Week 2 Exercises

# PLSQL Exercises

# Exercise 1: Control Structures

Create and Insert Statements for Control Structures  
  
CREATE TABLE Customers (  
 CustomerID NUMBER PRIMARY KEY,  
 Name VARCHAR2(50),  
 Age NUMBER,  
 Balance NUMBER,  
 IsVIP VARCHAR2(5)  
);  
  
CREATE TABLE Loans (  
 LoanID NUMBER PRIMARY KEY,  
 CustomerID NUMBER,  
 InterestRate NUMBER,  
 DueDate DATE  
);  
  
INSERT INTO Customers VALUES (1, 'Aarav ', 65, 15000, 'FALSE');  
INSERT INTO Customers VALUES (2, 'Sneha ', 45, 9500, 'FALSE');  
INSERT INTO Customers VALUES (3, 'Ravi ', 70, 12000, 'FALSE');  
  
INSERT INTO Loans VALUES (101, 1, 9.5, SYSDATE + 10);  
INSERT INTO Loans VALUES (102, 2, 10.0, SYSDATE + 40);  
INSERT INTO Loans VALUES (103, 3, 8.5, SYSDATE + 5);  
  
Exercise 1: Control Structures - Scenario 1

## Scenario:

The bank wants to apply a discount to loan interest rates for customers above 60 years old.

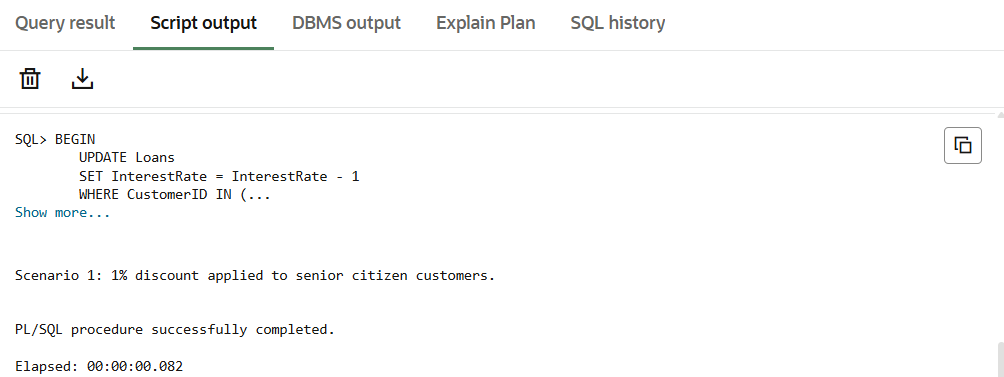
## Steps:

1. Identify customers whose age is greater than 60.
2. Update their loan interest rate by subtracting 1%.
3. Display a message that discount is applied.

## Code:

**BEGIN  
 UPDATE Loans  
 SET InterestRate = InterestRate - 1  
 WHERE CustomerID IN (  
 SELECT CustomerID FROM Customers WHERE Age > 60  
 );  
 DBMS\_OUTPUT.PUT\_LINE('Scenario 1: 1% discount applied to senior citizen customers.');  
END;  
/**

## Output:



# Exercise 1: Control Structures - Scenario 2

## Scenario:

A customer can be promoted to VIP status based on their balance.

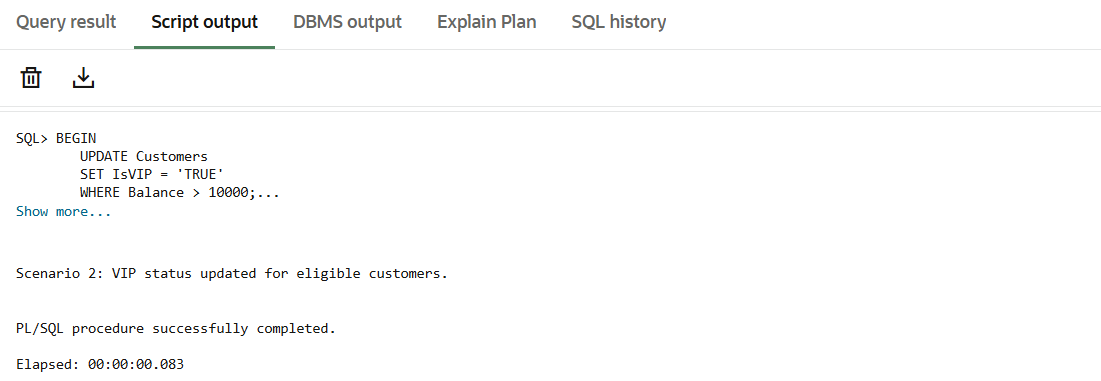
## Steps:

1. Find all customers with balance more than $10,000.
2. Set their IsVIP flag to TRUE.
3. Display a message that VIP status is updated.

