PL/SQL programming:

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.

**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.

**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.

CODE:

Scenario 1:

CREATE PROCEDURE ApplySeniorDiscount()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE v\_cust\_id INT;

DECLARE cust\_cursor CURSOR FOR

SELECT CustID FROM CUSTOMERS WHERE Age > 60;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN cust\_cursor;

read\_loop: LOOP

FETCH cust\_cursor INTO v\_cust\_id;

IF done THEN

LEAVE read\_loop;

END IF;

UPDATE LOANS

SET InterestRate = InterestRate - 1

WHERE CustID = v\_cust\_id;

END LOOP;

CLOSE cust\_cursor;

END;

Scenario 2:

CREATE PROCEDURE PromoteToVIP()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE v\_cust\_id INT;

DECLARE vip\_cursor CURSOR FOR

SELECT CustID FROM CUSTOMERS WHERE Balance > 10000;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN vip\_cursor;

read\_loop: LOOP

FETCH vip\_cursor INTO v\_cust\_id;

IF done THEN

LEAVE read\_loop;

END IF;

UPDATE CUSTOMERS

SET IsVIP = 'TRUE'

WHERE CustID = v\_cust\_id;

END LOOP;

CLOSE vip\_cursor;

END;

Scenario 3:

CREATE PROCEDURE SendLoanReminders()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE v\_loan\_id INT;

DECLARE v\_cust\_id INT;

DECLARE v\_name VARCHAR(100);

DECLARE v\_due\_date DATE;

DECLARE loan\_cursor CURSOR FOR

SELECT LoanID, CustID, DueDate

FROM LOANS

WHERE DueDate <= CURDATE() + INTERVAL 30 DAY;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN loan\_cursor;

read\_loop: LOOP

FETCH loan\_cursor INTO v\_loan\_id, v\_cust\_id, v\_due\_date;

IF done THEN

LEAVE read\_loop;

END IF;

-- Get customer name

SELECT Name INTO v\_name FROM CUSTOMERS WHERE CustID = v\_cust\_id;

-- Print reminder

SELECT CONCAT('Reminder: Loan ', v\_loan\_id, ' for ', v\_name,

' is due on ', DATE\_FORMAT(v\_due\_date, '%d-%b-%Y')) AS Reminder;

END LOOP;

CLOSE loan\_cursor;

END;

TO RUN:

CALL ApplySeniorDiscount();

CALL PromoteToVIP();

CALL SendLoanReminders();

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

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.

CODE:

Scenario 1:

CREATE PROCEDURE ProcessMonthlyInterest()

BEGIN

UPDATE ACCOUNTS

SET Balance = Balance \* 1.01

WHERE AccountType = 'SAVINGS';

END;

Scenario 2:

CREATE PROCEDURE UpdateEmployeeBonus(

IN dept\_name VARCHAR(50),

IN bonus\_percent DECIMAL(5,2)

)

BEGIN

UPDATE EMPLOYEES

SET Salary = Salary + (Salary \* bonus\_percent / 100)

WHERE Department = dept\_name;

END;

Scenario 3:

CREATE PROCEDURE TransferFunds(

IN from\_acc INT,

IN to\_acc INT,

IN amount DECIMAL(10,2)

)

BEGIN

DECLARE from\_balance DECIMAL(10,2);

SELECT Balance INTO from\_balance

FROM ACCOUNTS

WHERE AccountID = from\_acc;

IF from\_balance < amount THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Insufficient funds in source account.';

ELSE

UPDATE ACCOUNTS

SET Balance = Balance - amount

WHERE AccountID = from\_acc;

UPDATE ACCOUNTS

SET Balance = Balance + amount

WHERE AccountID = to\_acc;

END IF;

END;

TO RUN:

CALL UpdateEmployeeBonus('Sales', 10);

CALL TransferFunds(101, 202, 500.00);

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

TDD using JUnit5 and Mockito:

JUnit Basic Testing Exercise:

Exercise 1: Setting Up JUnit

**Scenario**: You need to set up JUnit in your Java project to start writing unit tests

**CODE:**

**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>com.example</groupId>

<artifactId>JunitDemo</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>21</maven.compiler.source>

<maven.compiler.target>21</maven.compiler.target>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**Class:**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**Test:**

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);

