**WEEK-2 JAVA FSE HANDS-ON**

**B.PRASANTH**

**6382932**

**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:**

CREATE TABLE CUSTOMERS (

    CUSTOMER\_ID   NUMBER PRIMARY KEY,

    NAME          VARCHAR2(50),

    AGE           NUMBER,

    BALANCE       NUMBER,

    INTEREST\_RATE NUMBER,

    ISVIP         VARCHAR2(5)

);

CREATE TABLE LOANS (

    LOAN\_ID     NUMBER PRIMARY KEY,

    CUSTOMER\_ID NUMBER,

    DUE\_DATE    DATE,

    FOREIGN KEY ( CUSTOMER\_ID )

        REFERENCES CUSTOMERS ( CUSTOMER\_ID )

);

INSERT INTO customers VALUES (1, 'Alice',   65, 12000, 9.5, 'FALSE');

INSERT INTO customers VALUES (2, 'Bob',     58, 8000,  10, 'FALSE');

INSERT INTO customers VALUES (3, 'Charlie', 70, 15000, 8.5, 'FALSE');

INSERT INTO customers VALUES (4, 'Diana',   45, 4000,  11, 'FALSE');

INSERT INTO loans VALUES (101, 1, SYSDATE + 10);

INSERT INTO loans VALUES (102, 2, SYSDATE + 40);

INSERT INTO loans VALUES (103, 3, SYSDATE + 5);

INSERT INTO loans VALUES (104, 4, SYSDATE + 60);

COMMIT;

**SCENARIO 1:**

BEGIN

  FOR customer\_rec IN (

    SELECT customer\_id, interest\_rate

    FROM customers

    WHERE age > 60)

    LOOP

    UPDATE customers

    SET interest\_rate = interest\_rate - 1

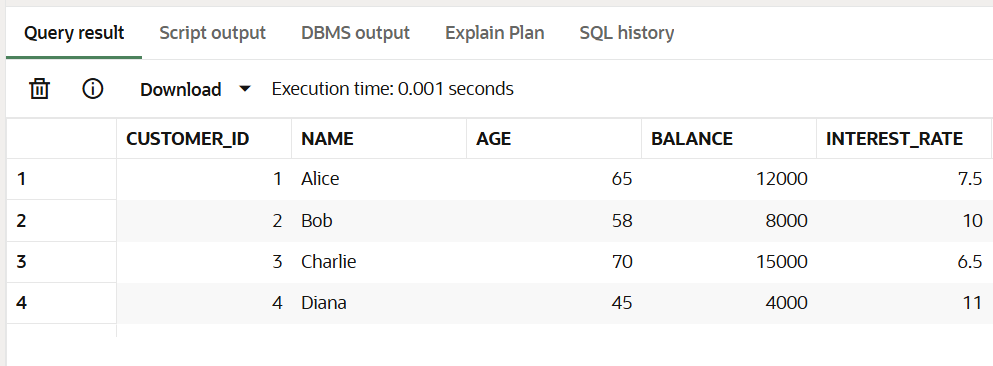
    WHERE customer\_id = customer\_rec.customer\_id;

  END LOOP;

  COMMIT;

END;

SELECT \* FROM customers;

**OUTPUT:**

**SCENARIO2:**

BEGIN

  FOR customer\_rec IN (

    SELECT customer\_id

    FROM customers

    WHERE balance > 10000

  ) LOOP

    UPDATE customers

    SET isvip = 'TRUE'

    WHERE customer\_id = customer\_rec.customer\_id;

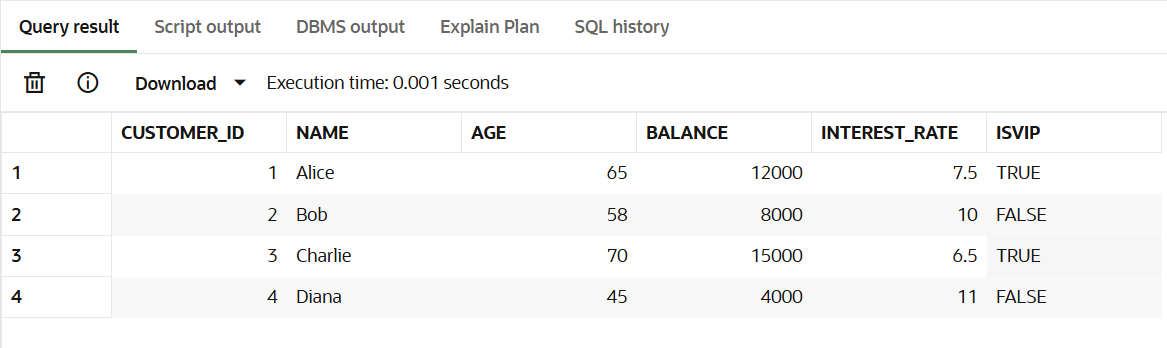
  END LOOP;

