**DIGITAL NURTURE 4.0 JavaFSE**

**WEEK 2**

**PL/SQL programming**

**Exercise 1:**

**Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

**Code :**

BEGIN

FOR loan\_rec IN (

SELECT l.LoanID, l.InterestRate, c.Age

FROM Loans l

JOIN Customers c ON l.CustID = c.CustID

) LOOP

IF loan\_rec.Age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = loan\_rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Discount applied to Loan ID: ' || loan\_rec.LoanID);

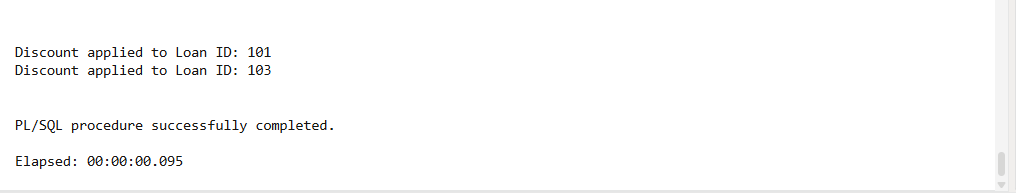
END IF;

END LOOP;

END;

/

**Output :**



**Scenario 2: A customer can be promoted to VIP status based on their balance.**

**Code:**

BEGIN

FOR cust\_rec IN (

SELECT CustID, Balance FROM Customers

) LOOP

IF cust\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustID = cust\_rec.CustID;

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || cust\_rec.CustID || ' promoted to VIP.');

END IF;

END LOOP;

END;

/

**Output :**



**Scenario 3: The bank wants to send reminders to customers whose loans are due within the next 30 days.**

**Code:**

BEGIN

FOR loan\_rec IN (

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

FROM Loans l

JOIN Customers c ON l.CustID = c.CustID

WHERE l.DueDate <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || loan\_rec.Name ||

', your loan (ID: ' || loan\_rec.LoanID ||

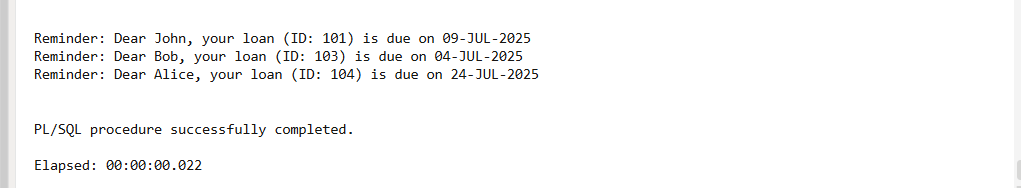
') is due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**Output :**



**Exercise 3: Stored Procedures**

**Scenario 1: The bank needs to process monthly interest for all savings accounts.**

**Code :**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

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 to Account ID: ' || acc.AccountID);

END LOOP;

END;

/

**Output :**



**Scenario 2: The bank wants to implement a bonus scheme for employees based on their performance.**

**Code :**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept IN VARCHAR2,

bonus\_percent IN NUMBER

) IS

BEGIN

FOR emp IN (

SELECT EmpID, Name FROM Employees WHERE Department = dept

) LOOP

UPDATE Employees

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

WHERE EmpID = emp.EmpID;

DBMS\_OUTPUT.PUT\_LINE('Bonus given to ' || emp.Name || ' in ' || dept);

END LOOP;

END;

/

**Output :**



**Scenario 3: Customers should be able to transfer funds between their accounts.**

**Code:**

CREATE SEQUENCE SEQ\_TransID START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE PROCEDURE TransferFunds(

from\_acc IN NUMBER,

to\_acc IN NUMBER,

amount IN NUMBER

) IS

insufficient\_balance EXCEPTION;

current\_balance NUMBER;

BEGIN

SELECT Balance INTO current\_balance

FROM Accounts

WHERE AccountID = from\_acc;

IF current\_balance < amount THEN

RAISE insufficient\_balance;

END IF;

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = from\_acc;

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = to\_acc;

INSERT INTO Transactions (TransID, FromAccountID, ToAccountID, Amount, TransDate)

