Schema to be Created

```
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)
);

CREATE TABLE Transactions (
   TransactionID NUMBER PRIMARY KEY,
   AccountID NUMBER,
   TransactionDate DATE,
   Amount NUMBER,
   TransactionType VARCHAR2(10),
   FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)
);

CREATE TABLE Loans (
   LoanID NUMBER PRIMARY KEY,
   CustomerID NUMBER,
   LoanAmount NUMBER,
   InterestRate NUMBER,
   StartDate DATE,
   EndDate DATE,
   FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

CREATE TABLE Employees (
   EmployeeID NUMBER PRIMARY KEY,
   Name VARCHAR2(100),
   Position VARCHAR2(50),
   Salary NUMBER,
   Department VARCHAR2(50),
   HireDate DATE
);
```

Example Scripts for Sample Data Insertion

```
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);

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

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

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'));
```

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.

Solution:

Scenario 1:

```
BEGIN
    FOR cust_rec IN (
        SELECT c.CustomerID, c.DOB, l.LoanID, l.InterestRate
        FROM Customers c
        JOIN Loans l ON c.CustomerID = l.CustomerID
    ) LOOP
        IF MONTHS_BETWEEN(SYSDATE, cust_rec.DOB) / 12 > 60 THEN
            UPDATE Loans
            SET InterestRate = InterestRate - 1
            WHERE LoanID = cust_rec.LoanID;

            DBMS_OUTPUT.PUT_LINE('1% discount applied to LoanID: ' ||
cust_rec.LoanID);
        END IF;
    END LOOP;
END;
```

Output:

There are no records of age greater than 60

```
PL/SQL procedure successfully completed.
Elapsed: 00:00:00.090
```

Scenario 2:

Query:

```
ALTER TABLE Customers ADD (IsVIP CHAR(1));
```

Output:

```
Table CUSTOMERS altered.

Elapsed: 00:00:00.027
```

Query:

```
BEGIN
    FOR cust_rec IN (
        SELECT CustomerID, Balance FROM Customers
    ) LOOP
        IF cust_rec.Balance > 10000 THEN
            UPDATE Customers
            SET IsVIP = 'Y'
            WHERE CustomerID = cust_rec.CustomerID;
```

```
            DBMS_OUTPUT.PUT_LINE('CustomerID ' || cust_rec.CustomerID
|| ' marked as VIP');
        END IF;
    END LOOP;
END;
```

Output:

```
PL/SQL procedure successfully completed.

Elapsed: 00:00:00.011
```

Scenario 3:

```
BEGIN
    FOR loan_rec IN (
        SELECT l.LoanID, l.CustomerID, l.EndDate, c.Name
        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: Loan ID ' || loan_rec.LoanID
|| ' for Customer ' || loan_rec.Name || ' is due on ' ||
TO_CHAR(loan_rec.EndDate, 'YYYY-MM-DD'));
    END LOOP;
END;
```

Output:

```
PL/SQL procedure successfully completed.

Elapsed: 00:00:00.006
```

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.

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.

Scenario 1:

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
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('Interest added to AccountID: ' ||
acc_rec.AccountID);
    END LOOP;
END;
```

Output:

```
EXEC ProcessMonthlyInterest;

Interest added to AccountID: 1
PL/SQL procedure successfully completed.
Elapsed: 00:00:00.013
```

Scenario 2:

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
    p_department IN VARCHAR2,
    p_bonus_percent IN NUMBER
) IS
BEGIN
    FOR emp_rec IN (
        SELECT EmployeeID, Salary
        FROM Employees
        WHERE Department = p_department
    ) LOOP
        UPDATE Employees
        SET Salary = Salary + (Salary * p_bonus_percent / 100)
        WHERE EmployeeID = emp_rec.EmployeeID;

        DBMS_OUTPUT.PUT_LINE('Bonus applied to EmployeeID: ' ||
emp_rec.EmployeeID);
    END LOOP;
END;
```

Output:

```
EXEC UpdateEmployeeBonus('IT', 10);

Bonus applied to EmployeeID: 2
PL/SQL procedure successfully completed.
Elapsed: 00:00:00.016
```

