**PL/SQL**

**Exercise 1: Control Structures**

**Source Code**

-- Create CUSTOMERS table

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

customer\_name VARCHAR2(100),

date\_of\_birth DATE,

account\_balance NUMBER(10,2),

is\_vip BOOLEAN DEFAULT FALSE,

email VARCHAR2(100)

);

-- Create LOANS table

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

interest\_rate NUMBER(5,2),

due\_date DATE,

amount\_due NUMBER(10,2),

status VARCHAR2(20) DEFAULT 'ACTIVE',

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

-- Insert sample customers

INSERT INTO customers VALUES (1, 'John Smith', TO\_DATE('1955-03-15', 'YYYY-MM-DD'), 5000.00, FALSE, 'john@example.com');

INSERT INTO customers VALUES (2, 'Alice Brown', TO\_DATE('1970-08-22', 'YYYY-MM-DD'), 15000.00, FALSE, 'alice@example.com');

INSERT INTO customers VALUES (3, 'Robert Davis', TO\_DATE('1948-11-05', 'YYYY-MM-DD'), 8000.00, FALSE, 'robert@example.com');

-- Insert sample loans

INSERT INTO loans VALUES (101, 1, 5.5, SYSDATE + 15, 2500.00, 'ACTIVE');

INSERT INTO loans VALUES (102, 2, 4.0, SYSDATE + 45, 10000.00, 'ACTIVE');

INSERT INTO loans VALUES (103, 3, 6.0, SYSDATE + 10, 5000.00, 'ACTIVE');

COMMIT;

DECLARE

CURSOR customer\_loans IS

SELECT c.customer\_id, c.date\_of\_birth, l.loan\_id, l.interest\_rate

FROM customers c

JOIN loans l ON c.customer\_id = l.customer\_id;

v\_age NUMBER;

v\_new\_rate NUMBER;

BEGIN

FOR cust\_rec IN customer\_loans LOOP

-- Calculate age

v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, cust\_rec.date\_of\_birth) / 12);

IF v\_age > 60 THEN

-- Apply 1% discount

v\_new\_rate := cust\_rec.interest\_rate - 1;

-- Ensure rate doesn't go below 0

IF v\_new\_rate < 0 THEN

v\_new\_rate := 0;

END IF;

-- Update the loan

UPDATE loans

SET interest\_rate = v\_new\_rate

WHERE loan\_id = cust\_rec.loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Applied discount to customer ' || cust\_rec.customer\_id ||

'. New rate: ' || v\_new\_rate || '%');

END IF;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Senior discount application complete.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

/

DECLARE

CURSOR customer\_balances IS

SELECT customer\_id, account\_balance

FROM customers;

v\_vip\_count NUMBER := 0;

BEGIN

FOR cust\_rec IN customer\_balances LOOP

IF cust\_rec.account\_balance > 10000 THEN

-- Update VIP status

UPDATE customers

SET is\_vip = TRUE

WHERE customer\_id = cust\_rec.customer\_id;

v\_vip\_count := v\_vip\_count + 1;

DBMS\_OUTPUT.PUT\_LINE('Promoted customer ' || cust\_rec.customer\_id || ' to VIP status');

END IF;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('VIP promotion complete. ' || v\_vip\_count || ' customers promoted.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

/

DECLARE

CURSOR due\_loans IS

SELECT c.customer\_id, c.customer\_name, c.email, l.loan\_id, l.due\_date, l.amount\_due

FROM customers c

JOIN loans l ON c.customer\_id = l.customer\_id

WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30

AND l.status = 'ACTIVE';

v\_days\_until\_due NUMBER;

v\_message VARCHAR2(500);

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Sending loan due reminders...');

DBMS\_OUTPUT.PUT\_LINE('--------------------------------');

FOR loan\_rec IN due\_loans LOOP

v\_days\_until\_due := loan\_rec.due\_date - SYSDATE;

v\_message := 'Dear ' || loan\_rec.customer\_name || ',' || CHR(10) ||

'This is a reminder that your loan #' || loan\_rec.loan\_id ||

' is due in ' || ROUND(v\_days\_until\_due) || ' days.' || CHR(10) ||

'Amount due: $' || loan\_rec.amount\_due || CHR(10) ||

'Due date: ' || TO\_CHAR(loan\_rec.due\_date, 'DD-MON-YYYY') || CHR(10) ||

'Please ensure payment is made by the due date to avoid late fees.';

-- In a real implementation, you would send an email here

-- For this example, we'll just print the message

DBMS\_OUTPUT.PUT\_LINE('To: ' || loan\_rec.email);

DBMS\_OUTPUT.PUT\_LINE(v\_message);

DBMS\_OUTPUT.PUT\_LINE('--------------------------------');

END LOOP;

