**PL/SQL PROGRAMMING**

**PL/SQL Exercises:**

**Exercise 1: Control Structures:**

-- Scenario 1: Apply a 1% discount to loan interest rates for customers above 60 years old.

-- This block assumes the existence of 'customers' and 'loans' tables.

-- 'customers' table: CUSTOMER\_ID (PK), AGE

-- 'loans' table: LOAN\_ID (PK), CUSTOMER\_ID (FK), INTEREST\_RATE

DECLARE

    CURSOR c\_customers IS

        SELECT customer\_id, age

        FROM customers;

    v\_customer\_id   NUMBER;

    v\_age           NUMBER;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('--- Scenario 1: Loan Interest Rate Discount ---');

    -- Loop through each customer

    FOR rec\_customer IN c\_customers

    LOOP

        v\_customer\_id := rec\_customer.customer\_id;

        v\_age := rec\_customer.age;

        -- Check if the customer is above 60 years old

        IF v\_age > 60 THEN

            -- Apply a 1% discount to their loan interest rates

            UPDATE loans

            SET interest\_rate = interest\_rate \* 0.99 -- 1% discount

            WHERE customer\_id = v\_customer\_id;

            DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id || ' (Age: ' || v\_age || ') - Discount applied.');

        ELSE

            DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id || ' (Age: ' || v\_age || ') - No discount.');

        END IF;

    END LOOP;

    COMMIT; -- Commit the changes

    DBMS\_OUTPUT.PUT\_LINE('--- Scenario 1: Completed ---');

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK; -- Rollback in case of an error

        DBMS\_OUTPUT.PUT\_LINE('An error occurred in Scenario 1: ' || SQLERRM);

END;

/

-- Scenario 2: Promote customers to VIP status based on their balance.

-- This block assumes the existence of a 'customers' table.

-- 'customers' table: CUSTOMER\_ID (PK), BALANCE, IS\_VIP (BOOLEAN/NUMBER, e.g., 0 for FALSE, 1 for TRUE)

DECLARE

    CURSOR c\_customers IS

        SELECT customer\_id, balance

        FROM customers;

    v\_customer\_id   NUMBER;

    v\_balance       NUMBER;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Scenario 2: VIP Status Promotion ---');

    -- Loop through each customer

    FOR rec\_customer IN c\_customers

    LOOP

        v\_customer\_id := rec\_customer.customer\_id;

        v\_balance := rec\_customer.balance;

        -- Check if the customer's balance is over $10,000

        IF v\_balance > 10000 THEN

            -- Set IsVIP flag to TRUE (e.g., 1)

            UPDATE customers

            SET is\_vip = 1

            WHERE customer\_id = v\_customer\_id;

            DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id || ' (Balance: $' || v\_balance || ') - Promoted to VIP.');

        ELSE

            -- Optionally, set to FALSE (e.g., 0) if they are not VIP

            -- UPDATE customers

            -- SET is\_vip = 0

            -- WHERE customer\_id = v\_customer\_id;

            DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || v\_customer\_id || ' (Balance: $' || v\_balance || ') - Not VIP.');

        END IF;

    END LOOP;

    COMMIT; -- Commit the changes

    DBMS\_OUTPUT.PUT\_LINE('--- Scenario 2: Completed ---');

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK; -- Rollback in case of an error

        DBMS\_OUTPUT.PUT\_LINE('An error occurred in Scenario 2: ' || SQLERRM);

END;

/

-- Scenario 3: Send reminders to customers whose loans are due within the next 30 days.

-- This block assumes the existence of 'loans' and 'customers' tables.

