**WEEK-2(PLSQL AND JUNIT):**

**TABLE CREATION:**

**TABLE 1:CUSTOMERS**

CREATE TABLE Customers (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance NUMBER,

    LastModified DATE

);

**TABLE 2:ACCOUNTS**

CREATE TABLE Accounts (

    AccountID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    AccountType VARCHAR2(20),

    Balance NUMBER,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

)

**TABLE 3:TRANSACTIONS**

CREATE TABLE Transactions (

    TransactionID NUMBER PRIMARY KEY,

    AccountID NUMBER,

    TransactionDate DATE,

    Amount NUMBER,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

**TABLE 4:LOANS**

CREATE TABLE Loans (

    LoanID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    LoanAmount NUMBER,

    InterestRate NUMBER,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

**TABLE 5:EMPLOYEES**

CREATE TABLE Employees (

    EmployeeID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DepartmentID NUMBER,

    Salary NUMBER

);

**INSERTION:**

**CUSTOMERS:**

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

VALUES (1, 'SSS', TO\_DATE('1990-06-15', 'YYYY-MM-DD'), 7000, SYSDATE);

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

VALUES (2, 'SHA', TO\_DATE('1962-05-15', 'YYYY-MM-DD'), 15000, SYSDATE);

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

VALUES (3, 'ALINA', TO\_DATE('1962-04-23', 'YYYY-MM-DD'), 10000, SYSDATE);

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

VALUES (4, 'TOBA', TO\_DATE('1963-02-28', 'YYYY-MM-DD'), 13000, SYSDATE);

SELECT \* FROM CUSTOMERS;

**LOANS:**

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, 8000, 7, SYSDATE, ADD\_MONTHS(SYSDATE, 70));

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

VALUES (3, 3, 10000, 6, SYSDATE, ADD\_MONTHS(SYSDATE, 80));

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

VALUES (4, 4, 6000, 3, SYSDATE, ADD\_MONTHS(SYSDATE, 40));

SELECT \* FROM Loans;

**ACCOUNTS:**

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

VALUES (1, 1, 'Savings', 7000, SYSDATE);

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

VALUES (2, 2, 'Current', 15000, SYSDATE);

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

VALUES (3, 3, 'Savings', 10000, SYSDATE);

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

VALUES (4, 4, 'Savings', 13000, SYSDATE);

SELECT \* FROM Accounts;

**EMPLOYEES:**

INSERT INTO Employees VALUES (1, 'Alice', 101, 50000);

INSERT INTO Employees VALUES (2, 'Bob', 101, 60000);

INSERT INTO Employees VALUES (3, 'Charlie', 102, 55000);

SELECT \* FROM Employees;

**HANDS-ON 1: Control Structures**

**Scenario 1:**

BEGIN

    FOR rec IN (

        SELECT I.LoanID, D.CustomerID, I.InterestRate,

               TRUNC(MONTHS\_BETWEEN(SYSDATE, D.DOB) / 12) AS age

        FROM Customers D

        JOIN Loans I ON D.CustomerID = I.CustomerID

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

    )

    LOOP

        UPDATE Loans

        SET InterestRate = InterestRate - 1

        WHERE LoanID = rec.LoanID;

        DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount to Loan ID: ' || rec.LoanID ||

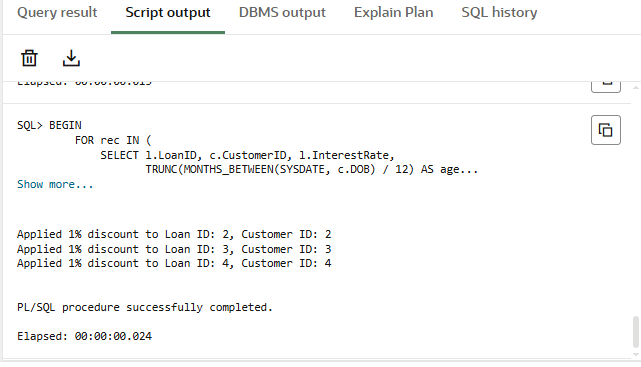
                             ', Customer ID: ' || rec.CustomerID);

    END LOOP;