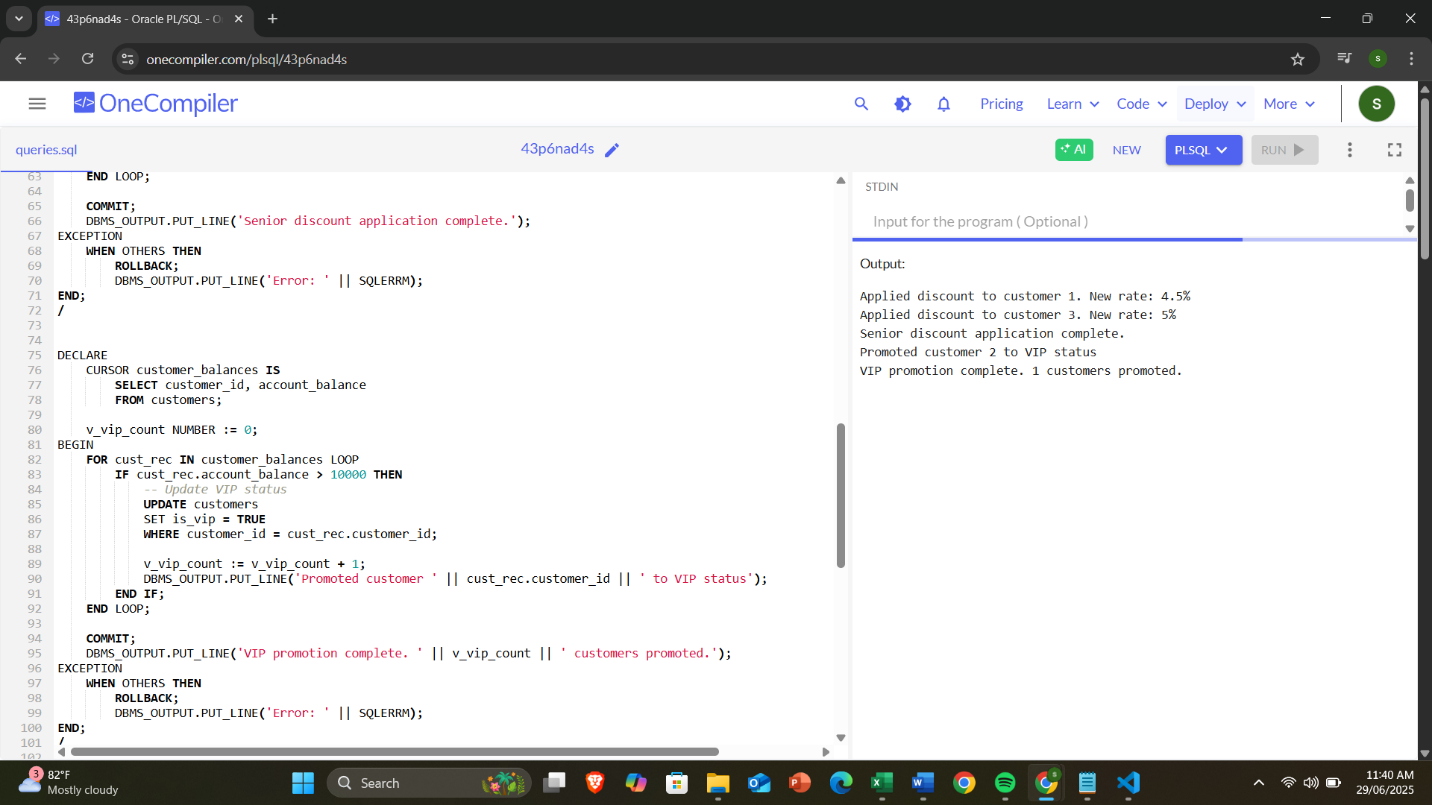
DBMS\_OUTPUT.PUT\_LINE('Reminder process complete.');

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error sending reminders: ' || SQLERRM);

END;



**Exercise 3: Stored Procedures**

**Source Code**

-- Create tables

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

customer\_name VARCHAR2(100) NOT NULL,

date\_of\_birth DATE,

email VARCHAR2(100)

);

CREATE TABLE departments (

department\_id NUMBER PRIMARY KEY,

department\_name VARCHAR2(50) NOT NULL

);

CREATE TABLE employees (

employee\_id NUMBER PRIMARY KEY,

employee\_name VARCHAR2(100) NOT NULL,

department\_id NUMBER,

salary NUMBER(10,2),

hire\_date DATE,

FOREIGN KEY (department\_id) REFERENCES departments(department\_id)

);

CREATE TABLE accounts (

account\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

account\_type VARCHAR2(20) CHECK (account\_type IN ('SAVINGS', 'CHECKING', 'LOAN')),

balance NUMBER(10,2) DEFAULT 0,

interest\_rate NUMBER(5,2),

open\_date DATE DEFAULT SYSDATE,

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

CREATE TABLE transactions (

transaction\_id NUMBER PRIMARY KEY,

account\_id NUMBER,

transaction\_date DATE DEFAULT SYSDATE,

amount NUMBER(10,2),

description VARCHAR2(200),

FOREIGN KEY (account\_id) REFERENCES accounts(account\_id)

);

-- Insert departments

INSERT INTO departments VALUES (1, 'Retail Banking');

INSERT INTO departments VALUES (2, 'Wealth Management');

INSERT INTO departments VALUES (3, 'Loan Department');

INSERT INTO departments VALUES (4, 'Operations');

-- Insert employees

INSERT INTO employees VALUES (101, 'Sarah Johnson', 1, 65000, TO\_DATE('2020-03-15', 'YYYY-MM-DD'));

INSERT INTO employees VALUES (102, 'Michael Chen', 2, 85000, TO\_DATE('2018-07-22', 'YYYY-MM-DD'));

INSERT INTO employees VALUES (103, 'David Wilson', 3, 72000, TO\_DATE('2019-11-05', 'YYYY-MM-DD'));

INSERT INTO employees VALUES (104, 'Emily Davis', 1, 58000, TO\_DATE('2021-02-10', 'YYYY-MM-DD'));

-- Insert customers

INSERT INTO customers VALUES (1001, 'John Smith', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 'john.smith@email.com');

