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.

Answer:

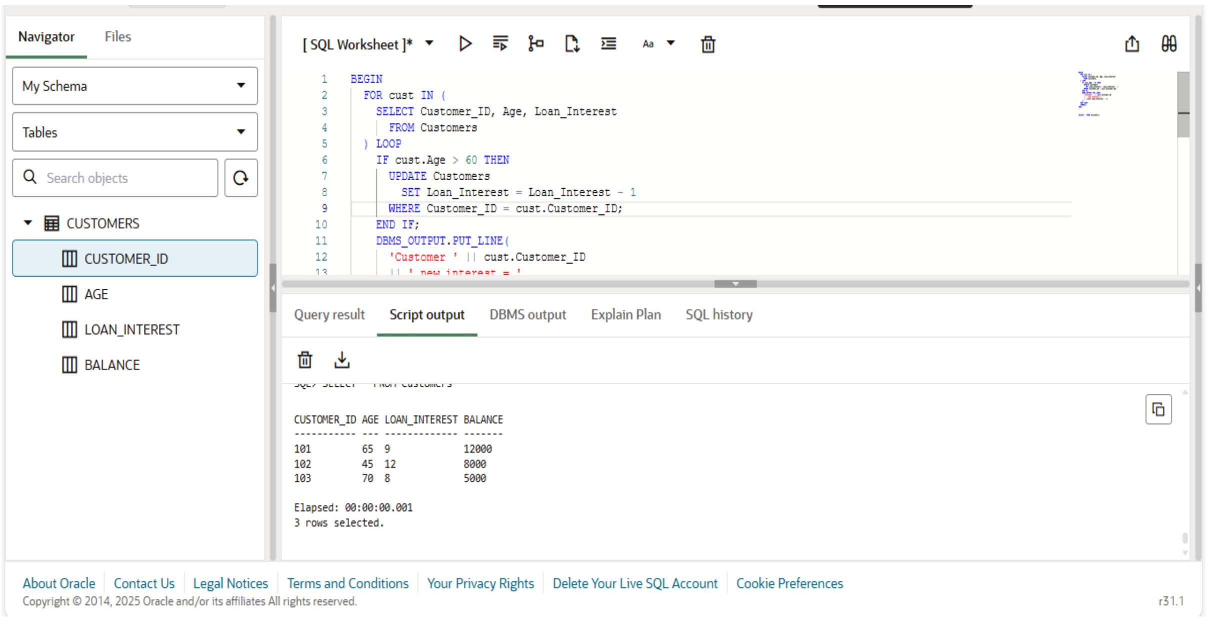
1.Scenario 1:

```
BEGIN
 FOR cust IN (
  SELECT Customer_ID, Age, Loan_Interest
   FROM Customers
 ) LOOP
  IF cust.Age > 60 THEN
   UPDATE Customers
    SET Loan_Interest = Loan_Interest - 1
   WHERE Customer_ID = cust.Customer_ID;
  END IF;
  DBMS_OUTPUT.PUT_LINE(
   'Customer ' || cust.Customer_ID
   || ' new interest = '
   || (cust.Loan_Interest - 1)
  );
 END LOOP;
```

```
  COMMIT;
END;
/
```

```
SELECT * FROM Customers;
```

Output:



```
BEGIN
  FOR cust IN (
    SELECT Customer_ID, Age, Loan_Interest
      FROM Customers
  ) LOOP
    IF cust.Age > 60 THEN
      UPDATE Customers
        SET Loan_Interest = Loan_Interest - 1
      WHERE Customer_ID = cust.Customer_ID;
    END IF;
    DBMS_OUTPUT.PUT_LINE(
      'Customer ' || cust.Customer_ID
      || ' new interest = '
```

```
CUSTOMER_ID AGE LOAN_INTEREST BALANCE
----------- --- ------------- -------
101         65  9             12000
102         45  12            8000
103         70  8             5000

Elapsed: 00:00:00.001
3 rows selected.
```

About Oracle
Contact Us
Legal Notices
Terms and Conditions
Your Privacy Rights
Delete Your Live SQL Account
Cookie Preferences
Copyright © 2014, 2025 Oracle and/or its affiliates All rights reserved.
r31.1

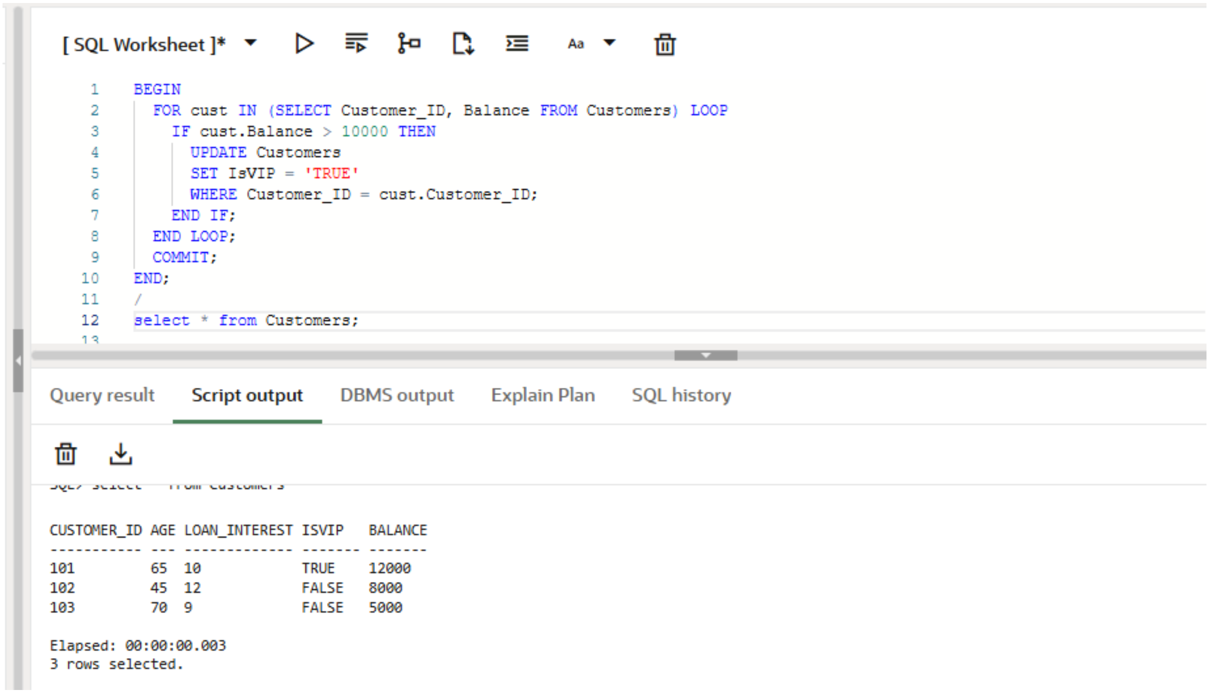
2.Scenario 2:

```
BEGIN
  FOR cust IN (SELECT Customer_ID, Balance FROM Customers) LOOP
    IF cust.Balance > 10000 THEN
      UPDATE Customers
      SET IsVIP = 'TRUE'
      WHERE Customer_ID = cust.Customer_ID;
    END IF;
  END LOOP;
  COMMIT;
END;
```

