WEEK 2

PL/SQL programming

Exercise 1: Control Structures

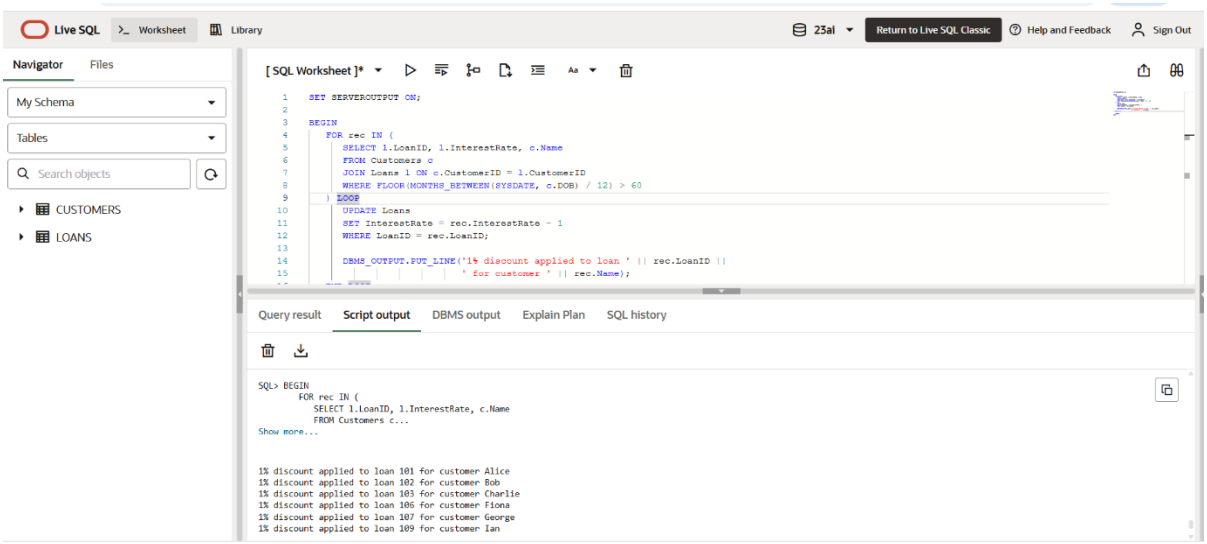
Scenario 1: The bank wants to apply a discount to loan interest rates for customers above 60 years old.

```
CREATE TABLE Customers (
    CustomerID NUMBER PRIMARY KEY,
    Name VARCHAR2(100),
    DOB DATE,
    Balance NUMBER,
    LastModified DATE,
    IsVIP CHAR(1) DEFAULT 'N'
);
CREATE TABLE Loans (
    LoanID NUMBER PRIMARY KEY,
    CustomerID NUMBER,
    LoanAmount NUMBER,
    InterestRate NUMBER,
    StartDate DATE,
    EndDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

SET SERVEROUTPUT ON;

BEGIN
   FOR rec IN (
```

```
    SELECT l.LoanID, l.InterestRate, c.Name
    FROM Customers c
    JOIN Loans l ON c.CustomerID = l.CustomerID
    WHERE FLOOR(MONTHS_BETWEEN(SYSDATE, c.DOB) / 12) > 60
  ) LOOP
    UPDATE Loans
    SET InterestRate = rec.InterestRate - 1
    WHERE LoanID = rec.LoanID;
    DBMS_OUTPUT.PUT_LINE('1% discount applied to loan ' || rec.LoanID ||
                         ' for customer ' || rec.Name);
  END LOOP;
  COMMIT;
END;
```