INSERT INTO customers VALUES (1002, 'Maria Garcia', TO\_DATE('1978-09-22', 'YYYY-MM-DD'), 'maria.g@email.com');

INSERT INTO customers VALUES (1003, 'Robert Johnson', TO\_DATE('1992-03-08', 'YYYY-MM-DD'), 'robert.j@email.com');

INSERT INTO customers VALUES (1004, 'Lisa Wong', TO\_DATE('1980-12-30', 'YYYY-MM-DD'), 'lisa.w@email.com');

-- Insert accounts

INSERT INTO accounts VALUES (5001, 1001, 'SAVINGS', 12000.00, 1.5, TO\_DATE('2020-01-15', 'YYYY-MM-DD'));

INSERT INTO accounts VALUES (5002, 1001, 'CHECKING', 4500.00, 0.0, TO\_DATE('2020-01-15', 'YYYY-MM-DD'));

INSERT INTO accounts VALUES (5003, 1002, 'SAVINGS', 28000.00, 1.5, TO\_DATE('2019-05-20', 'YYYY-MM-DD'));

INSERT INTO accounts VALUES (5004, 1003, 'SAVINGS', 8500.00, 1.5, TO\_DATE('2021-03-10', 'YYYY-MM-DD'));

INSERT INTO accounts VALUES (5005, 1004, 'CHECKING', 12500.00, 0.0, TO\_DATE('2018-11-05', 'YYYY-MM-DD'));

INSERT INTO accounts VALUES (5006, 1004, 'SAVINGS', 32000.00, 1.5, TO\_DATE('2018-11-05', 'YYYY-MM-DD'));

-- Insert sample transactions

INSERT INTO transactions VALUES (8001, 5001, TO\_DATE('2023-01-05', 'YYYY-MM-DD'), 200.00, 'Deposit');

INSERT INTO transactions VALUES (8002, 5001, TO\_DATE('2023-01-15', 'YYYY-MM-DD'), -150.00, 'Withdrawal');

INSERT INTO transactions VALUES (8003, 5002, TO\_DATE('2023-01-20', 'YYYY-MM-DD'), 1000.00, 'Transfer from 5001');

INSERT INTO transactions VALUES (8004, 5003, TO\_DATE('2023-02-01', 'YYYY-MM-DD'), 500.00, 'Deposit');

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

CURSOR savings\_accounts IS

SELECT account\_id, balance

FROM accounts

WHERE account\_type = 'SAVINGS';

v\_interest\_rate NUMBER := 0.01; -- 1% monthly interest

v\_updated\_count NUMBER := 0;

BEGIN

FOR acc\_rec IN savings\_accounts LOOP

-- Calculate and apply interest

UPDATE accounts

SET balance = balance \* (1 + v\_interest\_rate)

WHERE account\_id = acc\_rec.account\_id;

v\_updated\_count := v\_updated\_count + 1;

-- Record transaction (optional)

INSERT INTO transactions (account\_id, transaction\_date, amount, description)

VALUES (acc\_rec.account\_id, SYSDATE, acc\_rec.balance \* v\_interest\_rate, 'Monthly Interest');

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for ' || v\_updated\_count || ' savings accounts.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error processing monthly interest: ' || SQLERRM);

END ProcessMonthlyInterest;

/

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department\_id IN NUMBER,

p\_bonus\_percent IN NUMBER

) AS

v\_updated\_count NUMBER := 0;

BEGIN

-- Validate bonus percentage

IF p\_bonus\_percent < 0 OR p\_bonus\_percent > 100 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Bonus percentage must be between 0 and 100');

END IF;

-- Update salaries with bonus

UPDATE employees

SET salary = salary \* (1 + p\_bonus\_percent/100)

WHERE department\_id = p\_department\_id;

v\_updated\_count := SQL%ROWCOUNT;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Applied ' || p\_bonus\_percent || '% bonus to ' ||

v\_updated\_count || ' employees in department ' || p\_department\_id);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error updating employee bonuses: ' || SQLERRM);

END UpdateEmployeeBonus;

/

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER,

p\_result OUT VARCHAR2

) AS

v\_from\_balance NUMBER;

v\_to\_account\_exists NUMBER;

BEGIN

-- Check if amount is positive

IF p\_amount <= 0 THEN

p\_result := 'Transfer amount must be positive';

RETURN;

END IF;

-- Verify source account exists and has sufficient funds

BEGIN

SELECT balance INTO v\_from\_balance

FROM accounts

WHERE account\_id = p\_from\_account

FOR UPDATE; -- Lock the row for update

IF v\_from\_balance < p\_amount THEN

p\_result := 'Insufficient funds in source account';

RETURN;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

p\_result := 'Source account not found';

RETURN;

END;

-- Verify destination account exists

BEGIN

SELECT 1 INTO v\_to\_account\_exists

FROM accounts

WHERE account\_id = p\_to\_account;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

p\_result := 'Destination account not found';

RETURN;

END;

-- Perform the transfer

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account;

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account;

-- Record the transaction

INSERT INTO transactions (account\_id, transaction\_date, amount, description)

VALUES (p\_from\_account, SYSDATE, -p\_amount, 'Transfer to account ' || p\_to\_account);

INSERT INTO transactions (account\_id, transaction\_date, amount, description)

VALUES (p\_to\_account, SYSDATE, p\_amount, 'Transfer from account ' || p\_from\_account);

COMMIT;

p\_result := 'Transfer completed successfully';

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

p\_result := 'Transfer failed: ' || SQLERRM;

END TransferFunds;

