**PL SQL programming**

**Exercise 1: Control Structures**

**Scenario 1:**

**Code:**

CREATE TABLE CUSTOMERS (

CustomerID NUMBER,

Age NUMBER,

LoanInterestRate NUMBER

);

INSERT INTO CUSTOMERS VALUES (101, 65, 8.5);

INSERT INTO CUSTOMERS VALUES (102, 45, 9.0);

INSERT INTO CUSTOMERS VALUES (103, 70, 10.0);

INSERT INTO CUSTOMERS VALUES (104, 50, 7.5);

SET SERVEROUTPUT ON;

BEGIN

FOR rec IN (SELECT CustomerID, Age, LoanInterestRate FROM CUSTOMERS) LOOP

IF rec.Age > 60 THEN

UPDATE CUSTOMERS

SET LoanInterestRate = LoanInterestRate - 1

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Interest rate updated for CustomerID: ' || rec.CustomerID);

END IF;

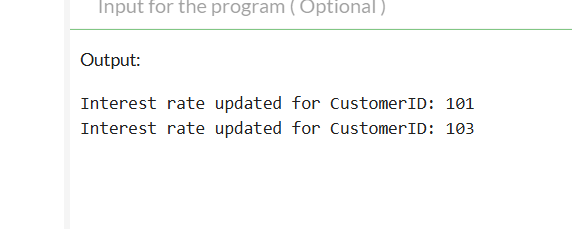
END LOOP;

COMMIT;

END;

/

**Output:**



**Scenario 2:**

**Code:**

CREATE TABLE CUSTOMERS (

CustomerID NUMBER,

Balance NUMBER,

IsVIP VARCHAR2(5)

);

INSERT INTO CUSTOMERS VALUES (101, 9500, 'FALSE');

INSERT INTO CUSTOMERS VALUES (102, 12000, 'FALSE');

INSERT INTO CUSTOMERS VALUES (103, 30000, 'FALSE');

INSERT INTO CUSTOMERS VALUES (104, 8000, 'FALSE');

SET SERVEROUTPUT ON;

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM CUSTOMERS) LOOP

IF rec.Balance > 10000 THEN

UPDATE CUSTOMERS

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('CustomerID ' || rec.CustomerID || ' is now a VIP.');

END IF;

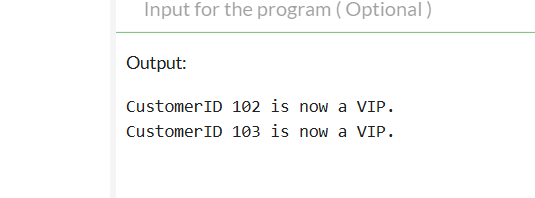
END LOOP;

COMMIT;

END;

/

**Output:**

****

**Scenario 3:**

**Code:**

CREATE TABLE LOANS (

LoanID NUMBER,

CustomerID NUMBER,

DueDate DATE

);

INSERT INTO LOANS VALUES (101, 1001, SYSDATE + 5);

INSERT INTO LOANS VALUES (102, 1002, SYSDATE + 15);

INSERT INTO LOANS VALUES (103, 1003, SYSDATE + 35);

INSERT INTO LOANS VALUES (104, 1004, SYSDATE + 25);

COMMIT;

BEGIN

FOR rec IN (

SELECT LoanID, CustomerID, DueDate

FROM LOANS

WHERE DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: LoanID ' || rec.LoanID ||

' for CustomerID ' || rec.CustomerID ||

' is due on ' || TO\_CHAR(rec.DueDate, 'DD-MON-YYYY')

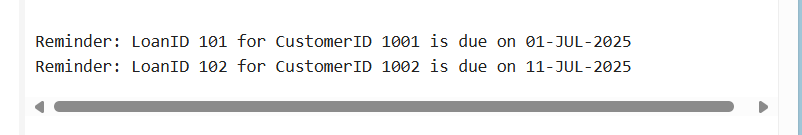
);

END LOOP;

END;

/

**Output:**

****

**Exercise 3: Stored Procedures**

**Code:**

CREATE TABLE ACCOUNTS (

AccountID NUMBER PRIMARY KEY,

AccountType VARCHAR2(20),

Balance NUMBER

);

INSERT INTO ACCOUNTS VALUES (101, 'SAVINGS', 10000);