/

select * from Customers;

Output:



```
BEGIN
  FOR cust IN (SELECT Customer_ID, Balance FROM Customers) LOOP
    IF cust.Balance > 10000 THEN
      UPDATE Customers
      SET IsVIP = 'TRUE'
      WHERE Customer_ID = cust.Customer_ID;
    END IF;
  END LOOP;
  COMMIT;
END;
/
select * from Customers;
```

```
CUSTOMER_ID AGE LOAN_INTEREST ISVIP   BALANCE
----------- --- ------------- ------- -------
101          65  10            TRUE    12000
102          45  12            FALSE   8000
103          70  9             FALSE   5000

Elapsed: 00:00:00.003
3 rows selected.
```

3.Session 3

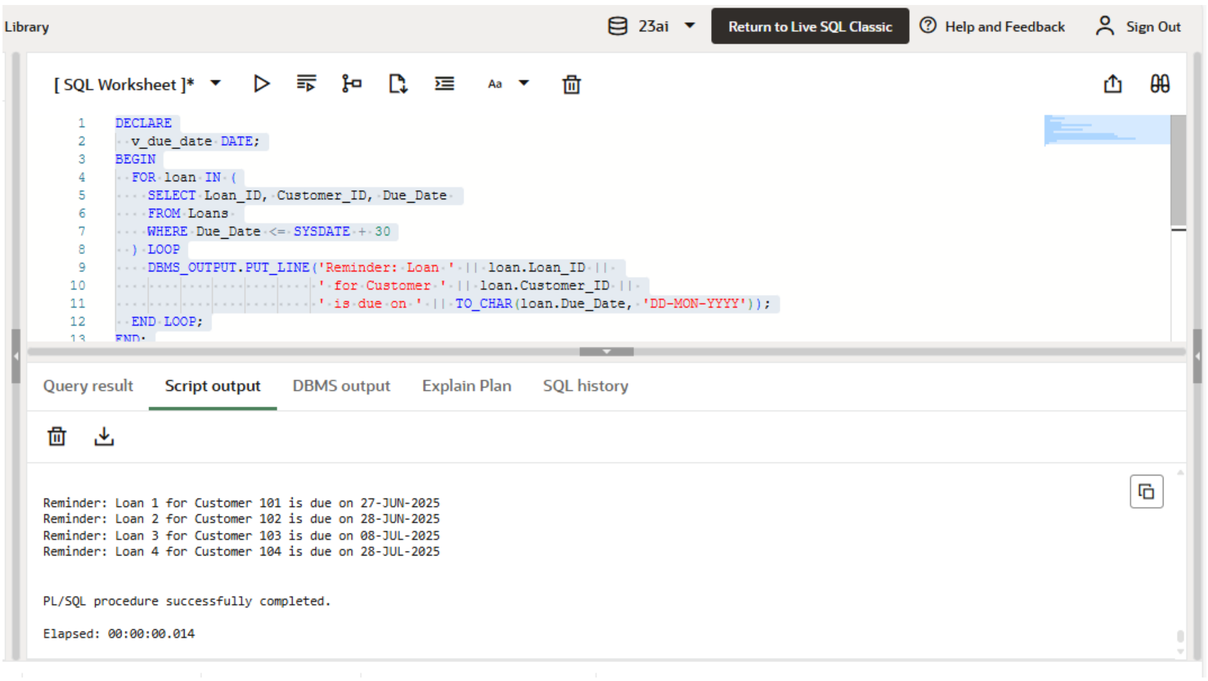
```
DECLARE
 v_due_date DATE;
BEGIN
 FOR loan IN (
  SELECT Loan_ID, Customer_ID, Due_Date
  FROM Loans
  WHERE Due_Date <= SYSDATE + 30
 ) LOOP
  DBMS_OUTPUT.PUT_LINE('Reminder: Loan ' || loan.Loan_ID ||
             ' for Customer ' || loan.Customer_ID ||
             ' is due on ' || TO_CHAR(loan.Due_Date, 'DD-MON-YYYY'));
```

END LOOP;

END;

/

Output:



```
DECLARE
  v_due_date DATE;
BEGIN
  FOR loan IN (
    SELECT Loan_ID, Customer_ID, Due_Date
    FROM Loans
    WHERE Due_Date <= SYSDATE + 30
  ) LOOP
    DBMS_OUTPUT.PUT_LINE('Reminder: Loan ' || loan.Loan_ID ||
                         ' for Customer ' || loan.Customer_ID ||
                         ' is due on ' || TO_CHAR(loan.Due_Date, 'DD-MON-YYYY'));
  END LOOP;
END;
```

```
Reminder: Loan 1 for Customer 101 is due on 27-JUN-2025
Reminder: Loan 2 for Customer 102 is due on 28-JUN-2025
Reminder: Loan 3 for Customer 103 is due on 08-JUL-2025
Reminder: Loan 4 for Customer 104 is due on 28-JUL-2025


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.014
```

Exercise 2: 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.

