**Digital nurture Deep skilling-Week2**

**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: List customers older than 25 (like applying senior discount)

SELECT

'Scenario 1 - Customers above age 25' AS Scenario,

customer\_id,

first\_name || ' ' || last\_name AS full\_name,

age

FROM Customers

WHERE age > 25;

-- Scenario 2: Promote customers to VIP if total order amount > 1000

SELECT

'Scenario 2 - VIP Customers (Spent > 1000)' AS Scenario,

C.customer\_id,

C.first\_name || ' ' || C.last\_name AS full\_nam

e,

SUM(O.amount) AS total\_spent

FROM Customers C

JOIN Orders O ON C.customer\_id = O.customer\_id

GROUP BY C.customer\_id

HAVING total\_spent > 1000;

-- Scenario 3: Remind customers with 'Pending' shipping status

SELECT

'Scenario 3 - Shipping Reminder' AS Scenario,

C.customer\_id,

C.first\_name || ' ' || C.last\_name AS full\_name,

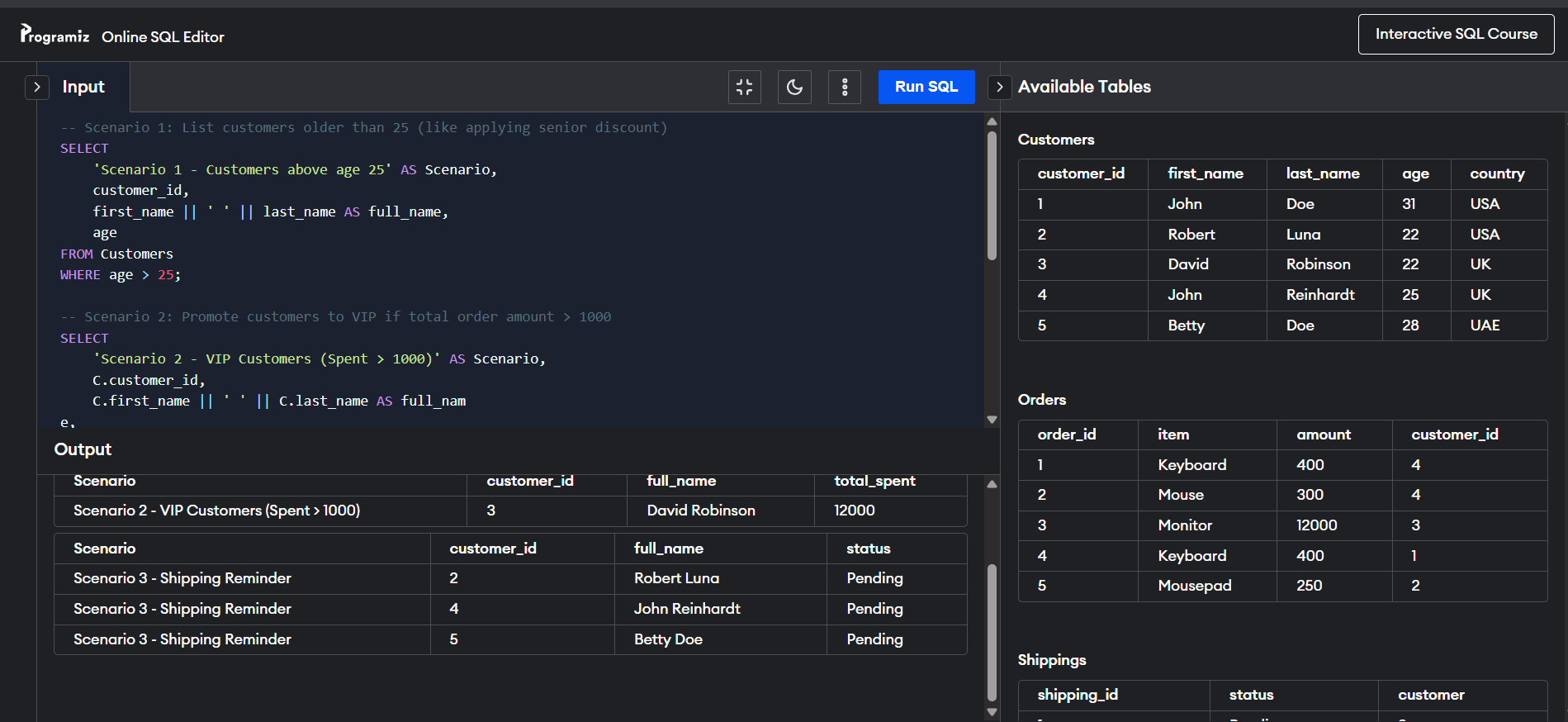
S.status

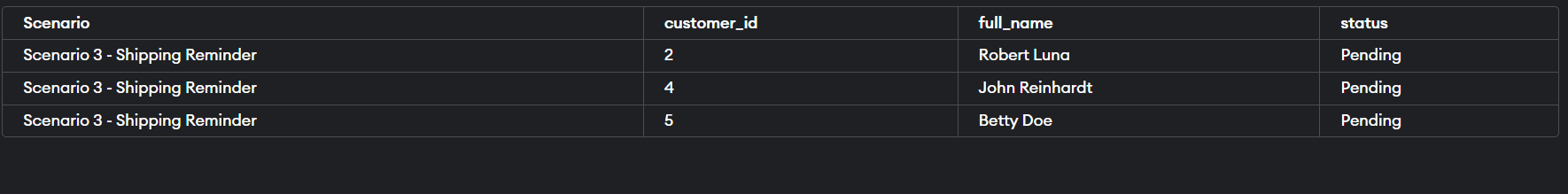
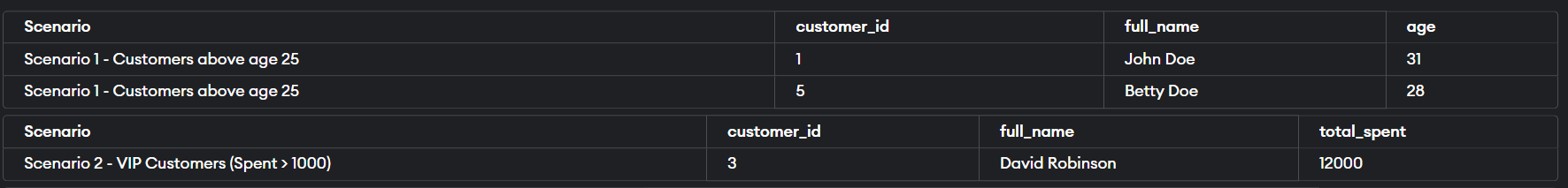
FROM Shippings S

JOIN Customers C ON S.customer = C.customer\_id

WHERE S.status = 'Pending';

**Output:**





**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:**-- Drop tables if they already exist

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE SavingsAccounts';

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- 1. Create SavingsAccounts table

CREATE TABLE SavingsAccounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER(10,2)

);

-- 2. Create Employees table

