WEEK 2 : COGNIZANT DN 4.0 FSE HANDS-ON EXERCISES

* **PL/SQL Exercises:**

**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.

**Solution:**

**-- Scenario 1: Apply 1% interest discount for customers above 60 years**

BEGIN

FOR cust\_rec IN (

SELECT CustomerID, Name, DOB FROM Customers

) LOOP

DECLARE

v\_age NUMBER;

BEGIN

v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, cust\_rec.DOB) / 12);

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Discount applied for customer: ' || cust\_rec.Name);

END IF;

END;

END LOOP;

END;

**-- Scenario 2: Set IsVIP = TRUE for customers with balance > 10,000**

BEGIN

FOR customer\_rec IN (

SELECT CustomerID, Balance FROM Customers

) LOOP

IF customer\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = customer\_rec.CustomerID;

END IF;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE('VIP status updated.');

END;

/

**-- Scenario 3: Print reminder messages for loans due in 30 days**

BEGIN

FOR loan\_rec IN (

SELECT l.LoanID, l.CustomerID, l.EndDate, c.Name

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan\_rec.LoanID ||

' for customer ' || loan\_rec.Name ||

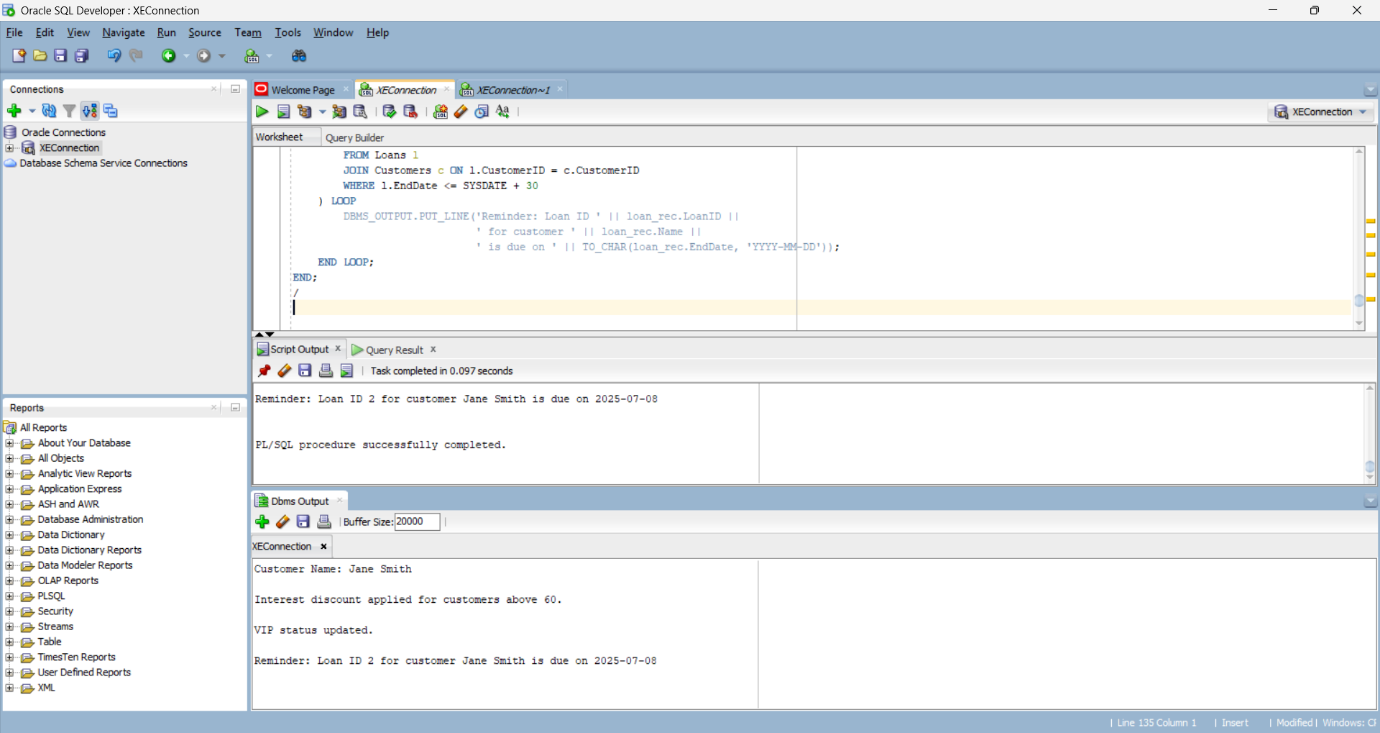
' is due on ' || TO\_CHAR(loan\_rec.EndDate, 'YYYY-MM-DD'));

