**Week 2**

**Mandatory Hands-On**

**-Abinaya B(Superset ID: 6388062)**

**PL SQL Exercises**

**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.

**CODE:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

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

VALUES (1, 'John Doe', TO\_DATE('1960-05-15', '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);

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

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

declare

cursor senior\_customers is

select c.CustomerID

from Customers c

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

begin

for cust in senior\_customers loop

update Loans

set InterestRate=InterestRate-1

where CustomerID=cust.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('1% discount applied to loans of customer ID: '||cust.CustomerID);

end loop;

end;

/

**OUTPUT:**

A screen shot of a computer

AI-generated content may be incorrect.

**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.

**CODE:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

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);

-- Add IsVIP column to Customers table

alter table Customers add(IsVIP VARCHAR2(5));

-- Make John's balance 15,000 (VIP)

update Customers set Balance=20000 where CustomerID=1;

-- Make Jane's balance 8000 (Not VIP)

update Customers set Balance=5000 where CustomerID=2;

commit;

begin

for cust in(select CustomerID,Name,Balance from Customers) loop

if cust.Balance>10000 then

update Customers

set IsVIP='TRUE'

where CustomerID=cust.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer '||cust.Name||' promoted to VIP.');

else

update Customers

set IsVIP='FALSE'

where CustomerID=cust.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer '||cust.Name||' is not VIP.');

end if;

end loop;

end;

/

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**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 (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

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);

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

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

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

VALUES (2, 2, 3000, 6.5, SYSDATE, SYSDATE + 10);

COMMIT;

BEGIN

for rec in(

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

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: Dear ' || rec.Name ||

', your loan (Loan ID: ' || rec.LoanID ||

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

'. Please ensure timely payment.');

end LOOP;

end;

/

**OUTPUT:**

A screen shot of a computer

AI-generated content may be incorrect.

**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.

**CODE:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

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);

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);

begin

-- Loop through all savings accounts

for acc in(

select AccountID,Balance

from Accounts

where UPPER(AccountType)='SAVINGS'

) LOOP

-- Calculate new balance with 1% interest

update Accounts

set Balance=Balance+(acc.Balance\*0.01),

LastModified=SYSDATE

where AccountID=acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID: ' || acc.AccountID ||

' | New Balance: ' || TO\_CHAR(acc.Balance + (acc.Balance \* 0.01), 'FM9999990.00'));

end LOOP;

end;

/

**OUTPUT:**

A screen shot of a computer

AI-generated content may be incorrect.

**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.

**CODE:**

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

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'));

create or replace procedure updateemployeebonus(

p\_department in varchar2,

p\_bonuspercent in number

) is

begin

for emp in (

select employeeid, name, salary

from employees

where upper(department) = upper(p\_department)

) loop

update employees

set salary = salary + (emp.salary \* (p\_bonuspercent / 100))

where employeeid = emp.employeeid;

dbms\_output.put\_line(

'Bonus applied to ' || emp.name ||

' | New Salary: ' || to\_char(emp.salary \* (1 + p\_bonuspercent / 100), 'fm9999990.00')

);

end loop;

end;

/

begin

updateemployeebonus('HR', 10);

end;

/

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**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:**

create table customers (

customerid number primary key,

name varchar2(100),

dob date,

balance number,

lastmodified date

);

create table accounts (

accountid number primary key,

customerid number,

accounttype varchar2(20),

balance number,

lastmodified date,

foreign key (customerid) references customers(customerid)

);

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);

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);

create or replace procedure transferfunds(

p\_from\_account in number,

p\_to\_account in number,

p\_amount in number

) is

v\_from\_balance accounts.balance%type;

begin

-- Fetch balance of the source account

select balance into v\_from\_balance

from accounts

where accountid=p\_from\_account;

-- Check if source has enough funds

if v\_from\_balance<p\_amount then

raise\_application\_error(-20001,'Insufficient balance in the source account.');

end if;

-- Deduct from source account

update accounts

set balance=balance-p\_amount,

lastmodified=sysdate

where accountid=p\_from\_account;

-- Credit to destination account

update accounts

set balance=balance+p\_amount,

lastmodified=sysdate

where accountid=p\_to\_account;

dbms\_output.put\_line(

'Transferred ' || p\_amount || ' from Account ID ' || p\_from\_account ||

' to Account ID ' || p\_to\_account

);

end;

/

begin

transferfunds(1, 2, 500);

end;

/

**OUTPUT:**A screen shot of a computer

AI-generated content may be incorrect.