-- 'loans' table: LOAN\_ID (PK), CUSTOMER\_ID (FK), DUE\_DATE

-- 'customers' table: CUSTOMER\_ID (PK), CUSTOMER\_NAME, EMAIL (or phone for contact)

DECLARE

    CURSOR c\_due\_loans IS

        SELECT

            l.loan\_id,

            l.due\_date,

            c.customer\_name,

            c.email -- Assuming an email column for sending reminders

        FROM

            loans l

        JOIN

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

        WHERE

            l.due\_date BETWEEN TRUNC(SYSDATE) AND TRUNC(SYSDATE) + 30; -- Loans due in the next 30 days

    v\_loan\_id       NUMBER;

    v\_due\_date      DATE;

    v\_customer\_name VARCHAR2(100);

    v\_email         VARCHAR2(100);

BEGIN

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Scenario 3: Loan Due Reminders ---');

    -- Loop through loans due in the next 30 days

    FOR rec\_loan IN c\_due\_loans

    LOOP

        v\_loan\_id := rec\_loan.loan\_id;

        v\_due\_date := rec\_loan.due\_date;

        v\_customer\_name := rec\_loan.customer\_name;

        v\_email := rec\_loan.email;

        -- Print a reminder message for each customer

        DBMS\_OUTPUT.PUT\_LINE('Reminder for ' || v\_customer\_name || ' (Email: ' || v\_email || '):');

        DBMS\_OUTPUT.PUT\_LINE('  Your loan (ID: ' || v\_loan\_id || ') is due on ' || TO\_CHAR(v\_due\_date, 'YYYY-MM-DD') || '.');

        DBMS\_OUTPUT.PUT\_LINE('  Please ensure timely repayment.');

        DBMS\_OUTPUT.PUT\_LINE('--------------------------------------');

    END LOOP;

    -- Check if any loans were found

    IF c\_due\_loans%NOTFOUND THEN

        DBMS\_OUTPUT.PUT\_LINE('No loans found due in the next 30 days.');

    END IF;

    DBMS\_OUTPUT.PUT\_LINE('--- Scenario 3: Completed ---');

EXCEPTION

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE('An error occurred in Scenario 3: ' || SQLERRM);

END;

/

**Exercise 3: Stored Procedures:**

-- Scenario 1: Process monthly interest for all savings accounts.

-- This procedure assumes the existence of an 'accounts' table.

-- 'accounts' table: ACCOUNT\_ID (PK), ACCOUNT\_TYPE (e.g., 'SAVINGS', 'CHECKING'), BALANCE

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

    v\_interest\_rate CONSTANT NUMBER := 0.01; -- 1% interest rate

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('--- Scenario 1: Processing Monthly Interest for Savings Accounts ---');

    -- Update the balance of all savings accounts

    UPDATE accounts

    SET balance = balance \* (1 + v\_interest\_rate)

    WHERE account\_type = 'SAVINGS';

    -- Check how many rows were updated

    IF SQL%ROWCOUNT > 0 THEN

        DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' savings account(s) updated with monthly interest.');

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('No savings accounts found or updated.');

    END IF;

    COMMIT; -- Commit the changes

    DBMS\_OUTPUT.PUT\_LINE('--- Scenario 1: Monthly Interest Processing Completed ---');

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK; -- Rollback in case of an error

        DBMS\_OUTPUT.PUT\_LINE('An error occurred in Scenario 1: ' || SQLERRM);

END ProcessMonthlyInterest;

/

-- To execute this procedure:

-- EXEC ProcessMonthlyInterest;

-- Scenario 2: Implement a bonus scheme for employees based on their performance.

-- This procedure assumes the existence of an 'employees' table.

-- 'employees' table: EMPLOYEE\_ID (PK), DEPARTMENT\_ID (FK), SALARY

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus

(

    p\_department\_id IN NUMBER,   -- The ID of the department to apply the bonus to

    p\_bonus\_percentage IN NUMBER -- The bonus percentage (e.g., 5 for 5%)

)

IS

    v\_rows\_updated NUMBER;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Scenario 2: Updating Employee Bonus ---');

    DBMS\_OUTPUT.PUT\_LINE('Applying ' || p\_bonus\_percentage || '% bonus to employees in Department ID: ' || p\_department\_id);

    -- Update the salary of employees in the given department by adding a bonus percentage

    UPDATE employees

    SET salary = salary \* (1 + (p\_bonus\_percentage / 100))

    WHERE department\_id = p\_department\_id;

    v\_rows\_updated := SQL%ROWCOUNT;

    IF v\_rows\_updated > 0 THEN

        DBMS\_OUTPUT.PUT\_LINE(v\_rows\_updated || ' employee(s) updated with bonus in Department ID: ' || p\_department\_id);