  COMMIT;

END;

SELECT \* FROM customers;

**OUTPUT:**

****

**SCENARIO3:**

BEGIN

  FOR loan\_rec IN (

    SELECT l.loan\_id, l.due\_date, c.name

    FROM loans l

    JOIN customers c ON l.customer\_id = c.customer\_id

    WHERE l.due\_date <= SYSDATE + 30

  ) LOOP

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

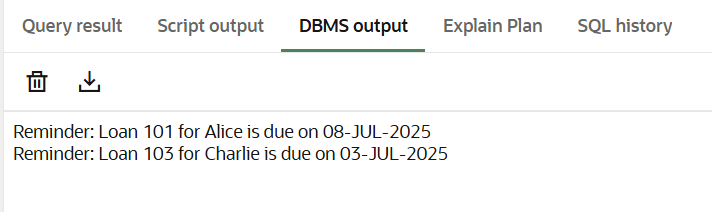
                         ' for ' || loan\_rec.name ||

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

  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.

**CODE:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE accounts';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE employees';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

account\_type VARCHAR2(20),

balance NUMBER

);

CREATE TABLE employees (

emp\_id NUMBER PRIMARY KEY,

name VARCHAR2(50),

department VARCHAR2(50),

salary NUMBER

);

INSERT INTO accounts VALUES (101, 1, 'savings', 10000);

INSERT INTO accounts VALUES (102, 2, 'savings', 5000);

INSERT INTO accounts VALUES (103, 1, 'checking', 8000);

INSERT INTO accounts VALUES (104, 3, 'savings', 20000);

INSERT INTO employees VALUES (1, 'Akash', 'HR', 50000);

INSERT INTO employees VALUES (2, 'Bonda', 'IT', 60000);

INSERT INTO employees VALUES (3, 'Prasanth', 'IT', 70000);

INSERT INTO employees VALUES (4, 'Dino', 'Sales', 55000);

**SCENARIO1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE accounts

SET balance = balance + (balance \* 0.01)

WHERE account\_type = 'savings';

COMMIT;

END;

/

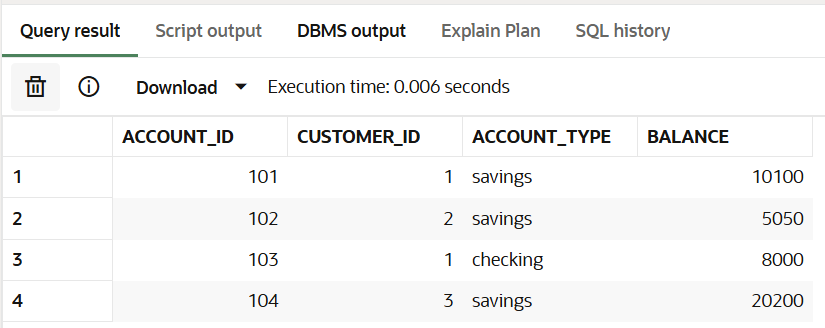
BEGIN

ProcessMonthlyInterest;

END;

SELECT \* FROM accounts;

**OUTPUT:**

****

**SCENARIO2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_name IN VARCHAR2,

bonus\_pct IN NUMBER

) IS

BEGIN

UPDATE employees

SET salary = salary + (salary \* bonus\_pct / 100)

WHERE department = dept\_name;

COMMIT;

END;

/

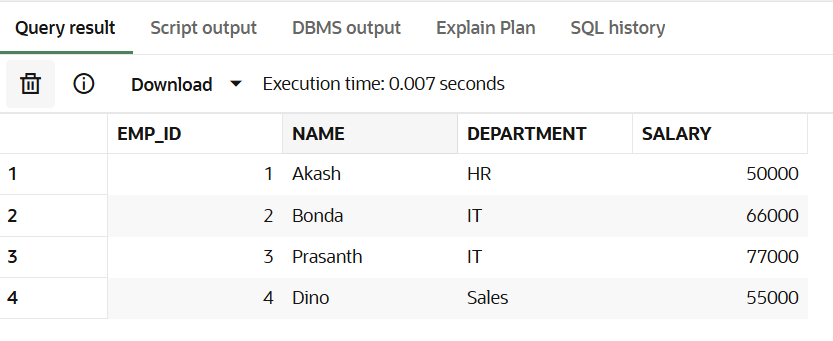
BEGIN

UpdateEmployeeBonus('IT', 10); -- Add 10% bonus to IT department

END;

