# PL/SQL Exercises

**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.

ANS:

BEGIN

    FOR cust IN (SELECT CustomerID FROM Customers WHERE Age > 60) LOOP

        UPDATE Customers

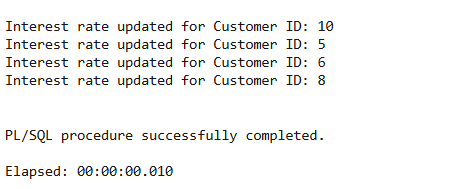
        SET InterestRate = InterestRate - 1

        WHERE CustomerID = cust.CustomerID;

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

    END LOOP;

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.

ANS:

BEGIN

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

UPDATE Customers

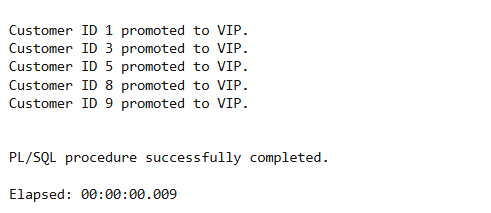
SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

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

END LOOP;

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.

ANS:  
  
BEGIN

    FOR loan\_rec IN (

        SELECT LoanID, CustomerID, DueDate

        FROM Loans

        WHERE DueDate <= SYSDATE + 30

    ) LOOP

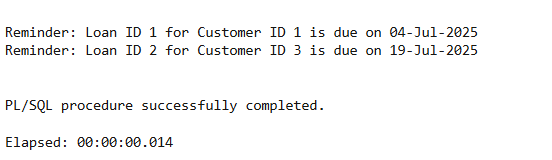
        DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan\_rec.LoanID ||

                             ' for Customer ID ' || loan\_rec.CustomerID ||

                             ' is due on ' || TO\_CHAR(loan\_rec.DueDate, '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.

ANS:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR cust IN (SELECT CustomerID, Balance FROM Customers) LOOP

UPDATE Customers

SET Balance = Balance + (Balance \* 0.01)

WHERE CustomerID = cust.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Applied 1% interest for Customer ID: ' || cust.CustomerID);

END LOOP;

END;

Running:

BEGIN

ProcessMonthlyInterest;

END;

Applied 1% interest for Customer ID: 1

Applied 1% interest for Customer ID: 10

Applied 1% interest for Customer ID: 2

Applied 1% interest for Customer ID: 3

Applied 1% interest for Customer ID: 5

Applied 1% interest for Customer ID: 6

Applied 1% interest for Customer ID: 7

Applied 1% interest for Customer ID: 4

Applied 1% interest for Customer ID: 8

Applied 1% interest for Customer ID: 9

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.006

**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.

1. CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
2. dept\_name IN VARCHAR2,
3. bonus\_percent IN NUMBER
4. ) IS
5. BEGIN
6. FOR emp IN (SELECT EmpID FROM Employees WHERE Department = dept\_name) LOOP
7. UPDATE Employees
8. SET Salary = Salary + (Salary \* bonus\_percent / 100)
9. WHERE EmpID = emp.EmpID;
10. DBMS\_OUTPUT.PUT\_LINE('Bonus applied to EmpID: ' || emp.EmpID);
11. END LOOP;
12. END;

Running:

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.012

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

* + **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.

ANS:

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_customer IN NUMBER,

to\_customer IN NUMBER,

amount IN NUMBER

) IS

insufficient\_balance EXCEPTION;

BEGIN

-- Check if sender has enough

DECLARE

sender\_balance NUMBER;

BEGIN

SELECT Balance INTO sender\_balance FROM Customers WHERE CustomerID = from\_customer;

IF sender\_balance < amount THEN

RAISE insufficient\_balance;

END IF;

END;

-- Deduct and add

UPDATE Customers SET Balance = Balance - amount WHERE CustomerID = from\_customer;

UPDATE Customers SET Balance = Balance + amount WHERE CustomerID = to\_customer;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amount || ' from Customer ' || from\_customer || ' to ' || to\_customer);

EXCEPTION

WHEN insufficient\_balance THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: insufficient funds in account ' || from\_customer);