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('No employees found or updated in Department ID: ' || p\_department\_id);

    END IF;

    COMMIT; -- Commit the changes

    DBMS\_OUTPUT.PUT\_LINE('--- Scenario 2: Employee Bonus Update Completed ---');

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK; -- Rollback in case of an error

        DBMS\_OUTPUT.PUT\_LINE('An error occurred in Scenario 2: ' || SQLERRM);

END UpdateEmployeeBonus;

/

-- To execute this procedure (example: 10% bonus for department 10):

-- EXEC UpdateEmployeeBonus(p\_department\_id => 10, p\_bonus\_percentage => 10);

-- Scenario 3: Customers should be able to transfer funds between their accounts.

-- This procedure assumes the existence of an 'accounts' table.

-- 'accounts' table: ACCOUNT\_ID (PK), BALANCE

CREATE OR REPLACE PROCEDURE TransferFunds

(

    p\_source\_account\_id IN NUMBER,

    p\_destination\_account\_id IN NUMBER,

    p\_amount IN NUMBER

)

IS

    v\_source\_balance NUMBER;

    e\_insufficient\_funds EXCEPTION; -- Custom exception for insufficient funds

BEGIN

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Scenario 3: Transferring Funds ---');

    DBMS\_OUTPUT.PUT\_LINE('Attempting to transfer $' || p\_amount || ' from Account ID ' || p\_source\_account\_id || ' to Account ID ' || p\_destination\_account\_id);

    -- Check if the transfer amount is positive

    IF p\_amount <= 0 THEN

        RAISE\_APPLICATION\_ERROR(-20001, 'Transfer amount must be positive.');

    END IF;

    -- Lock the source and destination accounts to prevent concurrency issues

    SELECT balance

    INTO v\_source\_balance

    FROM accounts

    WHERE account\_id = p\_source\_account\_id

    FOR UPDATE OF balance NOWAIT; -- NOWAIT will raise an error if the row is already locked

    -- Check if the source account has sufficient balance

    IF v\_source\_balance < p\_amount THEN

        RAISE e\_insufficient\_funds; -- Raise custom exception

    END IF;

    -- Deduct amount from source account

    UPDATE accounts

    SET balance = balance - p\_amount

    WHERE account\_id = p\_source\_account\_id;

    -- Add amount to destination account

    UPDATE accounts

    SET balance = balance + p\_amount

    WHERE account\_id = p\_destination\_account\_id;

    COMMIT; -- Commit the transaction if all updates are successful

    DBMS\_OUTPUT.PUT\_LINE('Successfully transferred $' || p\_amount || ' from Account ID ' || p\_source\_account\_id || ' to Account ID ' || p\_destination\_account\_id || '.');

EXCEPTION

    WHEN e\_insufficient\_funds THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in source account (ID: ' || p\_source\_account\_id || ').');

    WHEN NO\_DATA\_FOUND THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('Error: One or both account IDs not found.');

    WHEN OTHERS THEN

        ROLLBACK; -- Rollback in case of any other error

        DBMS\_OUTPUT.PUT\_LINE('An error occurred in Scenario 3: ' || SQLERRM);

END TransferFunds;

/

-- To execute this procedure (example: transfer $500 from account 101 to account 102):

-- EXEC TransferFunds(p\_source\_account\_id => 101, p\_destination\_account\_id => 102, p\_amount => 500);