**JUnit, Mockito and SL4J**

**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.

**CODE:**

Pom.xml:

<?xml version="1.0" encoding="UTF-8"?>  
<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>org.example</groupId>  
 <artifactId>junit\_mandatory\_basic\_1</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>17</maven.compiler.source>  
 <maven.compiler.target>17</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- JUnit 4 dependency -->  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
  
 <!-- Hamcrest for assertions -->  
 <dependency>  
 <groupId>org.hamcrest</groupId>  
 <artifactId>hamcrest</artifactId>  
 <version>2.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
</project>

NumberUtils

package org.example;  
public class NumberUtils{  
 public boolean isPalindrome(int number){  
 if (number<0) return false;  
 int original=number;  
 int reversed=0;  
 while(number!=0) {  
 int digit=number%10;  
 reversed=reversed\*10+digit;  
 number/=10;  
 }  
 return original==reversed;  
 }  
}

NumberUtilsTest

package org.example;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
public class NumberUtilsTest{  
 @Test  
 public void testIsPalindrome\_withPalindromeNumbers(){  
 NumberUtils utils=new NumberUtils();  
 assertTrue(utils.isPalindrome(121));  
 assertTrue(utils.isPalindrome(1221));  
 assertTrue(utils.isPalindrome(1));  
 assertTrue(utils.isPalindrome(0));  
 }  
 @Test  
 public void testIsPalindrome\_withNonPalindromeNumbers(){  
 NumberUtils utils=new NumberUtils();  
 assertFalse(utils.isPalindrome(123));  
 assertFalse(utils.isPalindrome(10));  
 assertFalse(utils.isPalindrome(-121));  
 }  
}

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**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()); } }

Pom.xml:

**CODE:**

<?xml version="1.0" encoding="UTF-8"?>  
<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>org.example</groupId>  
 <artifactId>junit\_mandatory\_basic\_1</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>17</maven.compiler.source>  
 <maven.compiler.target>17</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- JUnit Jupiter API -->  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-api</artifactId>  
 <version>5.9.3</version>  
 <scope>test</scope>  
 </dependency>  
  
 <!-- JUnit Jupiter Engine (required to run tests) -->  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-engine</artifactId>  
 <version>5.9.3</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.22.2</version>  
 </plugin>  
 </plugins>  
 </build>  
</project>

AssertionsTest:

package org.example;  
import static org.junit.jupiter.api.Assertions.\*;  
import org.junit.jupiter.api.Test;  
  
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:**

A screenshot of a computer program

AI-generated content may be incorrect.

**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.

Pom.xml:

**CODE:**

<?xml version="1.0" encoding="UTF-8"?>  
<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>org.example</groupId>  
 <artifactId>junit\_mandatory\_basic\_1</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>17</maven.compiler.source>  
 <maven.compiler.target>17</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- JUnit Jupiter API -->  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-api</artifactId>  
 <version>5.9.3</version>  
 <scope>test</scope>  
 </dependency>  
  
 <!-- JUnit Jupiter Engine (required to run tests) -->  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-engine</artifactId>  
 <version>5.9.3</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.22.2</version>  
 </plugin>  
 </plugins>  
 </build>  
</project>

PalindromeChecker.java:

package org.example;  
public class PalindromeChecker {  
 public boolean isPalindrome(int number){  
 int original=number;  
 int reversed=0;  
 while (number>0){  
 int digit=number%10;  
 reversed=reversed\*10+digit;  
 number=number/10;  
 }  
 return original==reversed;  
 }  
 public void reset(){  
 // No-op: For demo teardown purposes  
 }  
}