Scenario 3:

```
CREATE OR REPLACE PROCEDURE TransferFunds (
    p_from_account IN NUMBER,
    p_to_account IN NUMBER,
    p_amount IN NUMBER
) IS
    v_from_balance NUMBER;
BEGIN
    -- Get source account balance
    SELECT Balance INTO v_from_balance
    FROM Accounts
    WHERE AccountID = p_from_account;

    IF v_from_balance < p_amount THEN
```

```
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance in
source account.');
    END IF;

    -- Deduct from source
    UPDATE Accounts
    SET Balance = Balance - p_amount,
        LastModified = SYSDATE
    WHERE AccountID = p_from_account;

    -- Add to target
    UPDATE Accounts
    SET Balance = Balance + p_amount,
        LastModified = SYSDATE
    WHERE AccountID = p_to_account;

    DBMS_OUTPUT.PUT_LINE('Transferred ' || p_amount ||
                         ' from Account ' || p_from_account ||
                         ' to Account ' || p_to_account);
END;
```

Output:

```
EXEC TransferFunds(1, 2, 100);


Transferred 100 from Account 1 to Account 2
PL/SQL procedure successfully completed.
Elapsed: 00:00:00.010
```

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:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

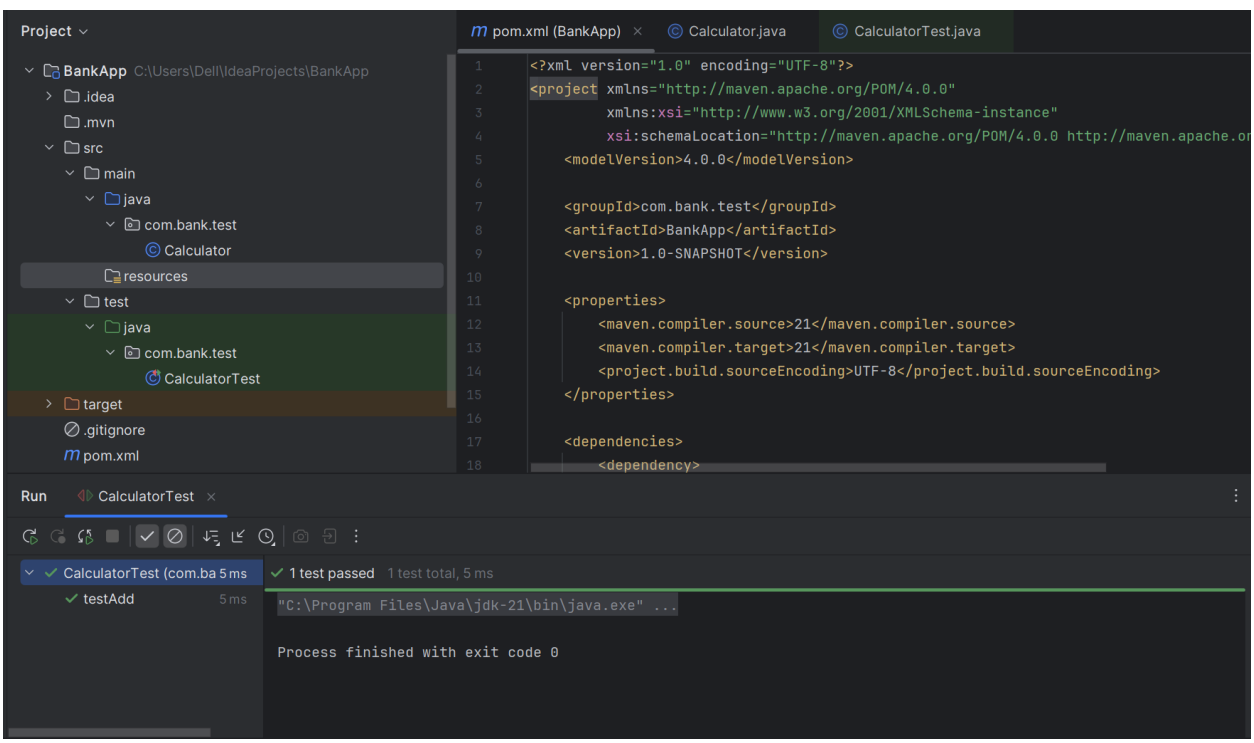
<version>4.13.2</version>

<scope>test</scope>

</dependency>

2. Create a new test class in your project.

Solution:



```
<?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>com.bank.test</groupId>
    <artifactId>BankApp</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

</project>
```