## Code:

BEGIN  
 UPDATE Customers  
 SET IsVIP = 'TRUE'  
 WHERE Balance > 10000;  
 DBMS\_OUTPUT.PUT\_LINE('Scenario 2: VIP status updated for eligible customers.');  
END;  
/

## Output:



# Exercise 1: Control Structures - Scenario 3

## Scenario:

The bank wants to send reminders to customers whose loans are due within the next 30 days.

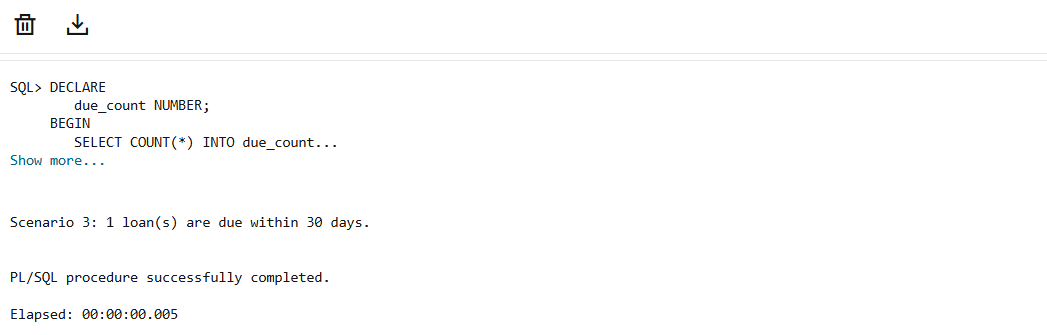
## Steps:

1. Identify loans that are due in the next 30 days.
2. Count how many such loans exist.
3. Print the number of loans due soon.

## Code:

DECLARE  
 due\_count NUMBER;  
BEGIN  
 SELECT COUNT(\*) INTO due\_count  
 FROM Loans  
 WHERE DueDate BETWEEN SYSDATE AND SYSDATE + 30;  
 DBMS\_OUTPUT.PUT\_LINE('Scenario 3: ' || due\_count || ' loan(s) are due within 30 days.');  
END;  
/

## Output:



# Exercise 3: Stored Procedures:

Create and Insert Statements for Stored Procedures  
  
CREATE TABLE SavingsAccounts (  
 AccountID NUMBER PRIMARY KEY,  
 Balance NUMBER  
);  
  
CREATE TABLE Employees (  
 EmployeeID NUMBER PRIMARY KEY,  
 Name VARCHAR2(50),  
 Salary NUMBER,  
 DepartmentID NUMBER  
);  
  
CREATE TABLE Accounts (  
 AccountID NUMBER PRIMARY KEY,  
 Balance NUMBER  
);  
  
INSERT INTO SavingsAccounts VALUES (201, 10000);  
INSERT INTO SavingsAccounts VALUES (202, 15000);  
  
INSERT INTO Employees VALUES (1, 'Priya ', 50000, 10);  
INSERT INTO Employees VALUES (2, 'Vikram ', 60000, 20);  
  
INSERT INTO Accounts VALUES (301, 8000);  
INSERT INTO Accounts VALUES (302, 5000);

# Exercise 3: Stored Procedures - Scenario 1

## Scenario:

The bank needs to process monthly interest for all savings accounts.

## Code:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS  
BEGIN  
 UPDATE SavingsAccounts  
 SET Balance = Balance \* 1.01;  
END;  
/

## Output:

## 

# Exercise 3: Stored Procedures - Scenario 2

## Scenario:

The bank wants to implement a bonus scheme for employees based on their performance.