WHEN OTHERS THEN

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

END;

Running:

Transferred 500 from Customer 1 to 2

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.014

# TDD using JUnit5 and Mockito

Exercise 1: Setting Up JUnit

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 {

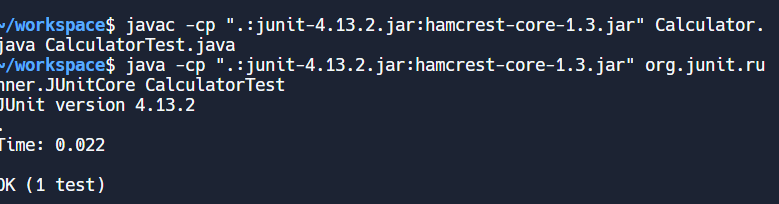
@Test

public void testAdd() {

Calculator calc = new Calculator();

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

}

}

Exercise 3: Assertions in JUnit

AssertionsTest.java

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testSimpleAssertions() {

assertEquals("Sum should be 5", 5, 2 + 3);

assertNotEquals("Should not be equal", 4, 2 + 3);

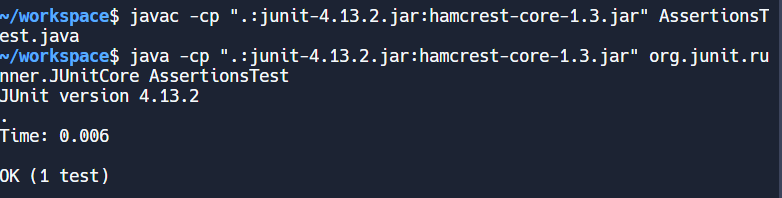
assertTrue("5 is greater than 3", 5 > 3);

assertFalse("3 is not greater than 5", 3 > 5);

assertNull("This should be null", null);

assertNotNull("This should not be null", "Hello");

}

}

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

BankAccount.java

public class BankAccount {

private int balance;

public BankAccount(int balance) {

this.balance = balance;

}

public void deposit(int amount) {

balance += amount;

}

public void withdraw(int amount) {

balance -= amount;

}

public int getBalance() {

return balance;

}

}

BankAccountTest.java

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class BankAccountTest {

private BankAccount account;

@Before

public void setUp() {

// Arrange

account = new BankAccount(1000);

}

@After

public void tearDown() {

account = null;

}

@Test

public void testWithdraw() {

// Act

account.withdraw(200);

// Assert

assertEquals("Balance after withdrawal should be 800", 800, account.getBalance());

}

@Test

public void testDeposit() {

// Act

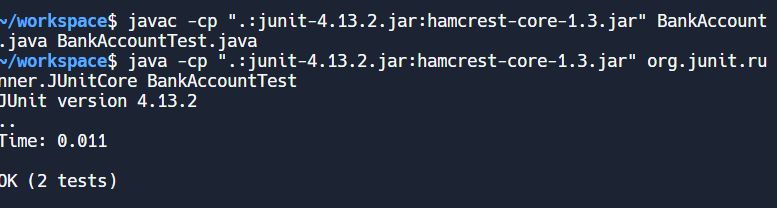
account.deposit(500);

// Assert

assertEquals("Balance after deposit should be 1500", 1500, account.getBalance());

}

}



# Mockito exercises

Exercise 1: Mocking and Stubbing, Exercise 2: Verifying Interactions

MyServiceTest.java

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest {

@Test

public void testMockingAndStubbing() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mocked Response", result);

}

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

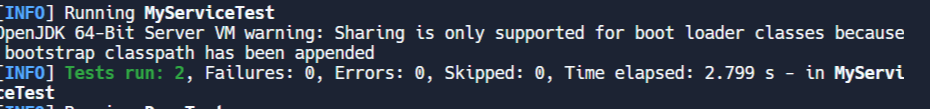
MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData(); // verifies interaction

}

}



# SLF4J logging framework

Exercise 1: Logging Error Messages and Warning Levels

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");

logger.info("ℹ️ This is an info message (optional)");

}

}