**MANDATORY HANDS** - **ON WEEK 2**

**PL/SQL**

**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.

Code:

-- Creating Tables and Inserting Values into Required Tables

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

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

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

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

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

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

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

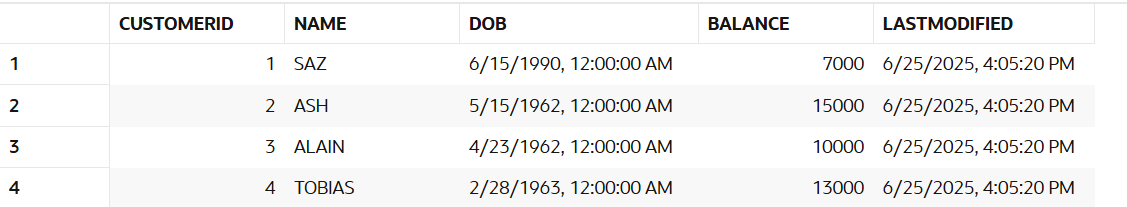
INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (2, 2, 7000, 7, SYSDATE, ADD\_MONTHS(SYSDATE, 70));

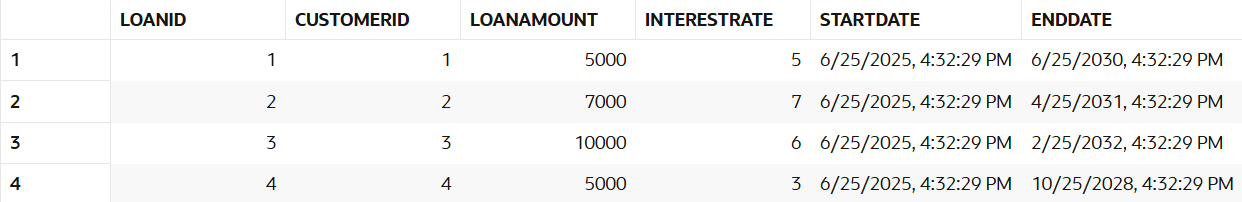
INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (3, 3, 10000, 6, SYSDATE, ADD\_MONTHS(SYSDATE, 80));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (4, 4, 5000, 3, SYSDATE, ADD\_MONTHS(SYSDATE, 40));

SELECT \* FROM CUSTOMERS;



SELECT \* FROM Loans;



BEGIN

FOR RECORDS IN (SELECT CustomerID, DOB FROM CUSTOMERS) LOOP

IF (MONTHS\_BETWEEN(SYSDATE, RECORDS.DOB))/12 > 60 THEN

UPDATE LOANS

SET InterestRate = InterestRate -1

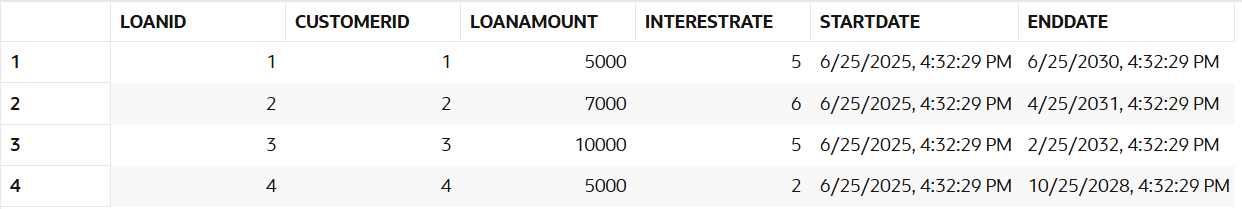
WHERE CustomerID = RECORDS.CustomerID;

END IF;

END LOOP;

END;

SELECT \* FROM Loans;



