**MODULE 3**

**EXERCISE 1:**

**IMPLEMENTATION:**

I implemented three different banking scenarios using T-SQL on OneCompiler. For each scenario, I created temporary in-memory tables using DECLARE @TableName TABLE (...) to simulate customer and loan data. I inserted sample records and used SQL logic to perform operations like updating loan interest for senior citizens (age > 60), promoting high-balance customers to VIP status, and generating loan reminders for due dates within the next 30 days. I used functions like DATEDIFF, GETDATE(), and conditional UPDATE/SELECT statements to handle each case efficiently.

**CODE:**

BEGIN

DECLARE @Customers TABLE (

CustomerID INT,

Name VARCHAR(100),

DOB DATE,

Balance DECIMAL(10,2)

);

DECLARE @Loans TABLE (

LoanID INT,

CustomerID INT,

InterestRate DECIMAL(5,2)

);

INSERT INTO @Customers VALUES

(1, 'John Doe', '1955-05-15', 15000.00),

(2, 'Jane Smith', '1990-07-20', 1500.00);

INSERT INTO @Loans VALUES

(1, 1, 5.00),

(2, 2, 4.50);

UPDATE @Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID IN (

SELECT CustomerID FROM @Customers

WHERE DATEDIFF(YEAR, DOB, GETDATE()) > 60

);

SELECT 'Applied 1% discount to seniors' AS Result;

SELECT \* FROM @Loans;

END;

GO

BEGIN

DECLARE @Customers TABLE (

CustomerID INT,

Name VARCHAR(100),

Balance DECIMAL(10,2),

IsVIP BIT DEFAULT 0

);

INSERT INTO @Customers VALUES

(1, 'John Doe', 15000.00, 0),

(2, 'Jane Smith', 1500.00, 0);

UPDATE @Customers

SET IsVIP = 1

WHERE Balance > 10000;

SELECT 'VIP promotions completed' AS Result;

SELECT \* FROM @Customers;

END;

GO

BEGIN

DECLARE @Loans TABLE (

LoanID INT,

CustomerID INT,

CustomerName VARCHAR(100),

EndDate DATE

);

INSERT INTO @Loans VALUES

(1, 1, 'John Doe', DATEADD(DAY, 15, GETDATE())), -- Due in 15 days

(2, 2, 'Jane Smith', DATEADD(MONTH, 6, GETDATE())); -- Due in 6 months

SELECT 'Loan due reminders:' AS Result;

SELECT

'Reminder for ' + CustomerName + ': Loan ' + CAST(LoanID AS VARCHAR) +

' due in ' + CAST(DATEDIFF(DAY, GETDATE(), EndDate) AS VARCHAR) + ' days' AS Message