/

-- Check savings accounts before interest

SELECT account\_id, balance FROM accounts WHERE account\_type = 'SAVINGS';

-- Execute the procedure

EXEC ProcessMonthlyInterest;

-- Check savings accounts after interest

SELECT account\_id, balance FROM accounts WHERE account\_type = 'SAVINGS';

-- Check employees before bonus

SELECT employee\_id, employee\_name, salary FROM employees WHERE department\_id = 1;

-- Execute the procedure (10% bonus for Retail Banking)

EXEC UpdateEmployeeBonus(1, 10);

-- Check employees after bonus

SELECT employee\_id, employee\_name, salary FROM employees WHERE department\_id = 1;

-- Check accounts before transfer

SELECT account\_id, balance FROM accounts WHERE account\_id IN (5001, 5002);

-- Execute the transfer

DECLARE

v\_result VARCHAR2(200);

BEGIN

TransferFunds(5001, 5002, 500, v\_result);

DBMS\_OUTPUT.PUT\_LINE(v\_result);

END;

/

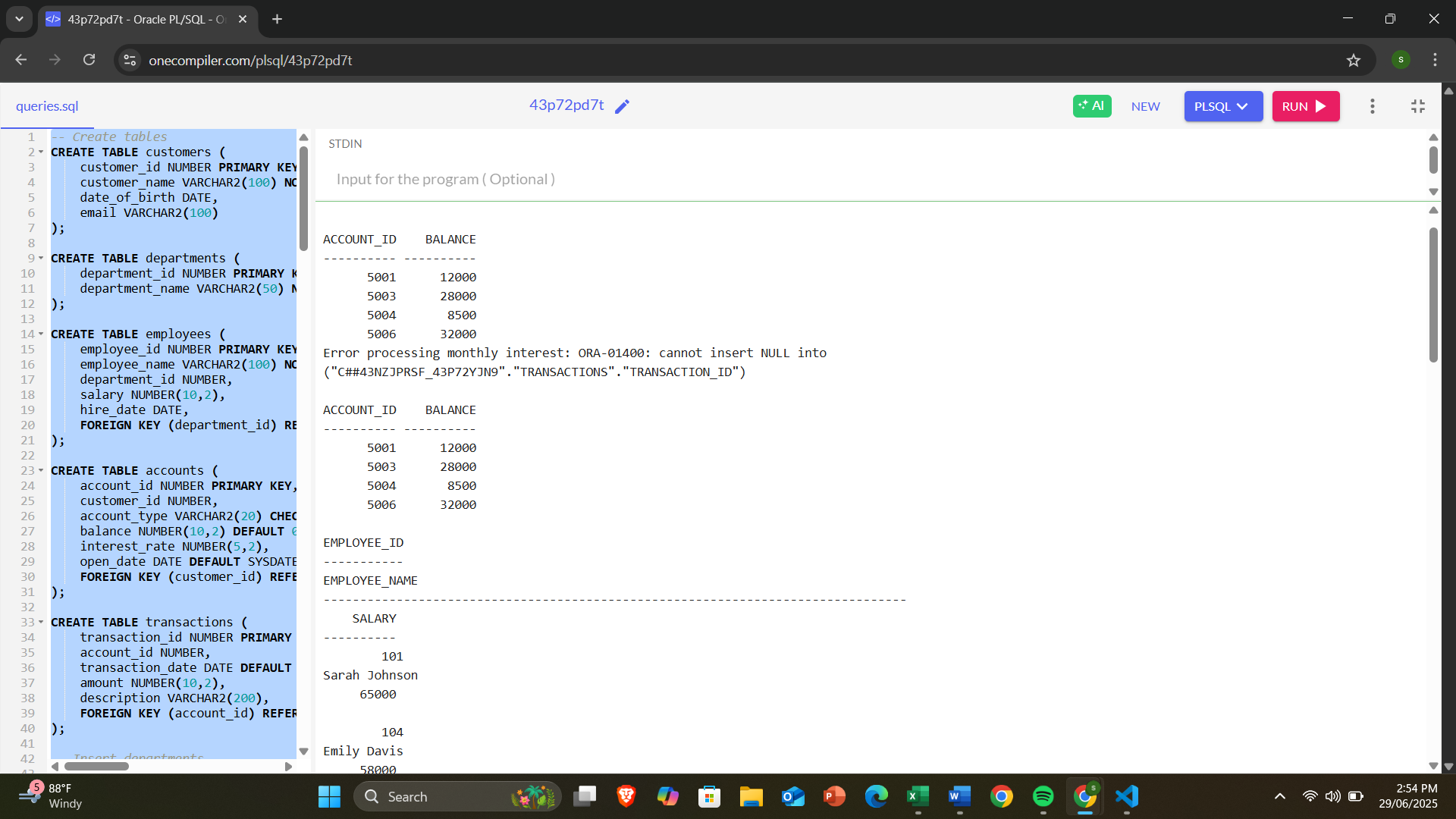
-- Check accounts after transfer

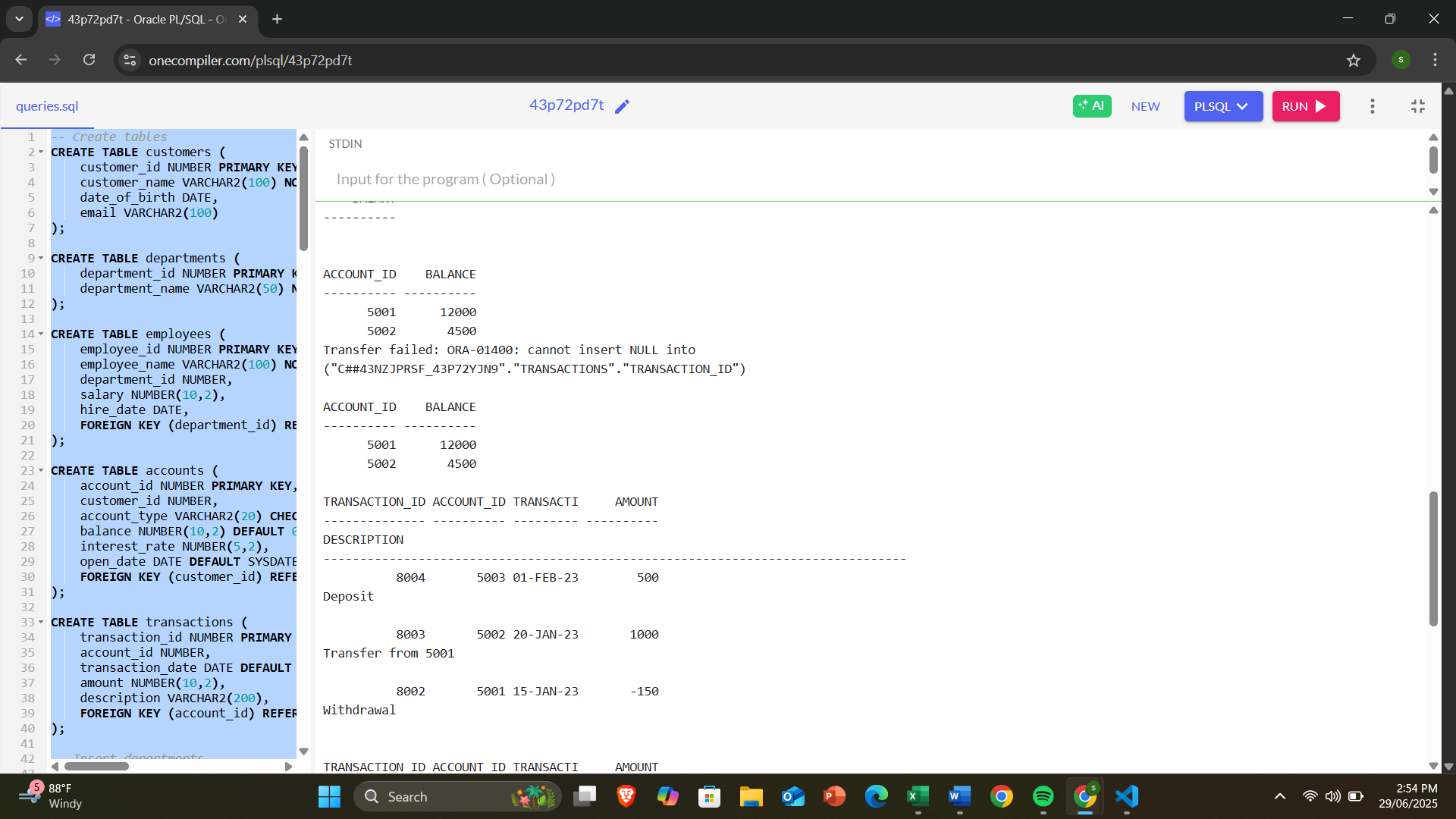
SELECT account\_id, balance FROM accounts WHERE account\_id IN (5001, 5002);