## Code:

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

## Output:

## 

# Exercise 3: Stored Procedures - Scenario 3

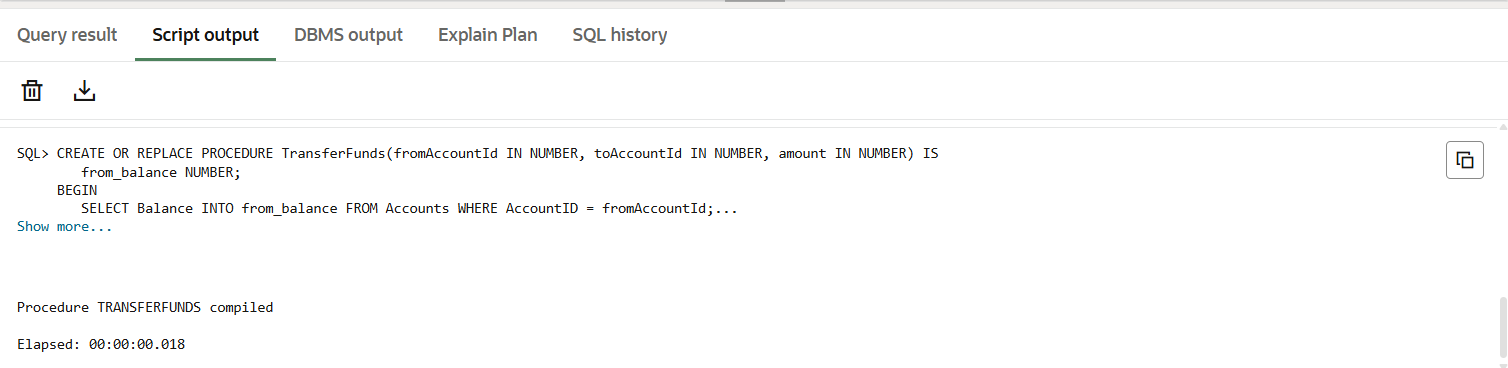
## Scenario:

Customers should be able to transfer funds between their accounts.

## Code:

CREATE OR REPLACE PROCEDURE TransferFunds(fromAccountId IN NUMBER, toAccountId IN NUMBER, amount IN NUMBER) IS  
 from\_balance NUMBER;  
BEGIN  
 SELECT Balance INTO from\_balance FROM Accounts WHERE AccountID = fromAccountId;  
 IF from\_balance >= amount THEN  
 UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = fromAccountId;  
 UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = toAccountId;  
 ELSE  
 RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance.');  
 END IF;  
END;  
/

## Output:



# 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:

* Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).
* Add JUnit dependency to your project. If you are using Maven, add the following to your

**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>junit-tests</artifactId>**

**<version>1.0-SNAPSHOT</version>**

**<dependencies>**

**<dependency>**

**<groupId>junit</groupId>**

**<artifactId>junit</artifactId>**

**<version>4.13.2</version>**

**<scope>test</scope>**

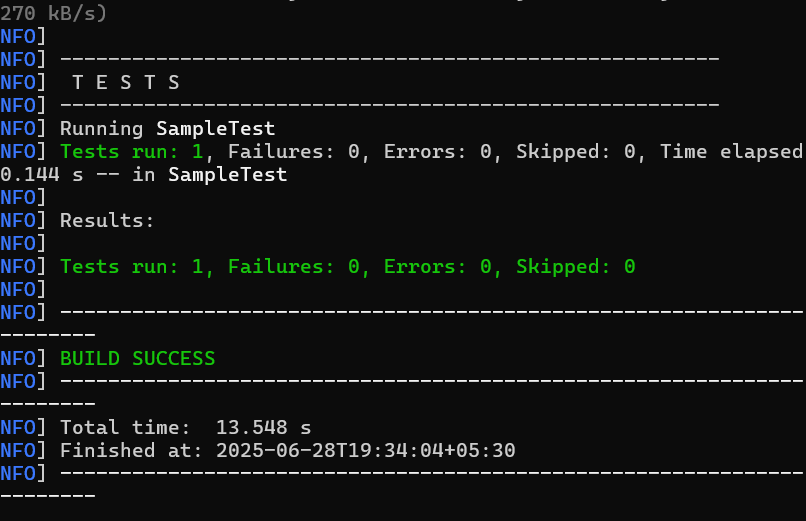
**</dependency>**

**</dependencies>**

**</project>**

* Create a new test class in your project.
* **File: src/test/java/SetupTest.java**
* **import org.junit.Test;**
* **import static org.junit.Assert.\*;**
* **public class SetupTest {**
* **@Test**
* **public void testEnvironment() {**
* **// This test checks if JUnit setup works**
* **assertTrue(true);**
* **}**
* **}**
* **File Path: src/main/java/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;**
* **}**
* **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;}}**