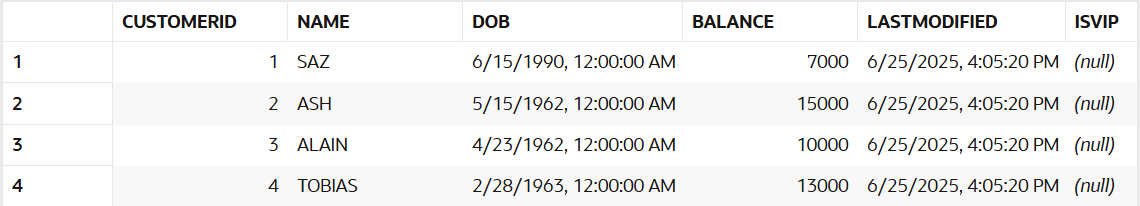
Scenario 2: A customer can be promoted to VIP status based on their balance.

* + 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:

ALTER TABLE Customers ADD IsVip VARCHAR2(3);

SELECT \* FROM Customers;



BEGIN

FOR RECORDS IN (SELECT BALANCE FROM Customers) LOOP

IF RECORDS.Balance > 10000 THEN

UPDATE Customers

SET IsVip = 'YES'

WHERE Balance = RECORDS.Balance;

ELSE

UPDATE Customers

SET IsVip = 'NO'

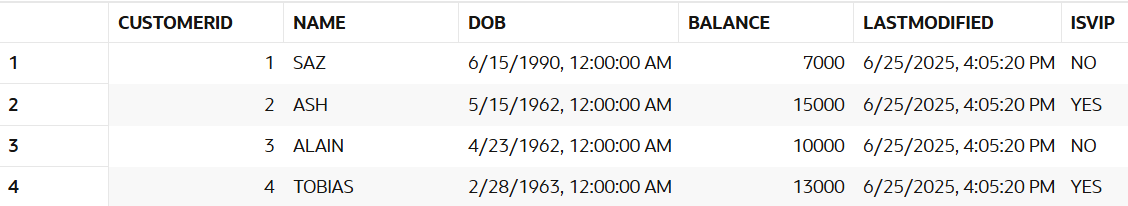
WHERE Balance = RECORDS.Balance;

END IF;

END LOOP;

END;

SELECT \* FROM Customers;



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:

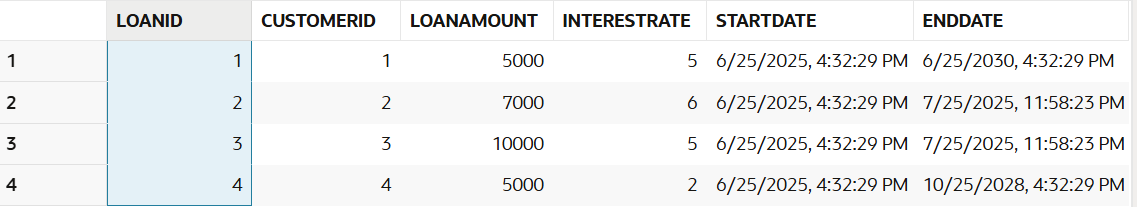
-- Updating End Dates to check the working of the query

UPDATE Loans

SET EndDate = SYSDATE + 30

WHERE LoanID BETWEEN 2 AND 3;

SELECT \* FROM Loans;



DECLARE

CUSTOMER\_NAME VARCHAR2(10);

BEGIN

FOR RECORDS IN (SELECT LoanID, CustomerID, EndDate FROM Loans WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP

SELECT Name INTO CUSTOMER\_NAME

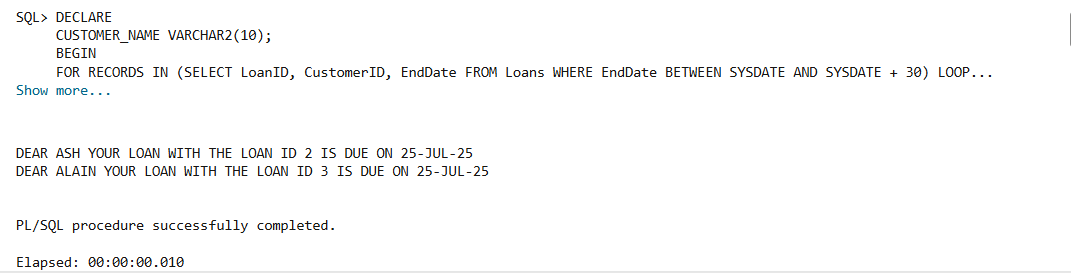
FROM Customers

WHERE CustomerID = RECORDS.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('DEAR ' || CUSTOMER\_NAME || ' YOUR LOAN WITH THE LOAN ID ' || RECORDS.LoanID || ' IS DUE ON ' || RECORDS.EndDate);

END LOOP;

END;



**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 Accounts (

    AccountID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    AccountType VARCHAR2(20),

    Balance NUMBER,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID));

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);

-- PROCEDURE -> ProcessMonthlyInterest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

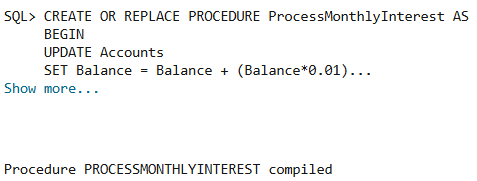
BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance\*0.01)

WHERE AccountType = 'Savings';

END;



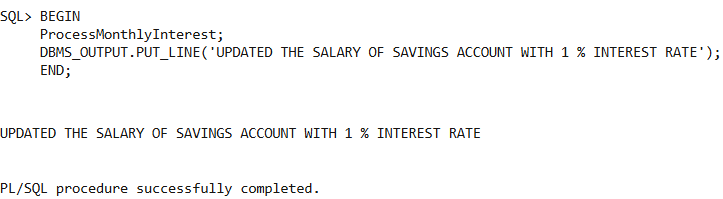
-- PL/SQL SCRIPT TO CALL THE PROCEUDRE

BEGIN

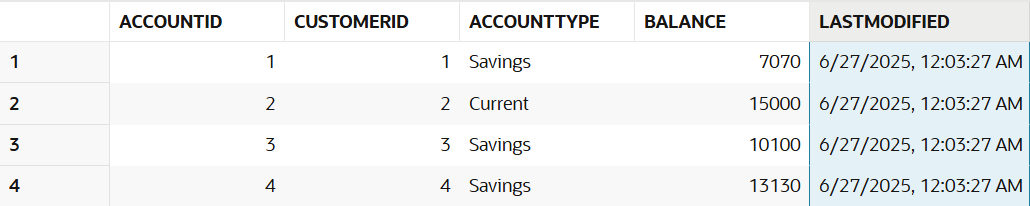
ProcessMonthlyInterest;

DBMS\_OUTPUT.PUT\_LINE('UPDATED THE SALARY OF SAVINGS ACCOUNT WITH 1 % INTEREST RATE');

END;



SELECT \* FROM Accounts;



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

)

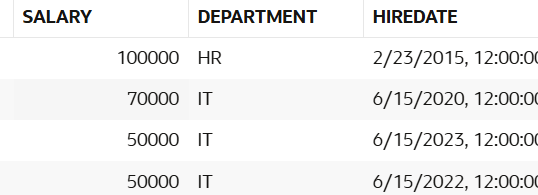
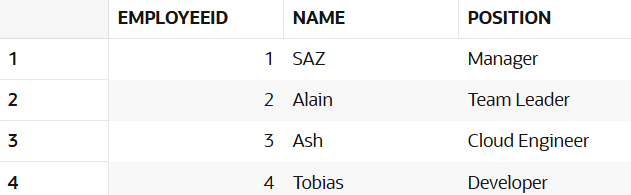
INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (1, 'SAZ', 'Manager', 100000, 'HR', TO\_DATE('2015-02-23', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (2, 'Alain', 'Team Leader', 70000, 'IT', TO\_DATE('2020-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (3, 'Ash', 'Cloud Engineer', 50000, 'IT', TO\_DATE('2023-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate) VALUES (4, 'Tobias', 'Developer', 50000, 'IT', TO\_DATE('2022-06-15', 'YYYY-MM-DD'));

