**Week 2 Mandatory Hands-On :**

**PL/SQL Programming :**

**Exercise 1 – Control Procedures**  
Scenario 1 :   
BEGIN  
  FOR cust\_rec IN (SELECT CustomerID, DOB FROM Customers) LOOP  
    IF MONTHS\_BETWEEN(SYSDATE, cust\_rec.DOB) / 12 > 60 THEN  
      UPDATE Loans  
      SET InterestRate = InterestRate - 1  
      WHERE CustomerID = cust\_rec.CustomerID;  
    END IF;  
  END LOOP;  
  COMMIT;  
END;  
/  
Output :  
Data before:  
John Doe DOB = '1985-05-15' → Age ~ 40 → No update  
Jane Smith DOB = '1990-07-20' → Age ~ 34 → No update   
  
Result:  
➤ No customers over 60 → No loan interest rate updated  
➤ No DBMS output unless you add it manually.   
  
Scenario 2 :   
BEGIN  
  FOR cust\_rec IN (SELECT CustomerID, Balance FROM Customers) LOOP  
    IF cust\_rec.Balance > 10000 THEN  
      UPDATE Customers  
      SET IsVIP = 'TRUE'  
      WHERE CustomerID = cust\_rec.CustomerID;  
    ELSE  
      UPDATE Customers  
      SET IsVIP = 'FALSE'  
      WHERE CustomerID = cust\_rec.CustomerID;  
    END IF;  
  END LOOP;  
  COMMIT;  
END;  
/  
Output :   
Set IsVIP to TRUE for Customers with Balance > $10,000   
  
Data before:  
John Doe: Balance = 1000  
Jane Smith: Balance = 1500   
  
IsVIP values:  
SELECT Name, Balance, IsVIP FROM Customers;   
  
Name Balance IsVIP   
John Doe 1000 FALSE  
Jane Smith 1500 FALSE  
  
➤ Output: No one is marked as VIP in your sample data.   
  
Scenario 3 :   
DECLARE  
  v\_name Customers.Name%TYPE;  
BEGIN  
  FOR loan\_rec IN (  
    SELECT CustomerID, LoanID, EndDate  
    FROM Loans  
    WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30  
  ) LOOP  
    SELECT Name INTO v\_name FROM Customers WHERE CustomerID = loan\_rec.CustomerID;  
      
    DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || loan\_rec.LoanID || ' for customer ' || v\_name || ' is due on ' || TO\_CHAR(loan\_rec.EndDate, 'DD-MON-YYYY'));  
  END LOOP;  
END;  
/  
Output :  
Sample Data:  
INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)  
VALUES (2, 2, 4000, 4.5, SYSDATE, SYSDATE + 15);

Output:  
Reminder: Loan 2 for customer Jane Smith is due on 10-JUL-2025  
  
**Ex 2 : Stored Procedures**   
Scenario 1 :  
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS  
BEGIN  
  FOR acc\_rec IN (  
    SELECT AccountID, Balance  
    FROM Accounts  
    WHERE AccountType = 'Savings'  
  ) LOOP  
    UPDATE Accounts  
    SET Balance = Balance + (Balance \* 0.01)  
    WHERE AccountID = acc\_rec.AccountID;  
  END LOOP;  
  COMMIT;  
END;  
/  
Output :  
Before:  
SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings';  
-- ID 1: Balance = 1000   
  
After:  
EXEC ProcessMonthlyInterest;  
SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings';  
-- ID 1: Balance = 1010 (1000 + 1%)   
  
Scenario 2 :   
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;  
/  
Output :   
Before:  
Bob Brown (IT): 60,000   
  
After:  
Bob Brown (IT): 66,000  
  
SELECT Name, Department, Salary FROM Employees;   
  
Name Department Salary   
  
Alice HR 70000  
Bob IT 66000  
  
Scenario 3 :  
CREATE OR REPLACE PROCEDURE TransferFunds (  
  p\_from\_account IN NUMBER,  
  p\_to\_account IN NUMBER,  
  p\_amount IN NUMBER  
) IS  
  v\_balance NUMBER;  
BEGIN  
  -- Get source account balance  
  SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_account;   
  
  IF v\_balance >= p\_amount THEN  
    -- Deduct from source  
    UPDATE Accounts  
    SET Balance = Balance - p\_amount  
    WHERE AccountID = p\_from\_account;   
  
    -- Add to destination  
    UPDATE Accounts  
    SET Balance = Balance + p\_amount  
    WHERE AccountID = p\_to\_account;   
  
    COMMIT;  
    DBMS\_OUTPUT.PUT\_LINE('Transfer successful.');  
  ELSE  
    DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in source account.');  
  END IF;  
EXCEPTION  
  WHEN NO\_DATA\_FOUND THEN  
    DBMS\_OUTPUT.PUT\_LINE('Invalid account ID.');  
  WHEN OTHERS THEN  
    DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  
END;  
/  
Output :  
Before:  
SELECT \* FROM Accounts;   
  
-- AccountID 1 (John): 1010  
-- AccountID 2 (Jane): 1500   
Output:  
Transfer successful.   
After:  
SELECT \* FROM Accounts;   
  
-- AccountID 1: 510  
-- AccountID 2: 2000   
  
> If Account 1 had less than $500:  
  
Insufficient balance in source account.