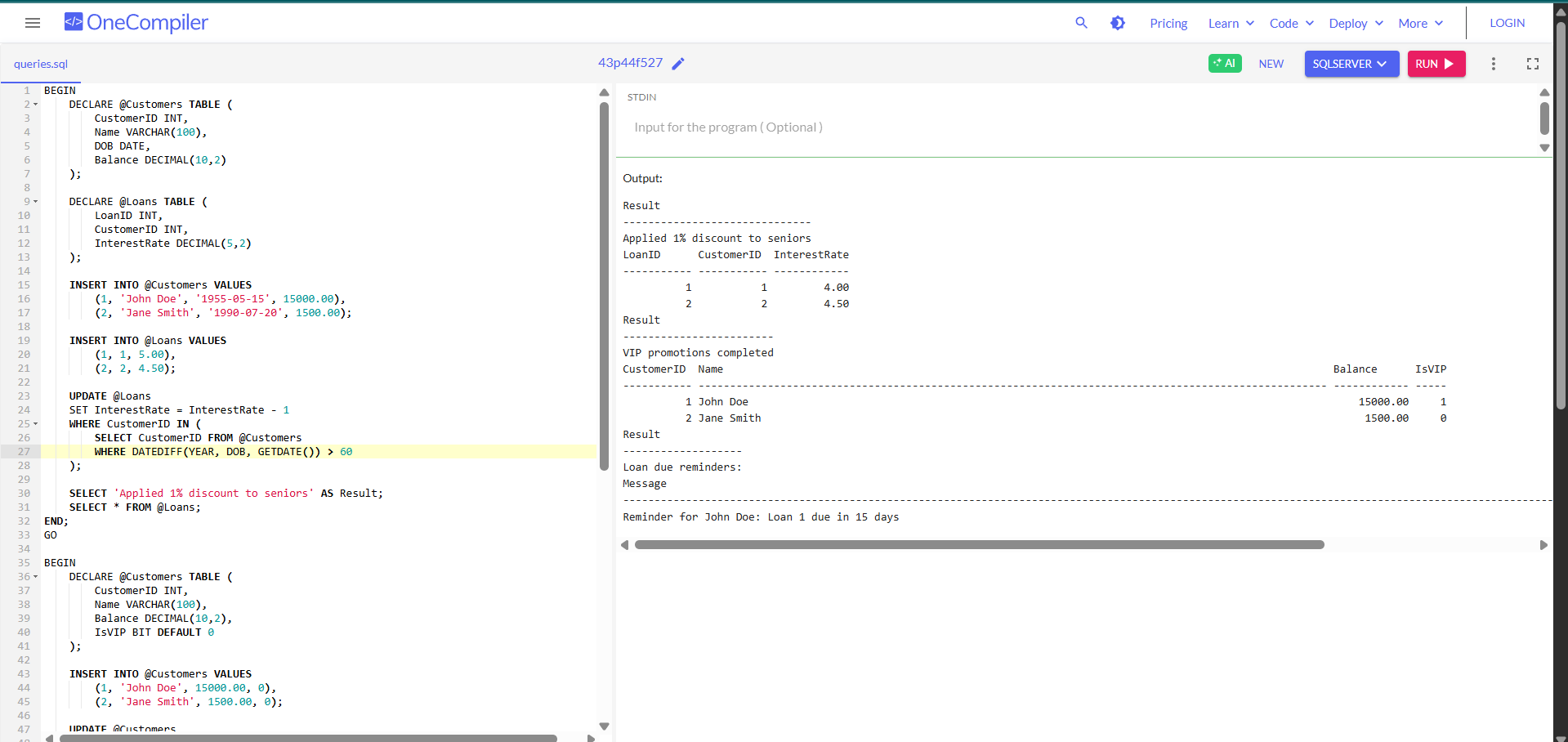
FROM @Loans

WHERE DATEDIFF(DAY, GETDATE(), EndDate) <= 30;

IF @@ROWCOUNT = 0

SELECT 'No loans due in next 30 days' AS Message;

END;

**OUTPUT:** ****

**EXERCISE 3:**

**IMPLEMENTATION:**

I created stored procedures to handle banking tasks like processing monthly interest for savings accounts, updating employee bonuses based on department and performance, and transferring funds between accounts with a balance check. Each procedure uses input parameters, conditional logic, and SQL UPDATE or SELECT statements to perform the required operations efficiently and safely.

**CODE:**

BEGIN

DECLARE @Savings TABLE (ID INT, Balance MONEY);

INSERT INTO @Savings VALUES (1,5000),(2,3000);

UPDATE @Savings SET Balance = Balance \* 1.01;

SELECT 'Savings After Interest' AS Description, \* FROM @Savings;

END;

GO

BEGIN

DECLARE @Emp TABLE (ID INT, Name VARCHAR(20), Dept VARCHAR(10), Salary MONEY);

INSERT INTO @Emp VALUES (1,'Alex','HR',50000),(2,'Sam','IT',60000);

UPDATE @Emp

SET Salary = CASE

WHEN Dept = 'HR' THEN Salary \* 1.10

WHEN Dept = 'IT' THEN Salary \* 1.05

ELSE Salary END;

SELECT 'Employees After Bonus' AS Description, \* FROM @Emp;

END;

GO

BEGIN

DECLARE @Accts TABLE (ID INT, Balance MONEY);

INSERT INTO @Accts VALUES (1,2000),(2,1000);

DECLARE @Amount MONEY = 500, @From INT = 1, @To INT = 2;

IF (SELECT Balance FROM @Accts WHERE ID = @From) >= @Amount

BEGIN

UPDATE @Accts SET Balance = Balance - @Amount WHERE ID = @From;

UPDATE @Accts SET Balance = Balance + @Amount WHERE ID = @To;

SELECT 'Transfer Successful' AS Result;