SELECT \* FROM Employees;



-- PROCEDURE -> UpdateEmployeeBonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (Dep IN VARCHAR2, Bonus IN NUMBER) AS

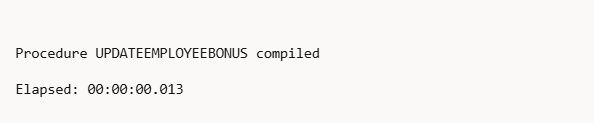
BEGIN

UPDATE Employees

SET SALARY = SALARY + (SALARY\*(Bonus\*0.01))

WHERE Department = Dep;

END;

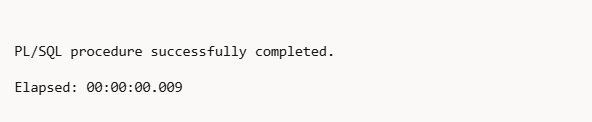


-- PL/SQL FOR CALLING THE PROCEDURE

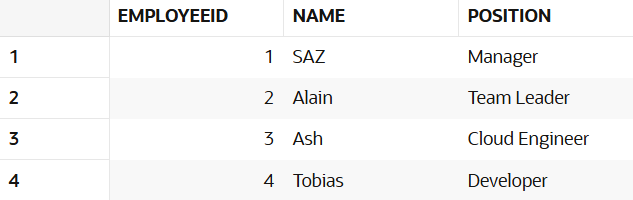
BEGIN

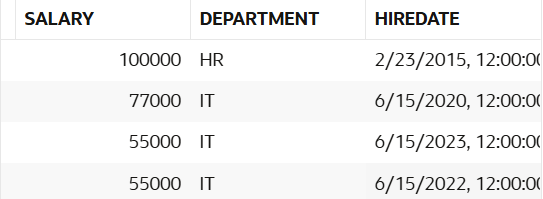
UpdateEmployeeBonus('IT', 10);

END;



SELECT \* FROM Employees;



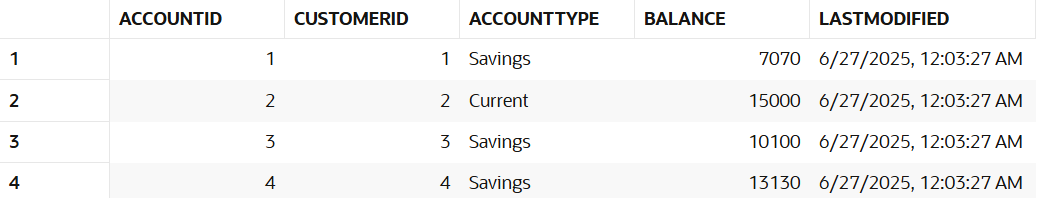


Scenario 3: Customers should be able to transfer funds between their accounts.

* + 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:

SELECT \* FROM Accounts;



CREATE OR REPLACE PROCEDURE TransferFunds (SENDER IN NUMBER, RECIEVER IN NUMBER, AMOUNT IN NUMBER) AS

SENDER\_AMOUNT NUMBER;

BEGIN

SELECT Balance INTO SENDER\_AMOUNT

FROM Accounts

WHERE AccountID = SENDER;

IF SENDER\_AMOUNT > AMOUNT THEN

UPDATE Accounts

SET Balance = Balance - AMOUNT

WHERE AccountID = SENDER;

UPDATE Accounts

SET Balance = Balance + AMOUNT

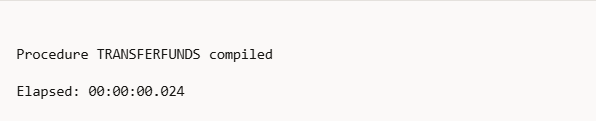
WHERE AccountID = RECIEVER;