## Output:



## Exercise 3: Assertions in JUnit

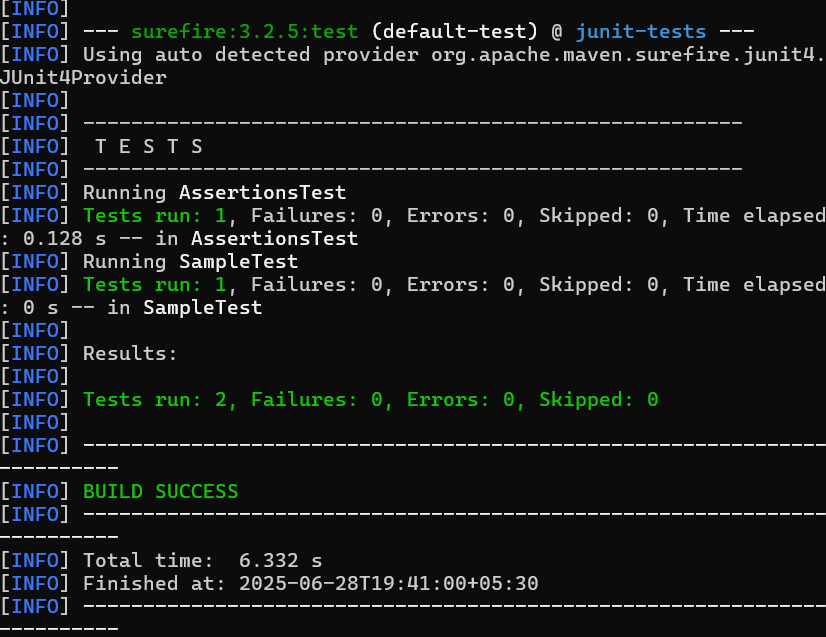
Scenario:  
You need to use different assertions in JUnit to validate your test results.

Steps:

* Write tests using various JUnit assertions.
* **File Path: src/main/java/AssertionsTest**

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:



## Exercise 4: AAA Pattern, Setup and Teardown

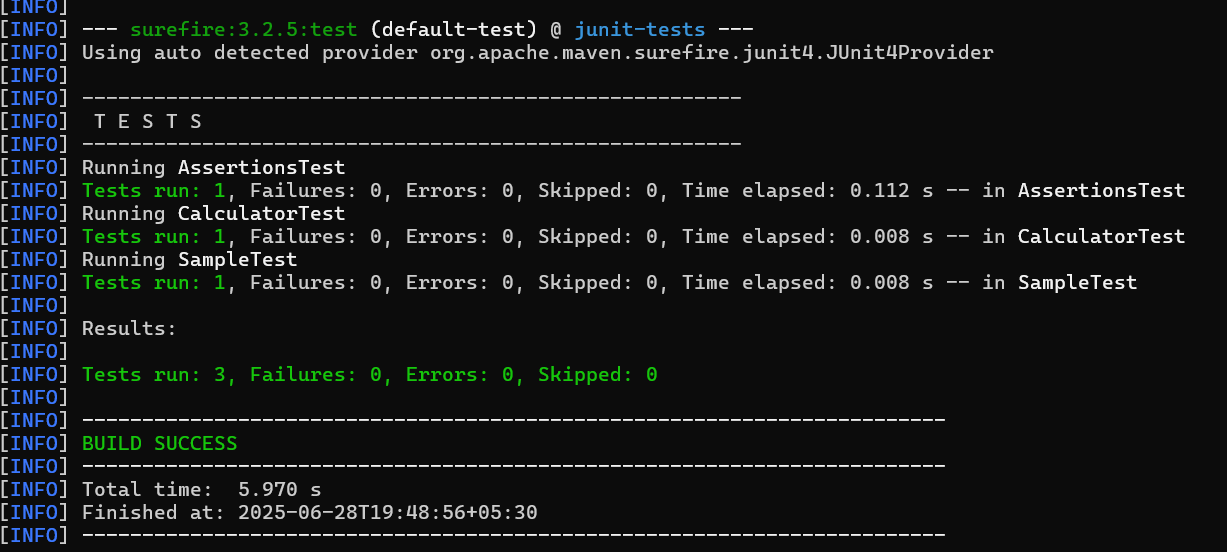
Scenario:  
You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

Steps:

* Write tests using the AAA pattern.
* Use @Before and @After annotations for setup and teardown methods.

