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.

QUERY:

```
DECLARE
CURSOR cust_cursor IS
SELECT c.CustomerID, l.LoanID, l.InterestRate
FROM Customers c
JOIN Loans l ON c.CustomerID = l.CustomerID
WHERE TRUNC(MONTHS_BETWEEN(SYSDATE, c.DOB) / 12) > 60;

BEGIN
FOR cust_rec IN cust_cursor LOOP
UPDATE Loans
SET InterestRate = cust_rec.InterestRate - 1
WHERE LoanID = cust_rec.LoanID;
END LOOP;
COMMIT;
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.

QUERY:

```
ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

DECLARE
  CURSOR vip_cursor IS
    SELECT CustomerID, Balance FROM Customers WHERE Balance > 10000;

BEGIN
  FOR vip_rec IN vip_cursor LOOP
    UPDATE Customers
    SET IsVIP = 'TRUE'
    WHERE CustomerID = vip_rec.CustomerID;
  END LOOP;

  COMMIT;
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-

```
DECLARE
  CURSOR loan_cursor IS
    SELECT l.LoanID, l.CustomerID, c.Name, l.EndDate
    FROM Loans l
    JOIN Customers c ON c.CustomerID = l.CustomerID
    WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN
  FOR rec IN loan_cursor LOOP
    DBMS_OUTPUT.PUT_LINE('Reminder: Loan ID ' || rec.LoanID ||
                         ' for Customer ' || rec.Name ||
                         ' is due on ' || TO_CHAR(rec.EndDate, 'DD-MON-YYYY'));
  END LOOP;
END;
/
```

OUTPUT:

SC1-

```
-- DOBs in Customers:
-- John Doe: 1985-05-15 → Age: 39
-- Jane Smith: 1990-07-20 → Age: 34
```

```
-- No changes made.
-- InterestRate remains 5 for John Doe (LoanID = 1).
```

SC2-

```
-- No changes made.
-- IsVIP will remain NULL (or default) for both customers.

-- Customer balances:
-- John Doe: 1000
-- Jane Smith: 1500
```

SC3-

```
-- No DBMS_OUTPUT messages displayed.
```

```
-- Only one loan:
-- LoanID: 1, EndDate: ADD_MONTHS(SYSDATE, 60) → 5 years in future.
```

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.

Ans->

Sc1- CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS
BEGIN
  UPDATE Accounts
  SET Balance = Balance + (Balance * 0.01)

```
  WHERE AccountType = 'Savings';
 COMMIT;
END;
/
```

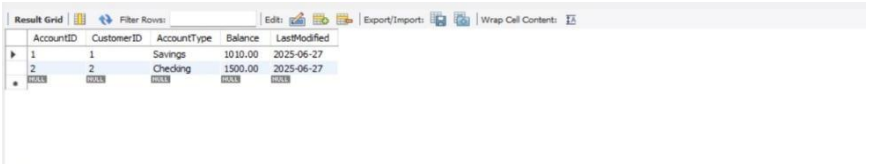
Sc2-

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
  p_Department IN VARCHAR2,
  p_BonusPercent IN NUMBER
) AS
BEGIN
  UPDATE Employees
  SET Salary = Salary + (Salary * p_BonusPercent / 100)
  WHERE Department = p_Department;
 COMMIT;
END;
/
```

Sc3-

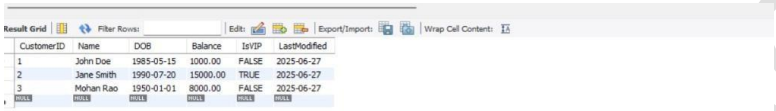
```
CREATE OR REPLACE PROCEDURE TransferFunds (
  p_FromAccountID IN NUMBER,
  p_ToAccountID IN NUMBER,
  p_Amount IN NUMBER
) AS
  v_FromBalance NUMBER;
BEGIN
  SELECT Balance INTO v_FromBalance
  FROM Accounts
  WHERE AccountID = p_FromAccountID
  FOR UPDATE;

  IF v_FromBalance < p_Amount THEN
    RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance for transfer.');
  END IF;

  UPDATE Accounts
  SET Balance = Balance - p_Amount
  WHERE AccountID = p_FromAccountID;

  UPDATE Accounts
  SET Balance = Balance + p_Amount
  WHERE AccountID = p_ToAccountID;

