**Exercise 1: Control Structures**

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

**Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

BEGIN

FOR cust\_rec IN (

SELECT CustomerID

FROM Customers

WHERE FLOOR(MONTHS\_BETWEEN(SYSDATE, DOB) / 12) > 60

)

LOOP

UPDATE Loans

SET InterestRate = InterestRate \* 0.99

WHERE CustomerID = cust\_rec.CustomerID;

END LOOP;

COMMIT;

END;

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

**Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

BEGIN

FOR vip\_candidate IN (

SELECT CustomerID

FROM Customers

WHERE Balance > 10000

)

LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = vip\_candidate.CustomerID;

END LOOP;

COMMIT;

END;

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

**Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

BEGIN

FOR rec IN (

SELECT L.LoanID, L.CustomerID, C.Name, L.EndDate

FROM Loans L

JOIN Customers C ON L.CustomerID = C.CustomerID

WHERE L.EndDate BETWEEN SYSDATE AND SYSDATE + 30

)

LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.LoanID ||

' for customer ' || rec.Name ||

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

END LOOP;

END;

**Exercise 3: Stored Procedures**

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

**Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (

SELECT AccountID

FROM Accounts

WHERE AccountType = 'Savings'

)

LOOP

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountID = acc.AccountID;

END LOOP;

COMMIT;

END;

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

**Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_percent / 100)

WHERE Department = p\_department;

COMMIT;

END;

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

* 1. **Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_source\_account IN NUMBER,

p\_target\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_source\_account;

IF v\_balance >= p\_amount THEN

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_source\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_target\_account;

COMMIT;

ELSE

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.');

END IF;

END;

**JUnit Testing Exercises**

**Exercise 1: Setting Up Junit**

**Calculator.java**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

**CalculatorTest.java**

package com.example;

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(3, 7);

assertEquals(10, result);

}

@Test

public void testSubtract() {

Calculator calc = new Calculator();

int result = calc.subtract(10, 4);

assertEquals(6, result);

}

}

**Exercise 3: Assertions in JUnit**

**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);

assertNull(null);

assertNotNull(new Object());

}

}

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

**CalculatorAAATest.java**

package com.example;

import org.junit.Before;

import org.junit.After;

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 testAddition() {

int a = 10;

int b = 5;

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

assertEquals(15, result);

}

public void testSubtraction() {

int a = 10;

int b = 4;

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

assertEquals(6, result);

}

}

**Mockito exercises**

**Exercise 1: Mocking and Stubbing**

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\_Mocking {

public interface ExternalApi {

String getData();

}

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

@Test

public void testExternalApiMockingAndStubbing() {

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);

}

}

**Exercise 2: Verifying Interactions**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest\_Verification {

public interface ExternalApi {

String getData();

}

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

@Test

public void testVerifyGetDataCalled() {

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

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

**SL4J Logging Exercise**

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

**pom.xml**

<dependencies>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

**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");

}

}

**Logback.xml**

<configuration>

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

<encoder>

<pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<root level="debug">

<appender-ref ref="STDOUT" />

</root>

</configuration>

**Output:**

HH:mm:ss.SSS [main] ERROR LoggingExample - This is an error message

HH:mm:ss.SSS [main] WARN LoggingExample - This is a warning message