Exercise 3: Assertions in JUnit

Scenario:

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

Steps:

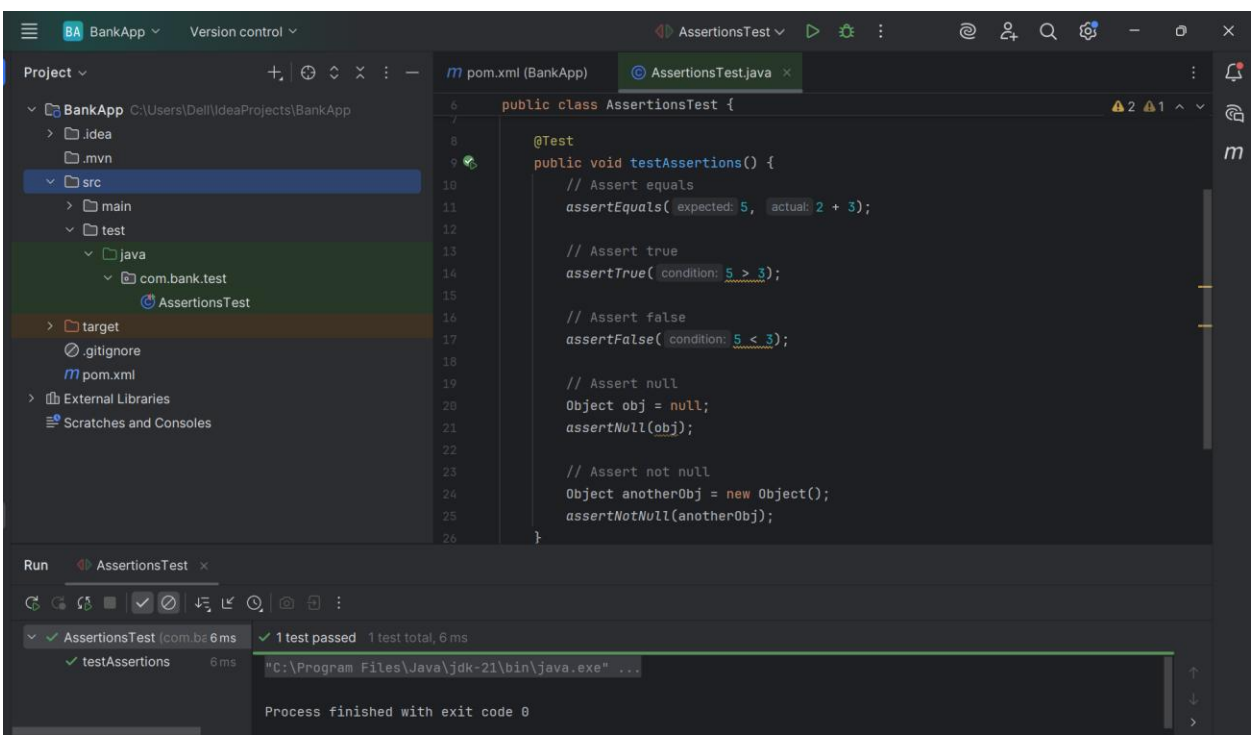
1. Write tests using various JUnit assertions.

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());
    }
}
```

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.

Steps:

1. Write tests using the AAA pattern.
2. Use @Before and @After annotations for setup and teardown methods.

Solutions:

Calculator.java

```
package com.bank.test;

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

```
package com.bank.test;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class CalculatorTest {

    private Calculator calc;

    @Before
    public void setUp() {
```

```java
        calc = new Calculator();
        System.out.println("Setup: Calculator object created");
    }

    @After
    public void tearDown() {
        calc = null;
        System.out.println("Teardown: Calculator object
destroyed");
    }

    @Test
    public void testAdd() {

        int result = calc.add(10, 5);

        assertEquals(15, result);
    }

    @Test
    public void testSubtract() {

        int result = calc.subtract(10, 3);

        assertEquals(7, result);
    }
}
```