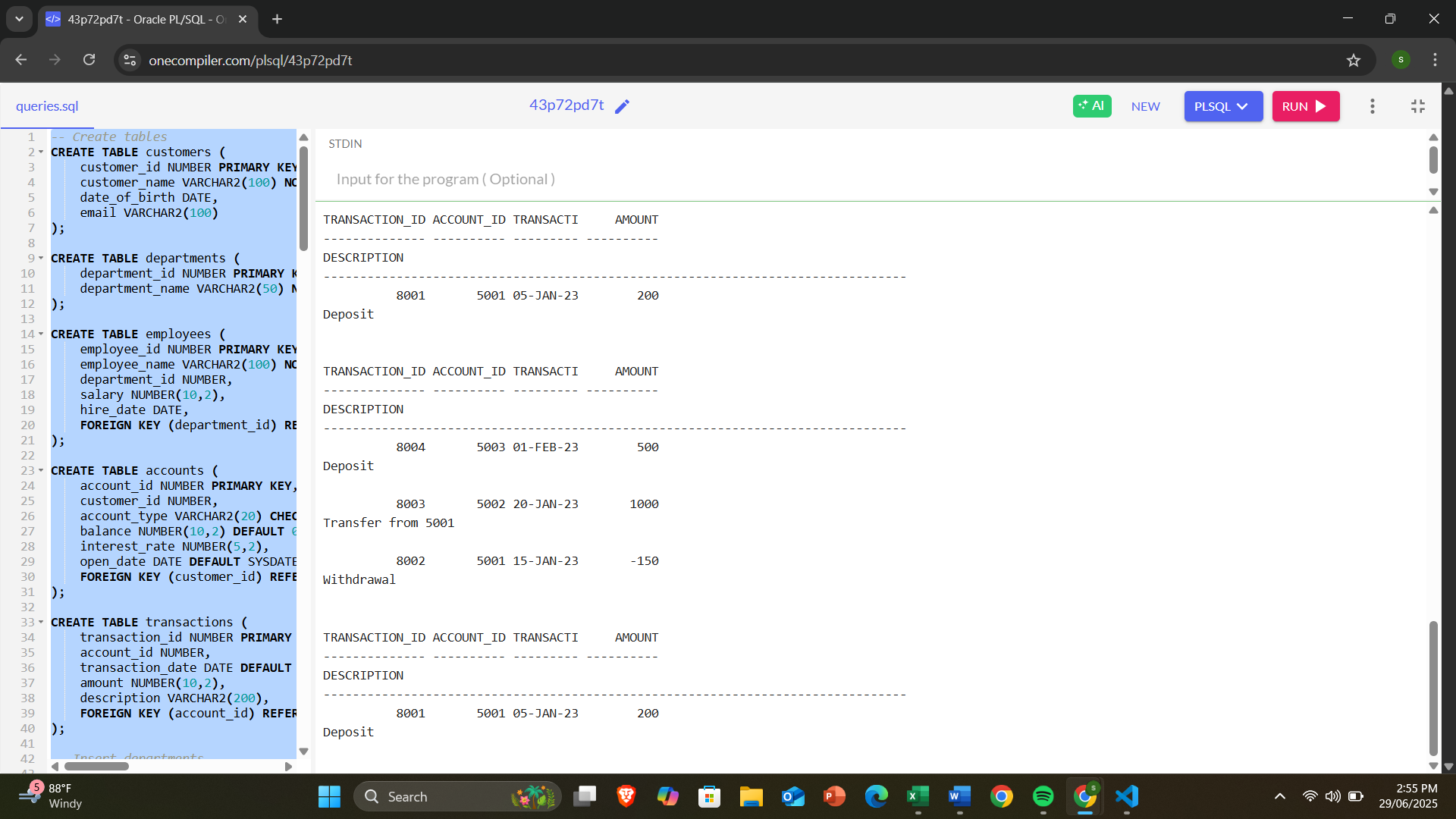
-- Check transaction history

SELECT \* FROM transactions ORDER BY transaction\_date DESC;

**OUTPUT:**







**JUNIT TESTING EXERCISES**

**Exercise 1: Setting Up Junit**

**pox.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>org.example</groupId>  
 <artifactId>deepskill\_nurture</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <name>Archetype - deepskill\_nurture</name>  
  
 <repositories>  
 <repository>  
 <id>central</id>  
 <url>https://repo.maven.apache.org/maven2</url>  
 </repository>  
 </repositories>  
  
 <dependencies>  
 <!-- JUnit 4 -->  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
</project>