END LOOP;

END;

/

**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.

**Solution:**

**-- Scenario 1: ProcessMonthlyInterest Procedure**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to savings accounts.');

END;

/

BEGIN

ProcessMonthlyInterest;

END;

/

**-- Scenario 2: UpdateEmployeeBonus Procedure**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department IN VARCHAR2,

p\_BonusPercent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* (p\_BonusPercent / 100))

WHERE Department = p\_Department;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to department: ' || p\_Department);

END;

/

**-- Scenario 3: TransferFunds Procedure**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_SourceAccount IN NUMBER,

p\_TargetAccount IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_SourceBalance NUMBER;

BEGIN

SELECT Balance INTO v\_SourceBalance

FROM Accounts

WHERE AccountID = p\_SourceAccount;

IF v\_SourceBalance < p\_Amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_SourceAccount;

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_TargetAccount;

DBMS\_OUTPUT.PUT\_LINE('Transfer completed.');

END;

/

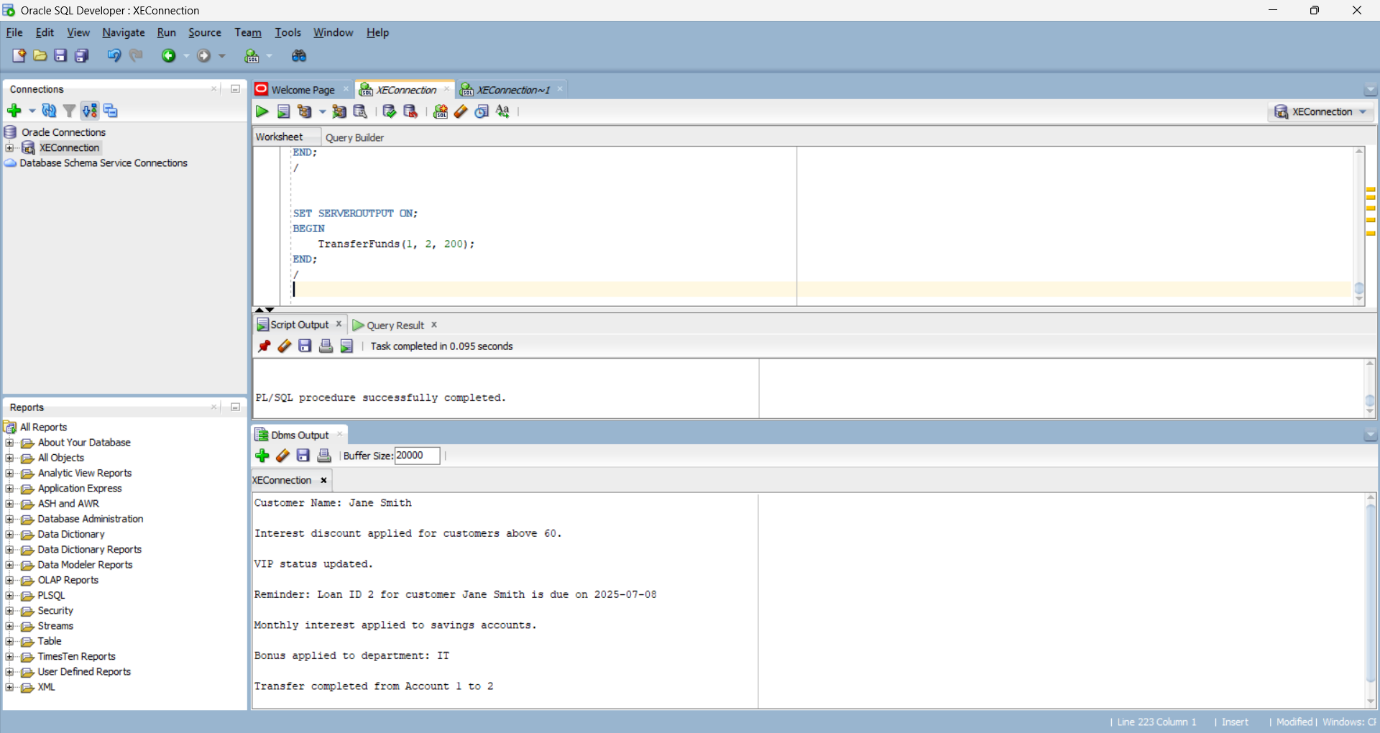
BEGIN

TransferFunds(1, 2, 200);

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:**

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.