  COMMIT;
END;
/
```

Output-

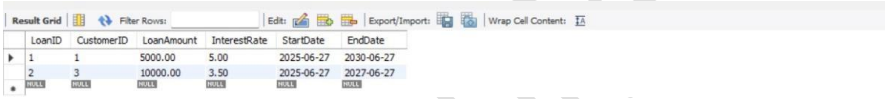


AccountID	CustomerID	AccountType	Balance	LastModified
1	1	Savings	1010.00	2025-06-27
2	2	Checking	1500.00	2025-06-27

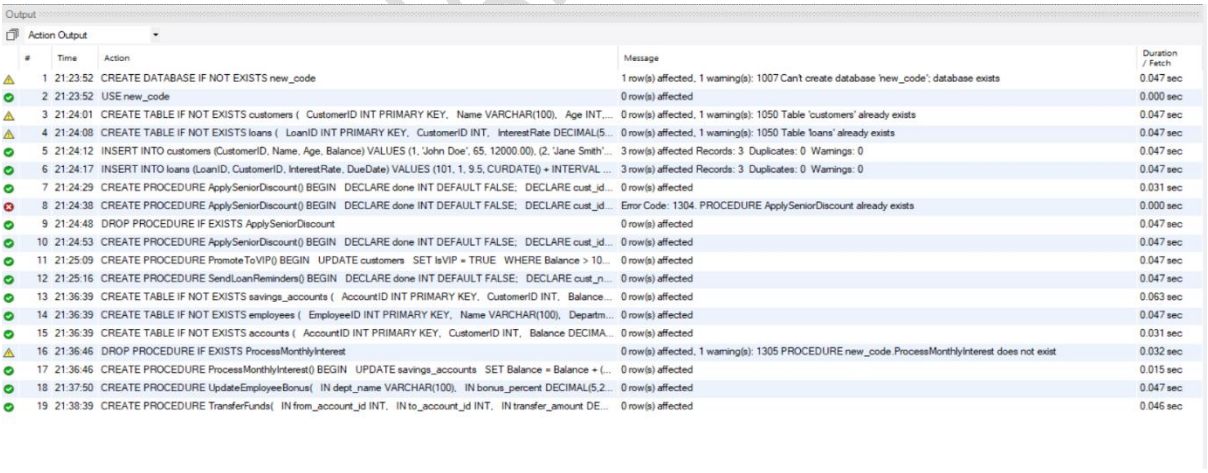
LoanID	CustomerName	EndDate



CustomerID	Name	DOB	Balance	IsVIP	LastModified
1	John Doe	1985-05-15	1000.00	FALSE	2025-06-27
2	Jane Smith	1990-07-20	15000.00	TRUE	2025-06-27
3	Mohan Rao	1950-01-01	8000.00	FALSE	2025-06-27



LoanID	CustomerID	LoanAmount	InterestRate	StartDate	EndDate
1	1	5000.00	5.00	2025-06-27	2030-06-27
2	3	10000.00	3.50	2025-06-27	2027-06-27



Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	21:23:52	CREATE DATABASE IF NOT EXISTS new_code	1 row(s) affected, 1 warning(s): 1007 Can't create database 'new_code'; database exists	0.047 sec
2	21:23:52	USE new_code	0 row(s) affected	0.000 sec
3	21:24:01	CREATE TABLE IF NOT EXISTS customers (CustomerID INT PRIMARY KEY, Name VARCHAR(100), Age INT,...	0 row(s) affected, 1 warning(s): 1050 Table 'customers' already exists	0.047 sec
4	21:24:08	CREATE TABLE IF NOT EXISTS loans (LoanID INT PRIMARY KEY, CustomerID INT, InterestRate DECIMAL(5...	0 row(s) affected, 1 warning(s): 1050 Table 'loans' already exists	0.047 sec
5	21:24:12	INSERT INTO customers (CustomerID, Name, Age, Balance) VALUES (1, 'John Doe', 65, 12000.00), (2, 'Jane Smith'...	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.047 sec
6	21:24:17	INSERT INTO loans (LoanID, CustomerID, InterestRate, DueDate) VALUES (101, 1, 9.5, CURDATE() + INTERVAL ...	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.047 sec
7	21:24:29	CREATE PROCEDURE ApplySeniorDiscount() BEGIN DECLARE done INT DEFAULT FALSE; DECLARE cust_id...	0 row(s) affected	0.031 sec
8	21:24:38	CREATE PROCEDURE ApplySeniorDiscount() BEGIN DECLARE done INT DEFAULT FALSE; DECLARE cust_id...	Error Code: 1304. PROCEDURE ApplySeniorDiscount already exists	0.000 sec
9	21:24:48	DROP PROCEDURE IF EXISTS ApplySeniorDiscount	0 row(s) affected	0.047 sec
10	21:24:53	CREATE PROCEDURE ApplySeniorDiscount() BEGIN DECLARE done INT DEFAULT FALSE; DECLARE cust_id...	0 row(s) affected	0.047 sec
11	21:25:09	CREATE PROCEDURE PromoteToVIP() BEGIN UPDATE customers SET IsVIP = TRUE WHERE Balance > 10...	0 row(s) affected	0.047 sec
12	21:25:16	CREATE PROCEDURE SendLoanReminders() BEGIN DECLARE done INT DEFAULT FALSE; DECLARE cust_n...	0 row(s) affected	0.047 sec
13	21:36:39	CREATE TABLE IF NOT EXISTS savings_accounts (AccountID INT PRIMARY KEY, CustomerID INT, Balance...	0 row(s) affected	0.063 sec
14	21:36:39	CREATE TABLE IF NOT EXISTS employees (EmployeeID INT PRIMARY KEY, Name VARCHAR(100), Departm...	0 row(s) affected	0.047 sec
15	21:36:39	CREATE TABLE IF NOT EXISTS accounts (AccountID INT PRIMARY KEY, CustomerID INT, Balance DECIMA...	0 row(s) affected	0.031 sec
16	21:36:46	DROP PROCEDURE IF EXISTS ProcessMonthlyInterest	0 row(s) affected, 1 warning(s): 1305 PROCEDURE new_code.ProcessMonthlyInterest does not exist	0.032 sec
17	21:36:46	CREATE PROCEDURE ProcessMonthlyInterest() BEGIN UPDATE savings_accounts SET Balance = Balance + (...	0 row(s) affected	0.015 sec
18	21:37:50	CREATE PROCEDURE UpdateEmployeeBonus(IN dept_name VARCHAR(100), IN bonus_percent DECIMAL(5,2...	0 row(s) affected	0.047 sec
19	21:38:39	CREATE PROCEDURE TransferFunds(IN from_account_id INT, IN to_account_id INT, IN transfer_amount DE...	0 row(s) affected	0.046 sec

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:

3. Create a new test class in your project

