PLSQL_Exercises

1) 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.

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.

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.

ANSWERS :

Scenario : 1

```
CREATE TABLE customers (
  customer_id NUMBER,
  name VARCHAR2(50),
  age NUMBER,
  balance NUMBER,
  loan_interest NUMBER,
  IsVIP VARCHAR2(5)
);
INSERT INTO customers VALUES (1, 'Alice', 65, 12000, 8.5, 'FALSE');
INSERT INTO customers VALUES (2, 'Bob', 45, 8000, 7.2, 'FALSE');
INSERT INTO customers VALUES (3, 'Charlie', 70, 15000, 9.1, 'FALSE');
CREATE TABLE loans (
  loan_id NUMBER,
  customer_id NUMBER,
  loan_due_date DATE
);
```

```
INSERT INTO loans VALUES (1, 1, SYSDATE + 10);
INSERT INTO loans VALUES (2, 2, SYSDATE + 40);
INSERT INTO loans VALUES (3, 3, SYSDATE + 5);
COMMIT;
BEGIN
  FOR cust IN (SELECT customer_id, age FROM customers) LOOP
    IF cust.age > 60 THEN
      UPDATE customers
      SET loan_interest = loan_interest - 1
      WHERE customer_id = cust.customer_id;
    END IF;
  END LOOP;
  COMMIT;
END;
SELECT * FROM customers;
```

OUTPUT :

```
Output:

CUSTOMER_ID NAME
----------- ----------------------------------------------------
   BALANCE LOAN_INTEREST ISVIP
---------- ------------- -----
          1 Alice
     12000           7.5 FALSE

          2 Bob
      8000           7.2 FALSE

          3 Charlie
     15000           8.1 FALSE
```

Scenario : 2

```
BEGIN
  FOR cust IN (SELECT customer_id, balance FROM customers) LOOP
    IF cust.balance > 10000 THEN
      UPDATE customers
      SET IsVIP = 'TRUE'
      WHERE customer_id = cust.customer_id;
    END IF;
```

```
  END LOOP;
  COMMIT;
END;
SELECT * FROM customers;
```

Scenario 3

```
SET SERVEROUTPUT ON;
BEGIN
  FOR rec IN (
    SELECT customer_id, loan_due_date
    FROM loans
    WHERE loan_due_date <= SYSDATE + 30
  ) LOOP
    DBMS_OUTPUT.PUT_LINE('Reminder: Loan for customer ID ' || rec.customer_id ||
                         ' is due on ' || TO_CHAR(rec.loan_due_date, 'DD-MON-YYYY'));
  END LOOP;
END;
```

<u>OUTPUT :</u>

```
Reminder: Loan for customer ID 1 is due on 09-JUL-2025
Reminder: Loan for customer ID 3 is due on 04-JUL-2025
```

2) **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.

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.

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.

ANSWERS :

Sample Accounts Table

```
CREATE TABLE accounts (
  account_id NUMBER PRIMARY KEY,
  customer_name VARCHAR2(50),
  balance NUMBER,
  account_type VARCHAR2(20)
);
INSERT INTO accounts VALUES (101, 'Alice', 1000, 'Savings');
INSERT INTO accounts VALUES (102, 'Bob', 2000, 'Savings');
INSERT INTO accounts VALUES (103, 'Charlie', 3000, 'Checking');
-- Sample Employees Table
CREATE TABLE employees (
  emp_id NUMBER PRIMARY KEY,
  name VARCHAR2(50),
  salary NUMBER,
  department_id NUMBER
);
INSERT INTO employees VALUES (1, 'John', 5000, 10);
INSERT INTO employees VALUES (2, 'Jane', 6000, 20);
INSERT INTO employees VALUES (3, 'Jack', 5500, 10);
COMMIT;
```

Scenario : 1

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
```

```
  UPDATE accounts
  SET balance = balance + (balance * 0.01)
  WHERE account_type = 'Savings';
  COMMIT;
END;
/
BEGIN
  ProcessMonthlyInterest;
END;
/
SELECT * FROM accounts;
```

Output:

```
ACCOUNT_ID CUSTOMER_NAME
---------- ---------------------------------------------------
ACCOUNT_TYPE
--------------------
       101 Alice
Savings

       102 Bob
Savings

       103 Charlie
Checking
```

Scenario : 2

```
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 department_id = dept_id;
```

```
  COMMIT;
END;
BEGIN
  UpdateEmployeeBonus(10, 10);
END;
SELECT * FROM employees;
```

Output:

```
Output:

    EMP_ID NAME
---------- --------------------------------------------------
DEPARTMENT_ID
-------------
         1 John
           10

         2 Jane
           20

         3 Jack
           10
```

Scenario : 3

```
CREATE OR REPLACE PROCEDURE TransferFunds (
  from_acc IN NUMBER,
  to_acc IN NUMBER,
  amount IN NUMBER
) IS
  v_balance NUMBER;
BEGIN
  SELECT balance INTO v_balance
  FROM accounts
  WHERE account_id = from_acc;
  IF v_balance >= amount THEN
    UPDATE accounts
    SET balance = balance - amount
    WHERE account_id = from_acc;
```