ELSE

DBMS\_OUTPUT.PUT\_LINE('INSUFFICIENT BALANCE');

END IF;

END;

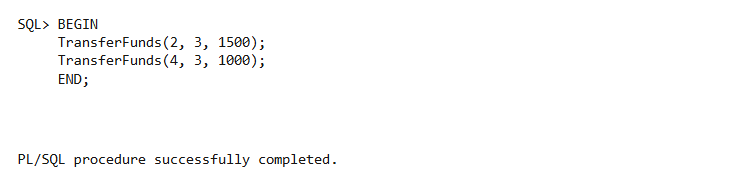


BEGIN

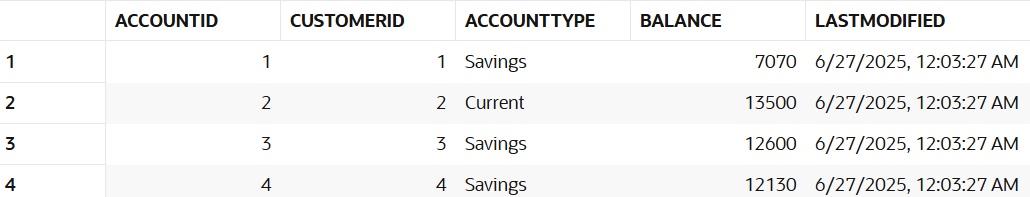
TransferFunds(2, 3, 1500);

TransferFunds(4, 3, 1000);

END;



SELECT \* FROM Accounts;



**TDD (JUNIT, MOCKITO), LOGGING**

DEPENDENCIES FORTESTING (XML FILE):

<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>UTF8 </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>

**JUNIT:**

Exercise 1: Setting Up JUnit

Code:

// Code block that is to be tested

public class prime\_test {

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 Class for testing code block(Tested using the libraries in maven dependencies)

import org.junit.Test;

import static org.junit.Assert.\*;

public class test {

prime\_test check = new prime\_test();

@Test

public void testPrime() {

assertTrue(check.PrimeCheck(3));

assertTrue(check.PrimeCheck(13));

assertTrue(check.PrimeCheck(97));

assertFalse(check.PrimeCheck(0));

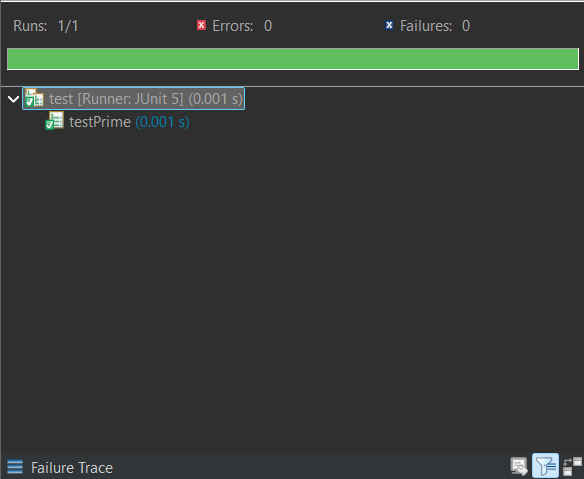
assertFalse(check.PrimeCheck(1));

assertFalse(check.PrimeCheck(4));

assertFalse(check.PrimeCheck(100));

}

}

Output:

Exercise 3: Assertions in JUnit

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

Code:

// Code block that is to be tested

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 class to test the above code block

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));

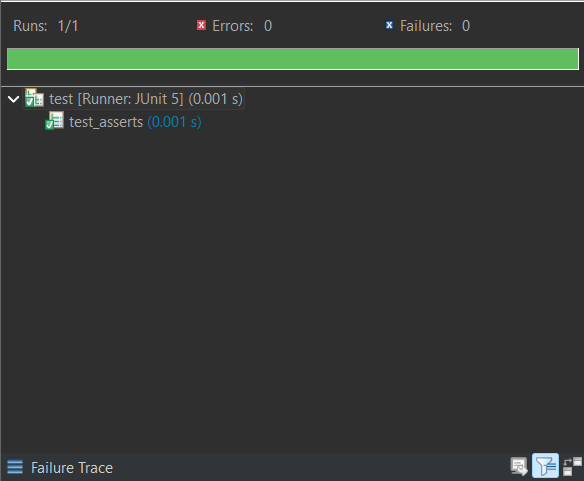
assertFalse(c.even(3));

assertNull(null);

assertNotNull(c);

}

}

Output: 

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.