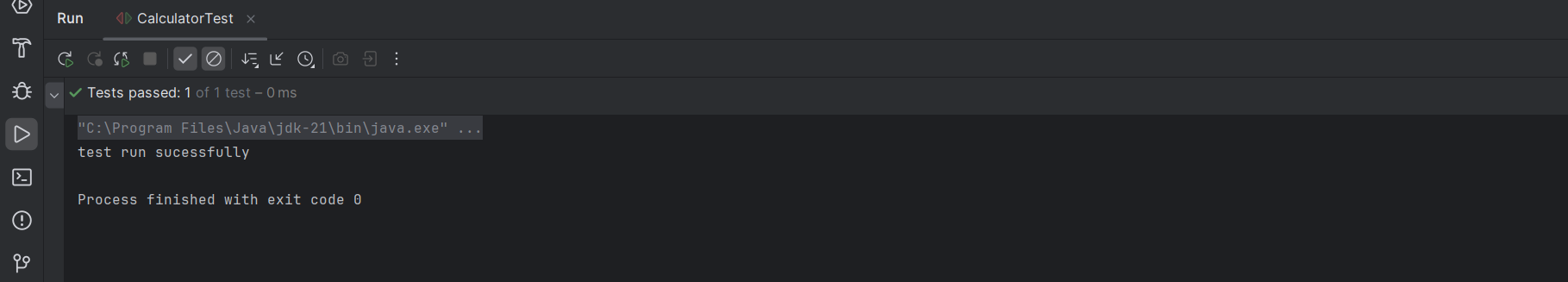
assertEquals(5, result);

System.out.println("test run sucessfully ");

    }

}

**OUTPUT:**



Exercise 3: Assertions in JUnit

**Scenario:** You need to write basic JUnit tests for a simple Java class.

**CODE:**

**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>com.example</groupId>

<artifactId>JunitDemo</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>21</maven.compiler.source>

<maven.compiler.target>21</maven.compiler.target>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**Test:**

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionTest {

@Test

public void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

assertFalse(5 < 3);

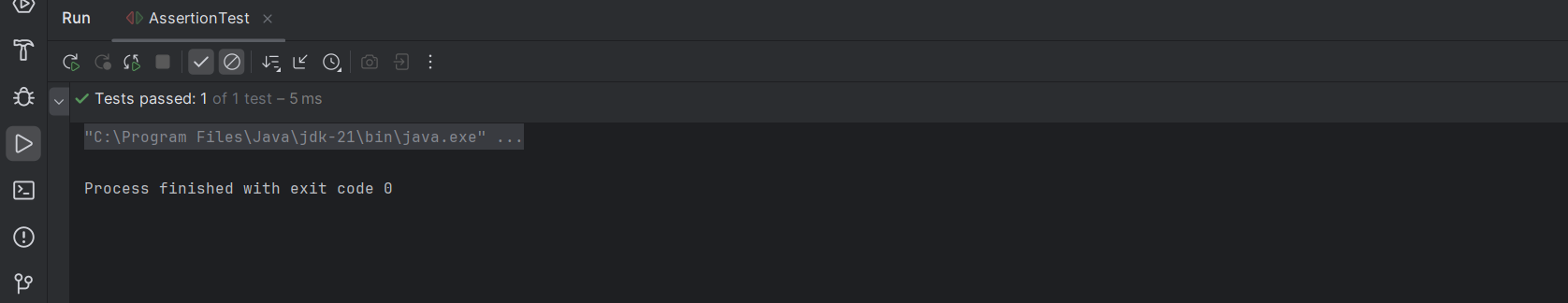
assertNull(null);

assertNotNull(new Object());

    }

}

**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.

**CODE:**

**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>com.example</groupId>

<artifactId>AAAPattern</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>21</maven.compiler.source>

<maven.compiler.target>21</maven.compiler.target>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**Class:**

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;

    }

}

**Test:**

package com.example;

import org.junit.After;

import org.junit.Before;

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("Setting up...");

}

@After

public void tearDown() {

calculator = null;

System.out.println("Cleaning up...");

}

@Test

public void testAddition() {

int a = 5;

int b = 3;

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

assertEquals(8, result);

}

@Test

public void testSubtraction() {

int a = 10;

int b = 4;

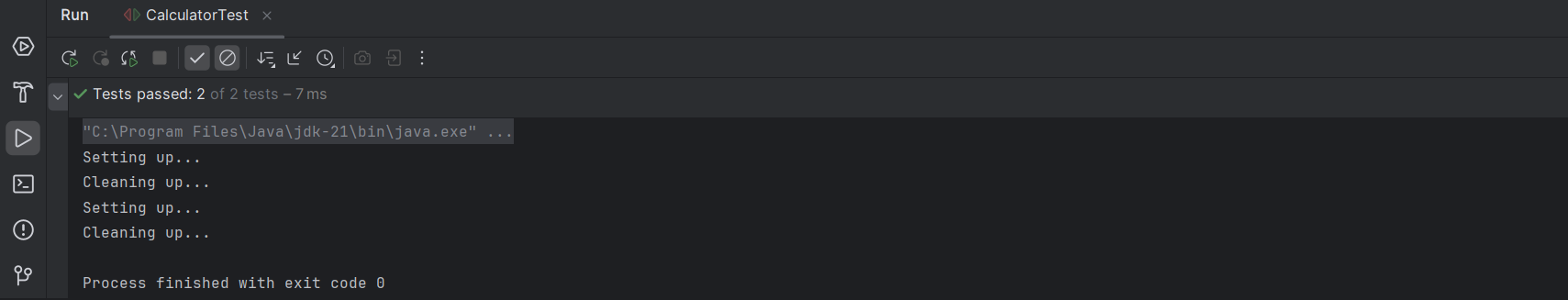
int result = calculator.subtract(a, b);

assertEquals(6, result);

}

}

**OUTPUT:**



Mockito Exercise:

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.