**J Unit Basic Examples :**  
Ex 1 :  
Calculator class :   
//src/main/java/com/example/Calculator.java  
package com.example;   
  
public class Calculator {  
    public int add(int a, int b) {  
        return a + b;  
    }  
}   
  
Corresponding test class :-  
//src/test/java/com/example/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(2, 3);  
        assertEquals(5, result);  
    }  
}  
Console output :  
-------------------------------------------------------  
T E S T S  
-------------------------------------------------------  
Running com.example.CalculatorTest  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.005 sec   
  
Results :   
  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0  
If you make the test fail :  
assertEquals(6, result); // wrong expected value  
Then output :   
java.lang.AssertionError:   
Expected :6  
Actual   :5   
  
Ex 3 :  
Updated and complete solution code :  
//src/test/java/com/example/AssertionsTest.java  
package com.example;   
  
import org.junit.Test;  
import static org.junit.Assert.\*;   
  
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());  
    }  
}  
Output (all Tests pass) :  
-------------------------------------------------------  
T E S T S  
-------------------------------------------------------  
Running com.example.AssertionsTest  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.002 sec   
  
Results :   
  
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0   
  
If a test fails (example) :  
If you change one of the assertions like :  
assertEquals(6, 2 + 3); // Wrong expected value  
Output :  
java.lang.AssertionError:  
Expected :6  
Actual   :5   
  
Ex 4 :  
Calculator class :  
//src/main/java/com/example/Calculator.java  
package com.example;   
  
public class Calculator {  
    public int add(int a, int b) {  
        return a + b;  
    }   
  
    public void reset() {  
        // Placeholder for reset logic  
        System.out.println("Calculator reset.");  
    }  
}  
Test class using AAA pattern @After and @Before  
// src/test/java/com/example/CalculatorTest.java  
package com.example;   
  
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() {  
        // Arrange: Initialize the calculator before each test  
        calculator = new Calculator();  
        System.out.println("Setup complete.");  
    }   
  
    @After  
    public void tearDown() {  
        // Cleanup logic after each test  
        calculator.reset();  
        System.out.println("Teardown complete.");  
    }   
  
    @Test  
    public void testAdd\_PositiveNumbers() {  
        // Arrange: Done in setUp   
  
        // Act  
        int result = calculator.add(10, 5);   
  
        // Assert  
        assertEquals(15, result);  
    }   
  
    @Test  
    public void testAdd\_NegativeNumbers() {  
        // Arrange: Done in setUp   
  
        // Act  
        int result = calculator.add(-3, -2);   
  
        // Assert  
        assertEquals(-5, result);  
    }  
}   
  
Console Output when Tests run :  
Setup complete.  
Calculator reset.  
Teardown complete.  
Setup complete.  
Calculator reset.  
Teardown complete.   
  
-------------------------------------------------------  
T E S T S  
-------------------------------------------------------  
Running com.example.CalculatorTest  
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 sec   
  
Results :   
  
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0

**Mockito Exercises :-**

Exercise 1:  
// 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();  
    }  
}  
Test code (mock+stub) :  
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() {  
        // Step 1: Create a mock  
        ExternalApi mockApi = Mockito.mock(ExternalApi.class);   
  
        // Step 2: Stub method  
        when(mockApi.getData()).thenReturn("Mock Data");   
  
        // Step 3: Use the mock  
        MyService service = new MyService(mockApi);  
        String result = service.fetchData();   
  
        // Step 4: Assert  
        assertEquals("Mock Data", result);  
    }  
}  
Output:  
Tests run: 1, Failures: 0  
✔ All tests passed  
  
Ex 2 :  
Test code (verification) :  
import static org.mockito.Mockito.\*;  
import org.junit.jupiter.api.Test;  
import org.mockito.Mockito;   
  
public class MyServiceTest {   
  
    @Test  
    public void testVerifyInteraction() {  
        // Step 1: Mock object  
        ExternalApi mockApi = Mockito.mock(ExternalApi.class);   
  
        // Step 2: Use it in service  
        MyService service = new MyService(mockApi);  
        service.fetchData();   
  
        // Step 3: Verify interaction  
        verify(mockApi).getData();  
    }  
}  
Output   
If getData() was called → test passes.   
  
If not :  
Wanted but not invoked:  
externalApi.getData();

**SLF4J – Logging Exercises –**

LoggingExample.java :

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

// Create logger instance

private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);

public static void main(String[] args) {

// Log an error message

logger.error("This is an error message");

// Log a warning 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 :

12:00:00.123 [main] ERROR LoggingExample - This is an error message

12:00:00.124 [main] WARN LoggingExample - This is a warning message