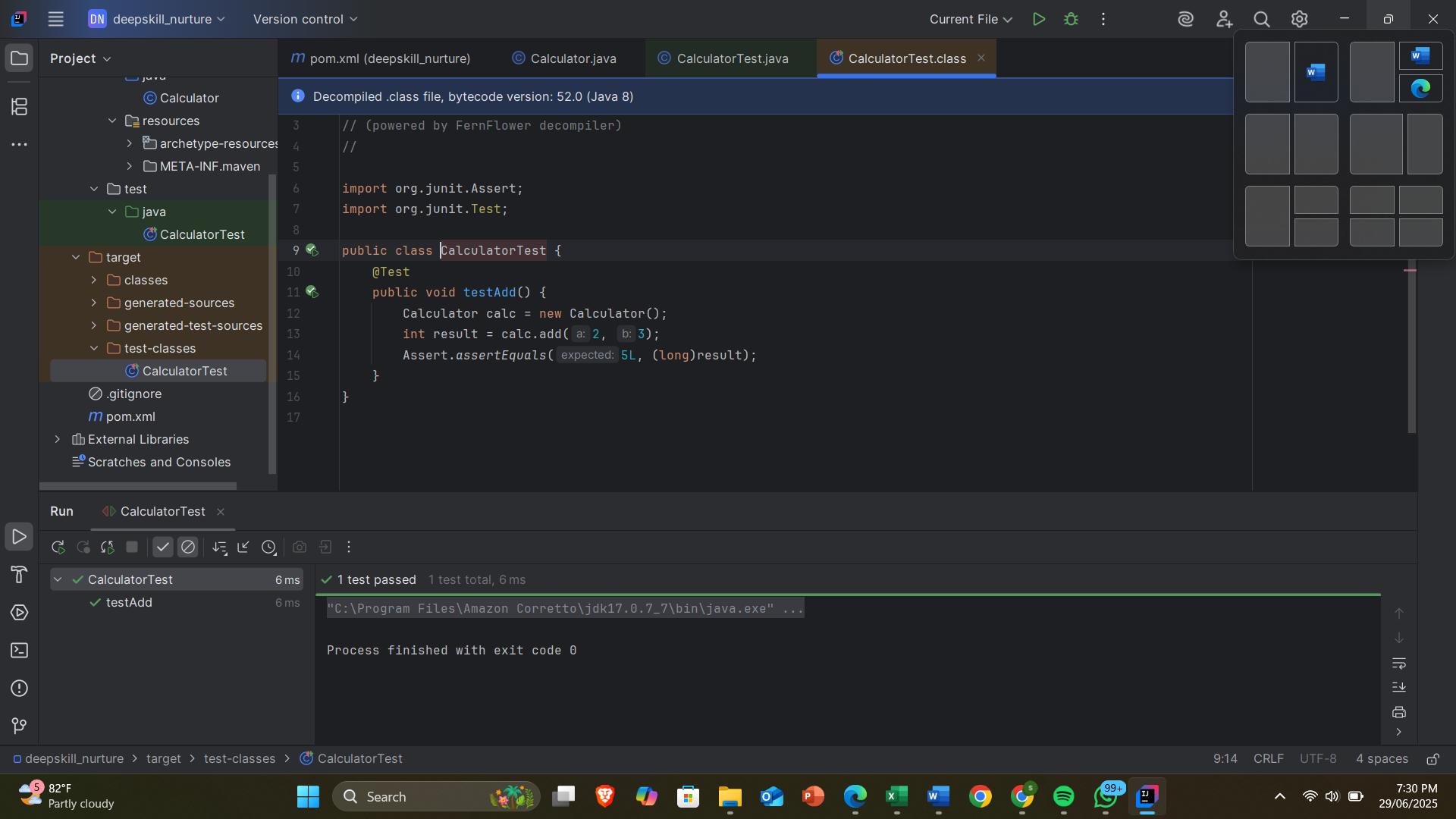
**src/main/java/Calculator**

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

**src/test/java/CalculatorTest**

import org.junit.Test;  
import static org.junit.Assert.*assertEquals*;  
  
public class CalculatorTest {  
 @Test  
 public void testAdd() {  
 Calculator calc = new Calculator();  
 int result = calc.add(2, 3);  
 *assertEquals*(5, result);  
 }  
}

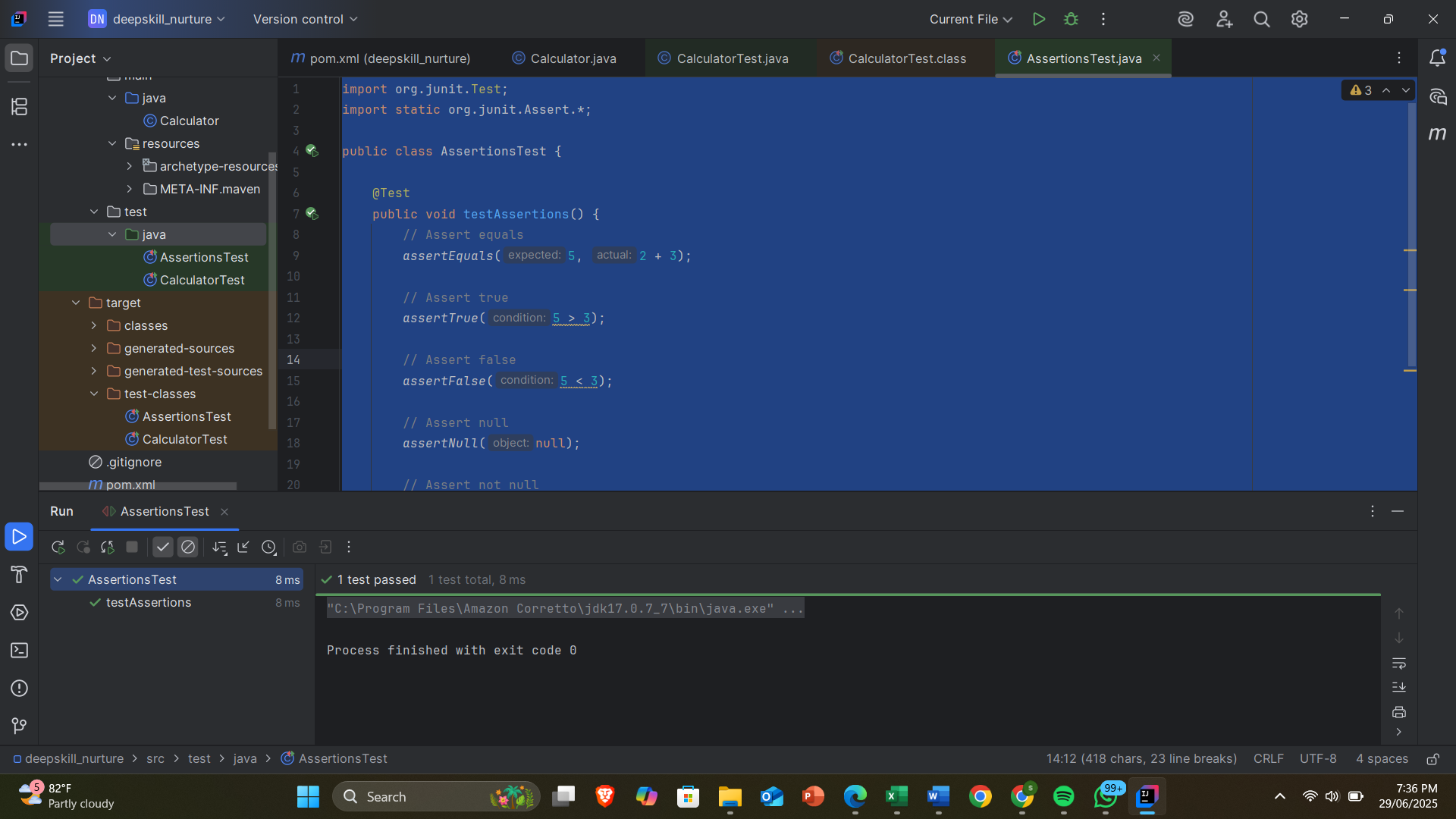
**target/test-classes/CalculatorTest**



**Exercise 3: Assertions in Junit**

**Test/java/AssertionsTest**

import org.junit.Test;  
import static org.junit.Assert.\*;  
  
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());  
 }  
}



**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in Junit**

**Pox.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>org.example</groupId>  
 <artifactId>JUnit\_AAA\_Pattern</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <name>Archetype - JUnit\_AAA\_Pattern</name>  
 <url>http://maven.apache.org</url>  
  