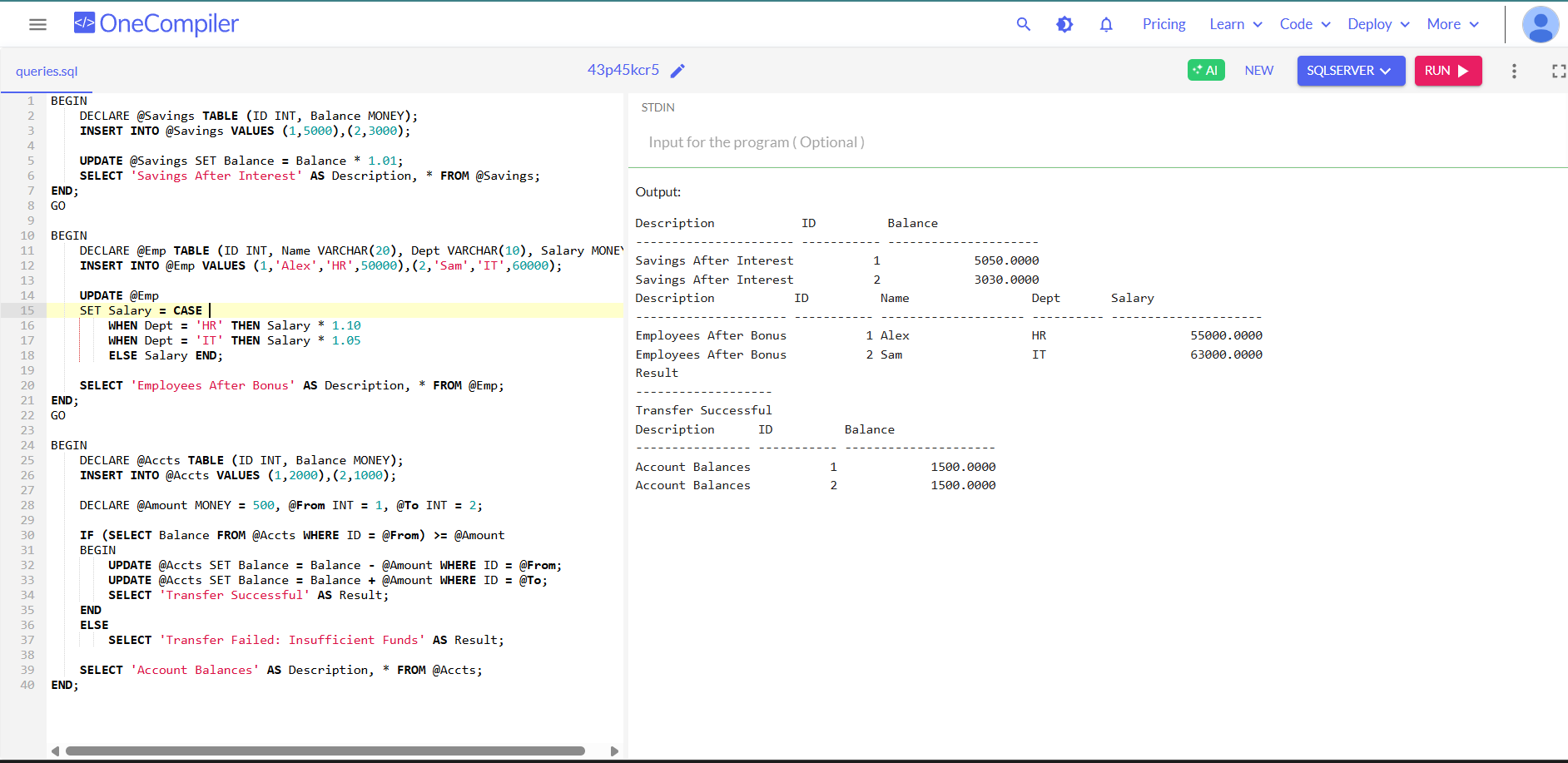
END

ELSE

SELECT 'Transfer Failed: Insufficient Funds' AS Result;

SELECT 'Account Balances' AS Description, \* FROM @Accts;

END;

**OUTPUT:** ****

**JUNIT TESTING EXERCISE:**

**EXERCISE 1:**

**CODE: SETTING UP JUNIT:**

**POM.XML:**

<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.sudherson</groupId>

<artifactId>junitdemo3</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- Single JUnit 4 dependency (removed duplicate) -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Maven Surefire Plugin to properly run JUnit 4 tests -->

<plugin>

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

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

<version>3.2.5</version>

<configuration>

<!-- Explicitly disable JUnit 5 (Jupiter) to force JUnit 4 -->

<useJUnitPlatform>false</useJUnitPlatform>

</configuration>

</plugin>

</plugins>

</build>

</project>

**CALCULATOR.JAVA**

package junitdemo3;

public class Calculator {

public int add(int a, int b) {

return a + b; // Ensure no hidden characters

}

}

**CALCULATORTEST.JAVA:**

package junitdemo3;

import org.junit.Test;

import static org.junit.Assert.\*;

import static org.junit.Assert.\*;

public class CalculatorTest {

*@Test*

public void testAdd() {

Calculator calc = new Calculator();

int result = calc.add(2, 3);

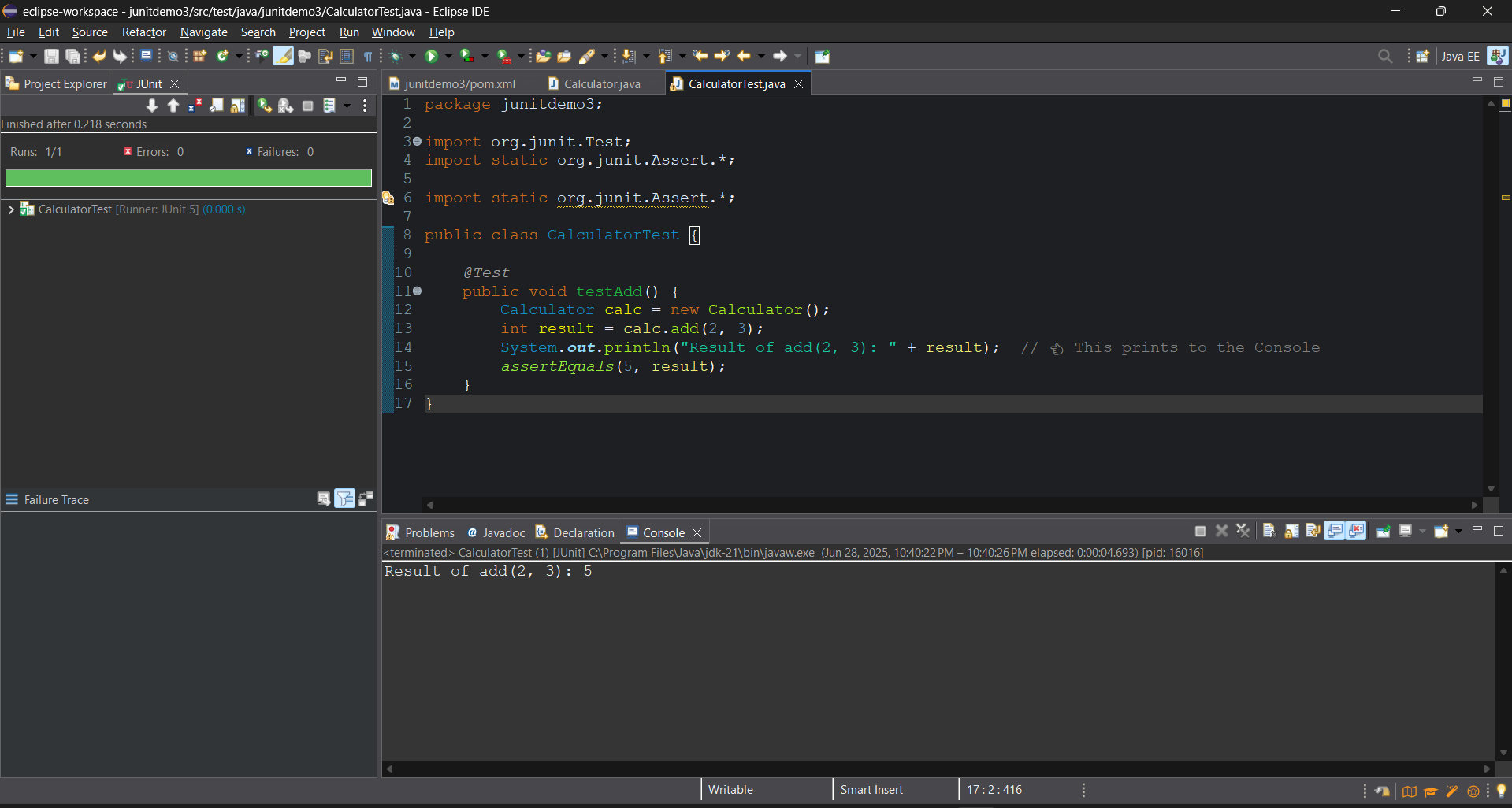
System.***out***.println("Result of add(2, 3): " + result); // 👈 This prints to the Console

