**Exercise 1: Control Structures**

CREATE TABLE Customers (

CustomerID NUMBER,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER(10, 2),

IsVIP CHAR(1),

LoanInterestRate NUMBER(5, 2)

);

INSERT INTO Customers VALUES (1, 'Tiyasha', 65, 12000, 'F', 8.5);

INSERT INTO Customers VALUES (2, 'Amit', 45, 9500, 'F', 9.0);

INSERT INTO Customers VALUES (3, 'Rita', 70, 15000, 'F', 10.0);

INSERT INTO Customers VALUES (4, 'Rahul', 30, 11000, 'F', 8.8);

CREATE TABLE Loans (

LoanID NUMBER,

CustomerID NUMBER,

DueDate DATE

);

INSERT INTO Loans VALUES (101, 1, SYSDATE + 10);

INSERT INTO Loans VALUES (102, 2, SYSDATE + 35);

INSERT INTO Loans VALUES (103, 3, SYSDATE + 5);

INSERT INTO Loans VALUES (104, 4, SYSDATE + 60);

COMMIT;

**Scenario1:**

BEGIN

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

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust.CustomerID;

END LOOP;

END;

/

**Output:**

| **CustomerID** | **Name** | **Old Rate** | **New Rate** |
| --- | --- | --- | --- |
| 1 | Tiyasha | 8.5 | 7.5 |
| 3 | Rita | 10.0 | 9.0 |

**Scenario2:**

BEGIN

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

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

END LOOP;

END;

/

**Output:**

| **CustomerID** | **Name** | **Age** | **Balance** | **IsVIP** | **LoanInterestRate** |
| --- | --- | --- | --- | --- | --- |
| 1 | Tiyasha | 65 | 12000 | T | 7.5 |
| 2 | Amit | 45 | 9500 | F | 9.0 |
| 3 | Rita | 70 | 15000 | T | 9.0 |
| 4 | Rahul | 30 | 11000 | T | 8.8 |

**Scenario3:**

DECLARE

v\_name VARCHAR2(100);

BEGIN

FOR loan IN (SELECT LoanID, CustomerID, DueDate FROM Loans

WHERE DueDate <= SYSDATE + 30) LOOP

SELECT Name INTO v\_name FROM Customers WHERE CustomerID = loan.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || v\_name ||

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

END LOOP;

END;

/

**Output:**

Reminder: Loan ID 101 for customer Tiyasha is due on 09-JUL-2025

Reminder: Loan ID 103 for customer Rita is due on 04-JUL-2025

**Exercise 3: Stored Procedures**

CREATE TABLE SavingsAccounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(50),

Salary NUMBER

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

INSERT INTO SavingsAccounts VALUES (101, 1, 10000);

INSERT INTO SavingsAccounts VALUES (102, 2, 15000);

INSERT INTO Employees VALUES (1, 'Ravi', 'IT', 40000);

INSERT INTO Employees VALUES (2, 'Neha', 'HR', 35000);

INSERT INTO Accounts VALUES (201, 1, 5000);

INSERT INTO Accounts VALUES (202, 1, 3000);

**Scenario1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR acc IN (SELECT AccountID, Balance FROM SavingsAccounts) LOOP

UPDATE SavingsAccounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountID = acc.AccountID;

END LOOP;

END;

/

**Output:**

| **AccountID** | **CustomerID** | **Balance** |
| --- | --- | --- |
| 101 | 1 | 10100 |
| 102 | 2 | 15150 |

**Scenario2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_pct IN NUMBER

) AS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

END;

/

**Output:**

| **EmployeeID** | **Name** | **Department** | **Salary** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Ravi | IT | 40000 |

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | Neha | HR | 38500 |

**Scenario3:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_acc IN NUMBER,

p\_to\_acc IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

-- Check if source has enough balance

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_acc;

IF v\_balance >= p\_amount THEN

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_acc;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_acc;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance');

END IF;

END;

/

**Output:**

| **AccountID** | **CustomerID** | **Balance** |
| --- | --- | --- |
| 201 | 1 | 4000 |
| 202 | 1 | 4000 |

**Exercise 1: Setting Up JUnit**

Changing pom.xml

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>untitled</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>24</maven.compiler.source>  
 <maven.compiler.target>24</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
  
  
</project>

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

import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class CalculatorTest {  
 public void testAdd() {  
 Calculator calc = new Calculator();  
 int result = calc.add(2, 3);  
 assertEquals(5, result);  
 }  
}

**Output:**

Addition Result: 5

Process finished with exit code 0

Tests passed: 1 of 1 test – CalculatorTest

**Exercise 3: Assertions in JUnit**

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());  
 }  
}

**Output:**

Tests passed: 1 of 1 test – AssertionsTest

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

public class Calculator {  
 public int add(int a, int b) {  
 return a + b;  
 }  
  
 public int subtract(int a, int b) {  
 return a - b;  
 }  
}

import org.junit.Before;  
import org.junit.After;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class CalculatorTest {  
  
 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 testSubtract() {  
   
 int result = calculator.subtract(10, 5);  
 assertEquals(5, result);  
 }  
}

**Output:**

Setup complete

Teardown complete

Setup complete

Teardown complete

Tests passed: 2 of 2 tests – CalculatorTest

**Exercise 1: Mocking and Stubbing**

public interface ExternalApi {  
 String getData();  
}

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

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:

Tests passed: 1 of 1 test – MyServiceTest

**Exercise 2: Verifying Interactions**

public interface ExternalApi {  
 String getData();  
}

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

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 testVerifyInteraction() {  
   
 ExternalApi mockApi = Mockito.mock(ExternalApi.class);  
 MyService service = new MyService(mockApi);  
  
 service.fetchData();  
  
 verify(mockApi).getData(); // ✔️ Verifies that getData() was called  
 }  
}

Output:

Tests passed: 1 of 1 test – MyServiceTest

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

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");   
 }  
}

<configuration>  
 <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">  
 <encoder>  
 <pattern>%d{HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n</pattern>  
 </encoder>  
 </appender>  
  
 <root level="debug">  
 <appender-ref ref="STDOUT" />  
 </root>  
 </configuration>

**Output:**

16:02:31 [main] ERROR LoggingExample - This is an error message

16:02:31 [main] WARN LoggingExample - This is a warning message

16:02:31 [main] INFO LoggingExample - This is an info message

16:02:31 [main] DEBUG LoggingExample - This is a debug message