**TDD using JUnit5 and Mockito**

**JUnit\_Basic Testing Exercises**

**Exercise 1: Setting Up Junit**

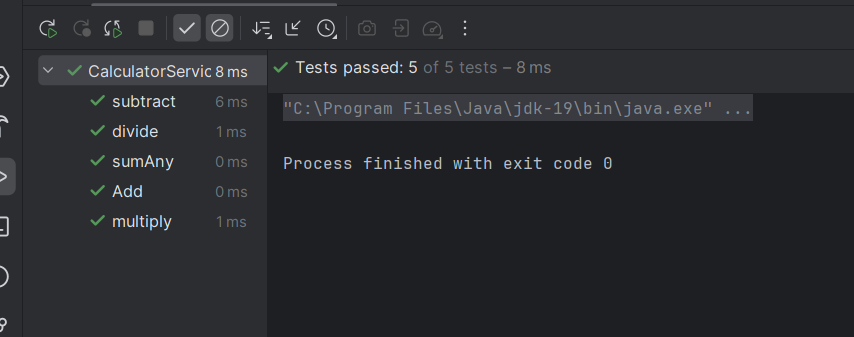
**Pom.xml**

<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.arijit.test</groupId>  
 <artifactId>Cognizant\_TestingExercises</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>Cognizant\_TestingExercises</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <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>  
 <dependency>  
 <groupId>org.jetbrains</groupId>  
 <artifactId>annotations</artifactId>  
 <version>RELEASE</version>  
 <scope>compile</scope>  
 </dependency>  
 </dependencies>  
</project>

**CalculatorService.java**

package com.arijit.test.services;  
  
public class CalculatorService {  
  
 public static int Add(int a, int b) {  
 return a + b;  
 }  
  
 public static int subtract(int a, int b) {  
 return a - b;  
 }  
  
 public static int multiply(int a, int b) {  
 return a \* b;  
 }  
  
 public static int divide(int a, int b) {  
 return a / b;  
 }  
  
 public static int sumAny(int ...num){  
 int sum = 0;  
 for(int i : num){  
 sum += i;  
 }  
 return sum;  
 }  
}

**OUTPUT:**

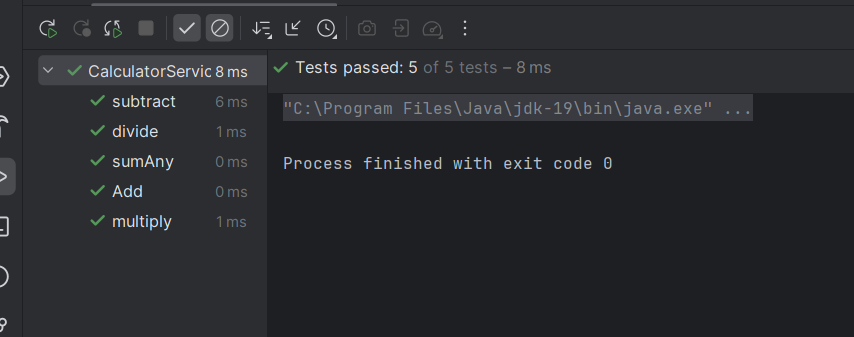


**Exercise 2: Writing Basic junit Tests:**

**CalculatorServiceTest.java**

package com.arijit.test.services;  
  
  
import org.junit.After;  
import org.junit.Assert;  
import org.junit.Before;  
import org.junit.Test;  
  
import java.sql.SQLOutput;  
  
public class CalculatorServiceTest {  
  
 //this will run before every test case  
 @Before  
 public void BeforeEachTest() {  
 System.*out*.println("Running upcoming Test case...");  
 }  
  
 @After  
 public void AfterEachTest() {  
 System.*out*.println("Running Test case ran successfully");  
 }  
 // test method of Add  
 @Test  
 public void Add(){  
  
 //actual result  
 int res = CalculatorService.*Add*(12,45);  
  
 //expected result  
 int expres = 57;  
 Assert.*assertEquals*(expres, res);  
  
 }  
  
 @Test  
 public void sumAny(){  
 //actual result  
 int res = CalculatorService.*sumAny*(2,7,8,9);  
  
 //expected result  
 int expres = 26;  
 Assert.*assertEquals*(expres, res);  
 }  
  
 @Test  
 public void subtract(){  
 int res = CalculatorService.*subtract*(45,12);  
 int expres = 33;  
 Assert.*assertEquals*(expres, res);  
 }  
  
 @Test  
 public void multiply(){  
 int res = CalculatorService.*multiply*(12,45);  
 int expres = 540;  
 Assert.*assertEquals*(expres, res);  
 }  
  
 @Test  
 public void divide(){  
 int res = CalculatorService.*divide*(60,20);  
 int expres = 3;  
 Assert.*assertEquals*(expres, res);  
 }  
  