*assertEquals*(5, result);

}

}

**OUTPUT**:



**EXERCISE 3:**

**CODE:**

**Pom.file:**

<project>

<modelVersion>4.0.0</modelVersion>

<groupId>junit.exercises</groupId> <!-- Changed to match package -->

<artifactId>junit-exercises</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>  
  
**StringUtils.java:**

package junitexercises;

public class StringUtils {

public static String reverse(String str) {

return new StringBuilder(str).reverse().toString();

}

}

**StringUtilsTest.java:**

package junitexercises;

import org.junit.Test;

import static org.junit.Assert.\*;

public class StringUtilsTest {

*@Test*

public void testReverse() {

System.***out***.println("--- Running testReverse ---");

String input = "hello";

String expected = "olleh";

String actual = StringUtils.*reverse*(input);

System.***out***.println("Input: " + input);

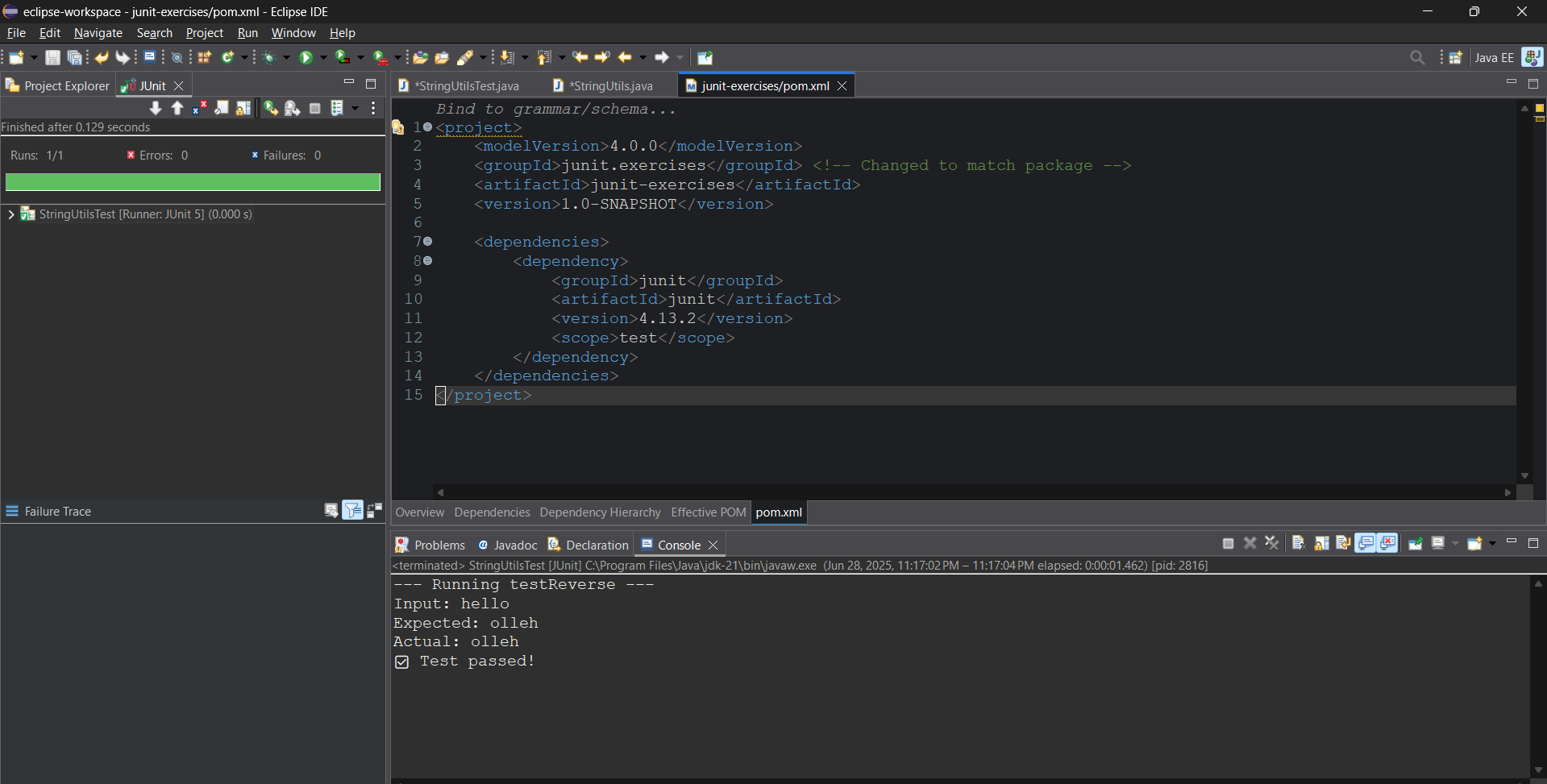
System.***out***.println("Expected: " + expected);

System.***out***.println("Actual: " + actual);

*assertEquals*(expected, actual);

}

}

**OUTPUT:** ****

**EXERCISE 4:**

**CODE:**

**Pom 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>bank-account-test</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