INSERT INTO ACCOUNTS VALUES (102, 'SAVINGS', 20000);

INSERT INTO ACCOUNTS VALUES (103, 'CURRENT', 15000);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE ACCOUNTS

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'SAVINGS';

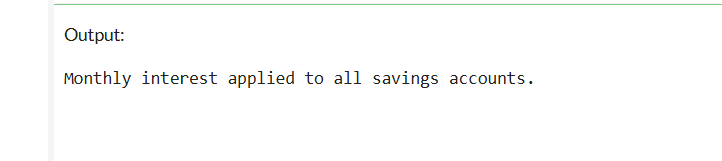
DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to all savings accounts.');

END;

/

EXEC ProcessMonthlyInterest;

**Output:**

****

**Scenario 2:**

**Code:**

CREATE TABLE EMPLOYEES (

EmployeeID NUMBER PRIMARY KEY,

DepartmentID NUMBER,

Salary NUMBER

);

INSERT INTO EMPLOYEES VALUES (201, 10, 50000);

INSERT INTO EMPLOYEES VALUES (202, 10, 60000);

INSERT INTO EMPLOYEES VALUES (203, 20, 55000);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_id IN NUMBER,

bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE EMPLOYEES

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

WHERE DepartmentID = dept\_id;

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || bonus\_percent || '% applied to department ' || dept\_id);

END;

/

BEGIN

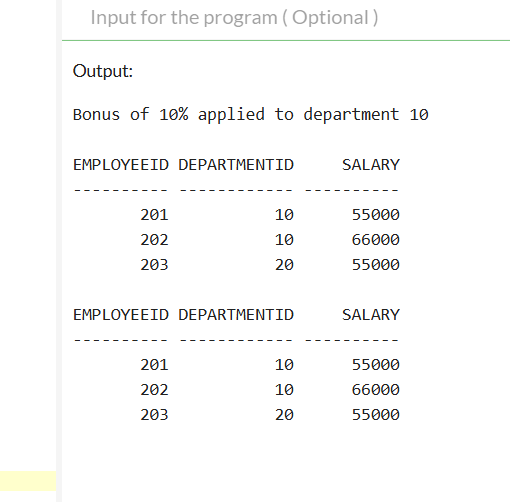
UpdateEmployeeBonus(10, 10); -- Apply 10% bonus to department 10

END;

/

SELECT \* FROM EMPLOYEES;

**Output:**

****

**Scenario 3:**

**Code:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE ACCOUNTS';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE ACCOUNTS (

AccountID NUMBER PRIMARY KEY,

Balance NUMBER

);

INSERT INTO ACCOUNTS VALUES (101, 10000);

INSERT INTO ACCOUNTS VALUES (102, 5000);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_acc IN NUMBER,

to\_acc IN NUMBER,

amt IN NUMBER

) IS

bal NUMBER;

BEGIN

SELECT Balance INTO bal FROM ACCOUNTS WHERE AccountID = from\_acc;

IF bal >= amt THEN

UPDATE ACCOUNTS SET Balance = Balance - amt WHERE AccountID = from\_acc;

UPDATE ACCOUNTS SET Balance = Balance + amt WHERE AccountID = to\_acc;

DBMS\_OUTPUT.PUT\_LINE( amt || ' transferred from ' || from\_acc || ' to ' || to\_acc);

ELSE

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

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Invalid account ID.');

END;

/

BEGIN

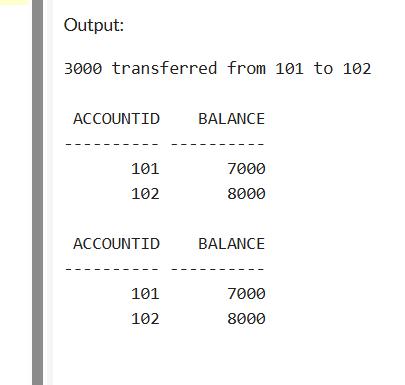
TransferFunds(101, 102, 3000);

END;

/

SELECT \* FROM ACCOUNTS;