VALUES (SEQ\_TransID.NEXTVAL, from\_acc, to\_acc, amount, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amount || ' from Account ' || from\_acc || ' to ' || to\_acc);

EXCEPTION

WHEN insufficient\_balance THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient balance in Account ID ' || from\_acc);

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);

END;

/

BEGIN

TransferFunds(101, 102, 2000);

END;

/

**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;

}

}

**CalculatorTest.java**

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

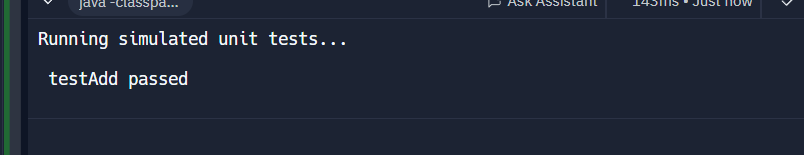
int result = calc.add(2, 3);

assertEquals(5, result);

}

}

**Output :**



**Exercise 3: Assertions in JUnit**

**Code :**

**AssertionsTest.java**

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

assertFalse(5 < 3);

String str = null;

assertNull(str);

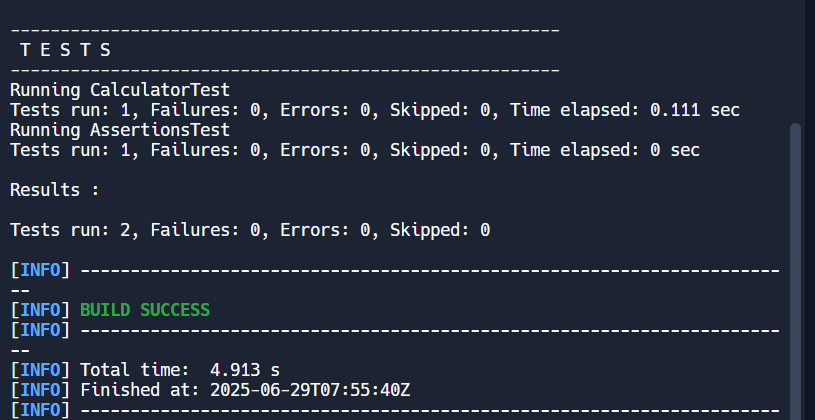
Object obj = new Object();

assertNotNull(obj);

}

}

**Output :**



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

**Teardown Methods in Junit**

**Code :**

**CalculatorAdvancedTest.java**

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorAdvancedTest {

private Calculator calculator;

@Before

public void setUp() {

calculator = new Calculator();

System.out.println(" Setup: Calculator instance created");

}

@After

public void tearDown() {

calculator = null;

System.out.println(" Teardown: Calculator instance cleared");

}

@Test

public void testAddPositiveNumbers() {

int a = 8;

int b = 12;

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

assertEquals(20, result);

}

@Test

public void testAddWithZero() {

int a = 0;

int b = 15;

int result = calculator.add(a, b)

assertEquals(15, result);

}

@Test

public void testAddNegative() {

int a = -7;

int b = -3;

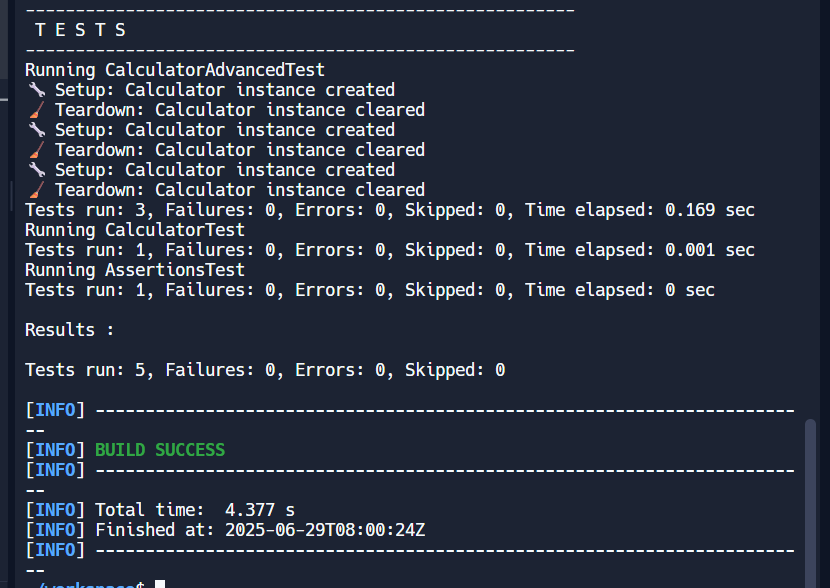
int result = calculator.add(a, b);