<junit.version>5.8.2</junit.version>

</properties>

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>  
  
**BankAccount :**

package bankaccounttest;

public class BankAccount {

private double balance;

public BankAccount(double initialBalance) {

this.balance = initialBalance;

}

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

}

}

public void withdraw(double amount) throws InsufficientFundsException {

if (amount > balance) {

throw new InsufficientFundsException();

}

balance -= amount;

}

public double getBalance() {

return balance;

}

}

**BankAccountTest:**

package bankaccounttest;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

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

public class BankAccountTest {

private BankAccount account;

private static final double ***INITIAL\_BALANCE*** = 100.0;

*@*BeforeEach

public void setUp() {

account = new BankAccount(***INITIAL\_BALANCE***);

System.***out***.println("Created new account with balance: " + ***INITIAL\_BALANCE***);

}

*@*AfterEach

public void tearDown() {

System.***out***.println("Final account balance: " + account.getBalance());

account = null;

}

*@*Test

public void testInitialBalance() {

double balance = account.getBalance();

assertEquals(***INITIAL\_BALANCE***, balance);

}

*@*Test

public void testDeposit() {

double depositAmount = 50.0;

double expectedBalance = ***INITIAL\_BALANCE*** + depositAmount;

account.deposit(depositAmount);

assertEquals(expectedBalance, account.getBalance());

}

*@*Test

public void testWithdraw() throws Exception {

// Arrange

double withdrawAmount = 30.0;

double expectedBalance = ***INITIAL\_BALANCE*** - withdrawAmount;

account.withdraw(withdrawAmount);

assertEquals(expectedBalance, account.getBalance());

}

*@*Test

public void testWithdrawInsufficientFunds() {

double withdrawAmount = 150.0;

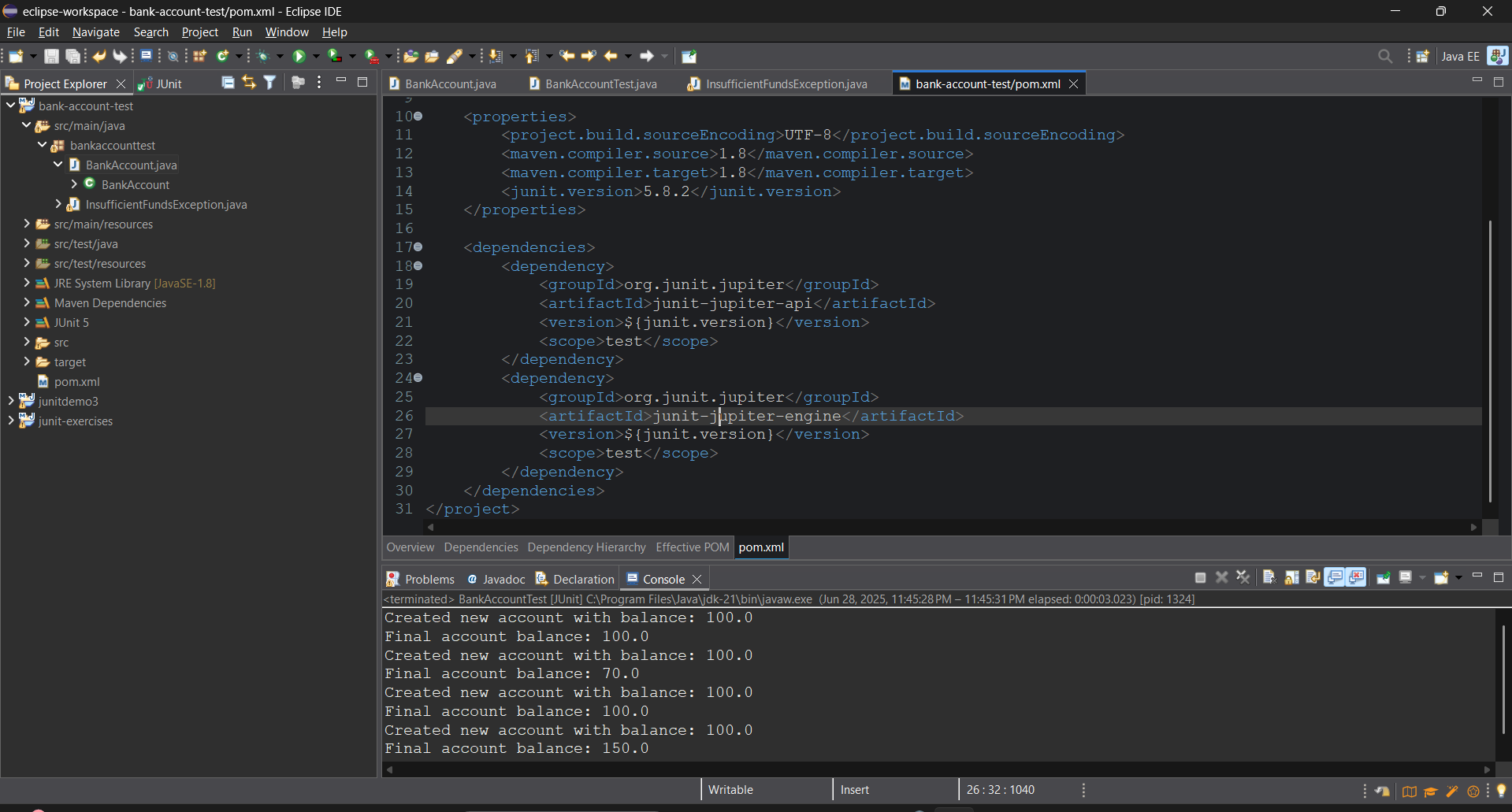
assertThrows(InsufficientFundsException.class, () -> {

account.withdraw(withdrawAmount);