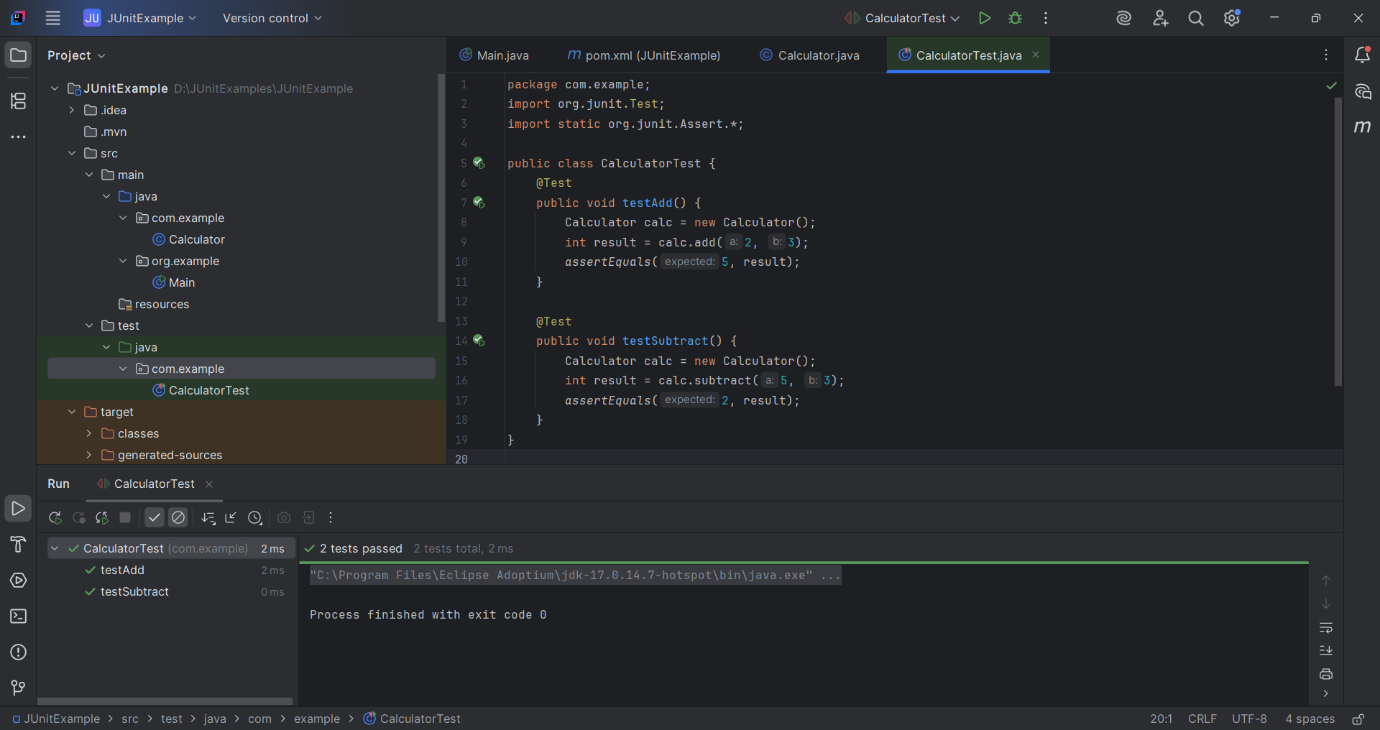
**Solution:**

**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;  
 }  
}

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