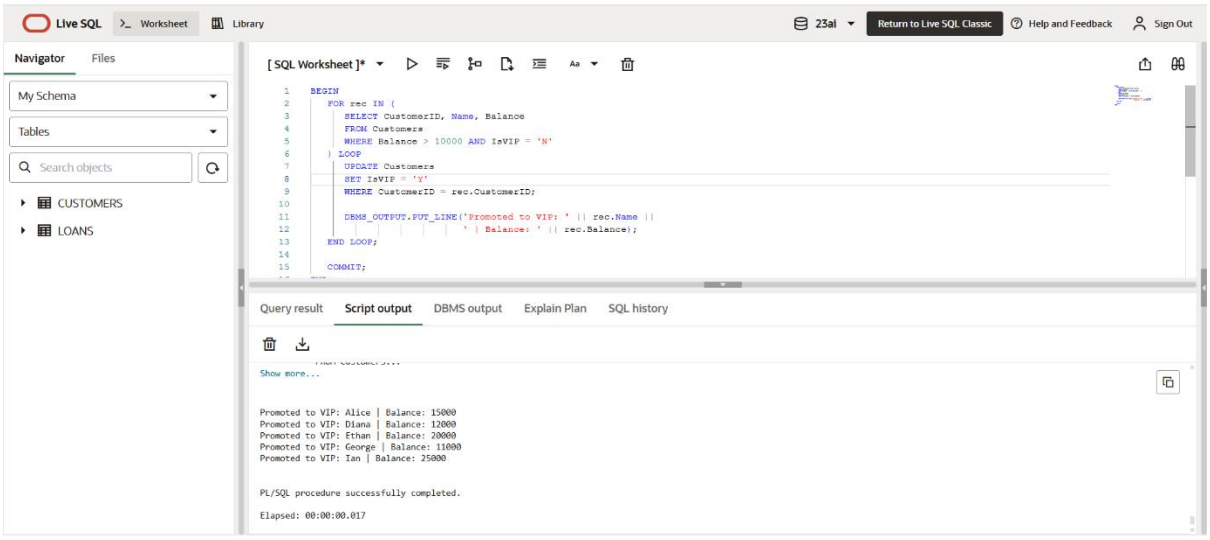
Scenario 2: A customer can be promoted to VIP status based on their balance.

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.

```
BEGIN
  FOR rec IN (
    SELECT CustomerID, Name, Balance
```

```
    FROM Customers
    WHERE Balance > 10000 AND IsVIP = 'N'
  ) LOOP
    UPDATE Customers
    SET IsVIP = 'Y'
    WHERE CustomerID = rec.CustomerID;

    DBMS_OUTPUT.PUT_LINE('Promoted to VIP: ' || rec.Name ||
                         ' | Balance: ' || rec.Balance);
  END LOOP;
  COMMIT;
END;
```



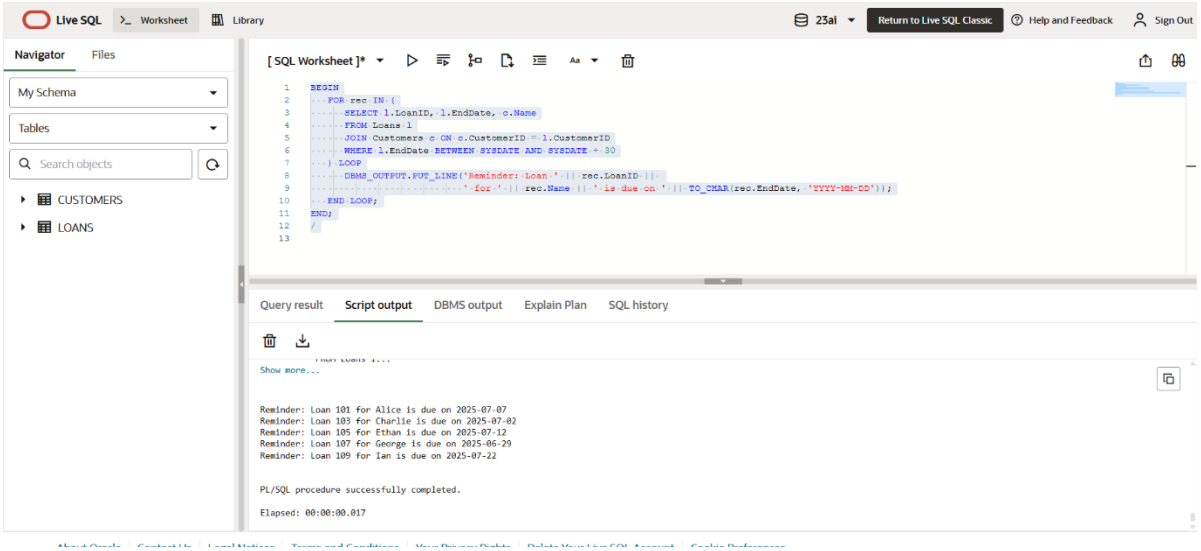
Scenario 3: The bank wants to send reminders to customers whose loans are due within the next 30 days.