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(50),

Salary NUMBER(10,2)

);

-- 3. Create Accounts table (for fund transfer)

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER(10,2)

);

-- Insert sample data for SavingsAccounts

INSERT INTO SavingsAccounts VALUES (101, 1, 5000.00);

INSERT INTO SavingsAccounts VALUES (102, 2, 3000.00);

INSERT INTO SavingsAccounts VALUES (103, 3, 10000.00);

-- Insert sample data for Employees

INSERT INTO Employees VALUES (1, 'Alice', 'IT', 60000);

INSERT INTO Employees VALUES (2, 'Bob', 'HR', 50000);

INSERT INTO Employees VALUES (3, 'Charlie', 'IT', 70000);

-- Insert sample data for Accounts (fund transfer)

INSERT INTO Accounts VALUES (101, 1, 8000.00);

INSERT INTO Accounts VALUES (202, 2, 2500.00);

COMMIT;

**-- Scenario 1: Monthly Interest on SavingsAccounts**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (SELECT AccountID, Balance FROM SavingsAccounts) LOOP

UPDATE SavingsAccounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountID = acc.AccountID;

END LOOP;

COMMIT;

END;

/

-- Call Scenario 1 Procedure

BEGIN

ProcessMonthlyInterest;

END;

/

-- Check updated balances

SELECT \* FROM SavingsAccounts;

**-- Scenario 2: Bonus for Employees in a Department**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_name IN VARCHAR2,

bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = dept\_name;

COMMIT;

END;

/

-- Call Scenario 2 Procedure (e.g. 10% bonus to IT dept)

BEGIN

UpdateEmployeeBonus('IT', 10);