PalindromeCheckerTest.java:

package org.example;  
import org.junit.jupiter.api.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
public class PalindromeCheckerTest{  
 private PalindromeChecker checker;  
 @BeforeEach  
 public void setUp(){  
 checker=new PalindromeChecker(); // Arrange  
 System.out.println("Setup done");  
 }  
 @AfterEach  
 public void tearDown(){  
 checker.reset(); // Simulate cleanup  
 System.out.println("Teardown done");  
 }  
 @Test  
 public void testPalindromeNumber(){  
 // Act  
 boolean result = checker.isPalindrome(121);  
 // Assert  
 assertTrue(result,"121 should be a palindrome");  
 }  
 @Test  
 public void testNonPalindromeNumber(){  
 // Act  
 boolean result=checker.isPalindrome(123);  
 // Assert  
 assertFalse(result,"123 is not a palindrome");  
 }  
 @Test  
 public void testSingleDigitNumber(){  
 // Act  
 boolean result = checker.isPalindrome(7);  
 // Assert  
 assertTrue(result, "Single-digit numbers are palindromes");  
 }  
}

**Output:**

A screenshot of a computer program

AI-generated content may be incorrect.

**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); } }

Pom.xml:

**CODE:**

<?xml version="1.0" encoding="UTF-8"?>  
<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>org.example</groupId>  
 <artifactId>junit\_mandatory\_basic\_1</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>17</maven.compiler.source>  
 <maven.compiler.target>17</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- JUnit 5 -->  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>5.9.3</version>  
 <scope>test</scope>  
 </dependency>  
  
 <!-- Mockito -->  
 <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>5.11.0</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.22.2</version>  
 </plugin>  
 </plugins>  
 </build>  
</project>

ExternalApi.java:

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

MyService.java:

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

MyServiceTest.java:

package org.example;  
  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testExternalApi() {  
 // Step 1: Create mock  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 // Step 2: Stub the method  
 *when*(mockApi.getData()).thenReturn("Mock Data");  
  
 // Step 3: Inject mock into service and test  
 MyService service = new MyService(mockApi);  
 String result = service.fetchData();  
  
 // Step 4: Assert the result  
 *assertEquals*("Mock Data", result);  
 }  
}

**OUTPUT:**

A screenshot of a computer program

AI-generated content may be incorrect.

**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.

**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();

}

}

Pom.XML:

<?xml version="1.0" encoding="UTF-8"?>  
<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>org.example</groupId>  
 <artifactId>junit\_mandatory\_basic\_1</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>17</maven.compiler.source>  
 <maven.compiler.target>17</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- JUnit 5 -->  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>5.9.3</version>  
 <scope>test</scope>  
 </dependency>  
  
 <!-- Mockito -->  
 <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>5.11.0</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.22.2</version>  
 </plugin>  
 </plugins>  
 </build>  
</project>

ExternalApi.java:

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

MyService.java:

package org.example;  
public class MyService{  
 private ExternalApi api;  
 public MyService(ExternalApi api){  
 this.api = api;  
 }  
 public String fetchData(){  
 return api.getData(); // delegate to external API  
 }  
}

MyServiceTest.java:

package org.example;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
public class MyServiceTest{  
 @Test  
 public void testVerifyInteraction(){  
 // Step 1: Create mock  
 ExternalApi mockApi=mock(ExternalApi.class);  
 // Step 2: Inject into service and call method  
 MyService service=new MyService(mockApi);  
 service.fetchData(); // should call mockApi.getData()  
 // Step 3: Verify interaction  
 verify(mockApi).getData(); // verify that getData() was called  
 }  
}

**OUTPUT:**

A screenshot of a computer program

AI-generated content may be incorrect.

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

Task: Write a Java application that demonstrates logging error messages and warning levels

using SLF4J.

**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");

}

}

Pom.XML:

<?xml version="1.0" encoding="UTF-8"?>  
<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>org.example</groupId>  
 <artifactId>Loging-demo</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>24</maven.compiler.source>  
 <maven.compiler.target>24</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
 <dependencies>  
 <!-- SLF4J API -->  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>1.7.30</version>  
 </dependency>  
  
 <!-- Logback (SLF4J Implementation) -->  
 <dependency>  
 <groupId>ch.qos.logback</groupId>  
 <artifactId>logback-classic</artifactId>  
 <version>1.2.3</version>  
 </dependency>  
 </dependencies>  
  
  
</project>

LoggingExample.java:

package org.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");  
 logger.info("This is an info message");  
 }  
}

**OUTPUT:**

A screenshot of a computer program

AI-generated content may be incorrect.