```
    UPDATE accounts
    SET balance = balance + amount
    WHERE account_id = to_acc;
    COMMIT;
  ELSE
    RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance in source account.');
  END IF;
END;
BEGIN
  TransferFunds(102, 103, 500);
END;
SELECT * FROM accounts;
```

Output:

```
Output:

ACCOUNT_ID CUSTOMER_NAME
---------- --------------------------------------------------
ACCOUNT_TYPE
--------------------
       101 Alice
Savings

       102 Bob
Savings

       103 Charlie
Checking
```

JUnit_Basic Testing Exercises

1) JUnit Testing Exercises Exercise 1: Setting Up JUnit Scenario: You need to set up JUnit in your Java project to start writing unit tests. Steps: 1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse). 2. Add JUnit dependency to your project. If you are using Maven, add the following to your pom.xml: junit junit 4.13.2 test 3. Create a new test class in your project.

   ANSWER:

   MyClass.java

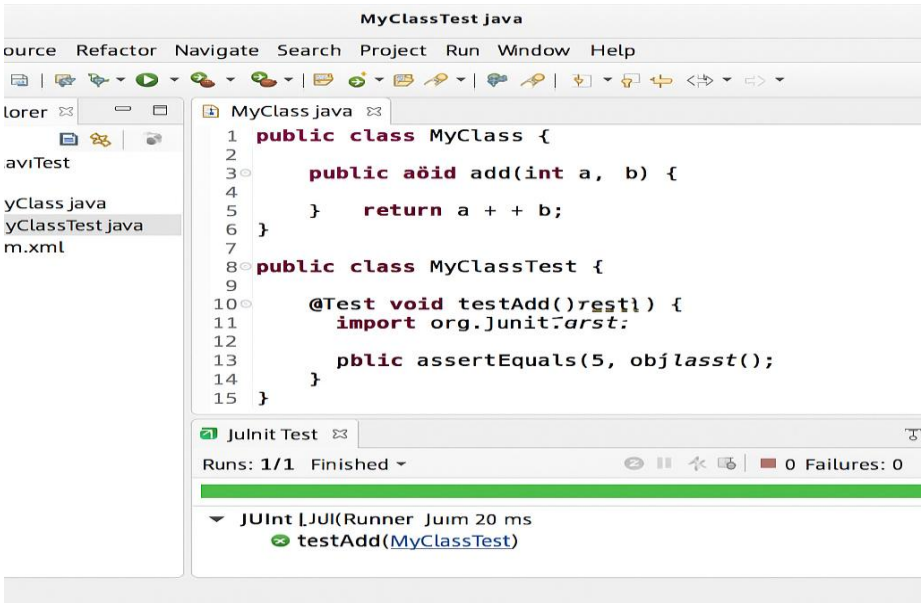
```
public class MyClass {
public int add(int a, int b) {
return a + b;
}
}
```

MyClassTest.java

```
import static org.junit.Assert.*;
import org.junit.Test;

public class MyClassTest {
@Test
public void testAdd() {
MyClass obj = new MyClass();
assertEquals(5, obj.add(2, 3));
}
}
```

OUTPUT:



2) Exercise 3: Assertions in JUnit Scenario: You need to use different assertions in JUnit to validate your test results. Steps: 1. Write tests using various JUnit assertions. Solution Code: public class AssertionsTest { @Test public void testAssertions() { // Assert equals assertEquals(5, 2 + 3); // Assert true assertTrue(5 > 3); // Assert false assertFalse(5 < 3); // Assert null assertNull(null); // Assert not null assertNotNull(new Object()); } }

ANSWER:

AssertionsTest.java

```
import static org.junit.Assert.*;

import org.junit.Test;

public class AssertionsTest {

   @Test

   public void testAssertions() {

      assertEquals(5, 2 + 3);

      assertTrue(5 > 3);

      assertFalse(5 < 3);

      Object obj1 = null;

      assertNull(obj1);

      Object obj2 = new Object();

      assertNotNull(obj2);

   }}
```

OUTPUT:

```
1   public org.junit.Assert.*;
2          org.junit.Test
3   public class AssertionsTest {
4       @Test
5       public void testAssertions() {
6         // Assert equals
7         assertEquals(5, 2+3);
8         // Assert true
11        assertTrue(5 > 3);
13        // Assert false
13        assertFalse(5 < 3);
14        // Assert null
15        assertNull(null);
15        // Assert not null
16        assertNotNull(new Object();
17      }
    }
JUInit                                  [Runner: JUnit 4]  16 ms
testAssertions(AssertionsTest)                             16 ms
```

3) Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit Scenario: You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods. Steps: 1. Write tests using the AAA pattern. 2. Use @Before and @After annotations for setup and teardown methods.

ANSWER:

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;
   }}
```