**Output:**

****

**Exercise 3: Assertions in Junit**

**Scenario:**

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

**Steps:**

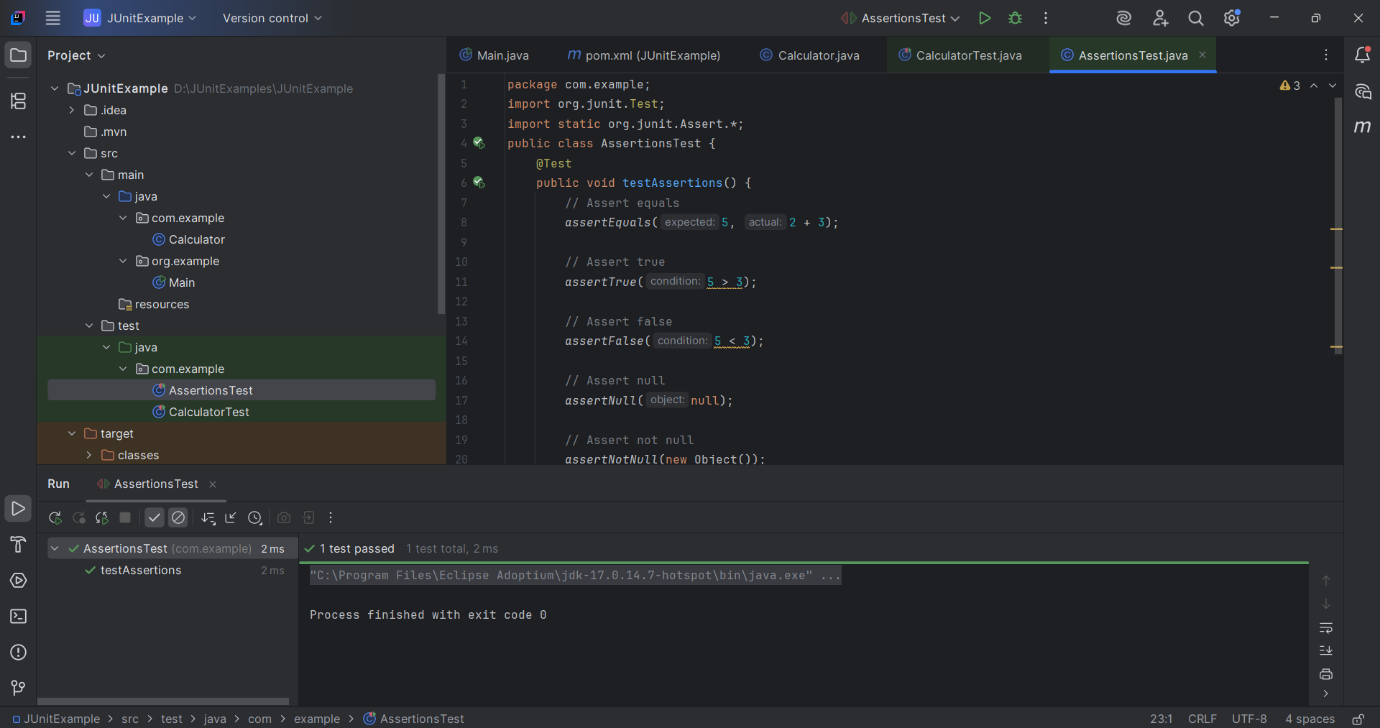
1. Write tests using various JUnit assertions.