END;

/

-- Check updated salaries

SELECT \* FROM Employees;

**-- Scenario 3: Transfer Funds Between Accounts**

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_account IN NUMBER,

to\_account IN NUMBER,

amount IN NUMBER

) IS

from\_balance NUMBER;

BEGIN

SELECT Balance INTO from\_balance

FROM Accounts

WHERE AccountID = from\_account;

IF from\_balance < amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance.');

END IF;

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = from\_account;

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = to\_account;

COMMIT;

END;

/

-- Call Scenario 3 Procedure (transfer $1000 from 101 to 202)

BEGIN

TransferFunds(101, 202, 1000);

END;

/

-- Check updated account balances

SELECT \* FROM Accounts;

**Output:**ACCOUNTID CUSTOMERID BALANCE

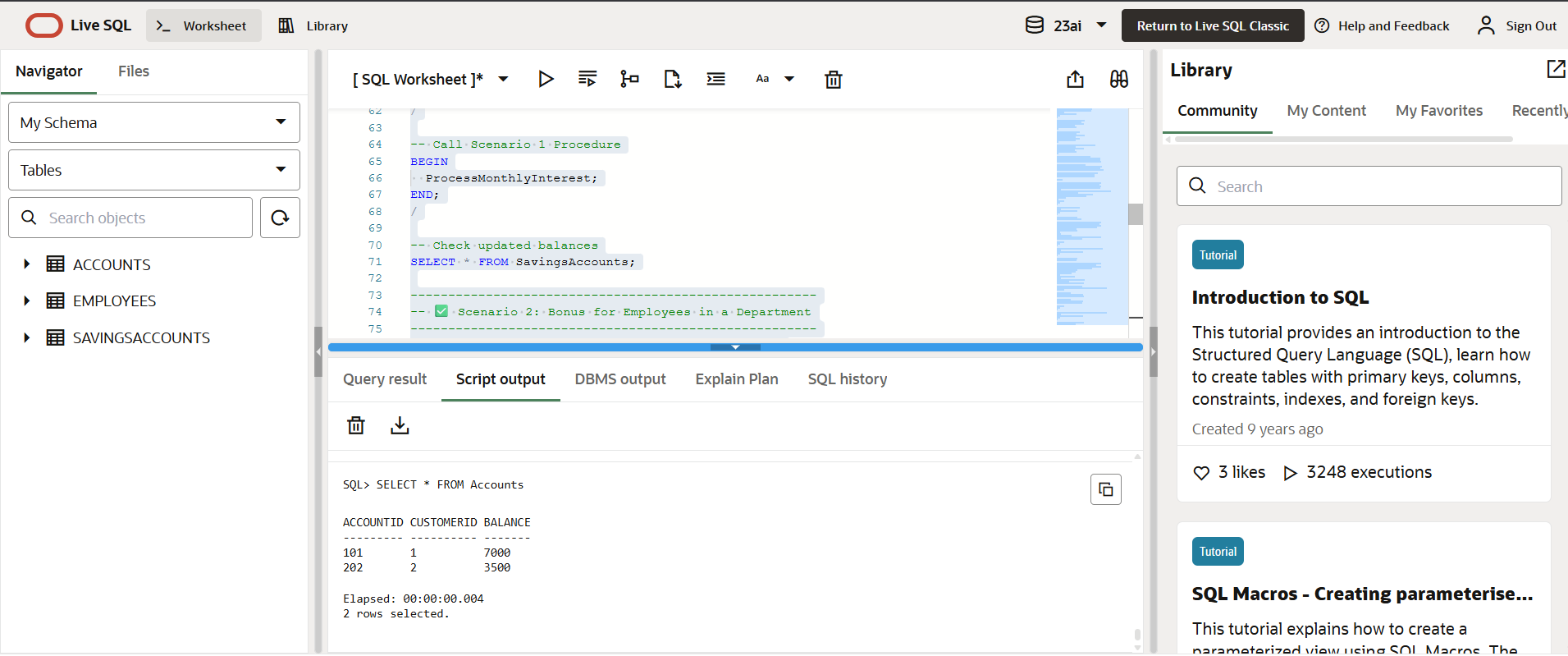
--------- ---------- -------

101 1 7000

202 2 3500

**Elapsed: 00:00:00.004**

**2 rows selected.**

****

# **JUnit Basic Testing**

**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:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

3. Create a new test class in your project

project.