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(); }  
 @After  
 public void tearDown() {  
 calculator = null;  
 }  
 @Test  
 public void testAddition() {  
 int result = calculator.add(2, 3);  
 assertEquals(5, result);  
 }  
}

## 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:

* Create a mock object for the external API.
* Stub the methods to return predefined values.
* Write a test case that uses the mock object.

**File Path: src/test/java/MyServiceTest.java**

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);}

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData(); }}

## Exercise 2: Verifying Interactions

Scenario:  
You need to ensure that a method is called with specific arguments.

Steps:

* Create a mock object.
* Call the method with specific arguments.
* Verify the interaction.
* **File Path: src/main/java/ExternalApi.java**
* public interface ExternalApi {
* String getData();
* }
* **File Path: src/main/java/MyService.java**
* public class MyService {
* private ExternalApi api;
* public MyService(ExternalApi api) {
* this.api = api;
* }
* public String fetchData() {
* return api.getData();
* }
* }
* **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>mockito-slf4j-example</artifactId>

## <version>1.0-SNAPSHOT</version>

## <dependencies>

## <!-- JUnit 5 -->

## <dependency>

## <groupId>org.junit.jupiter</groupId>

## <artifactId>junit-jupiter</artifactId>

## <version>5.9.1</version>

## <scope>test</scope>

## </dependency>

## <!-- Mockito -->

## <dependency>

## <groupId>org.mockito</groupId>

## <artifactId>mockito-core</artifactId>

## <version>4.11.0</version>

## <scope>test</scope>

## </dependency>

## <!-- SLF4J + Logback for logging -->

## <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>

## <build>

## <plugins>

## <!-- Needed to run main classes -->

## <plugin>

## <groupId>org.codehaus.mojo</groupId>

## <artifactId>exec-maven-plugin</artifactId>

## <version>3.1.0</version>

## <configuration>

## <mainClass>LoggingExample</mainClass>

## </configuration>

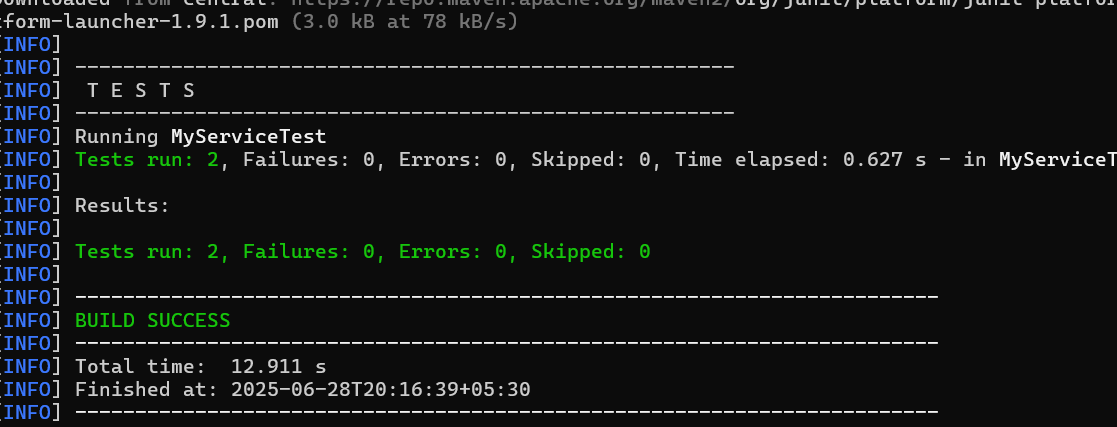
## </plugin>

## </plugins>

## </build>

## </project>

## Output:



# Logging using SLF4J

## Exercise 1: Logging Error Messages and Warning Levels

Scenario:  
Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

Steps:Add SLF4J and Logback dependencies to your 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>logging-example</artifactId>**
* **<version>1.0-SNAPSHOT</version>**
* **<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>**
* **<build>**
* **<plugins>**
* **<plugin>**
* **<groupId>org.codehaus.mojo</groupId>**
* **<artifactId>exec-maven-plugin</artifactId>**
* **<version>3.1.0</version>**
* **<configuration>**
* **<mainClass>LoggingExample</mainClass>**
* **</configuration>**
* **</plugin>**
* **</plugins> </build></project>**
* Create a Java class that uses SLF4J for logging:
* **File Path: src/main/java/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:

* 