**Solution:**

**AssertionsTest.java**

package com.example;  
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.

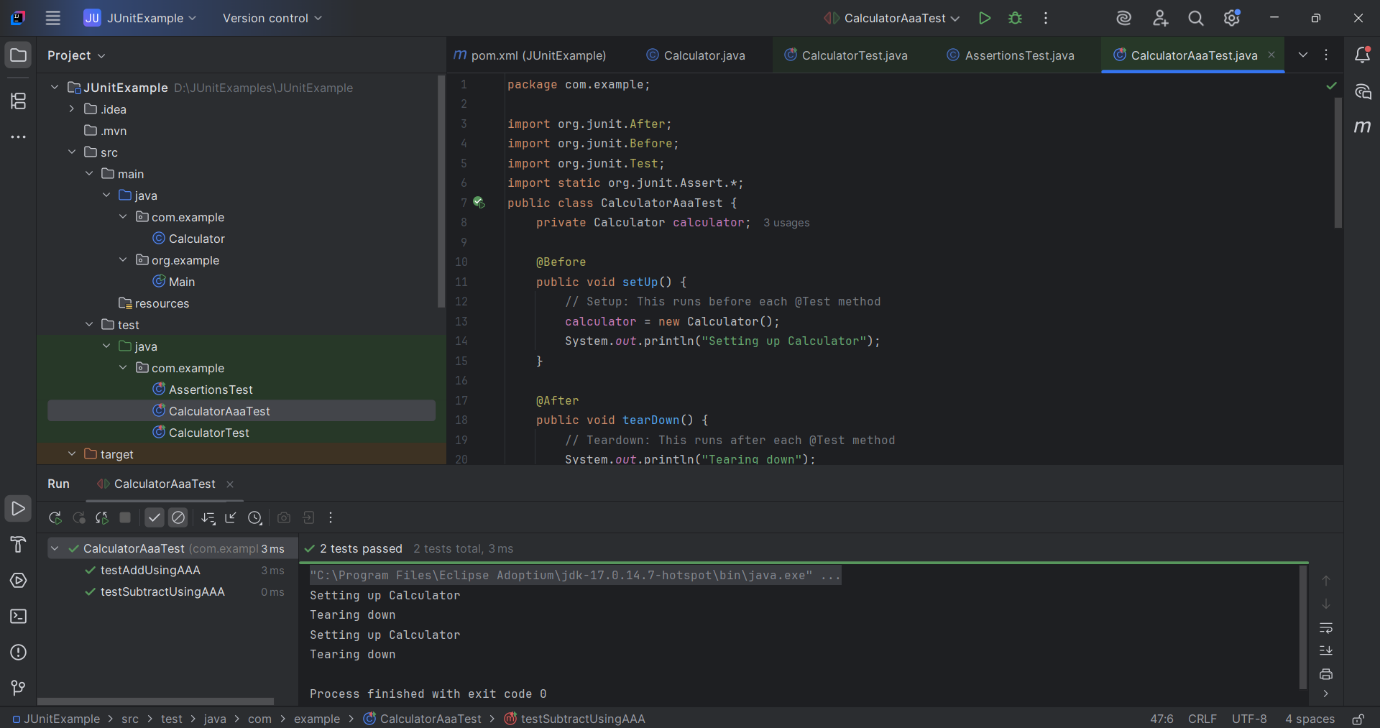
2. Use @Before and @After annotations for setup and teardown methods.