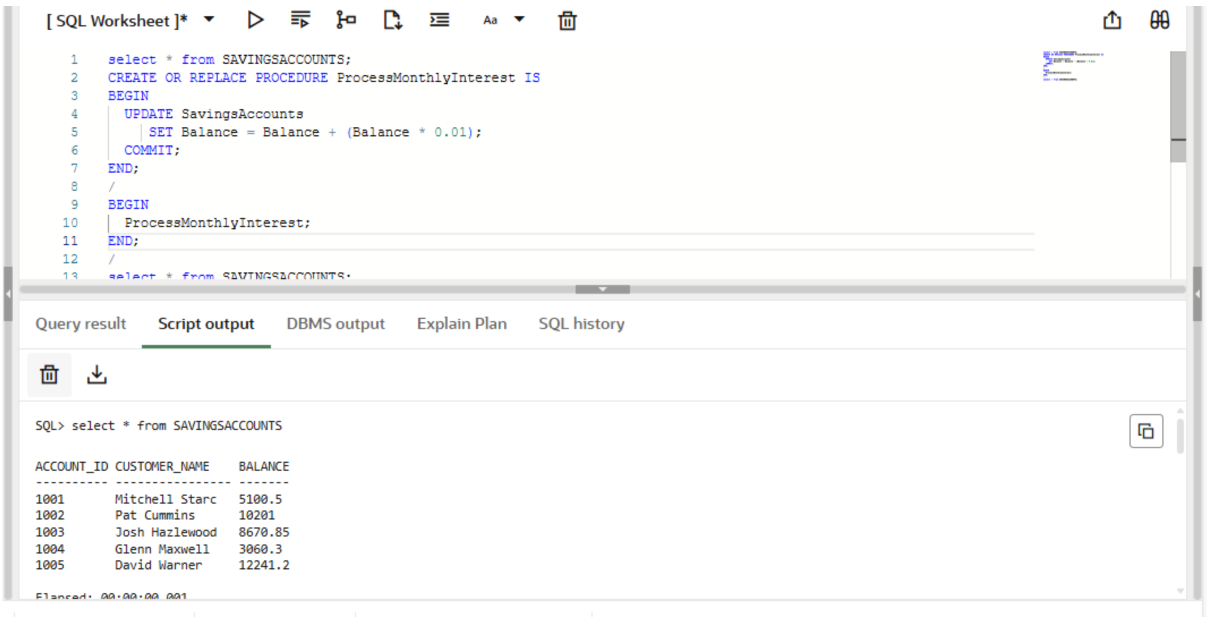
Answer:

Session 1:

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
  UPDATE SavingsAccounts
     SET Balance = Balance + (Balance * 0.01);
  COMMIT;
END;
/
```

Output:

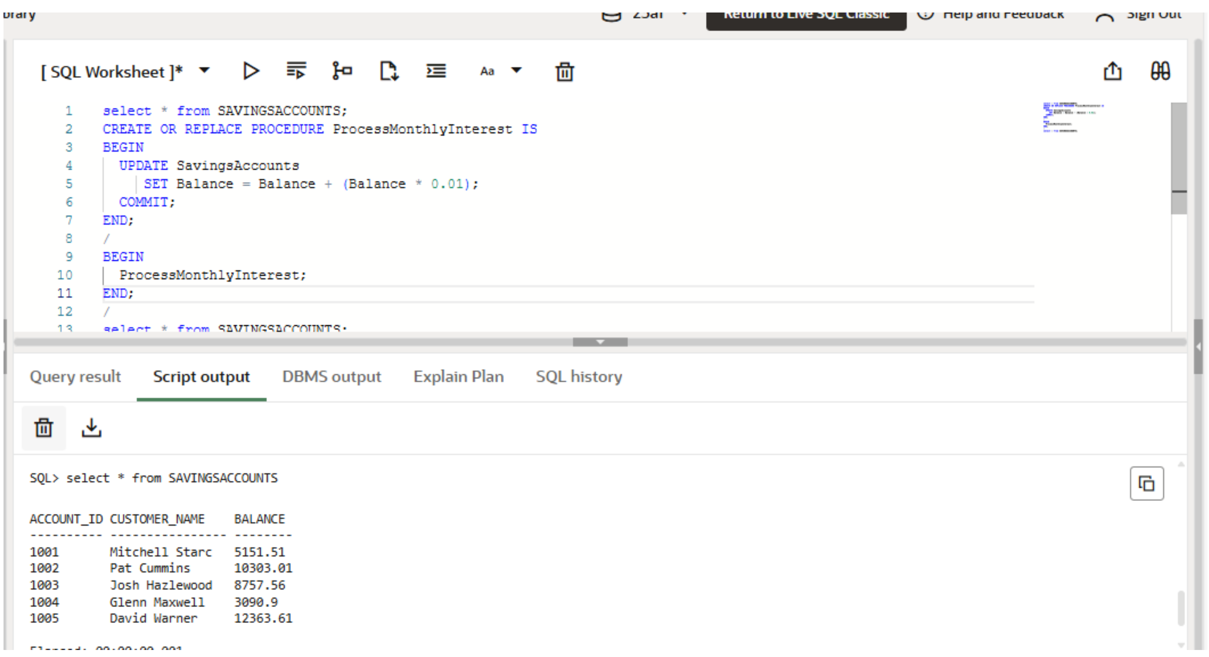
Before update:



```
select * from SAVINGSACCOUNTS;
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
  UPDATE SavingsAccounts
     SET Balance = Balance + (Balance * 0.01);
  COMMIT;
END;
/
BEGIN
  ProcessMonthlyInterest;
END;
/
select * from SAVINGSACCOUNTS;
```

```
SQL> select * from SAVINGSACCOUNTS

ACCOUNT_ID CUSTOMER_NAME    BALANCE
---------- --------------- -------
1001       Mitchell Starc   5100.5
1002       Pat Cummins      10201
1003       Josh Hazlewood   8670.85
1004       Glenn Maxwell    3060.3
1005       David Warner     12241.2

Elapsed: 00:00:00.001
```

After Update:



```
select * from SAVINGSACCOUNTS;
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
  UPDATE SavingsAccounts
     SET Balance = Balance + (Balance * 0.01);
  COMMIT;
END;
/
BEGIN
  ProcessMonthlyInterest;
END;
/
```

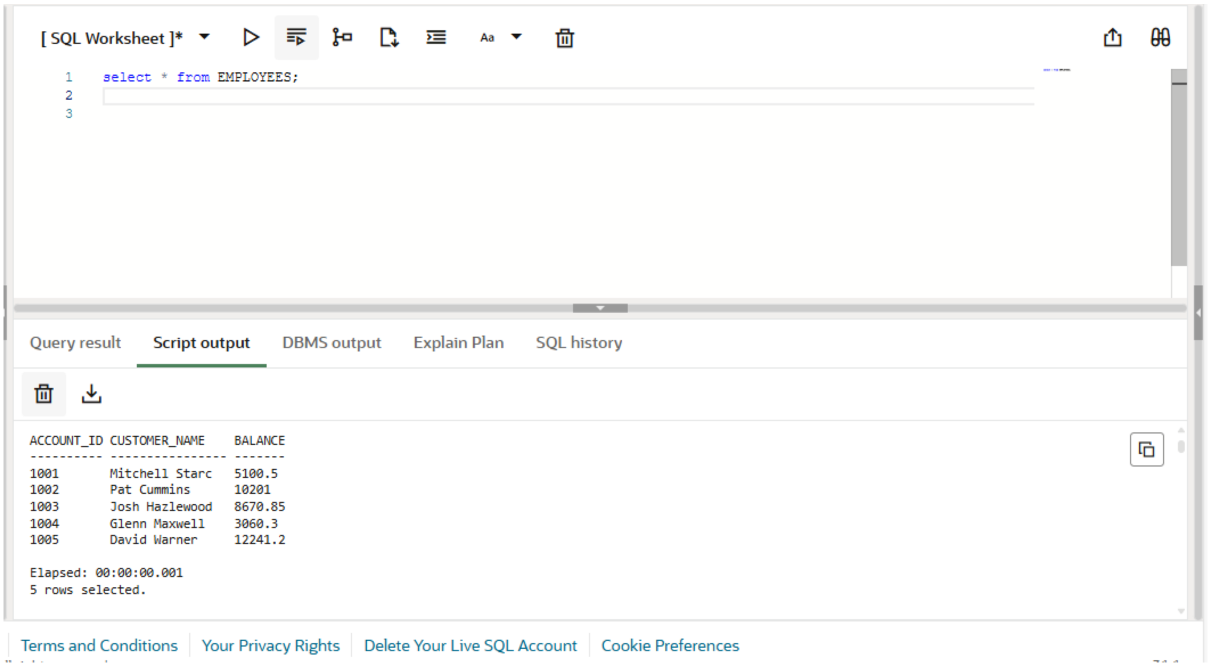
```
SQL> select * from SAVINGSACCOUNTS

ACCOUNT_ID CUSTOMER_NAME   BALANCE
---------- --------------- --------
1001       Mitchell Starc  5151.51
1002       Pat Cummins     10303.01
1003       Josh Hazlewood  8757.56
1004       Glenn Maxwell   3090.9
1005       David Warner    12363.61
```

2.Session 2:

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
  p_dept_id  IN  NUMBER,
  p_bonus_pc IN  NUMBER
) IS
BEGIN
  UPDATE Employees
     SET Salary = Salary + (Salary * p_bonus_pc / 100)
   WHERE Department_ID = p_dept_id;
  COMMIT;
END UpdateEmployeeBonus;
/
BEGIN
  UpdateEmployeeBonus(10, 10);
END;
/
select * from EMPLOYEES;
```

Output:

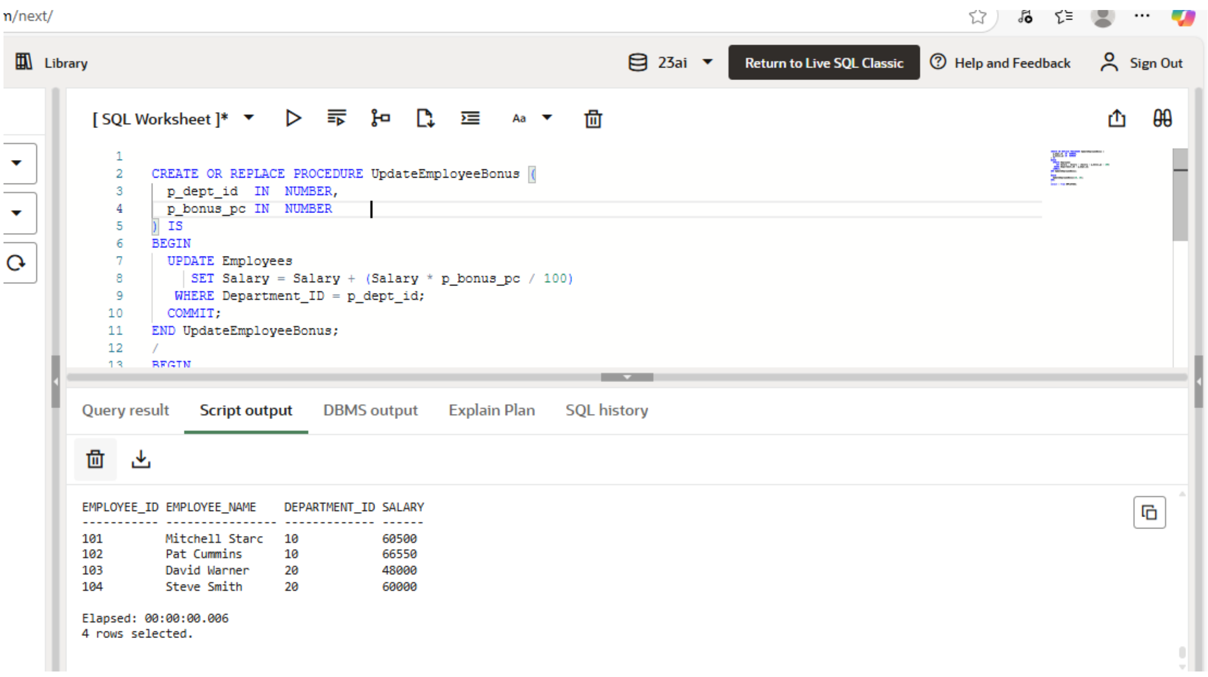
Before update:



```
select * from EMPLOYEES;
```

```
ACCOUNT_ID CUSTOMER_NAME    BALANCE
---------- ---------------- -------
1001       Mitchell Starc   5100.5
1002       Pat Cummins      10201
1003       Josh Hazlewood   8670.85
1004       Glenn Maxwell    3060.3
1005       David Warner     12241.2

Elapsed: 00:00:00.001
5 rows selected.
```