SELECT \* FROM employees;

**OUTPUT:**

****

**SCENARIO3:**

CREATE OR REPLACE PROCEDURE TransferFunds (

from\_acct IN NUMBER,

to\_acct IN NUMBER,

amount IN NUMBER

) IS

insufficient\_balance EXCEPTION;

BEGIN

-- Check if from account has enough balance

DECLARE

from\_balance NUMBER;

BEGIN

SELECT balance INTO from\_balance

FROM accounts

WHERE account\_id = from\_acct;

IF from\_balance < amount THEN

RAISE insufficient\_balance;

END IF;

-- Deduct from source

UPDATE accounts

SET balance = balance - amount

WHERE account\_id = from\_acct;

-- Add to destination

UPDATE accounts

SET balance = balance + amount

WHERE account\_id = to\_acct;

COMMIT;

EXCEPTION

WHEN insufficient\_balance THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient balance in source account.');

ROLLBACK;

END;

END;

/

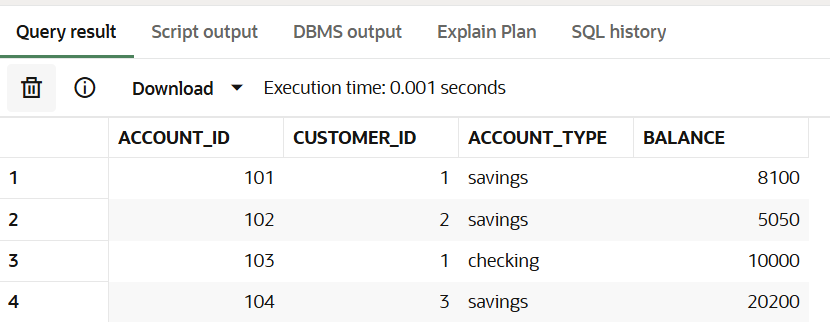
BEGIN

TransferFunds(101, 103, 2000); -- Transfer 2000 from account 101 to 103

END;

SELECT \* FROM accounts;