**Code:  
Pom.xml file:**

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>demo</groupId>

<artifactId>demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- ✅ JUnit 4 for Unit Testing -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- ✅ Maven Compiler Plugin -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Mytest.java:**

package demo;

import org.junit.Test;

import static org.junit.Assert.\*;

public class MyTest {

*@Test*

public void testAddition() {

int a = 2;

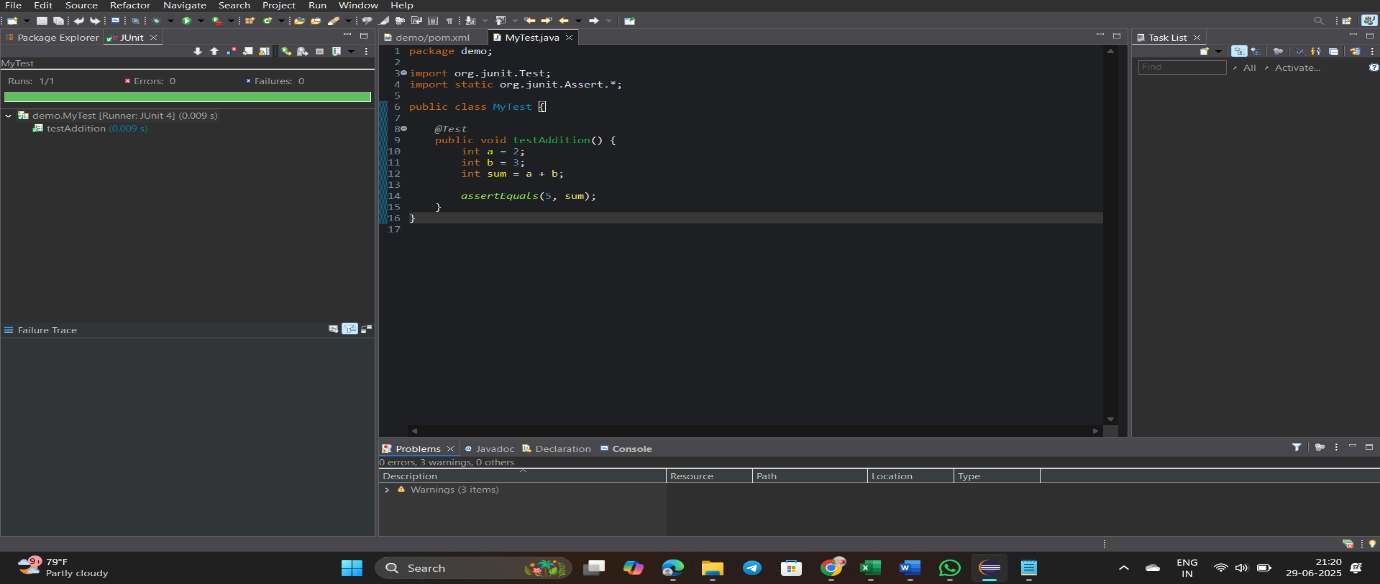
int b = 3;

int sum = a + b;

*assertEquals*(5, sum);

}

}

**Output: **

**Exercise 3: Assertions in JUnit**

Scenario:

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

Steps:

1. Write tests using various JUnit assertions.

Solution Code:

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());

}

}

**Code:  
AssertionsTest.java**

package demo;

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);