**Solution:**

**CalculatorAaaTest.java**

package com.example;  
  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
public class CalculatorAaaTest {  
 private Calculator calculator;  
  
 @Before  
 public void setUp() {  
 // Setup: This runs before each @Test method  
 calculator = new Calculator();  
 System.*out*.println("Setting up Calculator");  
 }  
  
 @After  
 public void tearDown() {  
 // Teardown: This runs after each @Test method  
 System.*out*.println("Tearing down");  
 }  
  
 @Test  
 public void testAddUsingAAA() {  
 // Arrange  
 int a = 2;  
 int b = 3;  
  
 // Act  
 int result = calculator.add(a, b);  
  
 // Assert  
 *assertEquals*(5, result);  
 }  
  
 @Test  
 public void testSubtractUsingAAA() {  
 // Arrange  
 int a = 10;  
 int b = 4;  
  
 // Act  
 int result = calculator.subtract(a, b);  
  
 // Assert  
 *assertEquals*(6, 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:**

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:**

**ExternalApi.java**

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

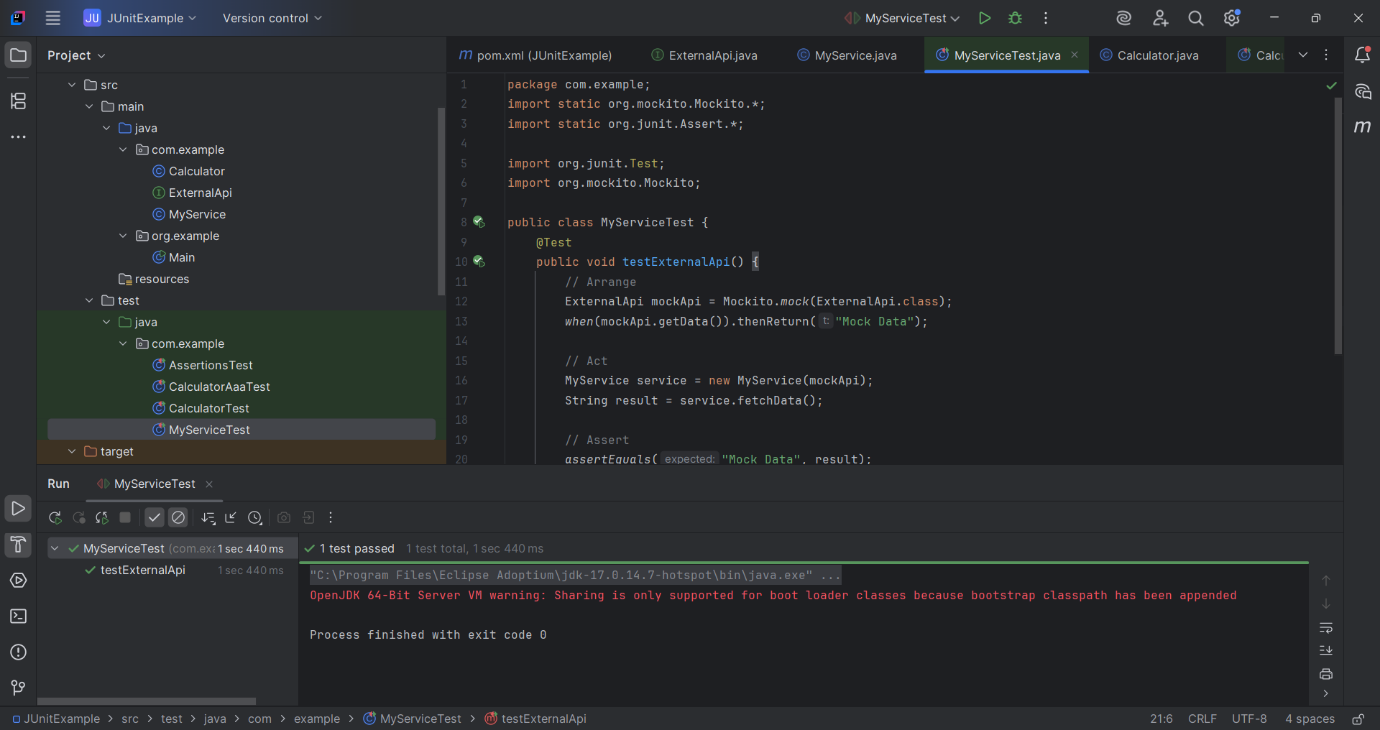
**MyService.java**

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

**MyServiceTest.java**

package com.example;  
import static org.mockito.Mockito.\*;  
import static org.junit.Assert.\*;  
  
import org.junit.Test;  
import org.mockito.Mockito;  
  
public class MyServiceTest {  
 @Test  
 public void testExternalApi() {  
 // Arrange  
 ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);  
 *when*(mockApi.getData()).thenReturn("Mock Data");  
  
 // Act  
 MyService service = new MyService(mockApi);  
 String result = service.fetchData();  
  
 // Assert  
 *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.

