**1)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.

-- Create Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

-- Create Accounts table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Create Transactions table

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

-- Create Loans table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Create Employees table

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

-- Insert sample customers

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

-- Senior citizen with high balance

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (3, 'David Raja', TO\_DATE('1954-06-01', 'YYYY-MM-DD'), 15000, SYSDATE);

-- Insert accounts

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

-- Insert transactions

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

-- Insert loans

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60)); -- Not due soon

-- Loan ending soon for David

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 3, 10000, 7, SYSDATE, SYSDATE + 10);

-- Insert employees

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

-- Enable output

SET SERVEROUTPUT ON;

-- Q1: Discount 1% for senior citizens (Age > 60)

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Q1: Discount for Senior Citizens ---');

FOR rec IN (

SELECT c.CustomerID, l.LoanID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE MONTHS\_BETWEEN(SYSDATE, c.DOB)/12 > 60

) LOOP

UPDATE Loans

SET InterestRate = rec.InterestRate - 1

WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Discount applied to Customer ID ' || rec.CustomerID);

END LOOP;

END;

/

-- Q2: VIP Check – Display only

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Q2: VIP Flag for High Balance Customers ---');

FOR rec IN (

SELECT CustomerID, Balance FROM Customers

) LOOP

IF rec.Balance > 10000 THEN

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || rec.CustomerID || ' has VIP balance: ₹' || rec.Balance);

END IF;

END LOOP;

END;

/

-- Q3: Reminders for loans due in next 30 days

BEGIN

DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Q3: Loan Due Reminders ---');

FOR rec IN (

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

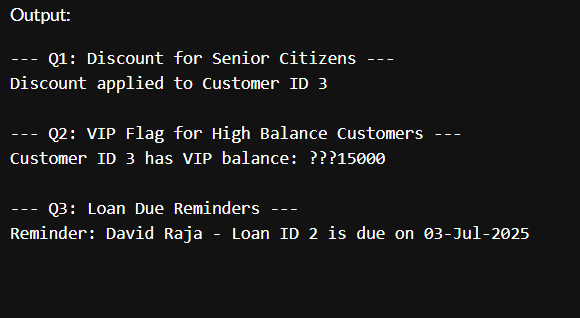
DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || rec.Name || ' - Loan ID ' || rec.LoanID ||

' is due on ' || TO\_CHAR(rec.EndDate, 'DD-Mon-YYYY'));

END LOOP;

END;

/



**2)EXERCISE 3: STORED PRODUCERS(PLSQL)**

* **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.

-- Create Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

-- Create Accounts table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Create Transactions table

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

-- Create Loans table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Create Employees table

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

-- Insert sample customers

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

-- Senior citizen with high balance

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (3, 'David Raja', TO\_DATE('1954-06-01', 'YYYY-MM-DD'), 15000, SYSDATE);

-- Insert accounts

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

-- Insert transactions

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

-- Insert loans

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60)); -- Not due soon

-- Loan ending soon for David

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 3, 10000, 7, SYSDATE, SYSDATE + 10);

-- Insert employees

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

-- Enable output display

SET SERVEROUTPUT ON;

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

-- Procedure 1: ProcessMonthlyInterest

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

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

) LOOP

UPDATE Accounts

SET Balance = acc.Balance + (acc.Balance \* 0.01)

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest added for Account ID ' || acc.AccountID);

END LOOP;

END;

/

-- Run Procedure 1

EXEC ProcessMonthlyInterest;

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

-- Procedure 2: UpdateEmployeeBonus

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

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept\_name IN VARCHAR2,

bonus\_pct IN NUMBER

) AS

BEGIN

FOR emp IN (

SELECT EmployeeID, Salary

FROM Employees

WHERE Department = dept\_name

) LOOP

UPDATE Employees

SET Salary = emp.Salary + (emp.Salary \* bonus\_pct / 100)

WHERE EmployeeID = emp.EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to Employee ID ' || emp.EmployeeID);

END LOOP;

END;

/

-- Run Procedure 2 (example: 10% bonus to IT department)

EXEC UpdateEmployeeBonus('IT', 10);

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

-- Procedure 3: TransferFunds

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

CREATE OR REPLACE PROCEDURE TransferFunds(

from\_acc IN NUMBER,

to\_acc IN NUMBER,

amount IN NUMBER

) AS

from\_balance NUMBER;