Code:

// Code block that is to be tested

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 class for testing the above code block

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class test{

// Arrange part of AAA Pattern

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() {

// Act part of AAA Pattern

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

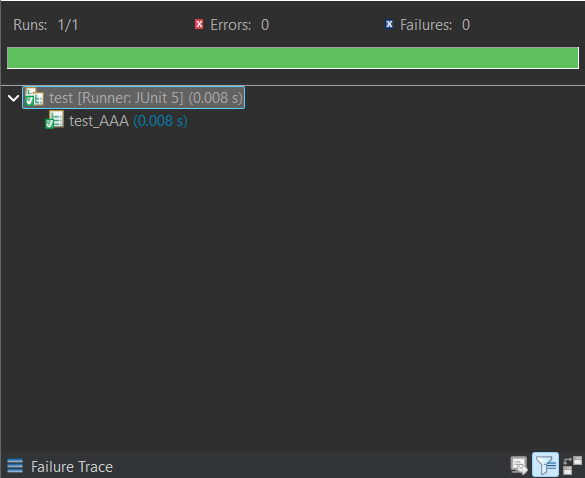
// Assert part of AAA Pattern

assertEquals(3,result);

}

}

Output:



**MOCKITO:**

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.

Code:

// API

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 class to test the API using Mockito

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() {

// Create mock

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

// Stub method

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

// Inject into service

MyService service = new MyService(mockApi);

// Act

String result = service.fetchData();

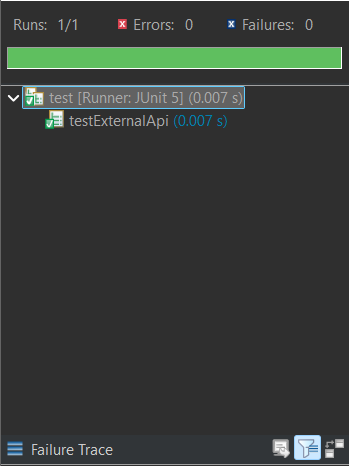
// Assert

assertEquals("Mock Data", result);

}

}

Output:



Exercise 2: Verifying Interactions

Scenario: You need to ensure that a method is called with specific arguments.

Code:

// API

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 Class to test using Mockito

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("SAZ")).thenReturn("Mock");

MyService service = new MyService(mockApi);

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

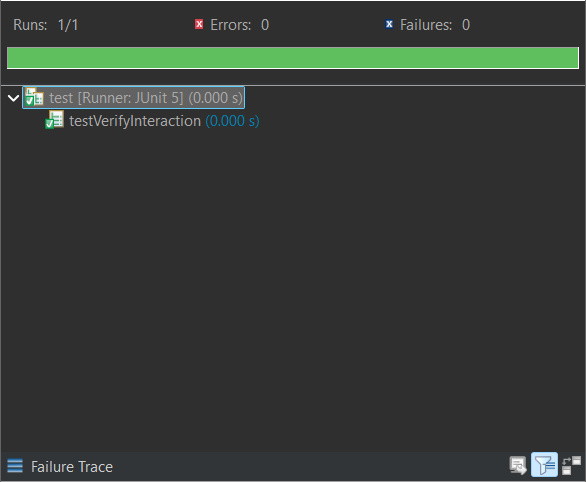
assertEquals("Mock ", result);

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

}

}

Output:



**LOGGING:**

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 {

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:

