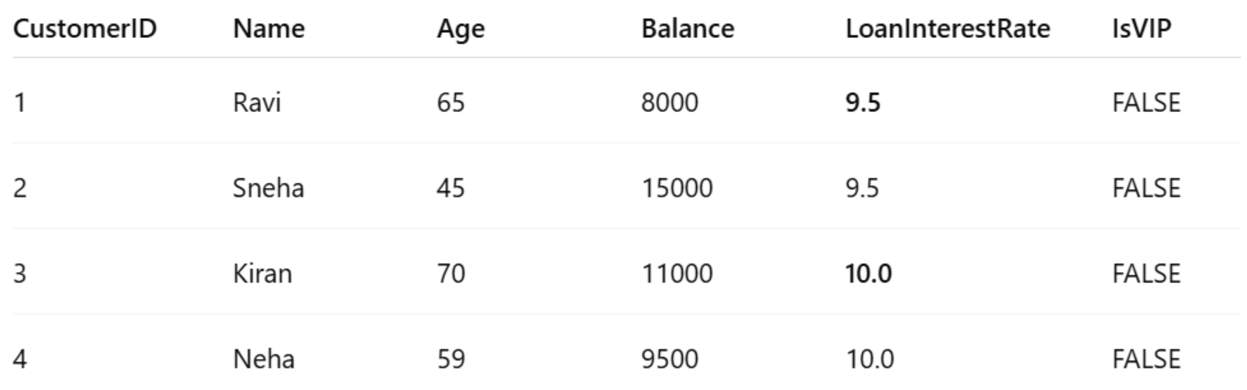
WEEK 2_PL SQL, Unit testing

Exercise 1: Control Structures

Scenario 1

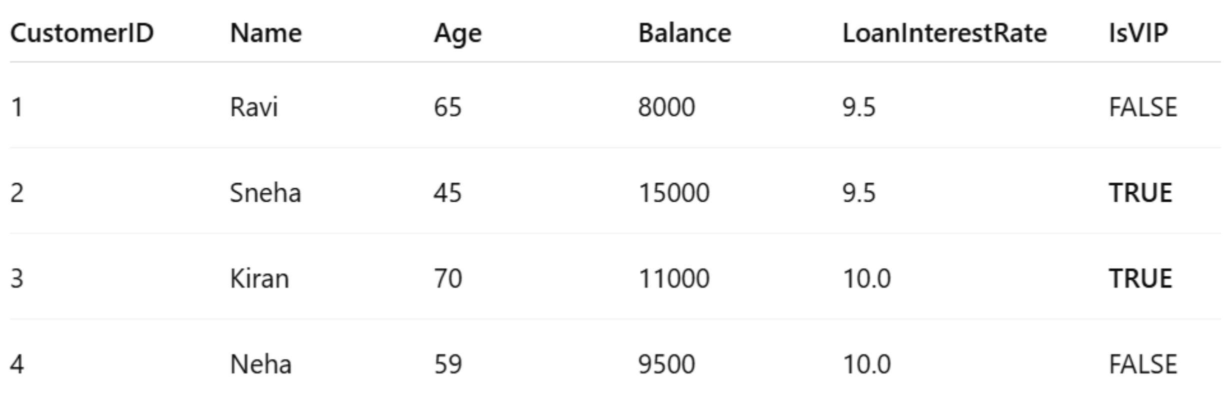
```
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE Customers';
EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF;
END;

CREATE TABLE Customers (
  CustomerID NUMBER PRIMARY KEY,
  Name VARCHAR2(50),
  Age NUMBER,
  Balance NUMBER,
  LoanInterestRate NUMBER,
  IsVIP VARCHAR2(5)
);

INSERT INTO Customers VALUES (1, 'Ravi', 65, 8000, 10.5, 'FALSE');
INSERT INTO Customers VALUES (2, 'Sneha', 45, 15000, 9.5, 'FALSE');
INSERT INTO Customers VALUES (3, 'Kiran', 70, 11000, 11.0, 'FALSE');
INSERT INTO Customers VALUES (4, 'Neha', 59, 9500, 10.0, 'FALSE');
COMMIT;

BEGIN
  FOR cust IN (SELECT CustomerID, Age FROM Customers) LOOP
    IF cust.Age > 60 THEN
      UPDATE Customers
      SET LoanInterestRate = LoanInterestRate - 1
      WHERE CustomerID = cust.CustomerID;
    END IF;
  END LOOP;
END;

SELECT * FROM Customers;
```