BEGIN

-- Get balance of source account

SELECT Balance INTO from\_balance

FROM Accounts

WHERE AccountID = from\_acc;

IF from\_balance < amount THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient balance.');

ELSE

-- Deduct amount from source account

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = from\_acc;

-- Add amount to destination account

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = to\_acc;

DBMS\_OUTPUT.PUT\_LINE(amount || ' transferred from Account ' || from\_acc || ' to Account ' || to\_acc);

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Account not found.');

WHEN OTHERS THEN

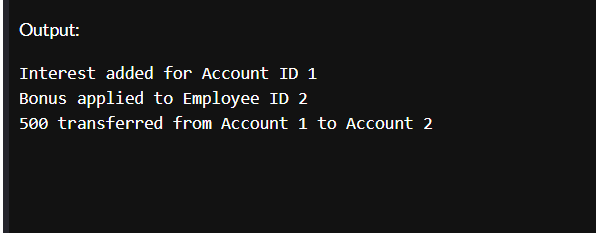
DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);

END;

/

-- Run Procedure 3 (example: transfer 500 from Account 1 to 2)

EXEC TransferFunds(1, 2, 500);



**3) EXERCISE 1: SETTING UP JUNIT(1. JUnit\_Basic Testing Exercise)**

* 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: junit junit 4.13.2 test
* 3. Create a new test class in your project.

**Code:**

public class MathHelper {

public boolean isEven(int number) {

return number % 2 == 0;

}

public boolean isOdd(int number) {

return number % 2 != 0;

}

public boolean isPositive(int number) {

return number > 0;

}

public boolean isNegative(int number) {

return number < 0;

}

public int square(int number) {

return number \* number;

}

}

**Test Code:**

import org.junit.Test;

import static org.junit.Assert.\*;

public class MathHelperTest {

@Test

public void testIsEven() {

MathHelper helper = new MathHelper();

assertTrue(helper.isEven(6));

assertFalse(helper.isEven(7));

}

@Test

public void testIsOdd() {

MathHelper helper = new MathHelper();

assertTrue(helper.isOdd(3));

assertFalse(helper.isOdd(8));

}

@Test

public void testIsPositive() {

MathHelper helper = new MathHelper();

assertTrue(helper.isPositive(5));

assertFalse(helper.isPositive(-2));

}

@Test

public void testIsNegative() {

MathHelper helper = new MathHelper();

assertTrue(helper.isNegative(-10));

assertFalse(helper.isNegative(3));

}

@Test

public void testSquare() {

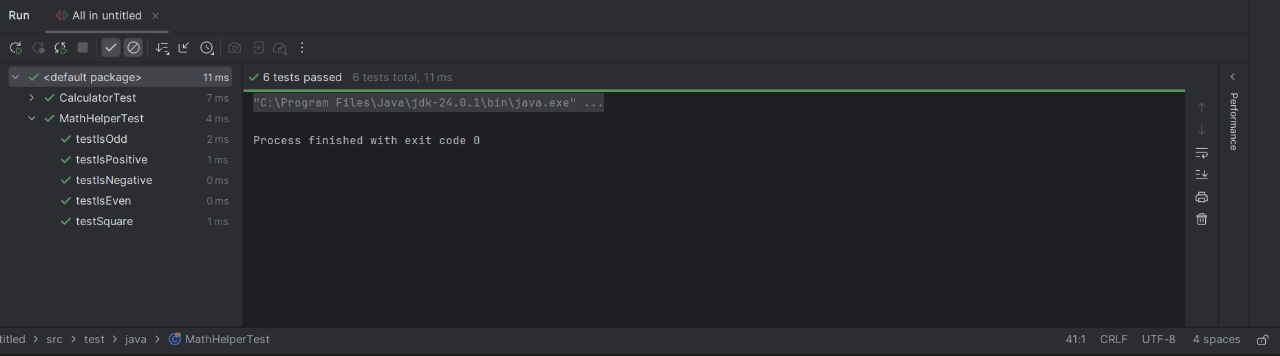
MathHelper helper = new MathHelper();

assertEquals(25, helper.square(5));

assertEquals(4, helper.square(2));

}

}.



**4) Exercise 3: Assertions in JUnit (1. JUnit\_Basic Testing Exercise)**

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

Steps: 1. Write tests using various JUnit assertions.