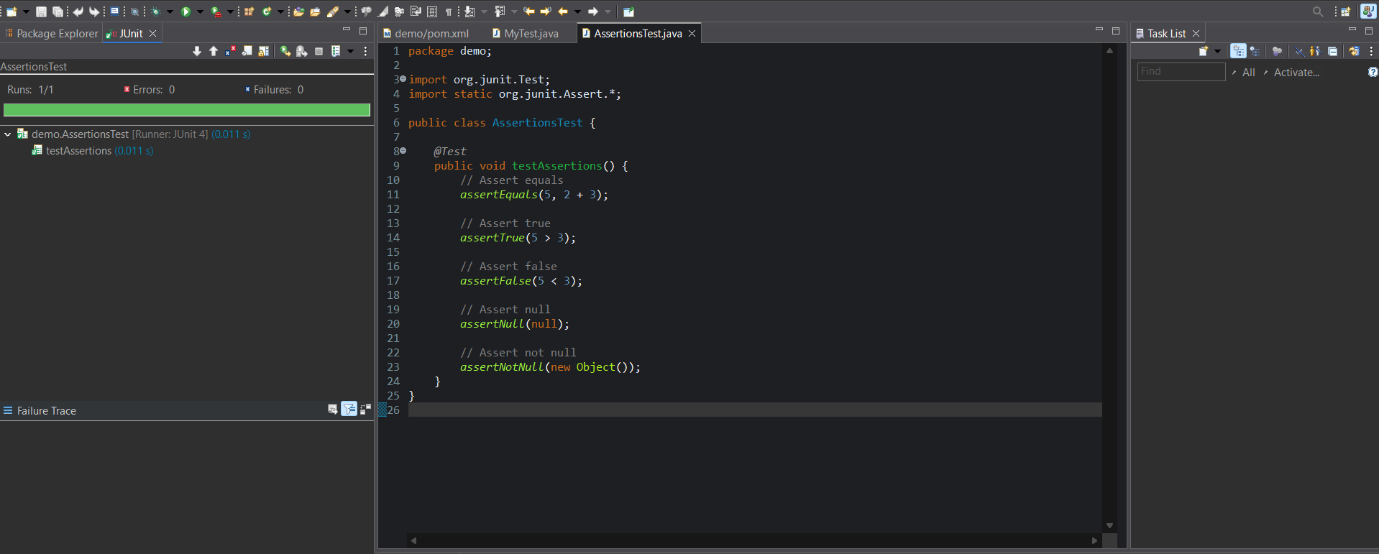
// Assert not null

*assertNotNull*(new Object());

}

}

**Output:**

****

**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.

Steps:

1. Write tests using the AAA pattern.

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

**Code:  
Calculator.java**

package demo;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a – b;

}

}

**CalculatorTest.java**

package demo;

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 eard() {

System.***out***.println(“Setting up…”);

calculator = new Calculator();

}

*@After*

public void eardown() {

System.***out***.println(“Cleaning up…”);

calculator = null;

}

*@Test*

public void testAddition() {

int result = calculator.add(2, 3);

*assertEquals*(5, result);

}

*@Test*

public void testSubtraction() {

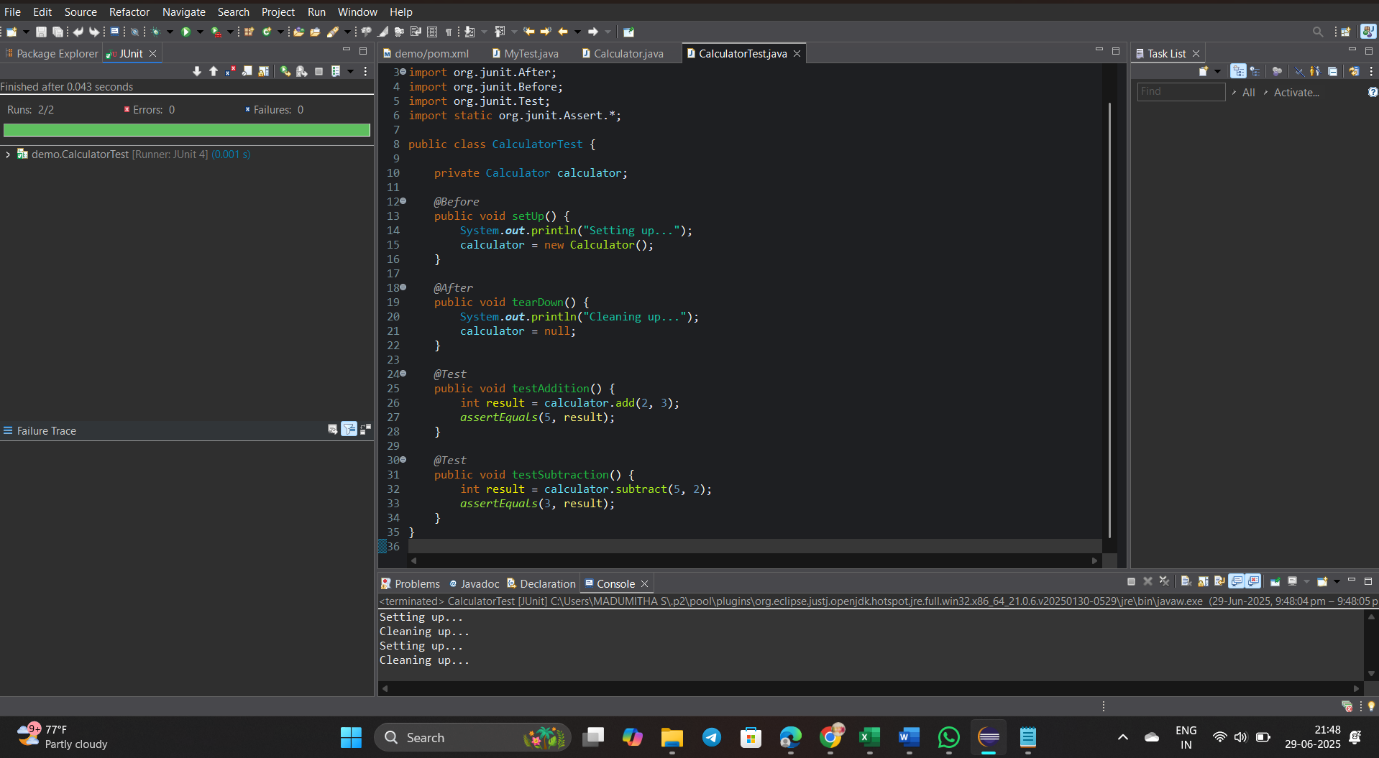
int result = calculator.subtract(5, 2);