- **Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

  Output:

```
BEGIN
  FOR rec IN (
    SELECT l.LoanID, l.EndDate, c.Name
```

```
    FROM Loans l
    JOIN Customers c ON c.CustomerID = l.CustomerID
    WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30
  ) LOOP
    DBMS_OUTPUT.PUT_LINE('Reminder: Loan ' || rec.LoanID ||
                         ' for ' || rec.Name || ' is due on ' || TO_CHAR(rec.EndDate, 'YYYY-MM-DD'));
  END LOOP;
END;
```



Exercise 3: Stored Procedures

```
CREATE TABLE Transactions (
    TransactionID NUMBER PRIMARY KEY,
    AccountID NUMBER,
    TransactionDate DATE,
    Amount NUMBER,
    TransactionType VARCHAR2(20)
);
CREATE TABLE Employees (
    EmployeeID NUMBER PRIMARY KEY,
    Name VARCHAR2(100),
```

```
    Position VARCHAR2(50),
    Salary NUMBER(10,2),
    Department VARCHAR2(50),
    HireDate DATE
);
```

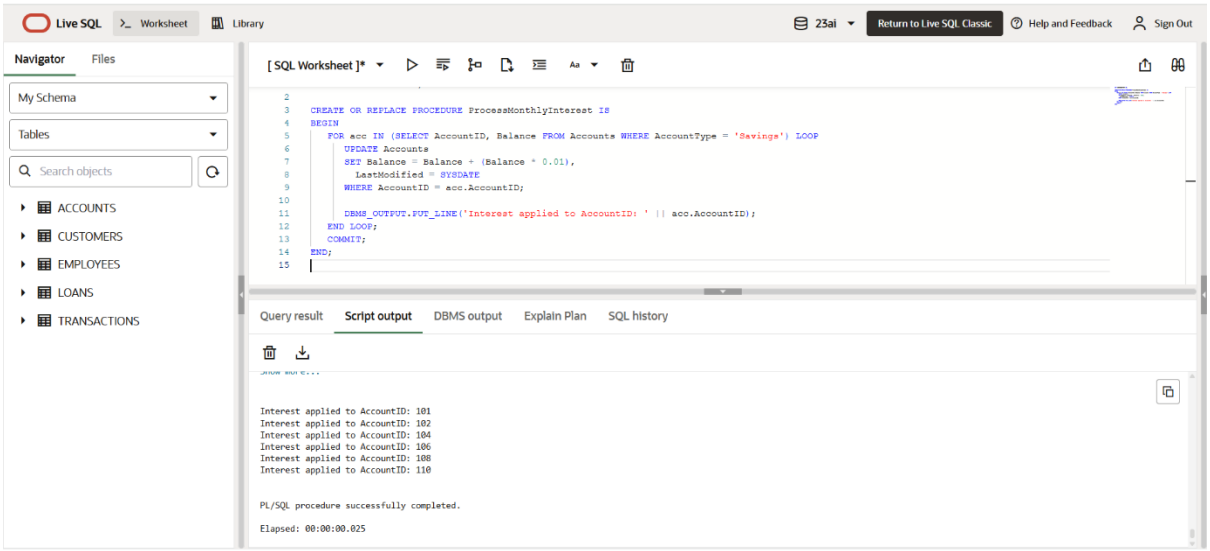
Scenario 1: The bank needs to process monthly interest for all savings accounts.

- **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.

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
  FOR acc IN (
    SELECT AccountID, Balance
    FROM Accounts
    WHERE AccountType = 'Savings'
  ) LOOP
    UPDATE Accounts
    SET Balance = Balance + (Balance * 0.01),
       LastModified = SYSDATE
    WHERE AccountID = acc.AccountID;

    DBMS_OUTPUT.PUT_LINE('Interest added for AccountID: ' ||
acc.AccountID ||
                         ' | New Balance: ' || (acc.Balance * 1.01));
  END LOOP;