**Test Code:**

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

// Assert equals

assertEquals("Sum of 2 + 3 must be 5", 5, 2 + 3);

// Assert true

assertTrue("5 is greater than 3", 5 > 3);

// Assert false

assertFalse("5 is not less than 3", 5 < 3);

// Assert null

Object obj1 = null;

assertNull("Object should be null", obj1);

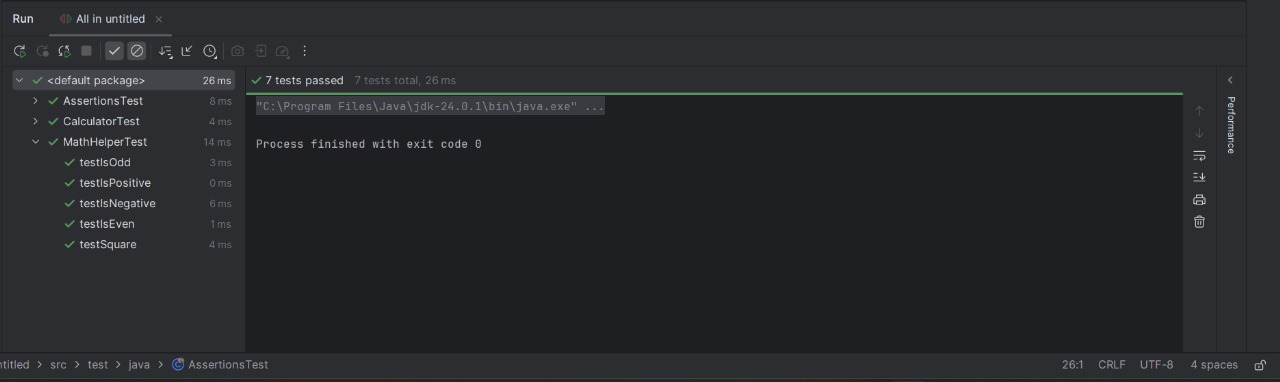
// Assert not null

Object obj2 = new Object();

assertNotNull("Object should not be null", obj2);

}

}



**5)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:**

public class BankAccount {

private int balance;

public BankAccount() {

this.balance = 0;

}

public void deposit(int amount) {

if (amount > 0) {

balance += amount;

}

}

public void withdraw(int amount) {

if (amount > 0 && amount <= balance) {

balance -= amount;

}

}

public int getBalance() {

return balance;

}

}

**Testcode:**

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class BankAccountTest {

private BankAccount account;

// Setup method – runs before each test

@Before

public void setUp() {

account = new BankAccount();

System.out.println("🔧 Bank account setup");

}

// Teardown method – runs after each test

@After

public void tearDown() {

account = null;

System.out.println("🧹 Bank account cleared");

}

// Test deposit using AAA pattern

@Test

public void testDeposit() {

// Arrange

int depositAmount = 100;

// Act

account.deposit(depositAmount);

// Assert

assertEquals(100, account.getBalance());

}

// Test withdraw using AAA pattern

@Test

public void testWithdraw() {

// Arrange

account.deposit(200);

// Act

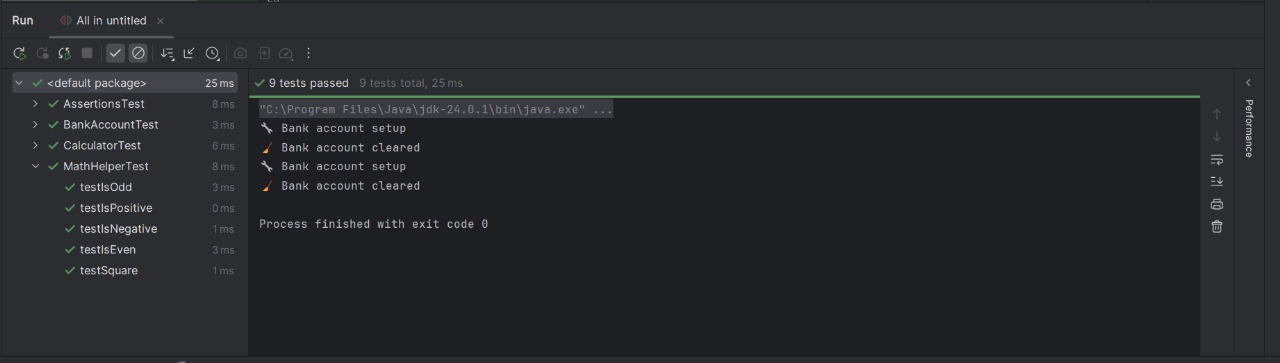
account.withdraw(50);