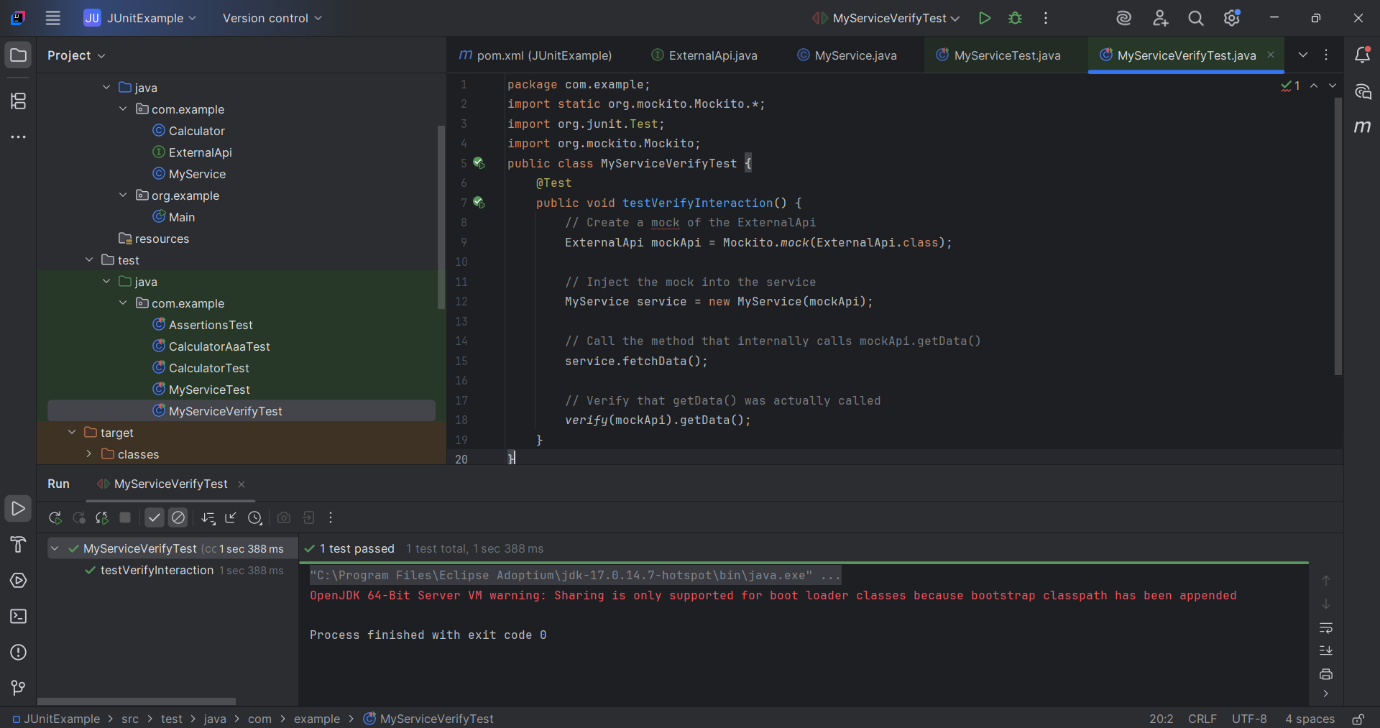
3. Verify the interaction.