**Output:**



JUnit, Mockito and SL4J

Exercise 1: Setting Up JUnit

**Calculator.java**

package com.example.JUnitDemo2;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java**

package com.example.JUnitDemo2;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

public void testAdd() {

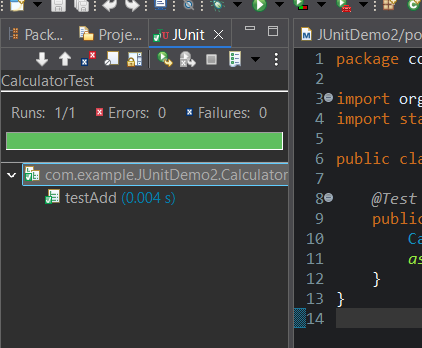
Calculator calc = new Calculator();

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

}

}

**Output:**

****

**Exercise 3: Assertions in Junit**

**AssertionsTest.java:**

**CODE:**

package com.example.JUnitDemo2;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

*@Test*

public void testAssertions() {

*assertEquals*(5, 2 + 3);

*assertNotEquals*(4, 2 + 3);

*assertTrue*("5 is greater than 3", 5 > 3);

*assertFalse*("2 is not greater than 3", 2 > 3);

String str = null;

*assertNull*(str);

String message = "Hello";

*assertNotNull*(message);

String a = "test";

String b = a;

*assertSame*(a, b);

String x = new String("test");

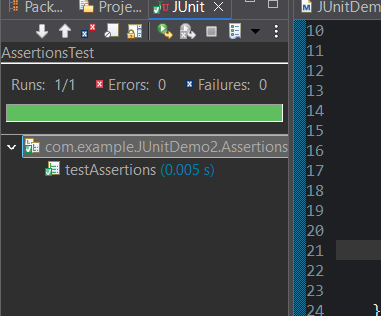
String y = new String("test");

*assertNotSame*(x, y);

}

}

**Output:**

****

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

**CalculatorAaaTest**

**CODE:**

package com.example.JUnitDemo2;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorAaaTest {

private Calculator calculator;

@Before

public void setUp() {

calculator = new Calculator();

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

}

@After

public void tearDown() {

calculator = null;

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

}

@Test

public void testAdd() {

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

assertEquals(15, result);

}

@Test

public void testAddWithZero() {

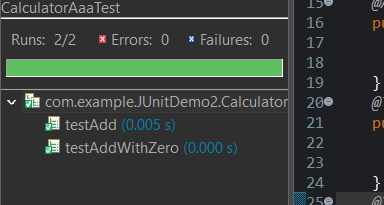
int result = calculator.add(0, 7);

assertEquals(7, result);

}

}

**OUTPUT:**

****

**Exercise 1: Mocking and Stubbing**

**ExternalApi:**

**CODE:**

package com.example;

public interface ExternalApi {

String getData();

}

**MyService:**

**CODE:**

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest:**

**CODE:**

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

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);

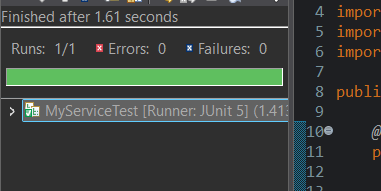
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**OUTPUT:**

****

**Exercise 2: Verifying Interactions with Mockito**

**MyServiceTest:**

**CODE:**

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

**import org.junit.jupiter.api.Test;**

**import org.mockito.Mockito;**

**public class MyServiceTest {**

**@Test**

**public void testVerifyInteraction() {**

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

**MyService service = new MyService(mockApi);**

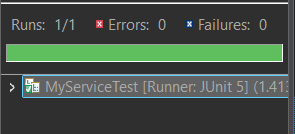
**service.fetchData();**

**verify(mockApi).getData();**

**}**

**}**

**OUTPUT:**



**SLF4JLoggingDemo:**

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

**LoggingExample:**

package com.example.SLF4JLoggingDemo;

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");

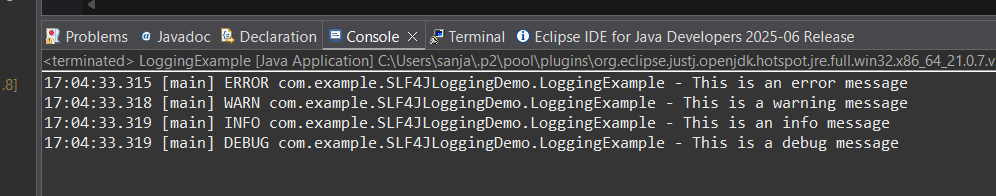
***logger***.info("This is an info message");

***logger***.debug("This is a debug message");

}

}

**OUTPUT:**

****