CustomerID	Name	Age	Balance	LoanInterestRate	IsVIP
1	Ravi	65	8000	**9.5**	FALSE
2	Sneha	45	15000	9.5	FALSE
3	Kiran	70	11000	**10.0**	FALSE
4	Neha	59	9500	10.0	FALSE

Scenario 2

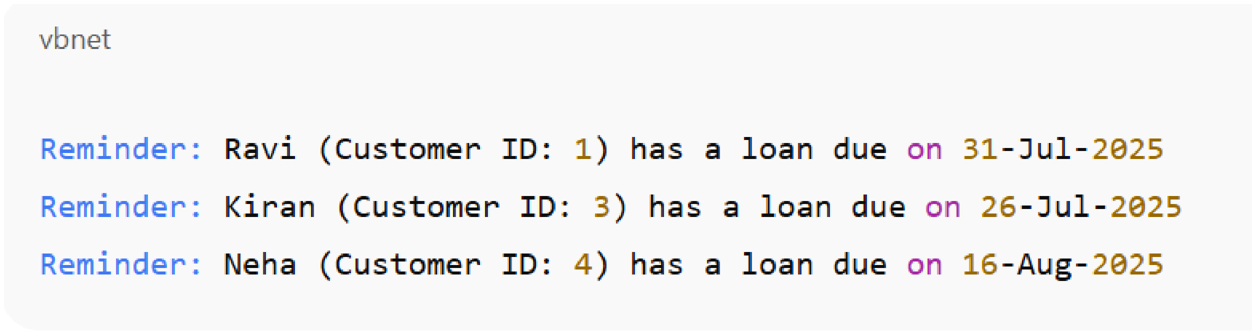
```
BEGIN
  FOR cust IN (SELECT CustomerID, Balance FROM Customers) LOOP
    IF cust.Balance > 10000 THEN
      UPDATE Customers
      SET IsVIP = 'TRUE'
      WHERE CustomerID = cust.CustomerID;
    END IF;
  END LOOP;
END;

SELECT * FROM Customers;
```



CustomerID	Name	Age	Balance	LoanInterestRate	IsVIP
1	Ravi	65	8000	9.5	FALSE
2	Sneha	45	15000	9.5	**TRUE**
3	Kiran	70	11000	10.0	**TRUE**
4	Neha	59	9500	10.0	FALSE

Scenario 2

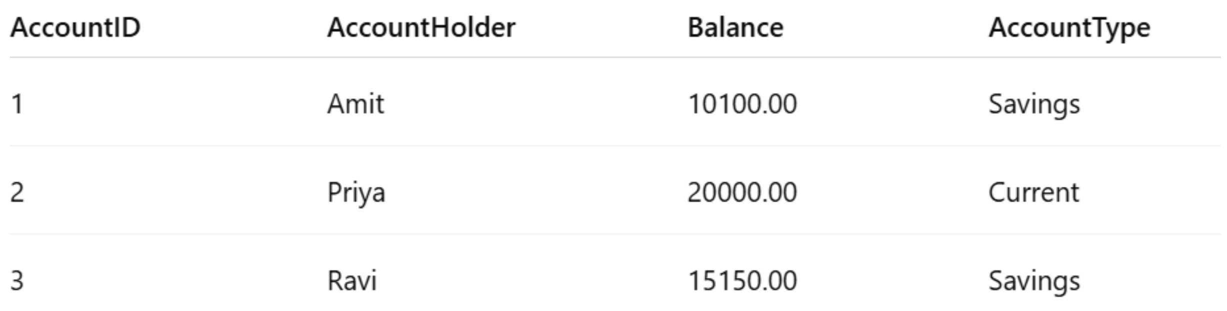
```
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE Loans';
EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF;
END;

CREATE TABLE Loans (
  LoanID NUMBER PRIMARY KEY,
  CustomerID NUMBER,
  DueDate DATE
);

INSERT INTO Loans VALUES (101, 1, SYSDATE + 10);  -- due soon
INSERT INTO Loans VALUES (102, 2, SYSDATE + 35);  -- not due soon
INSERT INTO Loans VALUES (103, 3, SYSDATE + 5);   -- due soon
INSERT INTO Loans VALUES (104, 4, SYSDATE + 25);  -- due soon
COMMIT;

SET SERVEROUTPUT ON;
BEGIN
  FOR loan IN (
   SELECT l.LoanID, l.CustomerID, c.Name, l.DueDate
   FROM Loans l
   JOIN Customers c ON l.CustomerID = c.CustomerID
   WHERE l.DueDate <= SYSDATE + 30
  ) LOOP
DBMS_OUTPUT.PUT_LINE('Reminder: ' || loan.Name || ' (Customer ID: ' || loan.CustomerID ||
') has a loan due on ' || TO_CHAR(loan.DueDate, 'DD-Mon-YYYY'));
END LOOP;
END;
```


vbnet

```
Reminder: Ravi (Customer ID: 1) has a loan due on 31-Jul-2025
Reminder: Kiran (Customer ID: 3) has a loan due on 26-Jul-2025
Reminder: Neha (Customer ID: 4) has a loan due on 16-Aug-2025
```

Exercise 3: Stored Procedures
Scenario 1

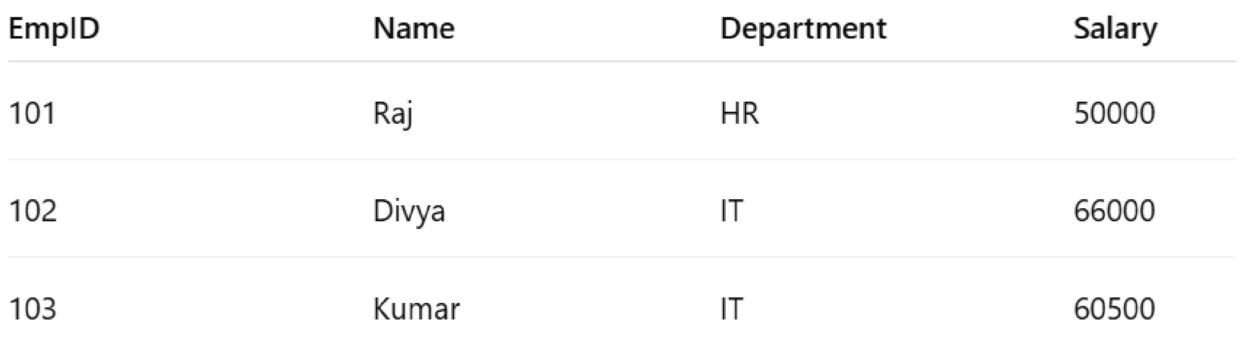
```
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE Accounts';
EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF;
END;

CREATE TABLE Accounts (
  AccountID NUMBER PRIMARY KEY,
  AccountHolder VARCHAR2(50),
  Balance NUMBER,
  AccountType VARCHAR2(20)
);

INSERT INTO Accounts VALUES (1, 'Amit', 10000, 'Savings');
INSERT INTO Accounts VALUES (2, 'Priya', 20000, 'Current');
INSERT INTO Accounts VALUES (3, 'Ravi', 15000, 'Savings');
COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
  FOR acc IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType =
'Savings') LOOP
    UPDATE Accounts
    SET Balance = Balance + (Balance * 0.01)
    WHERE AccountID = acc.AccountID;
  END LOOP;
END;

BEGIN
  ProcessMonthlyInterest;
END;

SELECT * FROM Accounts;
```



AccountID	AccountHolder	Balance	AccountType
1	Amit	10100.00	Savings
2	Priya	20000.00	Current
3	Ravi	15150.00	Savings

Scenario 2

```
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE Employees';
EXCEPTION WHEN OTHERS THEN IF SQLCODE != -942 THEN RAISE; END IF;
END;

CREATE TABLE Employees (
  EmpID NUMBER PRIMARY KEY,
  Name VARCHAR2(50),
  Department VARCHAR2(30),
  Salary NUMBER
);

INSERT INTO Employees VALUES (101, 'Raj', 'HR', 50000);
INSERT INTO Employees VALUES (102, 'Divya', 'IT', 60000);
INSERT INTO Employees VALUES (103, 'Kumar', 'IT', 55000);
COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
  deptName IN VARCHAR2,
  bonusPercent IN NUMBER
) IS
BEGIN
  UPDATE Employees
  SET Salary = Salary + (Salary * bonusPercent / 100)
  WHERE Department = deptName;
END;

BEGIN
  UpdateEmployeeBonus('IT', 10);
END;

SELECT * FROM Employees;
```

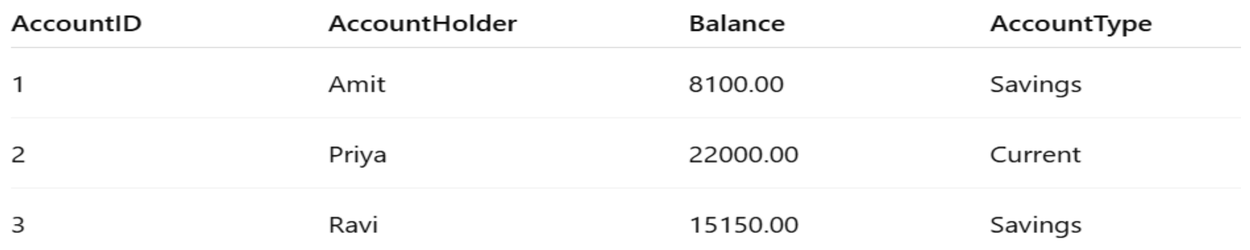


EmpID	Name	Department	Salary
101	Raj	HR	50000
102	Divya	IT	66000
103	Kumar	IT	60500

Scenario 3

```
CREATE OR REPLACE PROCEDURE TransferFunds (
  fromAccountID IN NUMBER,
  toAccountID IN NUMBER,
  amount IN NUMBER
) IS
  fromBalance NUMBER;
BEGIN
```

```
SELECT Balance INTO fromBalance FROM Accounts WHERE AccountID = fromAccountID;


IF fromBalance < amount THEN RAISE_APPLICATION_ERROR(-20001, 'Insufficient
balance in source account.'); END IF;

UPDATE Accounts
  SET Balance = Balance - amount
  WHERE AccountID = fromAccountID;


  UPDATE Accounts
  SET Balance = Balance + amount
  WHERE AccountID = toAccountID;
END;


BEGIN
  TransferFunds(1, 2, 2000);
END;


SELECT * FROM Accounts;
```



AccountID	AccountHolder	Balance	AccountType
1	Amit	8100.00	Savings
2	Priya	22000.00	Current
3	Ravi	15150.00	Savings

Exercise 1: Setting Up JUnit

1.

```
// File: src/main/java/com/example/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;
    }
    public int multiply(int a, int b) {
        return a * b;
    }
    public int divide(int a, int b) {
        if (b == 0)
            throw new IllegalArgumentException("Division by zero not allowed");
        return a / b;
    }
}
```

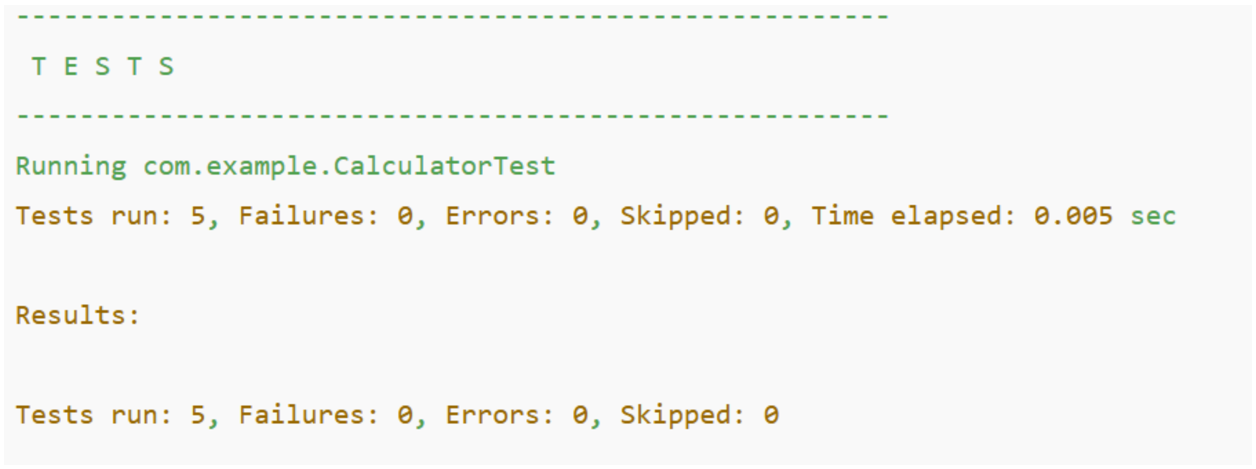
2.

```
// File: src/test/java/com/example/CalculatorTest.java
package com.example;
import org.junit.Test;
import static org.junit.Assert.*;
public class CalculatorTest {
    @Test
    public void testAdd() {
        Calculator calc = new Calculator();
        assertEquals(5, calc.add(2, 3));
    }
    @Test
    public void testSubtract() {
        Calculator calc = new Calculator();
        assertEquals(2, calc.subtract(5, 3));
    }
    @Test
    public void testMultiply() {
        Calculator calc = new Calculator();
        assertEquals(15, calc.multiply(3, 5));
    }
    @Test
    public void testDivide() {
        Calculator calc = new Calculator();
        assertEquals(2, calc.divide(10, 5));
    }
    @Test(expected = IllegalArgumentException.class)
    public void testDivideByZero() {
        Calculator calc = new Calculator();
        calc.divide(10, 0);
    }
}
```

3.

```
<dependencies>
```

```
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```



```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.example.CalculatorTest
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.005 sec

Results:

Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
```

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

Step 1: Business Logic - BankAccount.java

```
// File: src/main/java/com/example/BankAccount.java
package com.example;

public class BankAccount {
    private String owner;
    private double balance;

    public BankAccount(String owner, double initialBalance) {
        this.owner = owner;
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        if (amount <= 0)
            throw new IllegalArgumentException("Deposit must be positive");
        balance += amount;
    }

    public void withdraw(double amount) {
        if (amount > balance)
            throw new IllegalArgumentException("Insufficient balance");
        balance -= amount;
    }

    public double getBalance() {
        return balance;
    }

    public String getOwner() {
        return owner;
    }
}
```

Step 2: JUnit Test - BankAccountTest.java

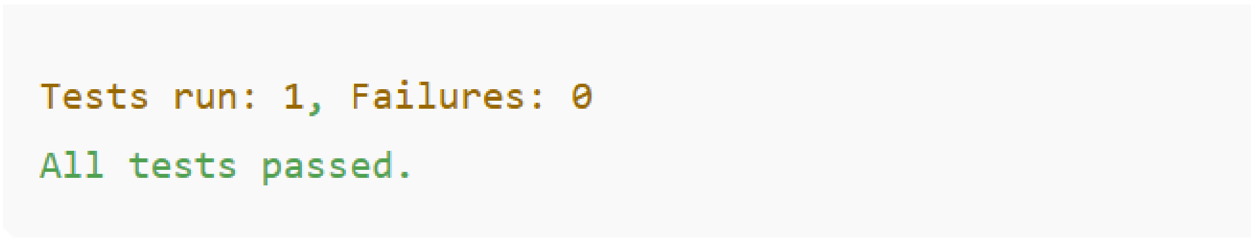
```
package com.example;


import org.junit.After;

import org.junit.Before;

import org.junit.Test;

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

```
public class BankAccountTest {
    private BankAccount account;
    // Setup method: runs before each test
    @Before
    public void setUp() {
        account = new BankAccount("Neethu", 1000.0);
        System.out.println("Setup complete: New account created");
    }
    // Teardown method: runs after each test
    @After
    public void tearDown() {
        account = null;
        System.out.println("Teardown complete: Account object destroyed");
    }
    @Test
    public void testDeposit() {
        // Arrange
        double depositAmount = 500.0;
        // Act
        account.deposit(depositAmount);

        // Assert
        assertEquals(1500.0, account.getBalance(), 0.001);
    }

    @Test
    public void testWithdraw() {
        // Arrange
        double withdrawAmount = 400.0;

        // Act
        account.withdraw(withdrawAmount);

        // Assert
        assertEquals(600.0, account.getBalance(), 0.001);
    }

    @Test(expected = IllegalArgumentException.class)
    public void testWithdrawInsufficientBalance() {
        // Act
        account.withdraw(2000.0);
    }
}
```

```
Setup complete: New account created
Teardown complete: Account object destroyed

Setup complete: New account created
Teardown complete: Account object destroyed

Setup complete: New account created
Teardown complete: Account object destroyed

Tests run: 3, Failures: 0, Errors: 0, Skipped: 0
```

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();
    }
}
<!-- pom.xml -->
<dependencies>
    <!-- JUnit 5 -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.10.0</version>
        <scope>test</scope>
    </dependency>

    <!-- Mockito -->
    <dependency>
        <groupId>org.mockito</groupId>
        <artifactId>mockito-core</artifactId>
        <version>5.12.0</version>
        <scope>test</scope>
    </dependency>
</dependencies>
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.when;

public class MyServiceTest {

    @Test
    public void testExternalApi() {
        // Step 1: Create a mock object
        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);
    }
}
```



```
Tests run: 1, Failures: 0
All tests passed.
```

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 org.junit.jupiter.api.Test;
import static org.mockito.Mockito.*;
public class MyServiceTest {
  @Test
  public void testVerifyInteraction() {
    ExternalApi mockApi = mock(ExternalApi.class);
    MyService service = new MyService(mockApi);
    service.fetchData();
    verify(mockApi).getData();
  }
}
```

```
✓ Test passed: method getData() was called exactly once as expected.
```

Exercise 1: Logging Error Messages and Warning Levels

```
<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>com.example</groupId>

    <artifactId>logging-example</artifactId>

    <version>1.0-SNAPSHOT</version>


    <dependencies>

        <!-- SLF4J API -->

        <dependency>

            <groupId>org.slf4j</groupId>

            <artifactId>slf4j-api</artifactId>

            <version>1.7.30</version>

        </dependency>


        <!-- Logback Classic Implementation -->

        <dependency>

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

            <artifactId>logback-classic</artifactId>

            <version>1.2.3</version>

        </dependency>

    </dependencies>
```

```
</project>
```

```
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");
    }
}
```

```
<configuration>
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%date [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <root level="debug">
        <appender-ref ref="CONSOLE"/>
    </root>
</configuration>
```

```
2025-07-21 19:04:00 [main] ERROR LoggingExample - This is an error message
2025-07-21 19:04:00 [main] WARN  LoggingExample - This is a warning message
```