After update:



```

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
  p_dept_id  IN  NUMBER,
  p_bonus_pc IN  NUMBER
) IS
BEGIN
  UPDATE Employees
     SET Salary = Salary + (Salary * p_bonus_pc / 100)
   WHERE Department_ID = p_dept_id;
  COMMIT;
END UpdateEmployeeBonus;
/
BEGIN
```

```
EMPLOYEE_ID EMPLOYEE_NAME    DEPARTMENT_ID SALARY
----------- ---------------- ------------- ------
101         Mitchell Starc   10            60500
102         Pat Cummins      10            66550
103         David Warner     20            48000
104         Steve Smith      20            60000

Elapsed: 00:00:00.006
4 rows selected.
```

3.Session 3:

CREATE OR REPLACE PROCEDURE TransferFunds (

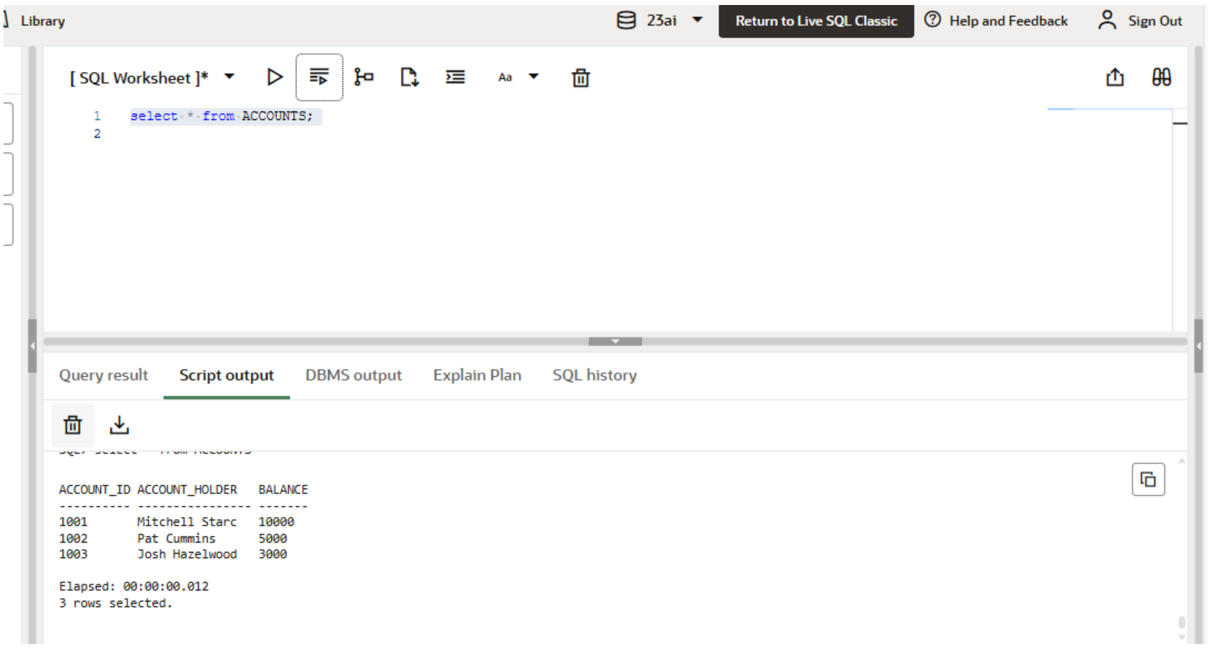
  p_from_acct IN NUMBER,

```
  p_to_acct    IN  NUMBER,
  p_amount     IN  NUMBER
) IS
  v_from_bal   NUMBER;
BEGIN
  SELECT Balance INTO v_from_bal
    FROM Accounts
   WHERE Account_ID = p_from_acct
   FOR UPDATE;

  IF v_from_bal < p_amount THEN
    RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds in source account');
  END IF;

  UPDATE Accounts
     SET Balance = Balance - p_amount
   WHERE Account_ID = p_from_acct;

  UPDATE Accounts
     SET Balance = Balance + p_amount
   WHERE Account_ID = p_to_acct;

  COMMIT;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20002, 'Account not found');
  WHEN OTHERS THEN
    ROLLBACK;
    RAISE;
END TransferFunds;
/
```

select * from ACCOUNTS;

Output:

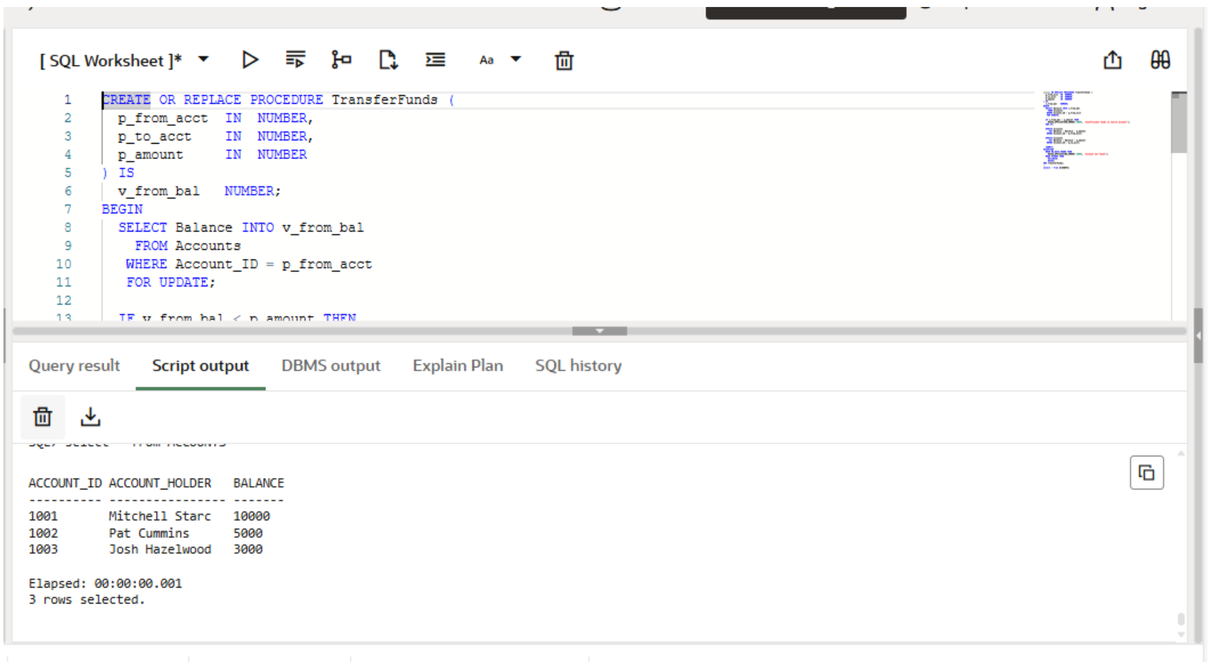
Before update:



```
select * from ACCOUNTS;
```

```
ACCOUNT_ID ACCOUNT_HOLDER   BALANCE
---------- --------------- -------
1001       Mitchell Starc   10000
1002       Pat Cummins      5000
1003       Josh Hazelwood   3000

Elapsed: 00:00:00.012
3 rows selected.
```