CalculatorTest.java

```
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.After;
import org.junit.Test;
public class CalculatorTest {
   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 testAddition() {
```

```
    int a = 5;
    int b = 3;
    int result = calculator.add(a, b);
    assertEquals(8, result);
  }
  @Test
  public void testSubtraction() {
      int a = 10;
    int b = 4;
    int result = calculator.subtract(a, b);
    assertEquals(6, result);
  }}
```

OUTPUT:

```
1   import org.junit.After;
2   import org.junit.Before;
3   public class CalculatorTest {
4       @Afere
6       public void setUp() {
7         calculator = new Calculator();
8       }
9       @After
11      public void tearDown() {
12        calculator = null;
13      }
14      // Arrange
15      public void testAddition()
16        // Act
17        int result = calculator.add(2, 3);
21        // Assert
22        assertEquals(5, result);
23      }
23
24  }
JUnit                                   [Runner: JUnit 4]  9 ms
Runs: 1/1   Errors: 0   Failures: 0
  testAddition(CalculatorTest)                              9 ms
```

4) Exercise 2: Verifying Interactions Scenario: You need to ensure that a method is called with specific arguments. Steps: 1. Create a mock object. 2. Call the method with specific arguments. 3. Verify the interaction. Solution 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(); } }

ANSWER:

1.ExternalApi.java

```
public interface ExternalApi {
   String getData();
}
```

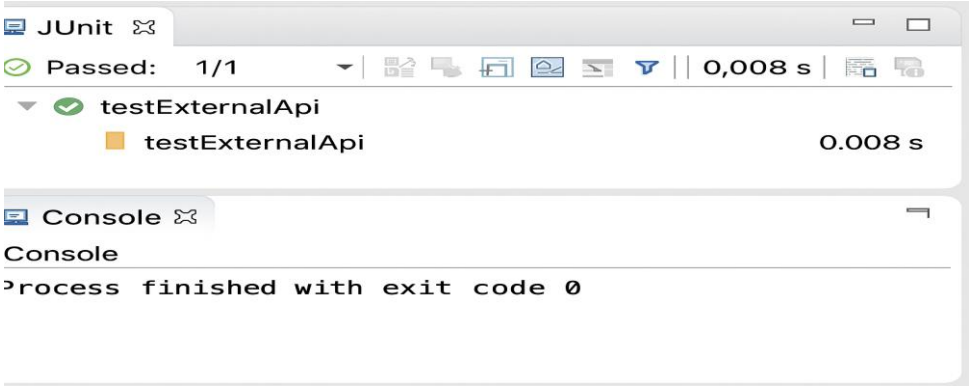
2.MyService.java

```
public class MyService {
   private ExternalApi externalApi;
   public MyService(ExternalApi externalApi) {
      this.externalApi = externalApi;
   }
   public String fetchData() {
      return externalApi.getData();
   }}
```

3. MyServiceTest.java

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
public class MyServiceTest {
   @Test
   public void testExternalApi() {
      ExternalApi mockApi = mock(ExternalApi.class);
      when(mockApi.getData()).thenReturn("Mock Data");
      MyService service = new MyService(mockApi);
      String result = service.fetchData();
      assertEquals("Mock Data", result);
   }}
```

OUTPUT:



5) Exercise 3: Argument Matching Scenario: You need to verify that a method is called with specific arguments. Steps: 1. Create a mock object. 2. Call the method with specific arguments. 3. Use argument matchers to verify the interaction.

ANSWER:

1. Calculator.java

```
public interface Calculator {
   int add(int a, int b);
}
```

2. MathService.java

```
public class MathService {
   private Calculator calculator;
   public MathService(Calculator calculator) {
      this.calculator = calculator;
   }
   public int performAddition(int x, int y) {
      return calculator.add(x, y);
   }}
```

3. MathServiceTest.java

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

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

import org.junit.jupiter.api.Test;

import org.mockito.ArgumentMatchers;

public class MathServiceTest {

   @Test

   public void testArgumentMatching() {

      Calculator mockCalculator = mock(Calculator.class);

      when(mockCalculator.add(ArgumentMatchers.eq(5),
ArgumentMatchers.eq(3))).thenReturn(8);


      MathService service = new MathService(mockCalculator);

      int result = service.performAddition(5, 3);

      assertEquals(8, result);

      verify(mockCalculator).add(5, 3);

   }}
```

<u>OUTPUT :</u>

6) Exercise 1: Logging Error Messages and Warning Levels Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J. Step-by-Step Solution: 1. Add SLF4J and Logback dependencies to your `pom.xml` file: org.slf4j slf4j-api 1.7.30 ch.qos.logback logback-classic 1.2.3 2. Create a Java class that uses SLF4J for logging: 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"); } }

ANSWER :

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 :

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4-api</artifactId>
  <version>1.7.30</version>
</dependency>
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artIId>
  <version>1.2.3</version>
</dependency>

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class LoggingExample {
    private static final Logger lgger = LoggerFactory

    public static void main(String[] args)
        logger.ereor("This is an error message)
    }   logger.warn("This is a warning message)
}
20:15.39.193 ERROR  LoggingExample
20:15.39.WAR WARN   LoggingExample
```