*assertEquals*(3, result);

}

}

**Output:**

****

**3. Mockito 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.

Solution Code:

import static org.mockito.Mockito.\*;

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);

}

}

**Code:  
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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>demo</groupId>

<artifactId>demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<!-- ✅ JUnit 5 API -->

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- ✅ JUnit Platform Engine -->

<dependency>

<groupId>org.junit.platform</groupId>

<artifactId>junit-platform-engine</artifactId>

<version>1.10.0</version>

<scope>test</scope>

</dependency>

<!-- ✅ JUnit Platform Launcher -->

<dependency>

<groupId>org.junit.platform</groupId>

<artifactId>junit-platform-launcher</artifactId>

<version>1.10.0</version>

<scope>test</scope>

</dependency>

<!-- ✅ Mockito for mocking -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version>

<scope>test</scope>

</dependency>

<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>

<!-- ✅ Compiler plugin -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<!-- ✅ Surefire plugin for JUnit 5 test running -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0-M9</version>

</plugin>

</plugins>

</build>

</project>

**ExternalApi.java**

package demo;

public interface ExternalApi {

String getData();

}

**MyService.java**

package demo;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

**package demo;**

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

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

*@Test*

public void testFetchData() {

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

*when*(mockApi.getData()).thenReturn("Mock Data");

MyService service = new MyService(mockApi);

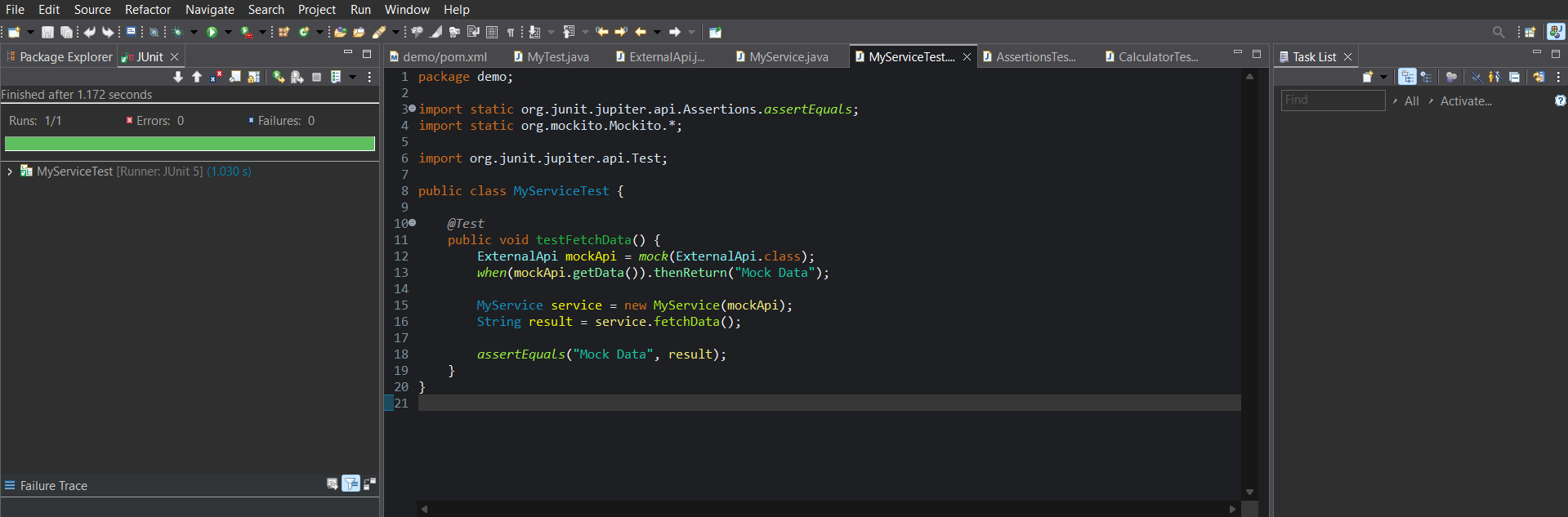
String result = service.fetchData();