// Assert

assertEquals(150, account.getBalance());

}

}



**6)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.

**External API:**

public interface ExternalApi {

String getData();

}

**Code:**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**Test code:**

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() {

// Step 1: Create mock of ExternalApi

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

// Step 2: Stub the method getData()

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

// Step 3: Use mock in MyService

MyService service = new MyService(mockApi);

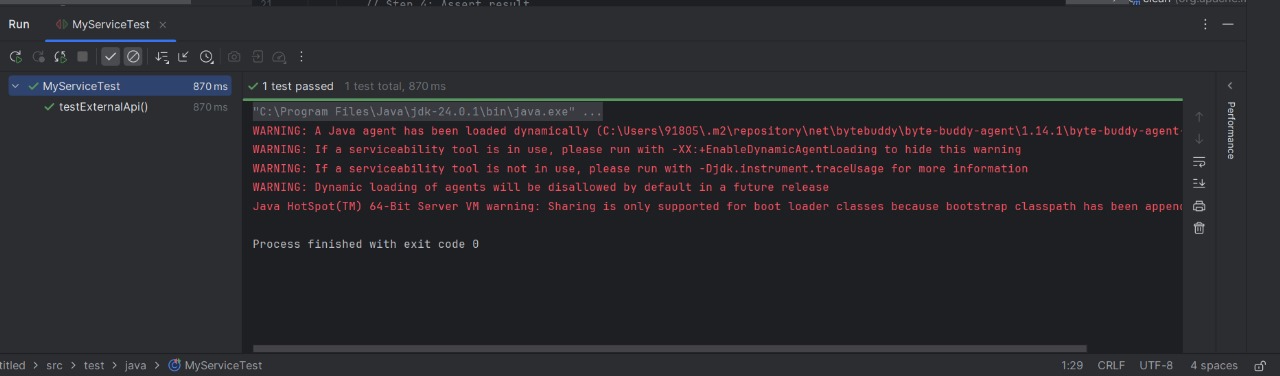
String result = service.fetchData();

// Step 4: Assert result

assertEquals("Mock Data", result);

}

}



**7)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.

**TestCode:**

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Step 1: Create mock

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

// Step 2: Create service and call method

MyService service = new MyService(mockApi);

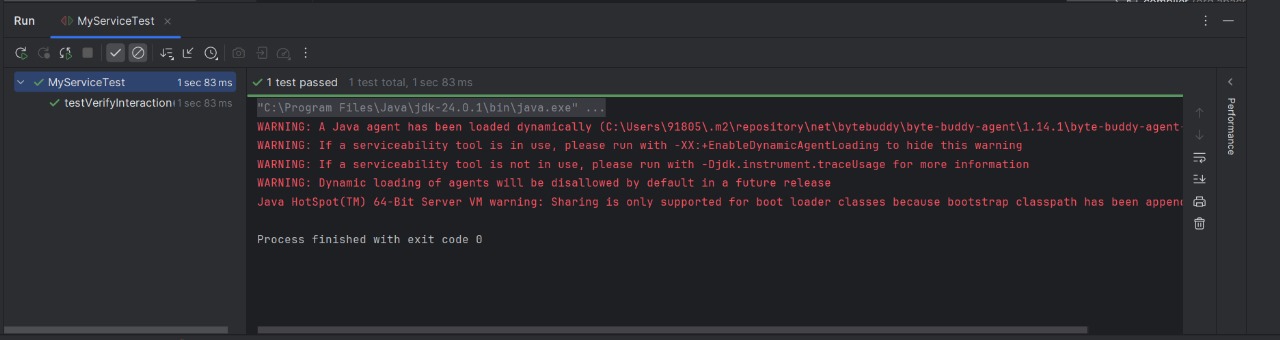
service.fetchData(); // this internally calls mockApi.getData()

// Step 3: Verify that mockApi.getData() was called

verify(mockApi).getData();

}

}



**8)Exercise 1: Logging Error Messages and Warning Levels Task**

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

**code:**

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

// Create logger

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");

}

}