    COMMIT;

END;

**OUTPUT:**

****

**Scenario 2:**

BEGIN

    FOR recs IN (

        SELECT CustomerID, Balance FROM Customers WHERE Balance > 10000

    )

    LOOP

        UPDATE Customers

        SET IsVIP = 'TRUE'

        WHERE CustomerID = recs.CustomerID;

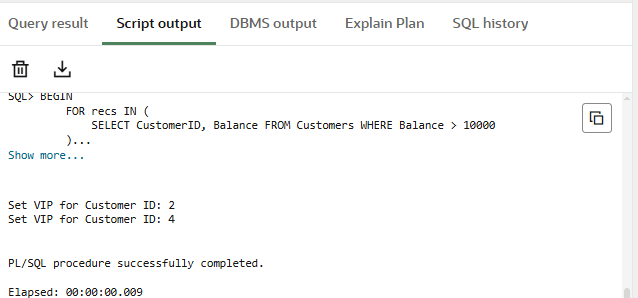
        DBMS\_OUTPUT.PUT\_LINE('Set VIP for Customer ID: ' || recs.CustomerID);

    END LOOP;

    COMMIT;

END;

**Output:**

****

**Scenario 3:**

BEGIN

    FOR rec IN (

        SELECT LoanID, CustomerID, EndDate

        FROM Loans

        WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30

    )

    LOOP

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

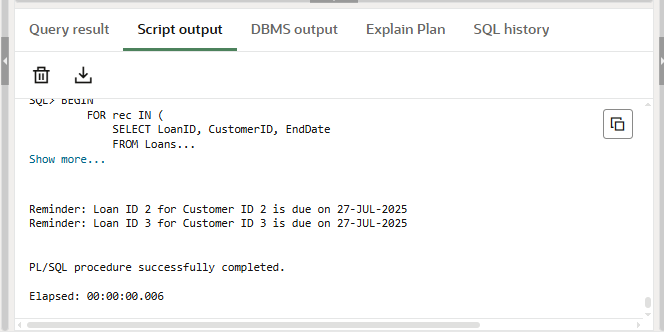
                             ' for Customer ID ' || rec.CustomerID ||

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

    END LOOP;

END;

**OUTPUT:**

****

**HANDS-ON 2: Stored Procedures**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

  FOR acc\_rec IN (

    SELECT AccountID, Balance

    FROM Accounts

    WHERE AccountType = 'Savings'

  )

  LOOP

    UPDATE Accounts

    SET Balance = Balance + (Balance \* 0.01),

        LastModified = SYSDATE

    WHERE AccountID = acc\_rec.AccountID;

    DBMS\_OUTPUT.PUT\_LINE('1% interest applied to Account ID: ' || acc\_rec.AccountID);

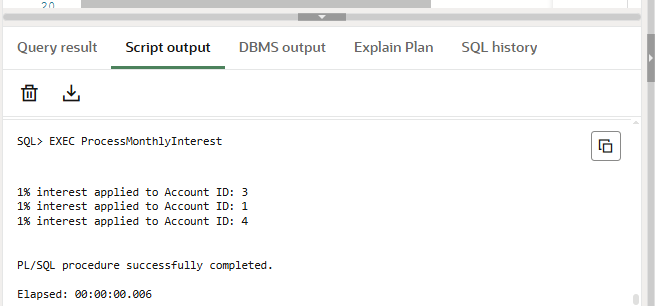
  END LOOP;

  COMMIT;

END;

EXEC ProcessMonthlyInterest;

**OUTPUT:**

****

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

    p\_DepartmentID IN NUMBER,

    p\_BonusPercent IN NUMBER

) AS

BEGIN

    FOR emp\_rec IN (

        SELECT EmployeeID, Salary

        FROM Employees

        WHERE DepartmentID = p\_DepartmentID

    )

    LOOP

        UPDATE Employees

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

        WHERE EmployeeID = emp\_rec.EmployeeID;