  COMMIT;
END;
```



Scenario 2: The bank wants to implement a bonus scheme for employees based on their performance.

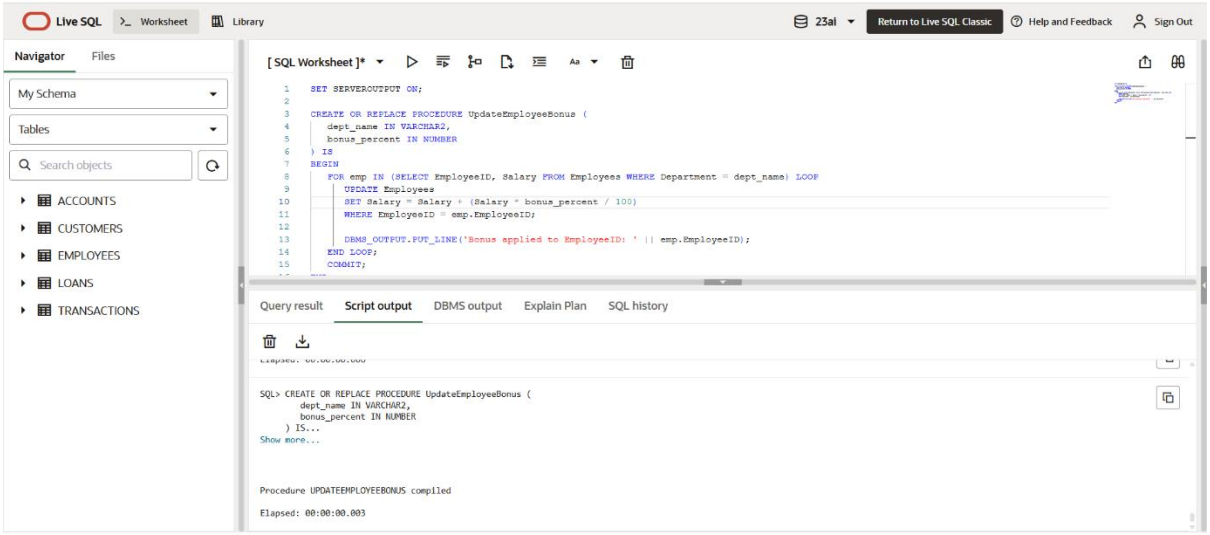
- **Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
  dept_name IN VARCHAR2,
  bonus_percent IN NUMBER
) IS
BEGIN
  FOR emp IN (
    SELECT EmployeeID, Salary
    FROM Employees
    WHERE Department = dept_name
  ) LOOP
    UPDATE Employees
```

```
    SET Salary = Salary + (Salary * bonus_percent / 100)
    WHERE EmployeeID = emp.EmployeeID;

    DBMS_OUTPUT.PUT_LINE('Bonus added for EmployeeID: ' || emp.EmployeeID ||
                         ' | New Salary: ' || (emp.Salary * (1 + bonus_percent / 100)));
  END LOOP;

  COMMIT;
END;
```