After update:



```
CREATE OR REPLACE PROCEDURE TransferFunds (
  p_from_acct   IN  NUMBER,
  p_to_acct     IN  NUMBER,
  p_amount      IN  NUMBER
) IS
  v_from_bal    NUMBER;
BEGIN
  SELECT Balance INTO v_from_bal
    FROM Accounts
   WHERE Account_ID = p_from_acct
   FOR UPDATE;

  IF v_from_bal < p_amount THEN
```

```
ACCOUNT_ID ACCOUNT_HOLDER   BALANCE
---------- --------------- -------
1001       Mitchell Starc   10000
1002       Pat Cummins      5000
1003       Josh Hazelwood   3000

Elapsed: 00:00:00.001
3 rows selected.
```

Exercise 3:

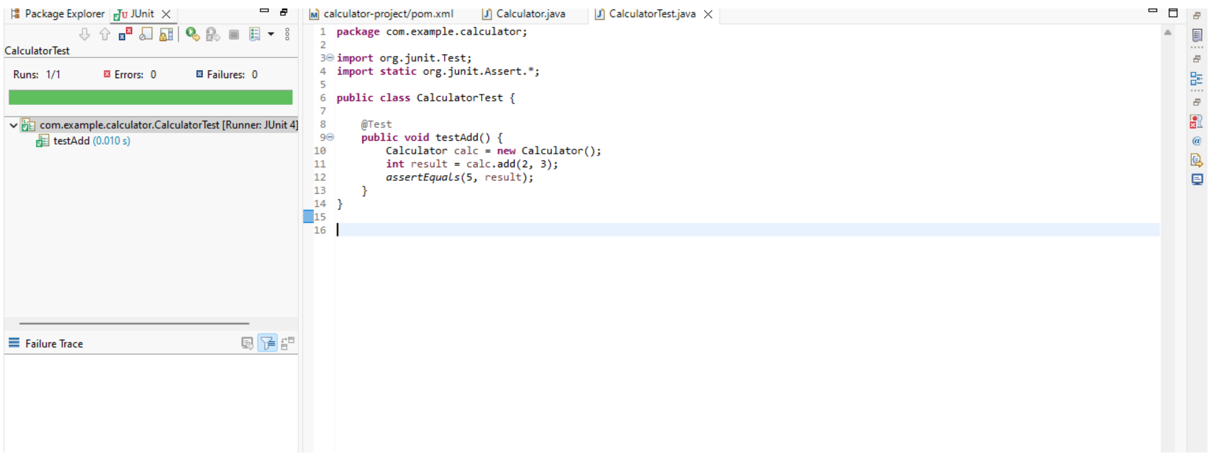
Setting Up JUnit Scenario: You need to set up JUnit in your Java project to start writing unit tests. Steps: 1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse). 2. Add JUnit dependency to your project. If you are

using Maven, add the following to your pom.xml: junit junit 4.13.2 test 3. Create a new test class in your project.

Answer:

```
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }
}
import org.junit.Test;
import static org.junit.Assert.*;

public class CalculatorTest {

    @Test
    public void testAdd() {
        Calculator calc = new Calculator();
        int result = calc.add(2, 3);
        assertEquals(5, result);
    }
}
```

Output:



Runs: 1/1 Errors: 0 Failures: 0

com.example.calculator.CalculatorTest [Runner: JUnit 4]
testAdd (0.010 s)

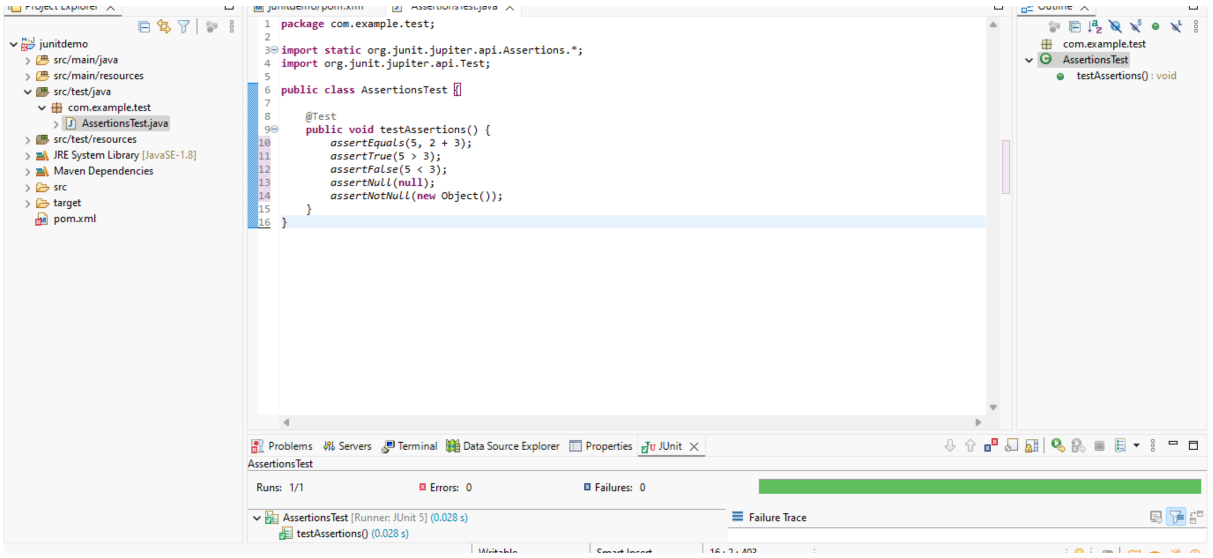
```java
package com.example.calculator;

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

public class CalculatorTest {

    @Test
    public void testAdd() {
        Calculator calc = new Calculator();
        int result = calc.add(2, 3);
        assertEquals(5, result);
    }
}

```

Exercise 4: 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.