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

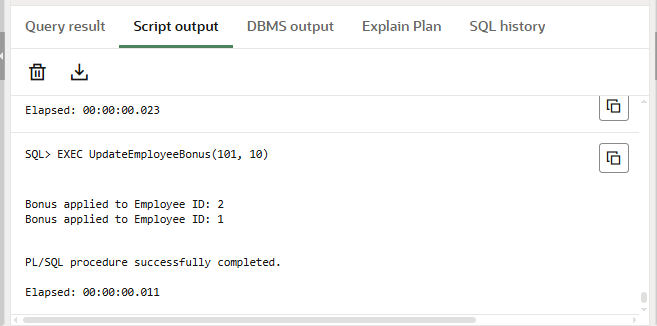
    END LOOP;

    COMMIT;

END;

EXEC UpdateEmployeeBonus(101, 10);

**OUTPUT:**

****

**Scenario 3:**

CREATE OR REPLACE PROCEDURE TransferFunds (

    p\_SourceAccountID IN NUMBER,

    p\_TargetAccountID IN NUMBER,

    p\_Amount IN NUMBER

) AS

    v\_SourceBalance NUMBER;

BEGIN

    IF p\_Amount <= 0 THEN

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

    END IF;

    SELECT Balance INTO v\_SourceBalance

    FROM Accounts

    WHERE AccountID = p\_SourceAccountID;

    IF v\_SourceBalance < p\_Amount THEN

        RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance in source account.');

    END IF;

    UPDATE Accounts

    SET Balance = Balance - p\_Amount,

        LastModified = SYSDATE

    WHERE AccountID = p\_SourceAccountID;

    UPDATE Accounts

    SET Balance = Balance + p\_Amount,

        LastModified = SYSDATE

    WHERE AccountID = p\_TargetAccountID;

    DBMS\_OUTPUT.PUT\_LINE('₹' || p\_Amount || ' transferred from Account ' ||

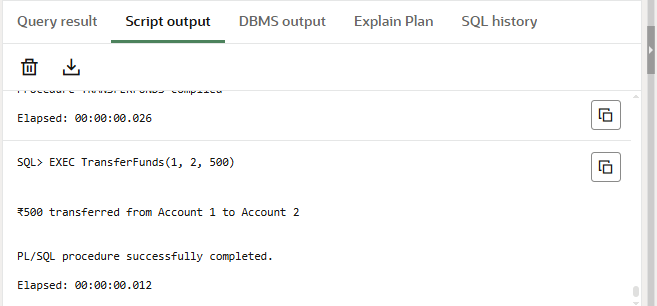
                         p\_SourceAccountID || ' to Account ' || p\_TargetAccountID);

    COMMIT;

END;

EXEC TransferFunds(1, 2, 500);

**OUTPUT:**

****

**DEPENDENCY:**

<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>my\_project</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</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>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- Mockito Core -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>4.11.0</version>

<scope>test</scope>

</dependency>

<!-- Mockito with JUnit 5 Integration -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-junit-jupiter</artifactId>

<version>4.11.0</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>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.1.2</version>

<configuration>

<useModulePath>**false**</useModulePath>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Hands-on 3: Setting Up JUnit**

* **Prime.java**

**public** **class** Prime{

**public** **boolean** PrimeCheck(**int** num) {

**if** (num <= 1) {

**return** **false**;

}

**if**(num == 2) {

**return** **true**;

}

**for** (**int** i = 2; i <= Math.*sqrt*(num); i++) {

**if** (num % i == 0) {

**return** **false**;

}

}

**return** **true**;

}

* **test.java**

**import** org.junit.Test;

**import** **static** org.junit.Assert.\*;

**public** **class** test {

Prime check = **new** Prime();

@Test

**public** **void** testPrime() {

assertTrue(check.PrimeCheck(3));

assertTrue(check.PrimeCheck(13));

assertTrue(check.PrimeCheck(97));

assertFalse(check.PrimeCheck(0));

assertFalse(check.PrimeCheck(1));

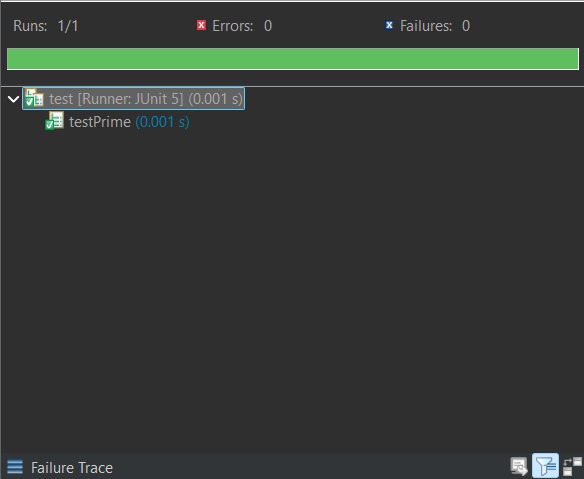
assertFalse(check.PrimeCheck(4));

assertFalse(check.PrimeCheck(100));

}

}

**Output:**

****

**Hands-on 4: Assertions in JUnit**

* **Code.java**

**public** **class** Code {

**public** **int** add (**int** a, **int** b) {

**return** a + b;

}

**public** **boolean** even(**int** c) {

**if** (c % 2 == 0) {

**return** **true**;

}

**return** **false**;

}

}

* **test.java**

**import** org.junit.Test;

**import** **static** org.junit.Assert.\*;

**public** **class** test {

Code c = **new** Code();

@Test

**public** **void** test\_asserts() {

assertEquals(3,c.add(2, 1));

assertNotEquals(6,c.add(2, 1));

assertTrue(c.even(2));

assertFalse(c.even(3));

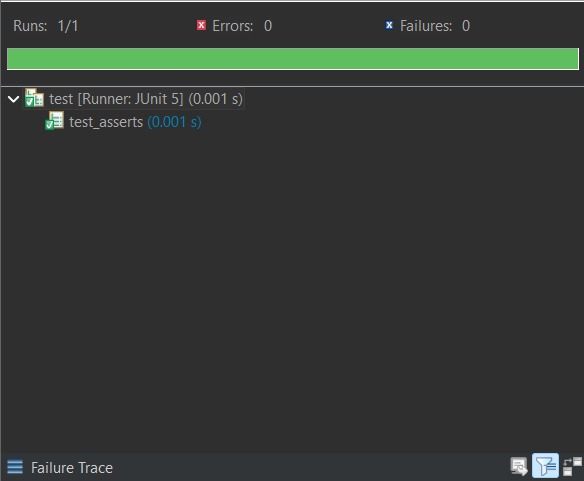
assertNull(**null**);

assertNotNull(c);

}

}

**OUTPUT:**

****

**Hands-on 5: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit**

* **Code.java**

**public** **class** Code {

**public** **int** add (**int** a, **int** b) {

**return** a + b;

}

**public** **boolean** even(**int** c) {

**if** (c % 2 == 0) {

**return** **true**;

}

**return** **false**;

}

}

* **Test.java**

**import** org.junit.Before;

**import** org.junit.After;

**import** org.junit.Test;

**import** **static** org.junit.Assert.\*;

**public** **class** test{

Code c = **new** Code();

@Before

**public** **void** setup() {

System.***out***.println("\*Setup (@Before)\*");

System.***out***.println("Setting up for tests");

System.***out***.println("Testing did not start yet");

}

@After

**public** **void** teardown() {

System.***out***.println("\*TearDown (@After)\*");

System.***out***.println("Testing Done, Check your code if error raises");

}

@Test

**public** **void** test\_AAA() {

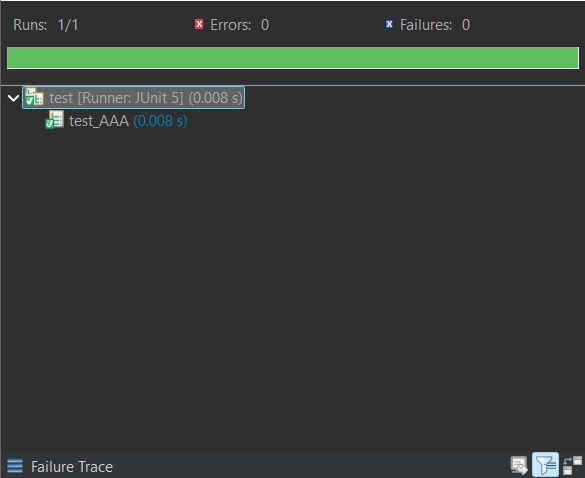
**int** result = c.add(2, 1);

assertEquals(3,result);