});

}

}

**OUTPUT:** ****

**MOCKITO EXERCISES**

**EXERCISE 1  
Mocking and Stubbing**

**CODE:**

**Pom.xml:**

<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>mockitoexercises</groupId>

<artifactId>mockito-exercises</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

<junit.version>5.8.2</junit.version>

<mockito.version>4.5.1</mockito.version>

</properties>

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>${mockito.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

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

<version>${mockito.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**MyService.java**

package mockitoexercises;

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

if (externalApi.connect("data-endpoint")) {

String data = externalApi.getData();

externalApi.disconnect();

return data;

}

return "Connection failed";

}

}

**MyServiceTest.java**

package mockitoexercises;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

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

import static org.mockito.Mockito.\*;

*@ExtendWith*(MockitoExtension.class)

public class MyServiceTest {

*@Mock*

private ExternalApi mockApi;

*@Test*

public void testFetchDataSuccess() {

System.***out***.println("\n=== Running testFetchDataSuccess ===");

// Arrange

System.***out***.println("Setting up mock to return success...");

*when*(mockApi.connect("data-endpoint")).thenReturn(true);

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

MyService service = new MyService(mockApi);

// Act

System.***out***.println("Calling fetchData()...");

String result = service.fetchData();

System.***out***.println("Verifying results...");

*assertEquals*("Mock Data", result);

*verify*(mockApi).disconnect();

System.***out***.println("Test passed successfully!");

}

*@Test*

public void testFetchDataConnectionFailure() {

System.***out***.println("\n=== Running testFetchDataConnectionFailure ===");

System.***out***.println("Setting up mock to return connection failure...");

*when*(mockApi.connect("data-endpoint")).thenReturn(false);

MyService service = new MyService(mockApi);

System.***out***.println("Calling fetchData()...");

String result = service.fetchData();

System.***out***.println("Verifying results...");

*assertEquals*("Connection failed", result);

*verify*(mockApi, *never*()).getData();

*verify*(mockApi, *never*()).disconnect();

System.***out***.println("Test passed successfully!");

}

*@Test*

public void testFetchDataWithDifferentResponses() {

System.***out***.println("\n=== Running testFetchDataWithDifferentResponses ===");

System.***out***.println("Setting up mock for multiple responses...");

*when*(mockApi.connect("data-endpoint")).thenReturn(true);

*when*(mockApi.getData())

.thenReturn("First Call")

.thenReturn("Second Call");

MyService service = new MyService(mockApi);

System.***out***.println("First call to fetchData()...");

*assertEquals*("First Call", service.fetchData());

System.***out***.println("Second call to fetchData()...");

*assertEquals*("Second Call", service.fetchData());

System.***out***.println("Test passed successfully!");

}

}

**EXTERNAL API:**

package mockitoexercises;