Answer:

```java
package com.example.test;


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

import org.junit.jupiter.api.Test;


public class AssertionsTest {


    @Test

    public void testAssertions() {

        assertEquals(5, 2 + 3);

        assertTrue(5 > 3);

        assertFalse(5 < 3);

        assertNull(null);
        assertNotNull(new Object());

    }

}
```

Output:



```
package com.example.test;

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

public class AssertionsTest {

    @Test
    public void testAssertions() {
        assertEquals(5, 2 + 3);
        assertTrue(5 > 3);
        assertFalse(5 < 3);
        assertNull(null);
        assertNotNull(new Object());
    }
}
```

```
AssertionsTest
Runs: 1/1        Errors: 0        Failures: 0
AssertionsTest [Runner: JUnit 5] (0.028 s)
    testAssertions() (0.028 s)
```

Exercise 5: 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.

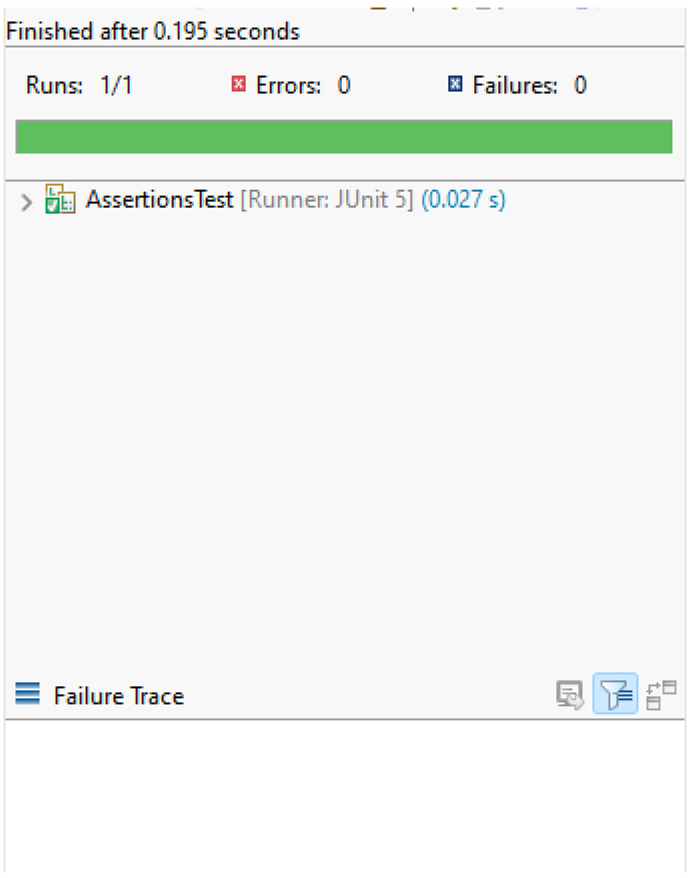
Answer:

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

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.AfterEach;

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

```
}

public class CalculatorTest {

  private Calculator calculator;

  @BeforeEach
  public void setUp() {
    System.out.println("Setting up...");
    calculator = new Calculator();
  }

  @AfterEach
  public void tearDown() {
    System.out.println("Cleaning up...");
    calculator = null;
  }

  @Test
  public void testAdd_PositiveNumbers() {
    int result = calculator.add(10, 20);
    assertEquals(30, result);
  }

  @Test
  public void testAdd_NegativeNumbers() {
    int result = calculator.add(-5, -3);
    assertEquals(-8, result);
  }

  @Test
```

```
    public void testAdd_PositiveAndNegative() {
        int result = calculator.add(5, -3);
        assertEquals(2, result);
    }
}
```

Output:



```
Finished after 0.195 seconds
Runs: 1/1        Errors: 0        Failures: 0
> AssertionsTest [Runner: JUnit 5] (0.027 s)
Failure Trace
```

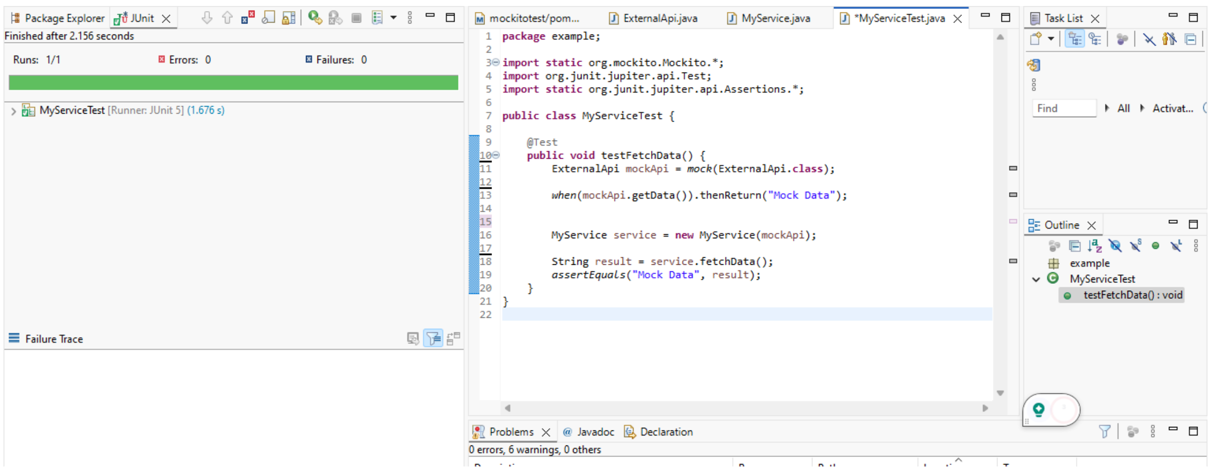
Exercise 6: Mocking and Stubbing Scenario: You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods. Steps: 1. Create a mock object for the external API. 2. Stub the methods to return predefined values. 3. Write a test case that uses the mock object.