}

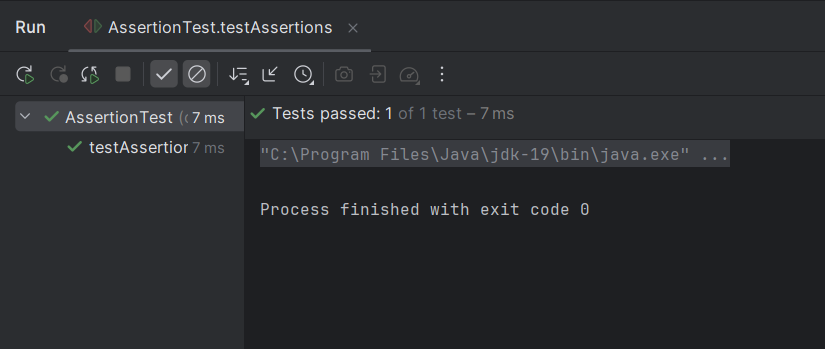
**OUTPUT**



**Exercise 3: Assertions in junit:**

package com.arijit.test.services;  
  
import org.junit.Assert;  
import org.junit.Test;  
  
public class AssertionTest {  
 @Test  
 public void testAssertions() {  
 Assert.*assertEquals*(5,2+3);  
 Assert.*assertTrue*(5>3);  
 Assert.*assertFalse*(5<3);  
 Assert.*assertNull*(null);  
 Assert.*assertNotNull*(new Object());  
 }  
}

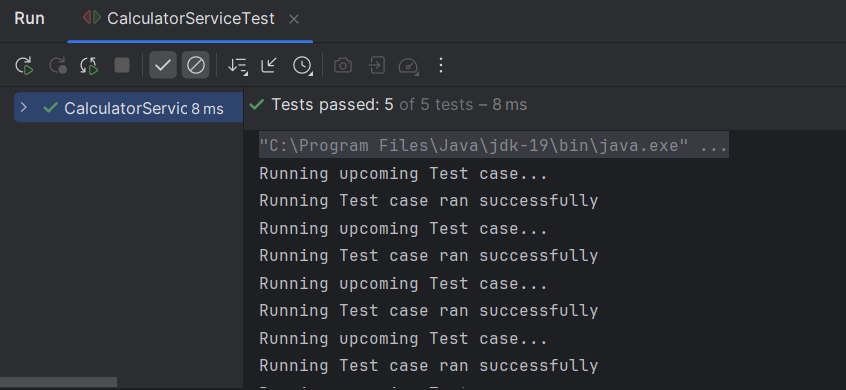
**OUTPUT:**



**Exercise 4: Arrange-Act-Assert(AAA) Pattern, Test Fixures, Setup and TearDown Methods in junit:**

package com.arijit.test.services;  
  
  
import org.junit.After;  
import org.junit.Assert;  
import org.junit.Before;  
import org.junit.Test;  
  
import java.sql.SQLOutput;  
  
public class CalculatorServiceTest {  
  
 //this will run before every test case  
 @Before  
 public void BeforeEachTest() {  
 System.*out*.println("Running upcoming Test case...");  
 }  
  
 @After  
 public void AfterEachTest() {  
 System.*out*.println("Running Test case ran successfully");  
 }  
 // test method of Add  
 @Test  
 public void Add(){  
  
 //actual result  
 int res = CalculatorService.*Add*(12,45);  
  
 //expected result  
 int expres = 57;  
 Assert.*assertEquals*(expres, res);  
  
 }  
  
 @Test  
 public void sumAny(){  
 //actual result  
 int res = CalculatorService.*sumAny*(2,7,8,9);  
  
 //expected result  
 int expres = 26;  
 Assert.*assertEquals*(expres, res);  
 }  
  
 @Test  
 public void subtract(){  
 int res = CalculatorService.*subtract*(45,12);  
 int expres = 33;  
 Assert.*assertEquals*(expres, res);  
 }  
  
 @Test  
 public void multiply(){  
 int res = CalculatorService.*multiply*(12,45);  
 int expres = 540;  
 Assert.*assertEquals*(expres, res);  
 }  
  