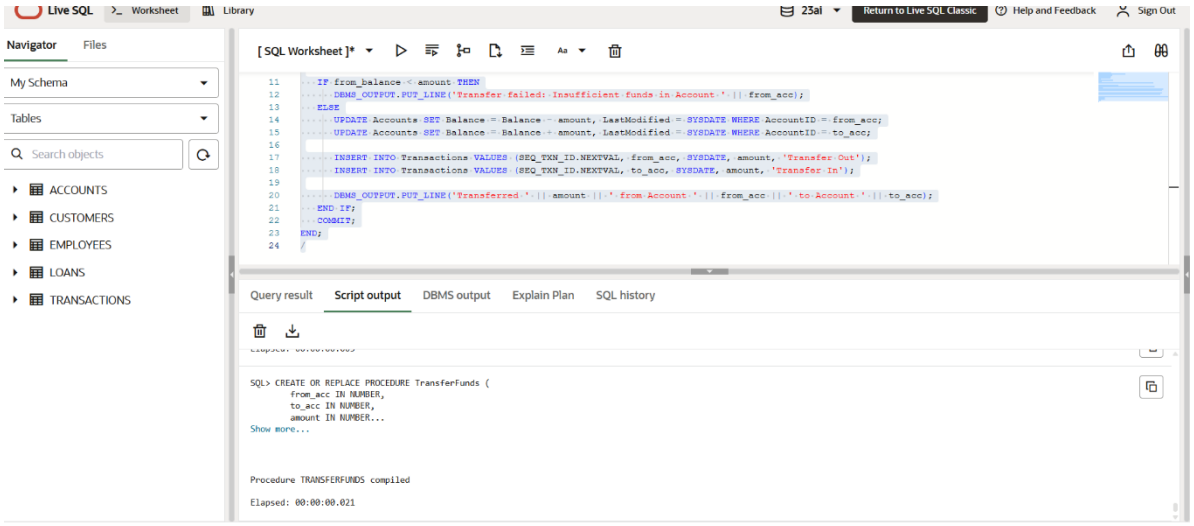


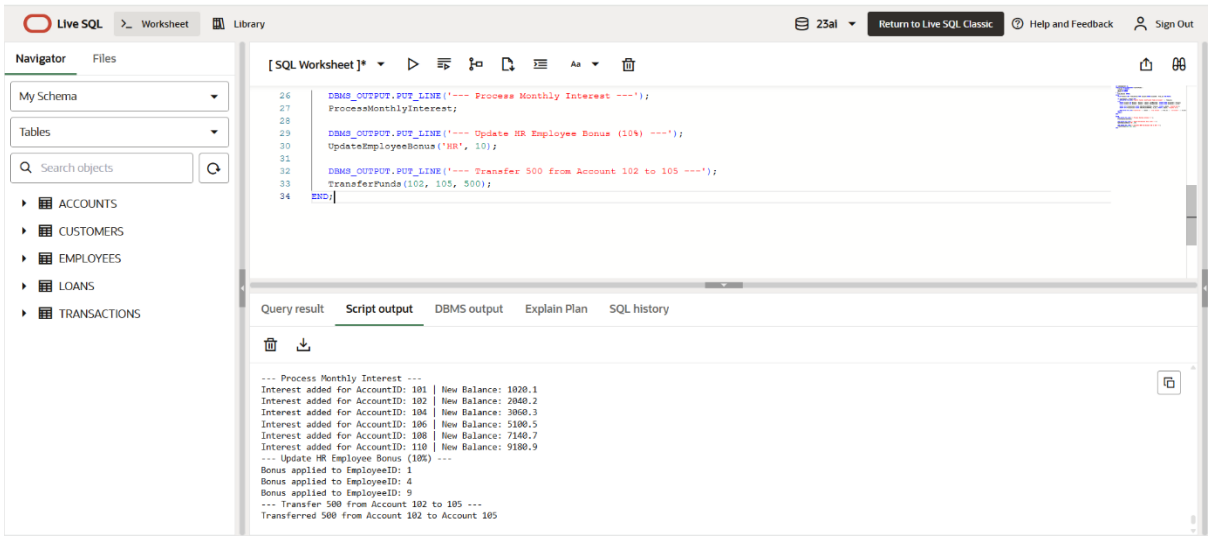
Scenario 3: Customers should be able to transfer funds between their accounts.

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.

```
CREATE OR REPLACE PROCEDURE TransferFunds (
  from_acc IN NUMBER,
  to_acc IN NUMBER,
  amount IN NUMBER
) IS
  from_balance NUMBER;
```

```
BEGIN
  SELECT Balance INTO from_balance FROM Accounts WHERE AccountID = from_acc FOR UPDATE;
  IF from_balance < amount THEN
    DBMS_OUTPUT.PUT_LINE('Transfer failed: insufficient balance.');
  ELSE
    UPDATE Accounts
    SET Balance = Balance - amount,
       LastModified = SYSDATE
    WHERE AccountID = from_acc;
    UPDATE Accounts
    SET Balance = Balance + amount,
       LastModified = SYSDATE
    WHERE AccountID = to_acc;
    INSERT INTO Transactions VALUES (SEQ_TXN_ID.NEXTVAL, from_acc, SYSDATE, amount, 'Transfer Out');
    INSERT INTO Transactions VALUES (SEQ_TXN_ID.NEXTVAL, to_acc, SYSDATE, amount, 'Transfer In');
    DBMS_OUTPUT.PUT_LINE('Transfer of $' || amount || ' from Account ' || from_acc || ' to Account ' || to_acc || ' successful.');
  END IF;
  COMMIT;
END;
```





TDD using JUnit5 and Mockito

Exercise 1: Setting Up JUnit

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.