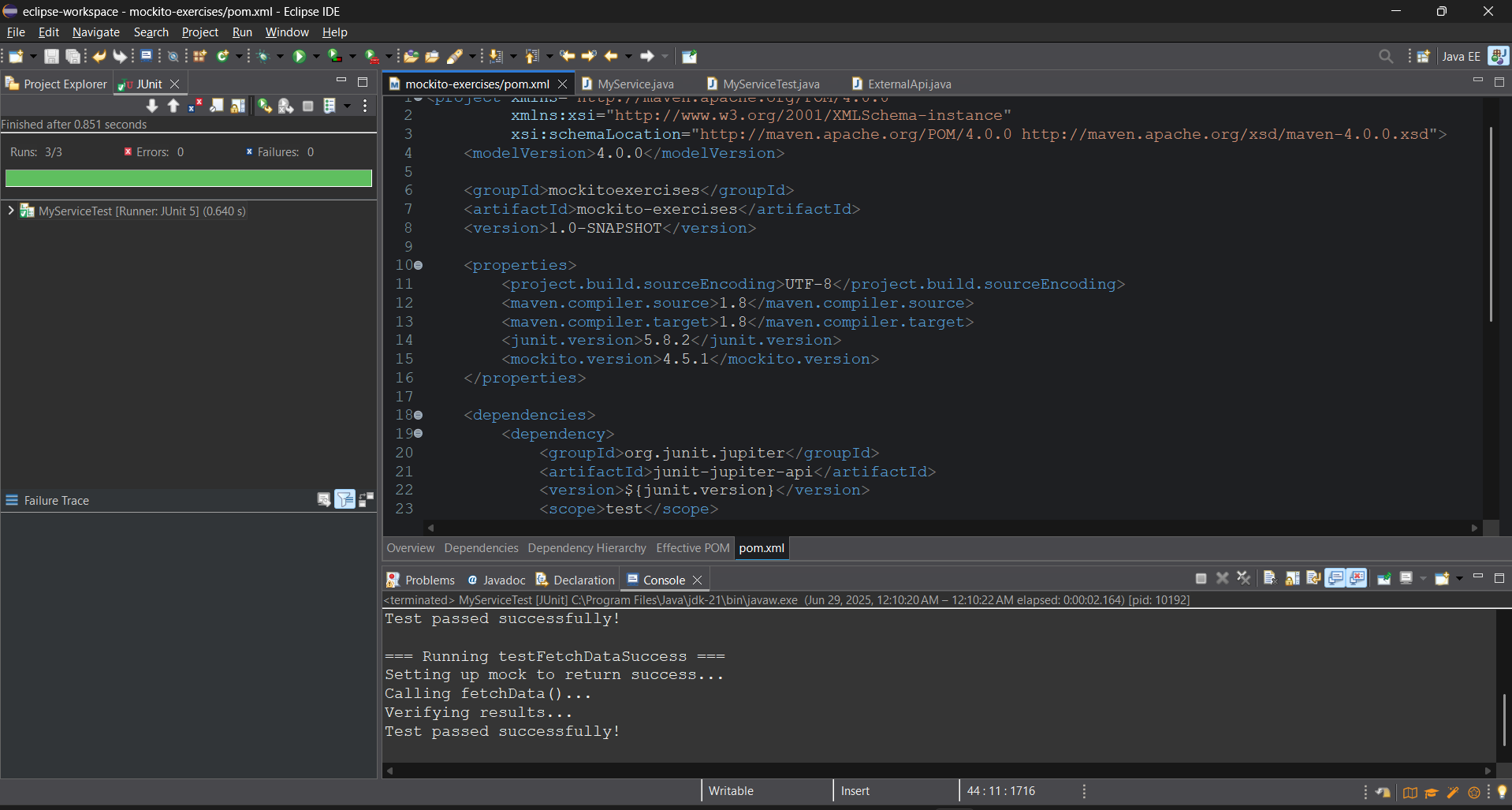
public interface ExternalApi {

String getData();

boolean connect(String endpoint);

void disconnect();

}

**OUTPUT:** ****

**EXERCISE :2**

**Verifying Interactions:**

**CODE:**

**Pom.xml:**

<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/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>mockitoverificationdemo</groupId>

<artifactId>mockito-verification-demo1</artifactId>

<version>1.0.0</version>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<junit.version>5.9.2</junit.version>

<mockito.version>5.3.1</mockito.version>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>${mockito.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>maven-compiler-plugin</artifactId>

<version>3.11.0</version>

</plugin>

</plugins>

</build>

</project>  
  
**MyService.java:**

package mockitoverificationdemo;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

System.out.println("MyService initialized with ExternalApi");

}

public String fetchData() {

System.out.println("Fetching data from external API...");

String data = api.getData();

System.out.println("Received data: " + data);

return data;

}

}

**External API:**

package mockitoverificationdemo;

public interface ExternalApi {

String getData();

}

**MyserviceTest.java:**

package mockitoverificationdemo;

import static org.mockito.Mockito.verify;

import static org.mockito.Mockito.when;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

System.out.println("\n=== Starting testVerifyInteraction ===");

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

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

System.out.println("Service returned: " + result);

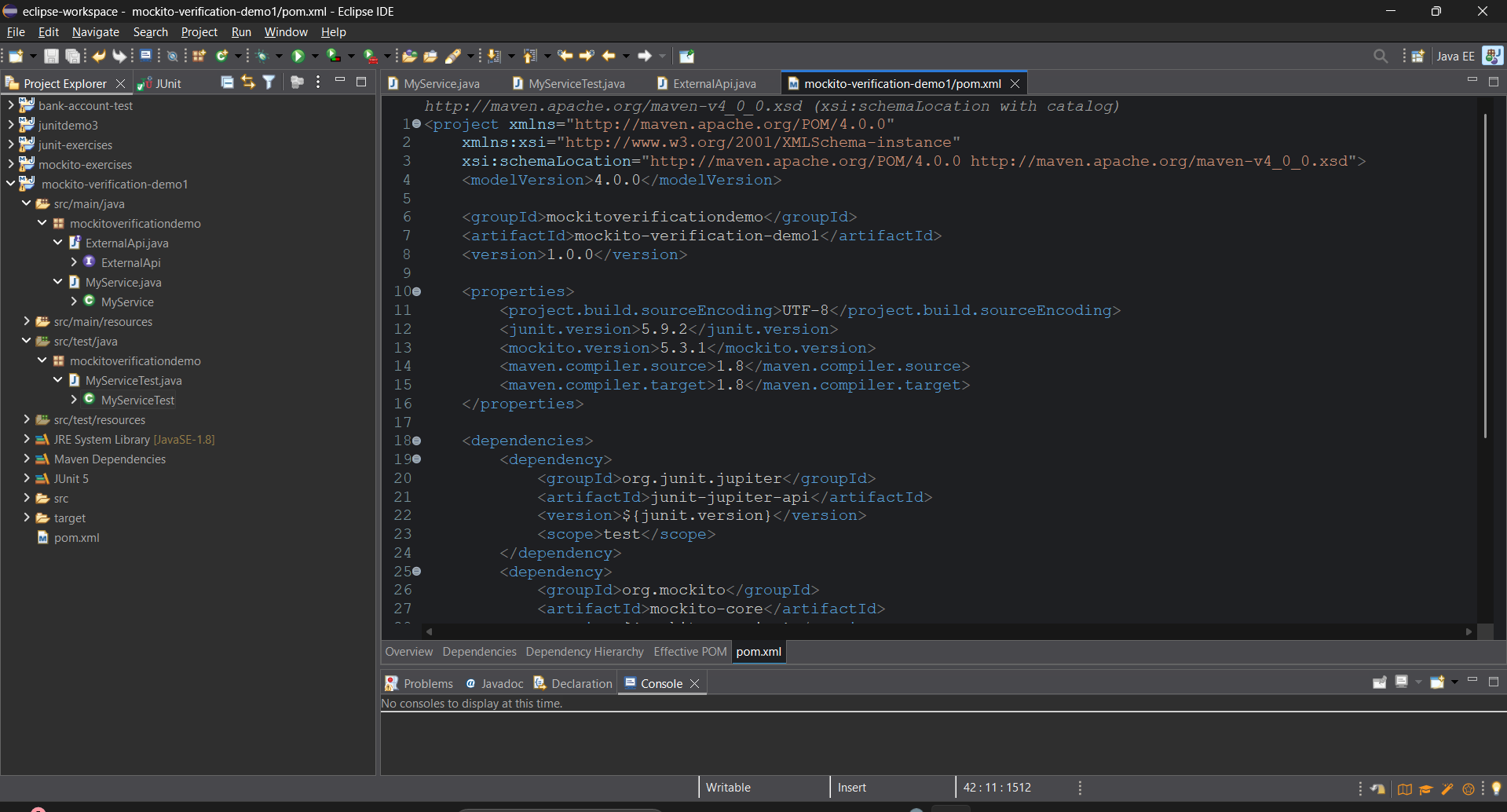
verify(mockApi).getData();