CODE:

**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-test-demo</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>17</maven.compiler.source>  
 <maven.compiler.target>17</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <junit.jupiter.version>5.9.3</junit.jupiter.version>  
 <mockito.version>5.11.0</mockito.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>5.12.0</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>5.10.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
   
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>3.0.0-M9</version>  
 </plugin>  
 </plugins>  
 </build>  
</project>

**Interface:**

package com.example;  
  
public interface ExternalApi {  
 String getData();  
}

**Class:**

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

**Test:**

import com.example.ExternalApi;  
import com.example.MyService;  
import org.junit.jupiter.api.Test;  
import org.mockito.Mockito;  
  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
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:

A computer screen shot of a keyboard

AI-generated content may be incorrect.

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.

CODE:

**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>com.example</groupId>  
 <artifactId>MockitoInteractionTest</artifactId>  
 <version>1.0-SNAPSHOT</version>

<properties>  
 <maven.compiler.source>17</maven.compiler.source>  
 <maven.compiler.target>17</maven.compiler.target>  
 <junit.jupiter.version>5.9.3</junit.jupiter.version>  
 <mockito.version>5.11.0</mockito.version>  
 </properties>  
  
 <dependencies>  
 <!-- JUnit 5 -->  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>${junit.jupiter.version}</version>  
 <scope>test</scope>  
 </dependency>  
  
 <!-- Mockito -->  
 <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>${mockito.version}</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <!-- Maven Surefire Plugin -->  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>3.0.0-M9</version>  
 </plugin>  
 </plugins>  
 </build>  
</project>

**Interface:**

package com.example;  
  
public interface ExternalApi {  
 String getData();  
}

**Class:**

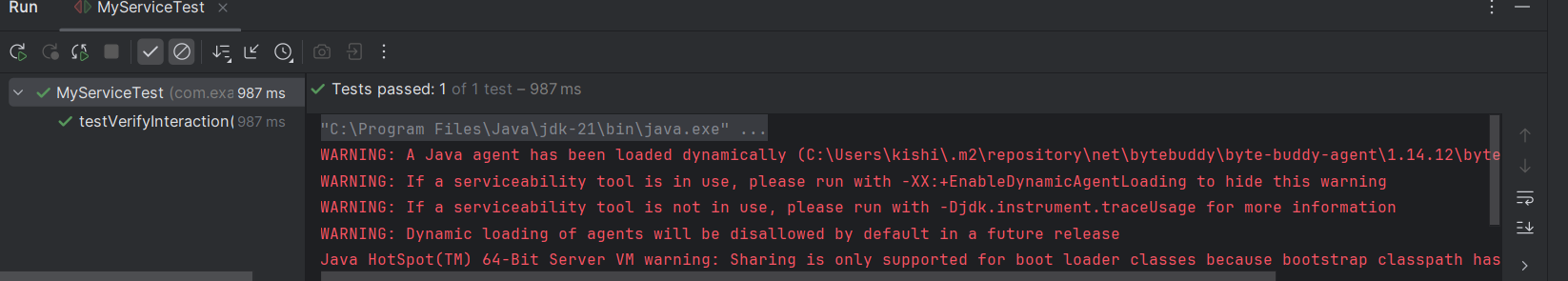
package com.example;  
  
public class MyService {  
 private ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;

}  
  
 public String fetchData() {  
 return api.getData();  
 }  
}

**Test:**

package com.example;  
  
import org.junit.jupiter.api.Test;  
import org.mockito.Mockito;  
  
import static 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();  
 }  
}

OUTPUT:



SLF4J logging framework

SL4J Logging exercises:

Exercise 1: Logging Error Messages and

Warning Levels

Task:

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

CODE:

**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>com.example</groupId>  
 <artifactId>sl4j</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>21</maven.compiler.source>  
 <maven.compiler.target>21</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
 <dependencies>  
 <!-- SLF4J API -->  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>1.7.36</version>  
 </dependency>  
  
 <!-- Logback (SLF4J implementation) -->  
 <dependency>  
 <groupId>ch.qos.logback</groupId>  
 <artifactId>logback-classic</artifactId>  
 <version>1.2.11</version>  
 </dependency>  
 </dependencies>  
</project>

**CLASS:**

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.");  
 *logger*.info("This is an informational message.");  
 *logger*.debug("This is a debug message.");  
 }  
}

OUTPUT:

A black screen with many small squares

AI-generated content may be incorrect.