**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: junit junit 4.13.2 test

3. Create a new test class in your project**.**

**Calculator.java:**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java:**

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

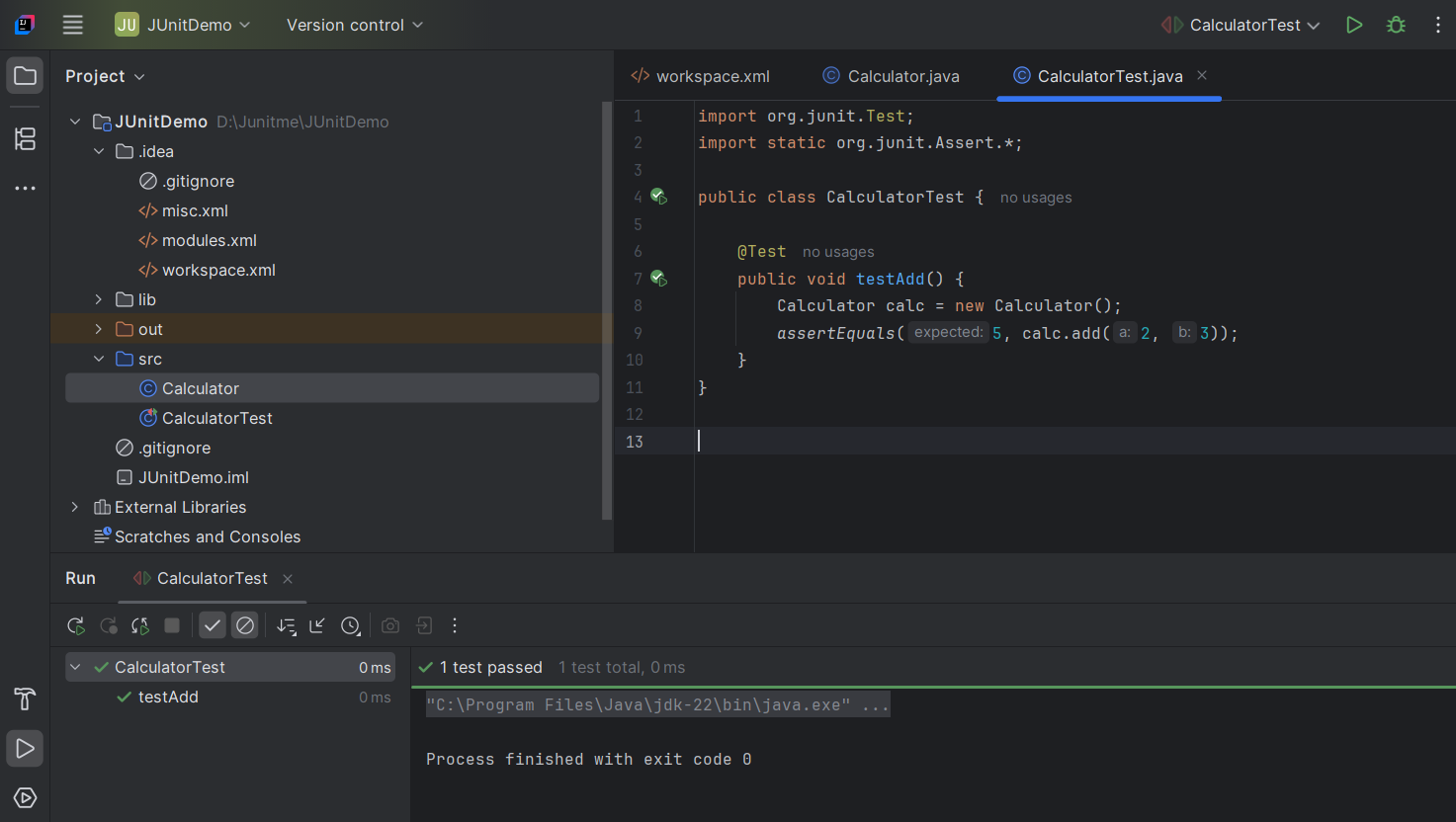
public void testAdd() {

Calculator calc = new Calculator();

assertEquals(5, calc.add(2, 3));

}

}



**Exercise 3: Assertions in JUnit Scenario: You need to use different assertions in JUnit to validate your test results.**

**AssertionsTest.java:**

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals("2 + 3 should be 5", 5, 2 + 3);

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

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

Object obj1 = null;

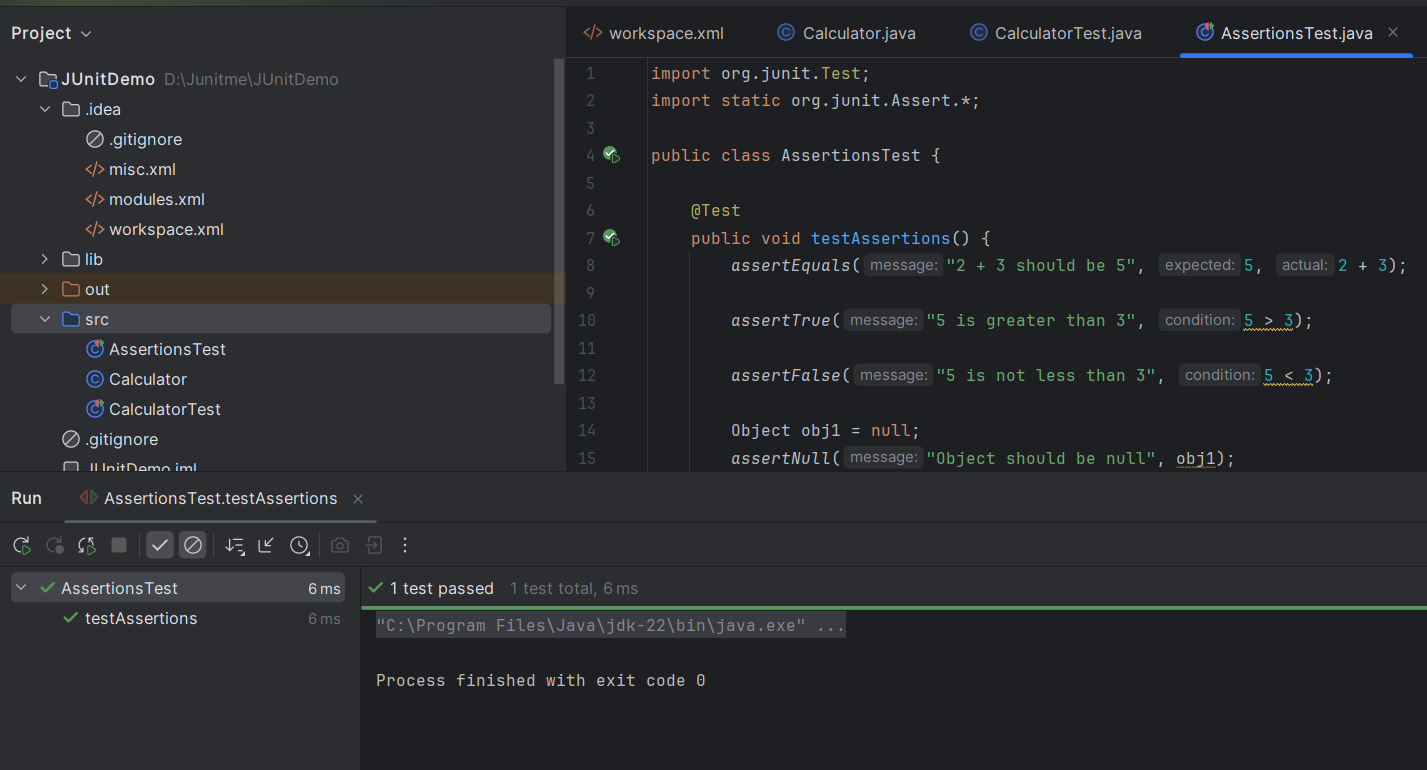
assertNull("Object should be null", obj1);

Object obj2 = new Object();

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

}

}



**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.**

**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;

}

}

**CalculatorTest.java:**

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();

System.out.println("Setup: Creating Calculator instance");

}

@After

public void tearDown() {

calculator = null;

System.out.println("Teardown: Clearing Calculator instance");

}

@Test

public void testAddition() {

int a = 2;

int b = 3;

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

assertEquals(5, result);

}

@Test

public void testSubtraction() {

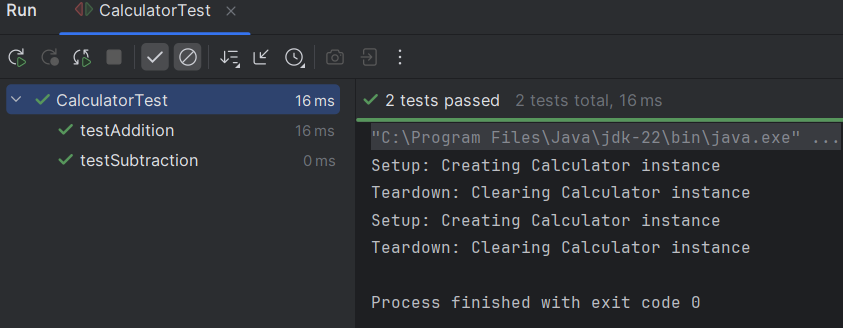
int a = 5;

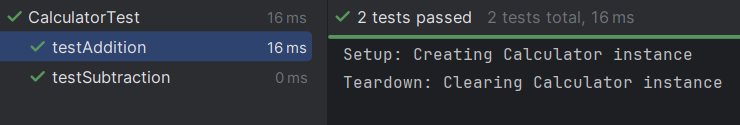
int b = 2;

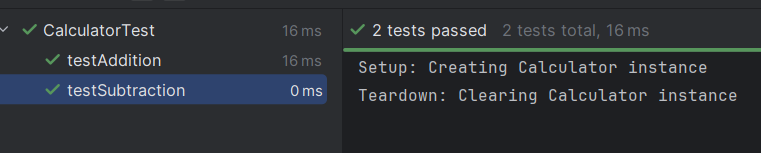
int result = calculator.subtract(a, b);

assertEquals(3, result);

}

}





**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**

**ExternalApi.java:**

public interface ExternalApi {

String getData();