assertEquals(-10, result);

}

}

**Output :**



**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

**Code:**

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

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

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);

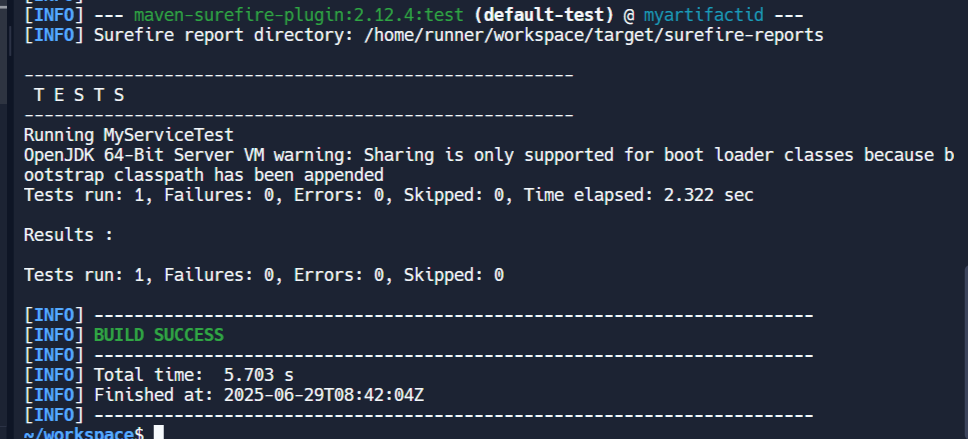
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**Output :**



**Exercise 2: Verifying Interactions**

**Code:**

**MyServiceTest2.java**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest2 {

@Test

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApi);

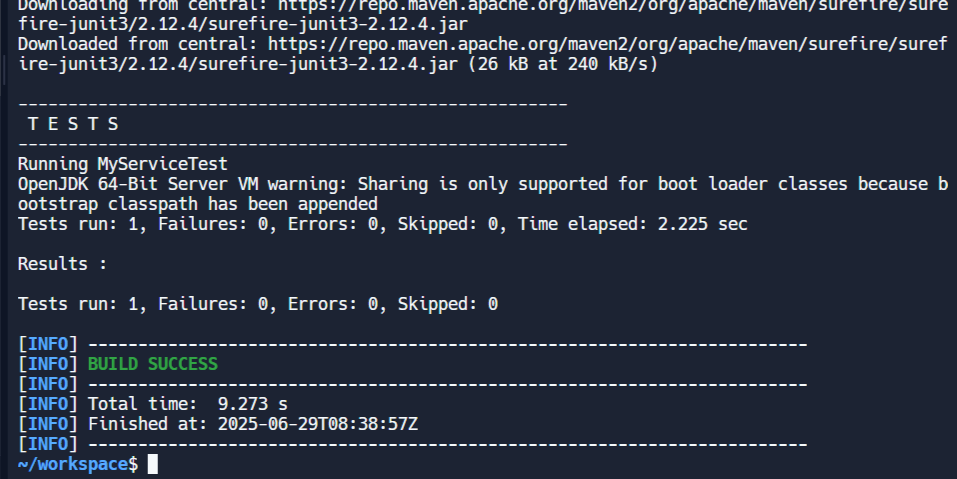
service.fetchData();

verify(mockApi).getData();

}

}

**Output :**



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

**Code :**

**LoggingExample.java**

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");

}

}

**Output :**