**Solution:**

**MyServiceVerifyTest.java**

package com.example;  
import static org.mockito.Mockito.\*;  
import org.junit.Test;  
import org.mockito.Mockito;  
public class MyServiceVerifyTest {  
 @Test  
 public void testVerifyInteraction() {  
 // Create a mock of the ExternalApi  
 ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);  
  
 // Inject the mock into the service  
 MyService service = new MyService(mockApi);  
  
 // Call the method that internally calls mockApi.getData()  
 service.fetchData();  
  
 // Verify that getData() was actually called  
 *verify*(mockApi).getData();  
 }  
}

**Output:**

****

* **Logging using SLF4J:**

**Exercise 1: Logging Error Messages and Warning Levels**

**Task:**

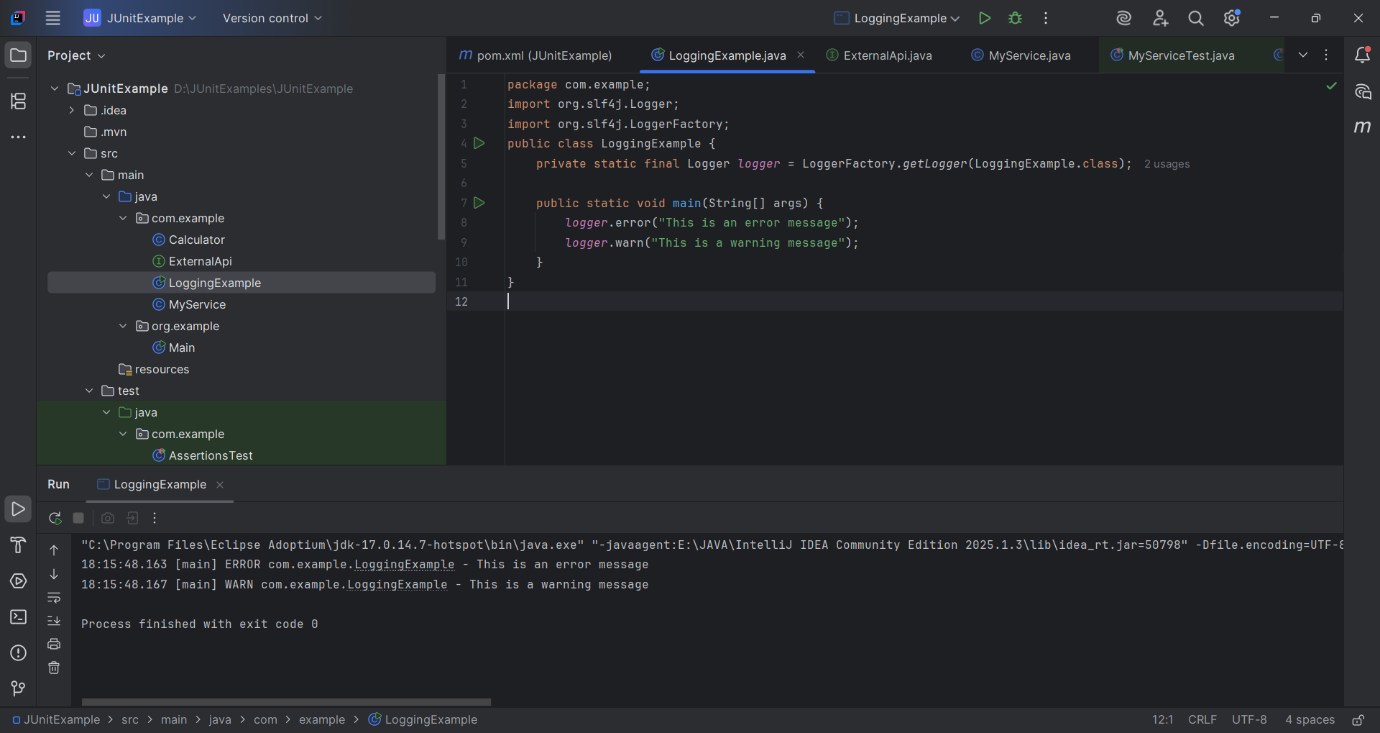
Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

**Solution:**

**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:**

****