*assertEquals*("Mock Data", result);

}

}

**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.

Solution Code:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import 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();

}

}

**Code:  
MyService.java**

package demo;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**ExternalApi.java**

package demo;

public interface ExternalApi {

String getData();

}

**MyServiceTest.java**

package demo;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

*@Test*

public void testVerifyInteraction() {

ExternalApi mockApi = *mock*(ExternalApi.class); // Step 1

MyService service = new MyService(mockApi);

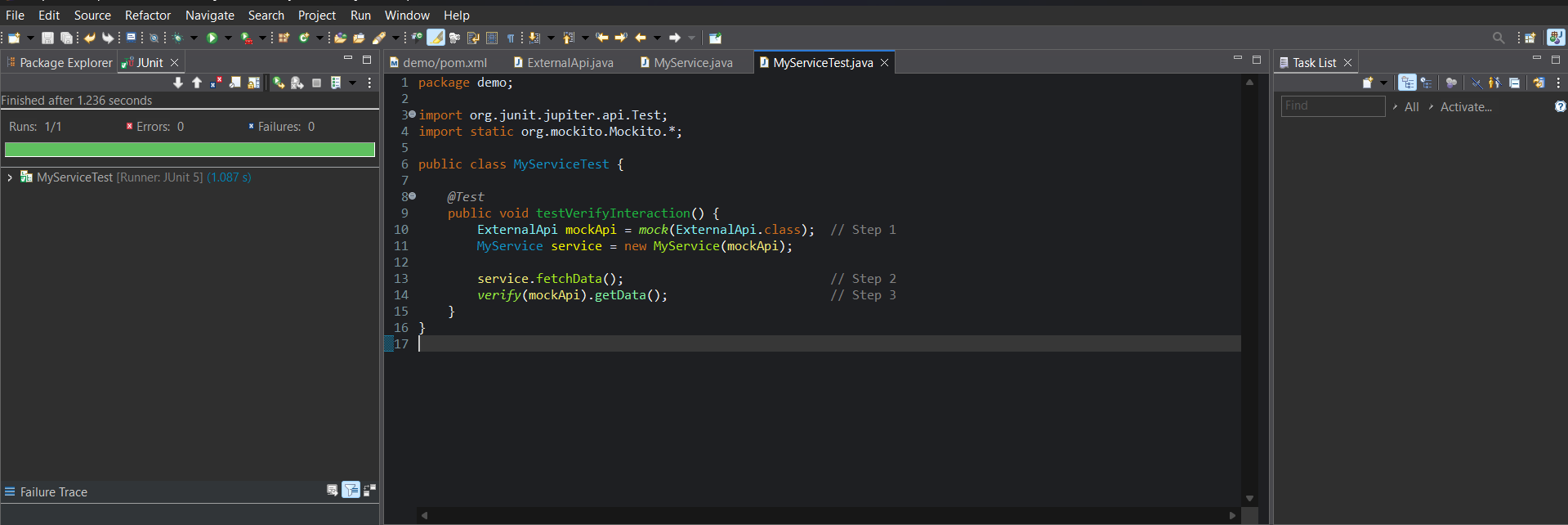
service.fetchData(); // Step 2

*verify*(mockApi).getData(); // Step 3

}

}

**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.

Step-by-Step Solution:

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

<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>

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

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");

}

}

**Code:  
LoggingExample.java**

package demo;

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");

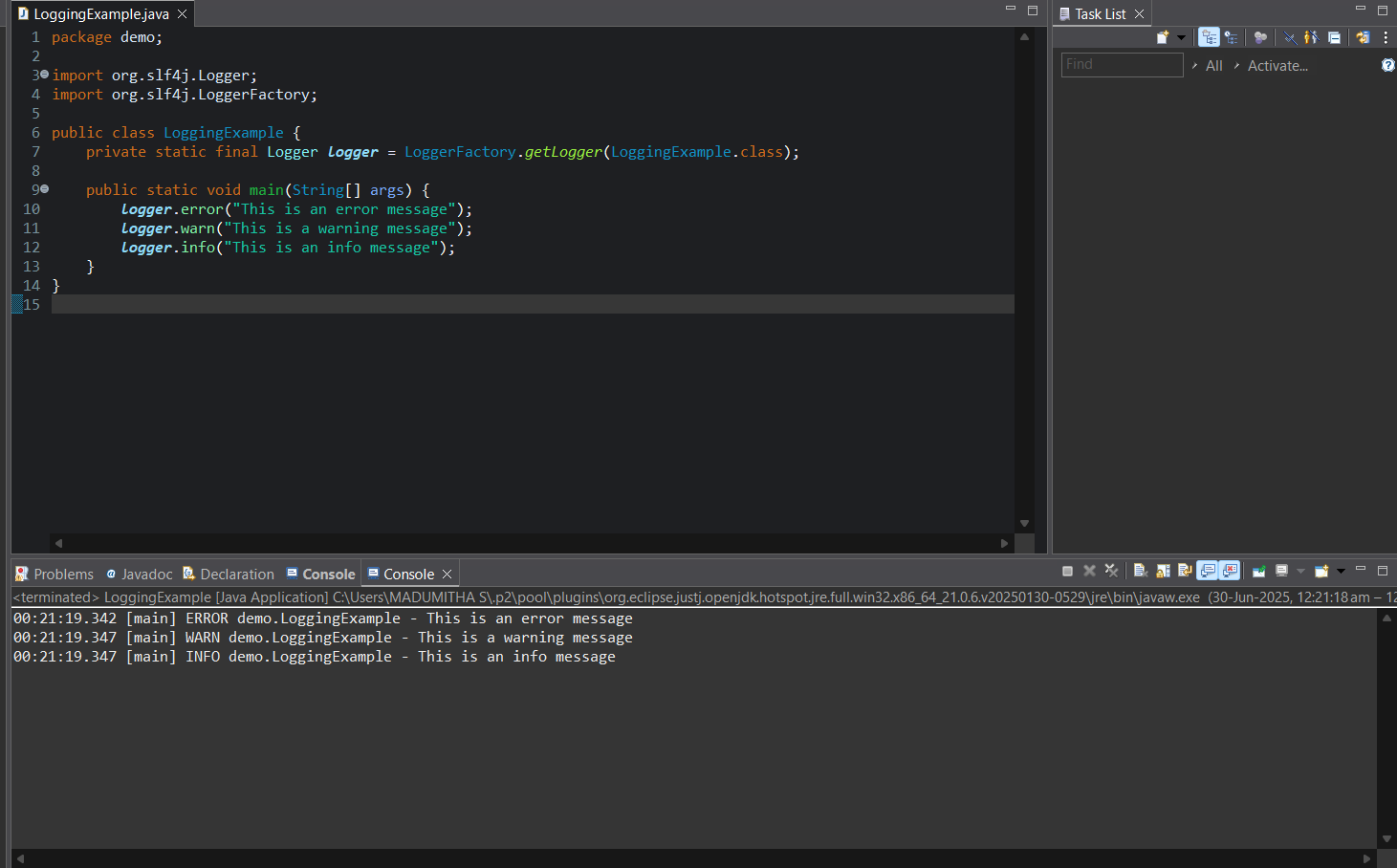
***logger***.warn("This is a warning message");

***logger***.info("This is an info message");

}

}

**Output:**

****