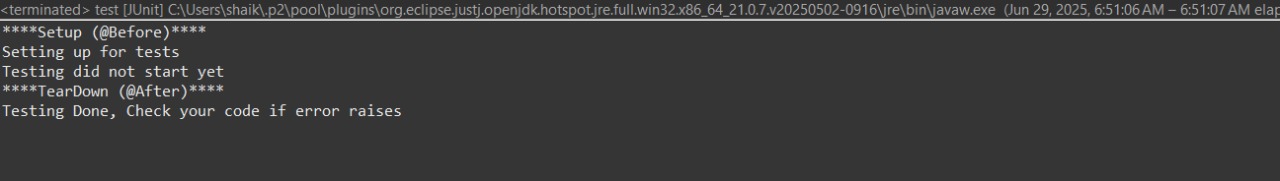
}

}

**Output 1:**

****

**Output 2:**

****

**Hands-on 6:** **Mocking and Stubbing**

**public** **interface** ExternalApi {

String getData();

}

**public** **class** MyService {

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** String fetchData() {

**return** api.getData();

}

}

* **Test.java**

**import** **static** org.mockito.Mockito.\*;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** org.junit.jupiter.api.Test;

**import** org.mockito.Mockito;

**public** **class** test {

@Test

**public** **void** testExternalApi() {

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

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

MyService service = **new** MyService(mockApi);

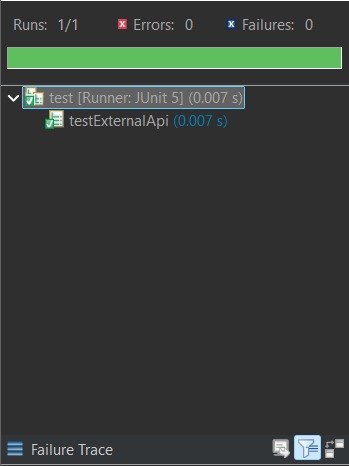
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**Output:**

****

**Hands-on 7:** **Verifying Interactions**

**public** **interface** ExternalApi {

String getData(String s);

}

**public** **class** MyService {

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** **void** fetchData(String s) {

api.getData(String s);

}

}

* **Test.java**

**import** **static** org.mockito.Mockito.\*;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** org.junit.jupiter.api.Test;

**import** org.mockito.Mockito;

**public** **class** test {

@Test

**public** **void** testVerifyInteraction() {

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

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

MyService service = **new** MyService(mockApi);

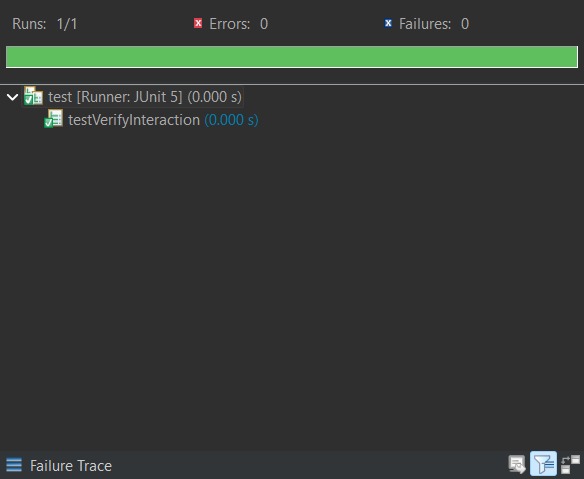
String result = service.fetchDataForUser("shafreen");

assertEquals("Mock ", result);

verify(mockApi).getData("shafreen");

    }

}

**Output:**

**Hands-on 8: Logging Error Messages and Warning Levels**

* **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) {

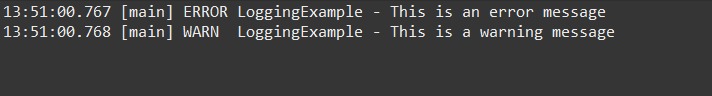
***logger***.error("This is an error message");

***logger***.warn("This is a warning message");

}

}

**Output:**

****