System.out.println("Verified that getData() was called");

System.out.println("=== Test Passed ===\n");

}

}

**OUTPUT:** ****

**SL4J LOGGING EXERCISES**

**EXERCISE 1:**

**CODE:**

**Pom.xml:**

<?xml version="1.0" encoding="UTF-8"?>

<project>

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>slf4j-demo</artifactId>

<version>1.0.0</version>

<dependencies>

<!-- SLF4J API -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<!-- Logback Implementation -->

<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-compiler-plugin</artifactId>

<version>3.11.0</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**LoggingExample.java:**

package 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");

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

// Fixed division operation

try {

int numerator = 10;

int denominator = 2; // Changed from 0 to prevent error

int result = numerator / denominator;

***logger***.info("Division result: {}", result);

} catch (Exception e) {

***logger***.error("Division error occurred", e);

}

}

}

**Logback.xml:**

<configuration>

<appender name="CONSOLE" 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">

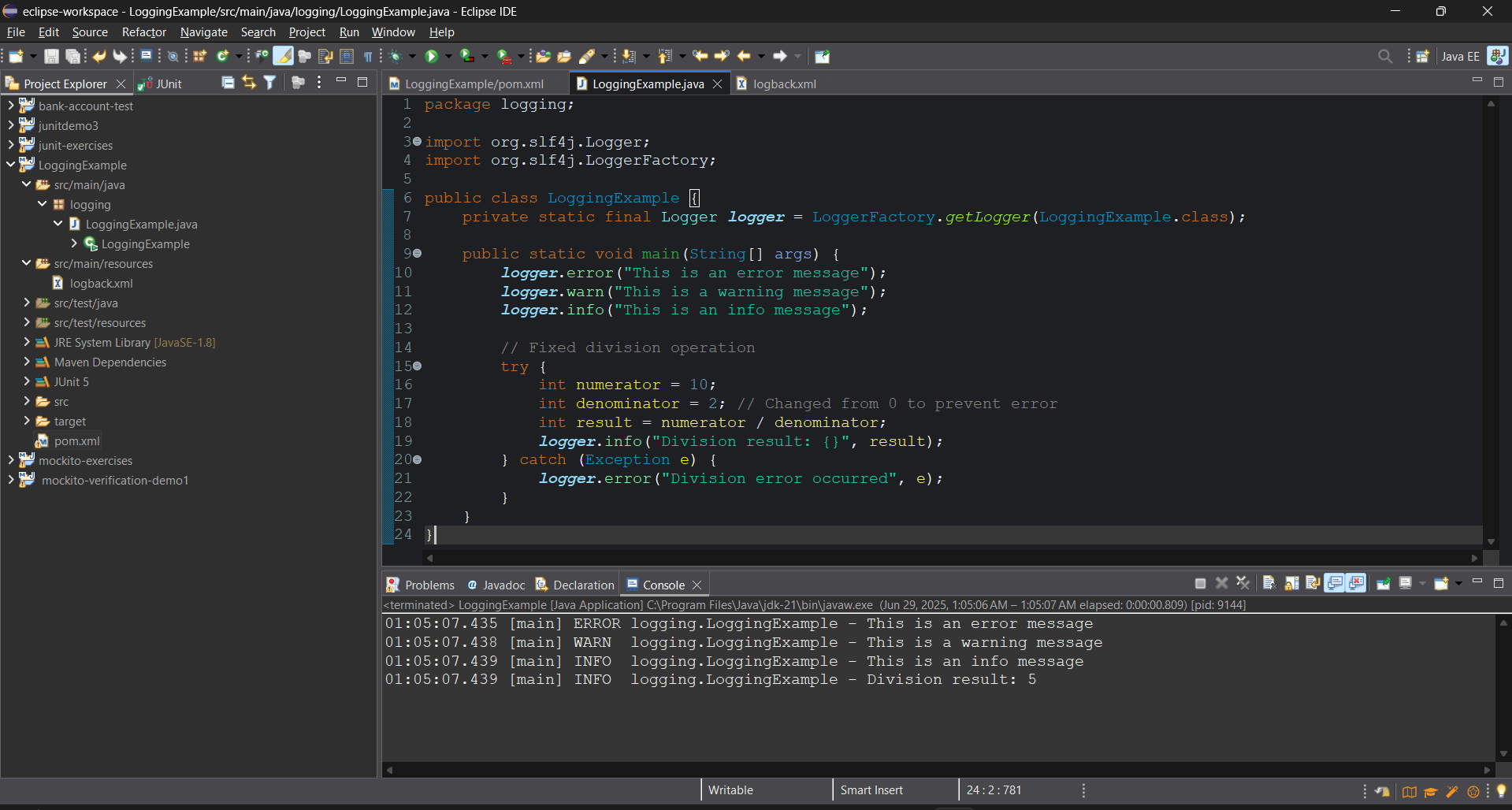
<appender-ref ref="CONSOLE" />

</root>

<!-- Add colored output for better visibility -->

<conversionRule conversionWord="clr" converterClass="org.springframework.boot.logging.logback.ColorConverter" />

</configuration>

**OUTPUT:** ****