example;

public interface ExternalApi {

String getData();

}

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

package com.example;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testExternalApi() {

// Step 1: Mock the external API

ExternalApi mockApi = mock(ExternalApi.class);

// Step 2: Stub the getData() method

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

// Step 3: Inject mock into service and call method

MyService service = new MyService(mockApi);

String result = service.fetchData();

// Optional: Print to console

System.out.println("Received: " + result);

// Assert that the stubbed value is returned

assertEquals("Mock Data", result);

}

}

**Exercise 2: Verifying Interactions**

**CODE:**

package com.example;

public interface ExternalApi {

String getData();

}

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData(); // External API call

}

}

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Step 1: Create mock

ExternalApi mockApi = mock(ExternalApi.class);

// Step 2: Stub method if needed

when(mockApi.getData()).thenReturn("Mocked Response");

// Step 3: Call the method

MyService service = new MyService(mockApi);

service.fetchData();

// Step 4: Verify interaction

verify(mockApi).getData(); // Checks getData() was called

System.out.println(" Verified: mockApi.getData() was called.");

}

}

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

**CODE:**

package com.example;

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");

logger.info("This is an info message"); // Optional

}

}

<configuration>

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%date [%level] [%logger] - %msg%n</pattern>

</encoder>

</appender>

<root level="debug">

<appender-ref ref="STDOUT"/>

</root>

</configuration>

**OUTPUT:**