Output:

```
CalculatorTest (com.ba 5 ms        2 tests passed  2 tests total, 5 ms
  testAdd              5 ms        "C:\Program Files\Java\jdk-21\bin\java.exe" ...
  testSubtract         0 ms        Setup: Calculator object created
                                   Teardown: Calculator object destroyed
                                   Setup: Calculator object created
                                   Teardown: Calculator object destroyed

                                   Process finished with exit code 0
```

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

```
<?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>com.bank.test</groupId>
    <artifactId>Services</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>5.10.0</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-core</artifactId>
            <version>5.11.0</version>
            <scope>test</scope>
        </dependency>
    </dependencies>


</project>
```

ExternalApi.java

```java
package com.bank.test;

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

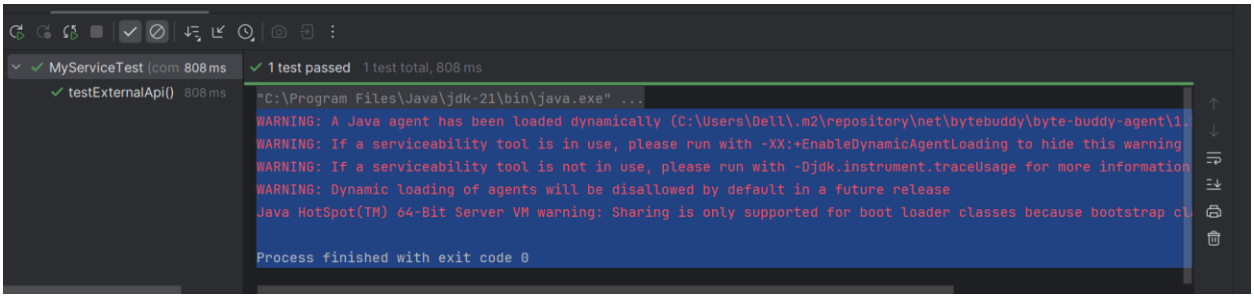
MyService.java

```java
package com.bank.test;

public class MyService {
    private ExternalApi api;

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

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

MyServiceTest.java

```java
package com.bank.test;

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

public class MyServiceTest {

    @Test
    public void testExternalApi() {

        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:



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.

Solution 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();
    }
}
```

ExternalApi.java

```
package com.bank.test;

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

Myservice.java

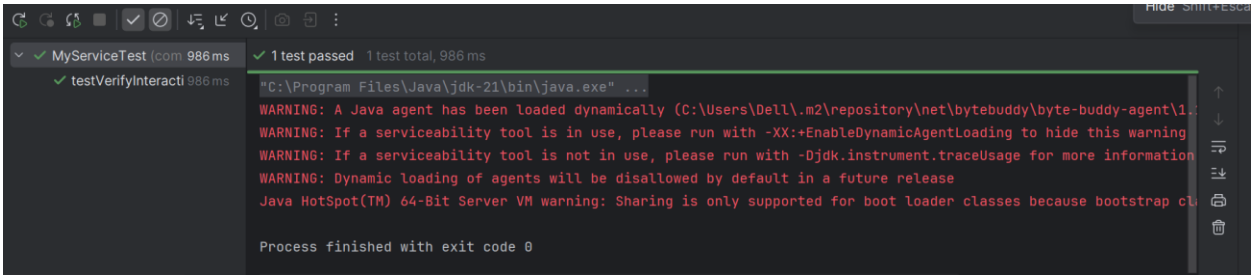
```
package com.bank.test;

public class MyService {
    private ExternalApi api;

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

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

MyServiceTest.java

```
package com.bank.test;

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

public class MyServiceTest {

    @Test
    public void testVerifyInteraction() {

        ExternalApi mockApi = mock(ExternalApi.class);

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

```
        service.fetchData();

        verify(mockApi).getData();
    }
}
```

Output:



Exercise 1: 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:

```
<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>com.bank.test</groupId>
    <artifactId>Services</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <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>



</project>
```

LoggingExample.java

```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:

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.3\lib\idea_rt.jar=5136
10:34:12.316 [main] ERROR LoggingExample - This is an error message
10:34:12.319 [main] WARN LoggingExample - This is a warning message

Process finished with exit code 0
```