Answer:

```
package example;
```

```java
public interface ExternalApi {
   String getData();
}
package example;
public class MyService {
   private ExternalApi api;

   public MyService(ExternalApi api) {
      this.api = api;
   }

   public String fetchData() {
      return api.getData();
   }
}
package example;

import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class MyServiceTest {

   @Test
   public void testFetchData() {
      ExternalApi mockApi = mock(ExternalApi.class);

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


      MyService service = new MyService(mockApi);

      String result = service.fetchData();
      assertEquals("Mock Data", result);
   }
}
```

Output:



```
Finished after 2.156 seconds
Runs: 1/1    Errors: 0    Failures: 0
MyServiceTest [Runner: JUnit 5] (1.676 s)
```

```
package example;

import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class MyServiceTest {

    @Test
    public void testFetchData() {
        ExternalApi mockApi = mock(ExternalApi.class);

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


        MyService service = new MyService(mockApi);

        String result = service.fetchData();
        assertEquals("Mock Data", result);
    }
}
```

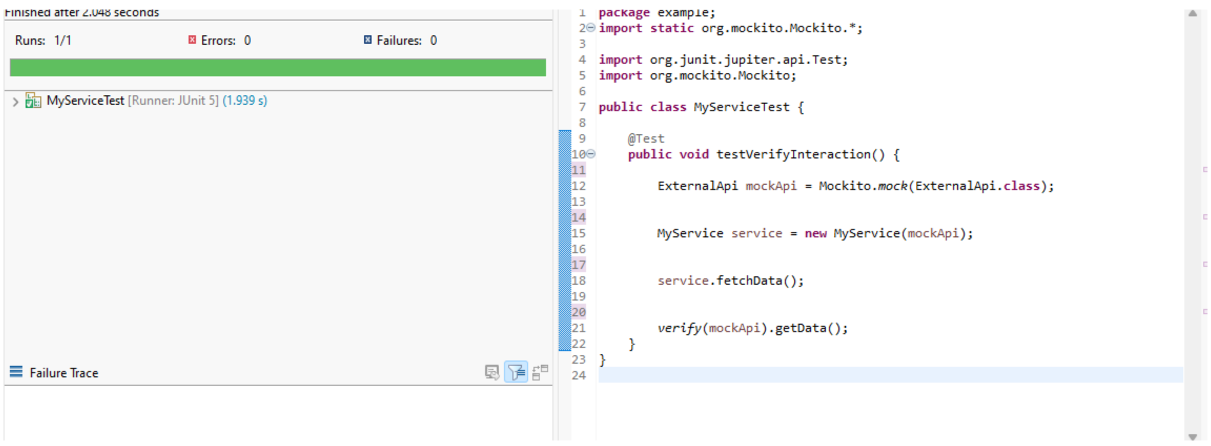
```
example
MyServiceTest
  testFetchData() : void
0 errors, 6 warnings, 0 others
```

Exercise 7: 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.

Answer:

```
package example;


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


public class MyService {
   private ExternalApi api;


   public MyService(ExternalApi api) {
      this.api = api;
   }


   public String fetchData() {
      return api.getData();
   }
}
package example;
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();
    }
}
```

Output:



```
Finished after 2.048 seconds
Runs: 1/1        Errors: 0        Failures: 0

> MyServiceTest [Runner: JUnit 5] (1.939 s)

Failure Trace
```

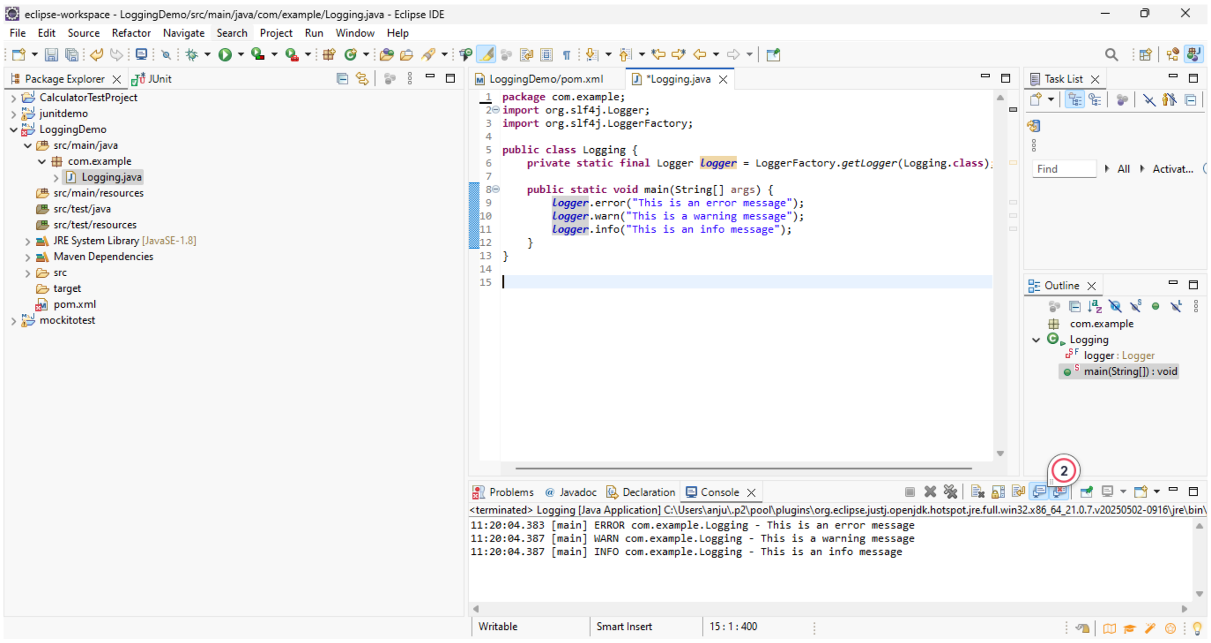
```
package example;
import static org.mockito.Mockito.*;

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();
    }
}
```

Exercise 8: Logging Error Messages and Warning Levels Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J. Step-by-Step Solution: 1. Add SLF4J and Logback dependencies to your `pom.xml` file: org.slf4j slf4j-api 1.7.30 ch.qos.logback logback-classic 1.2.3 2. Create a Java class that uses SLF4J for logging

Answer:

```
package com.example;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;


public class Logging {

    private static final Logger logger = LoggerFactory.getLogger(Logging.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:



eclipse-workspace - LoggingDemo/src/main/java/com/example/Logging.java - Eclipse IDE

```
package com.example;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Logging {
    private static final Logger logger = LoggerFactory.getLogger(Logging.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");
    }
}
```

```
<terminated> Logging [Java Application] C:\Users\anju\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\
11:20:04.383 [main] ERROR com.example.Logging - This is an error message
11:20:04.387 [main] WARN com.example.Logging - This is a warning message
11:20:04.387 [main] INFO com.example.Logging - This is an info message
```