```
<project>
    ...
    <dependencies>
        <!-- JUnit 4 -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

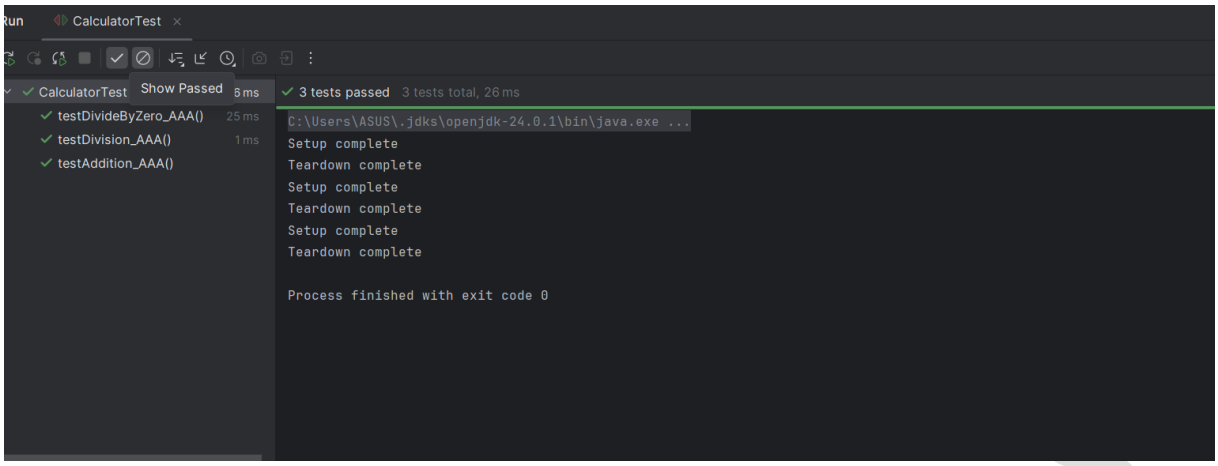
Calculator.java

```
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 testAddition() {
        // Arrange
        Calculator calculator = new Calculator();
        int a = 2;
        int b = 3;

        // Act
        int result = calculator.add(a, b);

        // Assert
        assertEquals(5, result);
    }
}
```

Output



Exercise 3: Assertions in JUnit Scenario:

You need to use different assertions in JUnit to validate your test results.

AssertionTest.java

```java
package com.example;

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

public class AssertionTest {

    @Test
    void testAssertions() {
        // Assert equals
        assertEquals(5, 2 + 3);

        // Assert true
        assertTrue(true);

        // Assert false
        assertFalse(false);

        // Assert null
        Object obj1 = null;
        assertNull(null);

        // Assert not null
        Object obj2 = new Object();
        assertNotNull(obj2);
    }
}
```

Output:

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.

1. Write tests using the AAA pattern.

2. Use @Before and @After annotations for setup and teardown methods.

Calculator.java

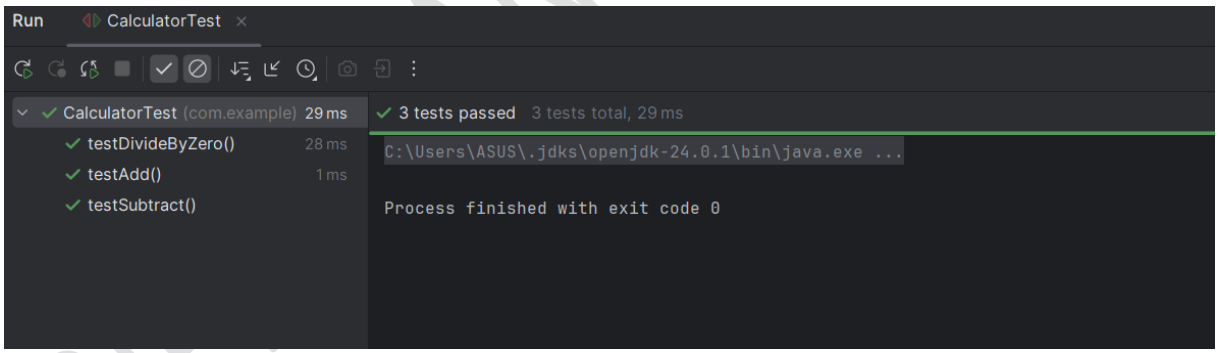
```
package com.example;

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("Cannot divide by
zero");
        return a/b;
    }
}
```

CalculatorTest.java

```
package com.example;

import org.junit.jupiter.api.*;

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

public class CalculatorTest {

    Calculator calc;

    @BeforeEach
    void setUp() {
        // Arrange: Initialize calculator before each test
        calc = new Calculator();
```

```
    }

    @AfterEach
    void tearDown() {
        // Optionally nullify or log cleanup
        calc = null;
    }

    @Test
    void testAdd() {
        // Act
        int result = calc.add(2, 3);
        // Assert
        assertEquals(5, result);
    }

    @Test
    void testSubtract() {
        int result = calc.subtract(5, 3);
        assertEquals(2, result);
    }

    @Test
    void testDivideByZero() {
        Exception exception = assertThrows(IllegalArgumentException.class,
() -> {
            calc.divide(10, 0);
        });
        assertEquals("Cannot divide by zero", exception.getMessage());
}
}
```

Output:



Mockito Hands-On Exercises

Exercise 1: Mocking and Stubbing Scenario: You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

Steps: 1. Create a mock object for the external API.