Calculator.java

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);
    }
}
```

```
> mvn test

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running AssertionsTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.012 sec

Results:

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

Exercise 3: Assertions in JUnit Scenario: You need to use different assertions in JUnit to validate your test results.

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

public class AssertionsTest {

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

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.

Calculator.java

```
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
```

```
        return a - b;
    }
}
```

CalculatorTest.java

```
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;

public class CalculatorTest {
    private Calculator calculator;
    @Before
    public void setUp() {
        System.out.println("Setting up Calculator...");
        calculator = new Calculator();
    }
    @After
    public void tearDown() {
        System.out.println("Tearing down Calculator...\n");
        calculator = null;
    }
    @Test
    public void testAddition() {
        int a = 5;
        int b = 3;
        int result = calculator.add(a, b);
        assertEquals(8, result);
    }
```

```
    @Test
    public void testSubtraction() {
        int a = 10;
        int b = 4;
        int result = calculator.subtract(a, b);
        assertEquals(6, result);
    }
}
```

```
> mvn test

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running CalculatorTest
Setting up Calculator
Running testAdd
Tearing down Calculator

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.014 sec
```

1. Mockito exercises

Exercise 1: Mocking and Stubbing

ExternalApi.java

```
public interface ExternalApi {
    String getData();
}
```

MyService.java

```
public class MyService {
    private ExternalApi api;
```

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

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

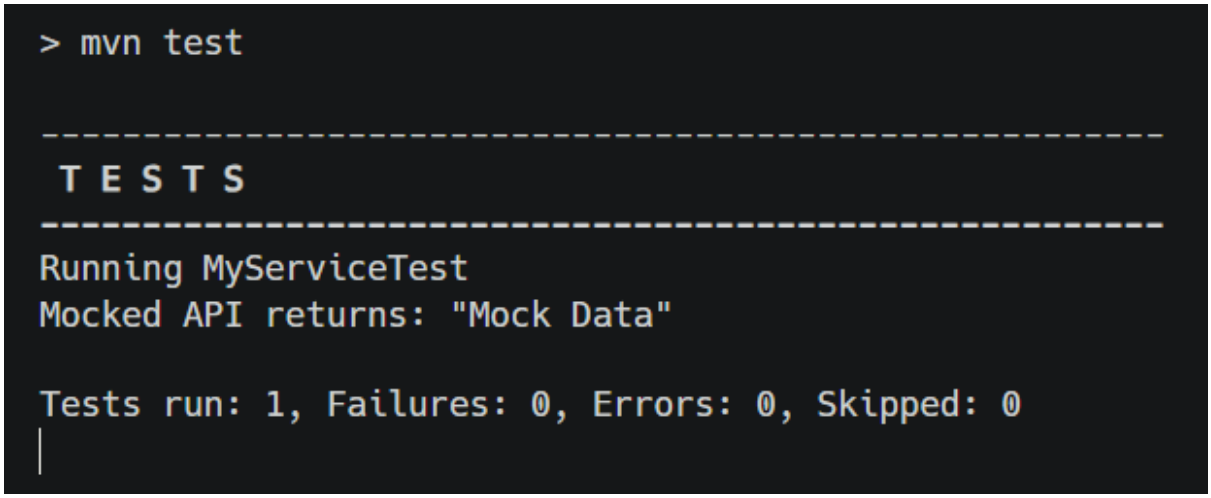
MyServiceTest.java

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

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

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);
        verify(mockApi).getData(); // optional: verify method was called
    }
}
```



Exercise 2: Verifying Interactions Scenario: You need to ensure that a method is called with specific arguments.

ExternalApi.java

```
public interface ExternalApi {
    String getData();
}
```

MyService.java

```
public class MyService {
    private ExternalApi api;

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

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

```
> mvn test

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running MyServiceTest
Verified that getData() was called exactly once.

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

Logging using SLF4J

Exercise 1: Logging Error Messages and Warning Levels Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

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");
        logger.info("This is an info message");
        logger.debug("This is a debug message (may not show without config change)");
    }
}
```

logback.xml

```
<configuration>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
    </encoder>
  </appender>

  <root level="debug">
    <appender-ref ref="STDOUT"/>
  </root>
</configuration>
```