 @Test  
 public void divide(){  
 int res = CalculatorService.*divide*(60,20);  
 int expres = 3;  
 Assert.*assertEquals*(expres, res);  
 }  
  
}

**OUTPUT:**



**Mockito Exercises:**

**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.arijit.test</groupId>  
 <artifactId>Mockito\_Exercises</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>Mockito\_Exercises</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- Mockito Core -->  
 <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>5.11.0</version> <!-- You can check for latest version -->  
 <scope>test</scope>  
 </dependency>  
  
 <!-- JUnit Jupiter API (JUnit 5) -->  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>5.10.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
</project>

**Exercise 1: Mocking and Stubbing:**

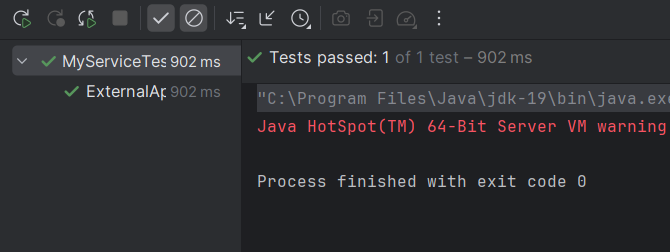
**MyService.java**

package com.arijit.test.services;  
  
interface ExternalApi {  
 String getData(String input);  
}  
  
public class MyService {  
 private final ExternalApi externalApi;  
  
 public MyService(ExternalApi externalApi) {  
 this.externalApi = externalApi;  
 }  
  
 public String fetchData() {  
 return externalApi.getData("input");  
 }  
}

**MyServiceTest.java**

package com.arijit.test.services;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void ExternalApi() {  
 // Mock the external API  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 *when*(mockApi.getData("input")).thenReturn("Mock Data");  
 MyService service = new MyService(mockApi);  
  
  
 String result = service.fetchData();  
  
  
 *assertEquals*("Mock Data", result);  
  
 }  
}

**OUTPUT:**



**Exercise 2: Verifying interactions:**

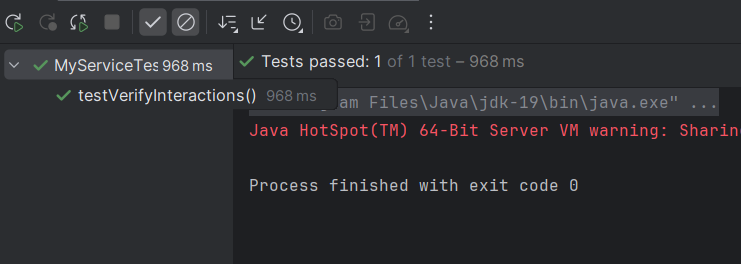
**MyService.java**

package com.arijit.test.services;  
  
interface ExternalApi {  
 String getData(String input);  
}  
  
public class MyService {  
 private final ExternalApi externalApi;  
  
 public MyService(ExternalApi externalApi) {  
 this.externalApi = externalApi;  
 }  
  
 public String process(String input) {  
 String data = externalApi.getData(input);  
 return "Processed: " + data;  
 }  
}

**MyServiceTest.java**

package com.arijit.test.services;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testVerifyInterations() {  
 // Mock the external API  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 *when*(mockApi.getData("input")).thenReturn("mocked-data");  
  
 // Inject mock into the service  
 MyService service = new MyService(mockApi);  
  
 // Call the method  
 String result = service.process("input");  
  
 // Verify results  
 *assertEquals*("Processed: mocked-data", result);  
 *verify*(mockApi).getData("input");  
 }  
}

**OUTPUT:**



**sl4j Logging Exercises:**

**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.arijit.test</groupId>  
 <artifactId>Mockito\_Exercises</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>Mockito\_Exercises</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- Mockito Core -->  
 <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>5.11.0</version> <!-- You can check for latest version -->  
 <scope>test</scope>  
 </dependency>  
  
 <!-- JUnit Jupiter API (JUnit 5) -->  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>5.10.2</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>  
  
</project>

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

**LoggingExample.java:**

package com.arijit.test.services;  
  
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:**