2. Stub the methods to return predefined values.

3. Write a test case that uses the mock object

Interface

```java
package com.example;

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

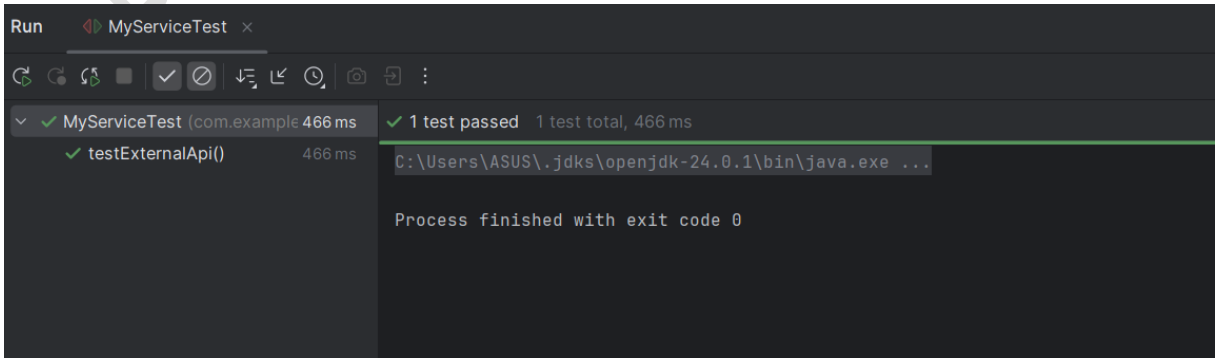
MyServiceTest.java

```java
package com.example;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

public class MyServiceTest {

    @Test
    public void testExternalApi() {
        // Step 1: Create mock object
        ExternalApi mockApi = mock(ExternalApi.class);

        // Step 2: Stub method
        when(mockApi.getData()).thenReturn("Mock Data");

        // Step 3: Use in service and test
        MyService service = new MyService(mockApi);
        String result = service.fetchData();

        assertEquals("Mock Data", result);
}
}
```

MyService

```java
package com.example;

public class MyService {

    private ExternalApi api;

    public MyService(ExternalApi api) {
        this.api = api;
    }

    public String fetchData() {
        return api.getData(); // This would normally call an external API
}
}
```

Output



Exercise 2: Verifying Interactions Scenario: You need to ensure that a method is called with specific arguments.

Steps: 1. Create a mock object.

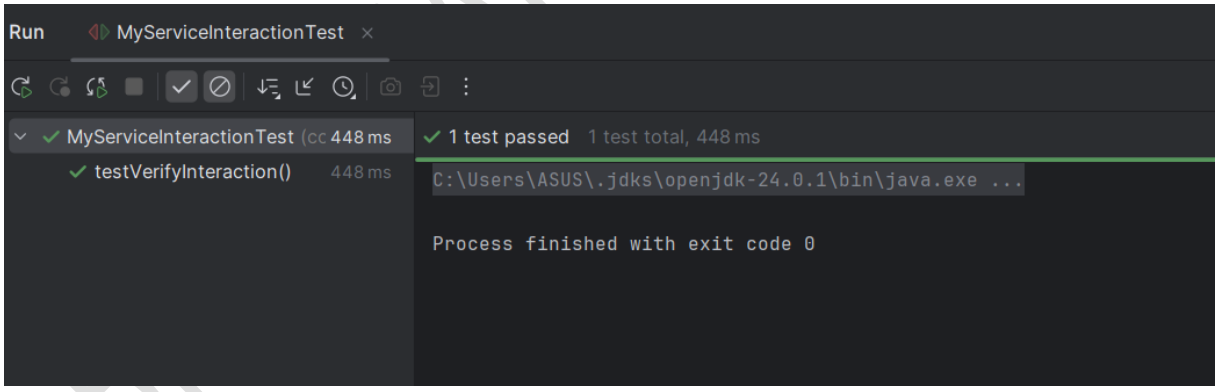
2. Call the method with specific arguments.

3. Verify the interaction.

MyServiceInteractionTest.java

```
package com.example;

import org.junit.jupiter.api.Test;
import static org.mockito.Mockito.*;

public class MyServiceInteractionTest {

    @Test
    public void testVerifyInteraction() {
        // Step 1: Create mock
        ExternalApi mockApi = mock(ExternalApi.class);

        // Step 2: Call fetchData()
        MyService service = new MyService(mockApi);
        service.fetchData();

        // Step 3: Verify interaction
        verify(mockApi, times(1)).getData();  // 👍 THIS LINE
 }
}
```

Output:



Logging using SLF4J

Exercise 1: Logging Error Messages and Warning Levels Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

1. **Add SLF4J and Logback dependencies to your `pom.xml` file:**

2. **Create a Java class that uses SLF4J for logging:**

Pom.xml

```
<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>LoggingExample</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

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

LoggingExample.java

```
package com.example;

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