}

**MyService.java:**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java:**

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

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

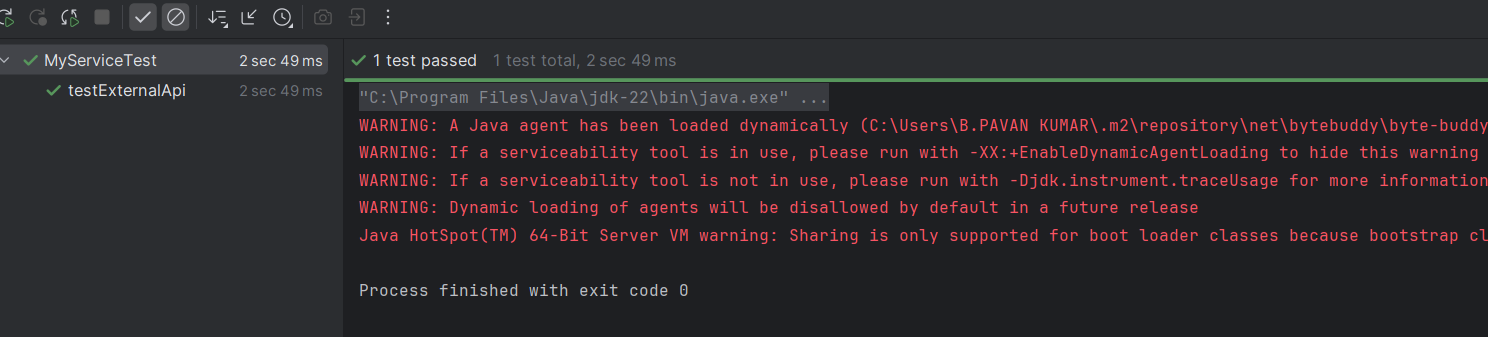
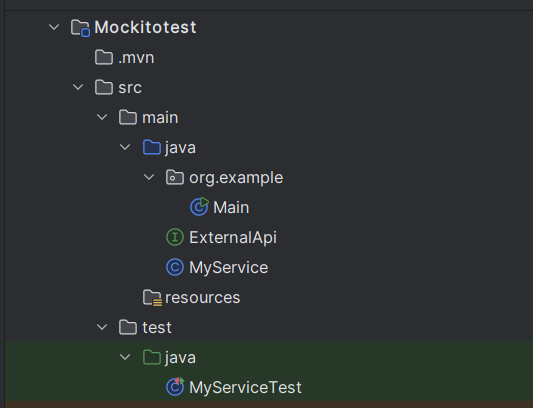
MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}



**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.**

**ExternalApi.java:**

public interface ExternalApi {

String getData();

}

**MyService.java:**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

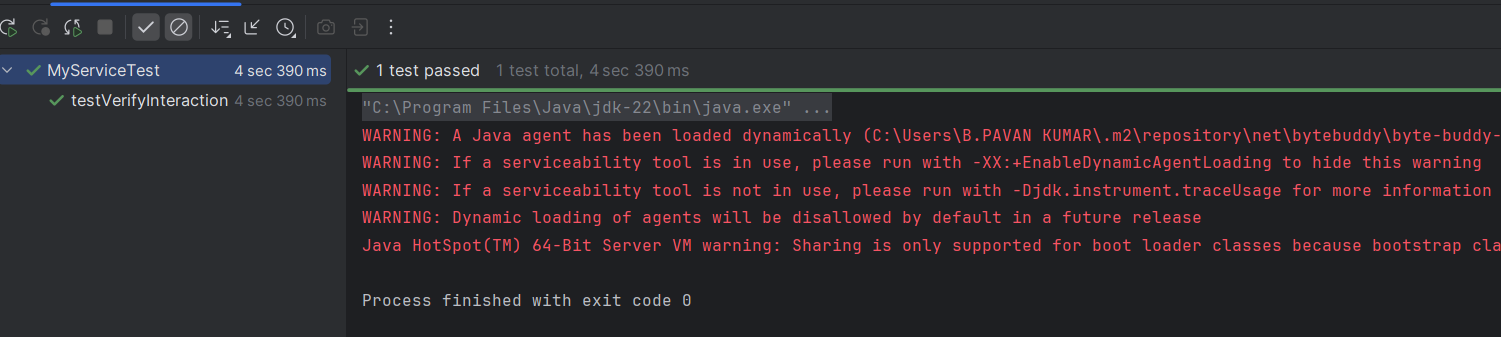
MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}



**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.**

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

**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>LoggingExample</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>11</maven.compiler.source>  
 <maven.compiler.target>11</maven.compiler.target>  
 </properties>  
  
 <dependencies>  
 <!-- SLF4J API -->  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>1.7.30</version>  
 </dependency>  
  
 <!-- Logback implementation for SLF4J -->  
 <dependency>  
 <groupId>ch.qos.logback</groupId>  
 <artifactId>logback-classic</artifactId>  
 <version>1.2.3</version>  
 </dependency>  
 </dependencies>  
</project>