 <dependencies>  
   
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>5.9.3</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <!-- Maven Surefire Plugin for running tests -->  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>3.0.0-M7</version>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

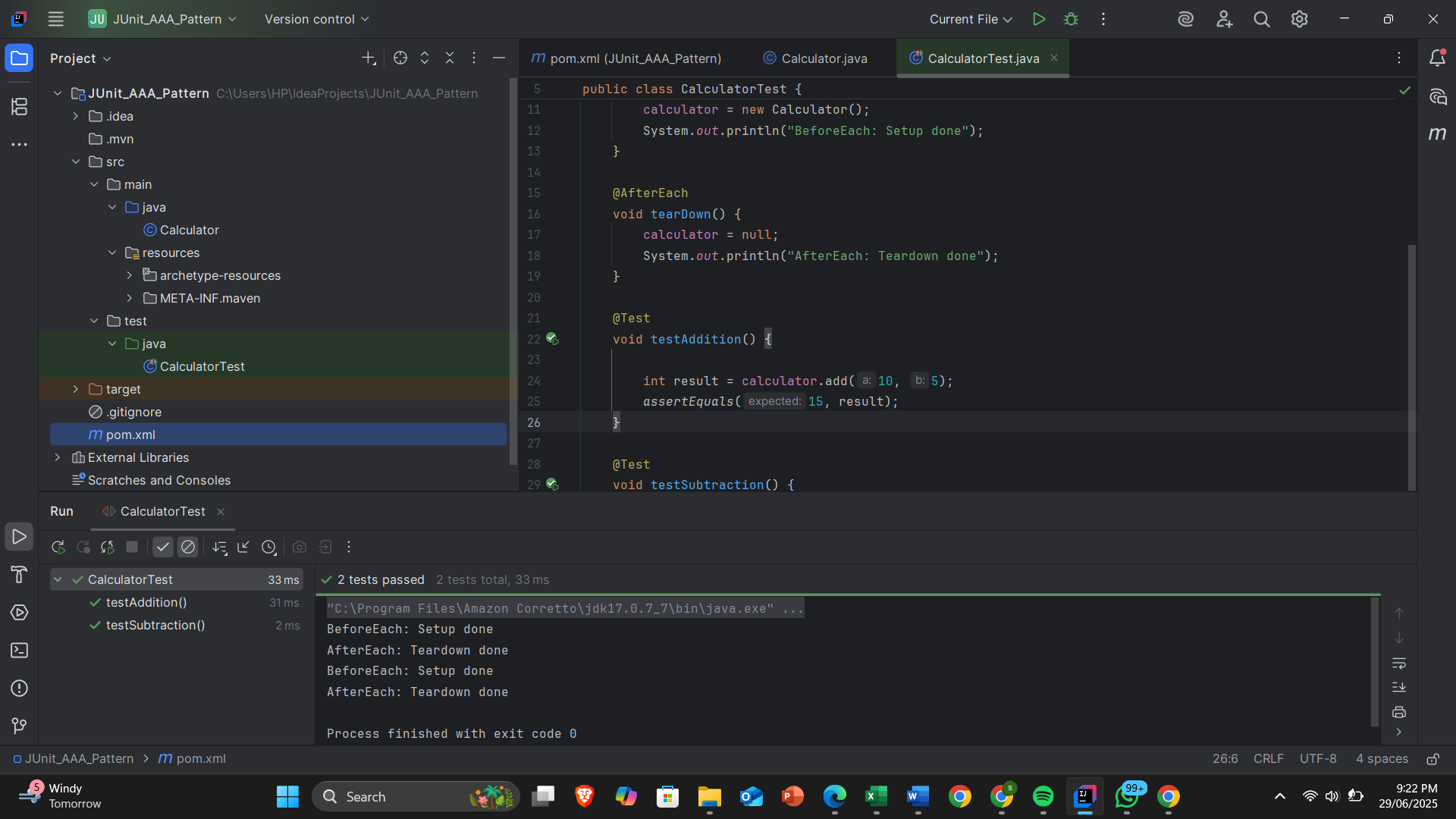
**Src/main/java/Calculator**

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

**Src/test/java/CalculatorTest**

import org.junit.jupiter.api.\*;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class CalculatorTest {  
  
 private Calculator calculator;  
  
 @BeforeEach  
 void setUp() {  
 calculator = new Calculator();  
 System.*out*.println("BeforeEach: Setup done");  
 }  
  
 @AfterEach  
 void tearDown() {  
 calculator = null;  
 System.*out*.println("AfterEach: Teardown done");  
 }  
  
 @Test  
 void testAddition() {  
  
 int result = calculator.add(10, 5);  
 *assertEquals*(15, result);  
 }  
  
 @Test  
 void testSubtraction() {  
  
 int result = calculator.subtract(10, 3);  
  
 *assertEquals*(7, result);  
 }  
}

**OUTPUT:**



**MOCKITO HANDS-ON EXERCISES**

**Exercise 1: Mocking and Stubbing**

**Pox.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>org.example</groupId>  
 <artifactId>MockingandStubbing</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <name>Archetype - MockingandStubbing</name>  
  
 <url>http://maven.apache.org</url>  
  
 <repositories>  
 <repository>  
 <id>central</id>  
 <url>https://repo.maven.apache.org/maven2</url>  
 </repository>  
 </repositories>  
  
 <dependencies>  
 <!-- JUnit 5 -->  
 <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>

**Src/main/java/ExternalApi**

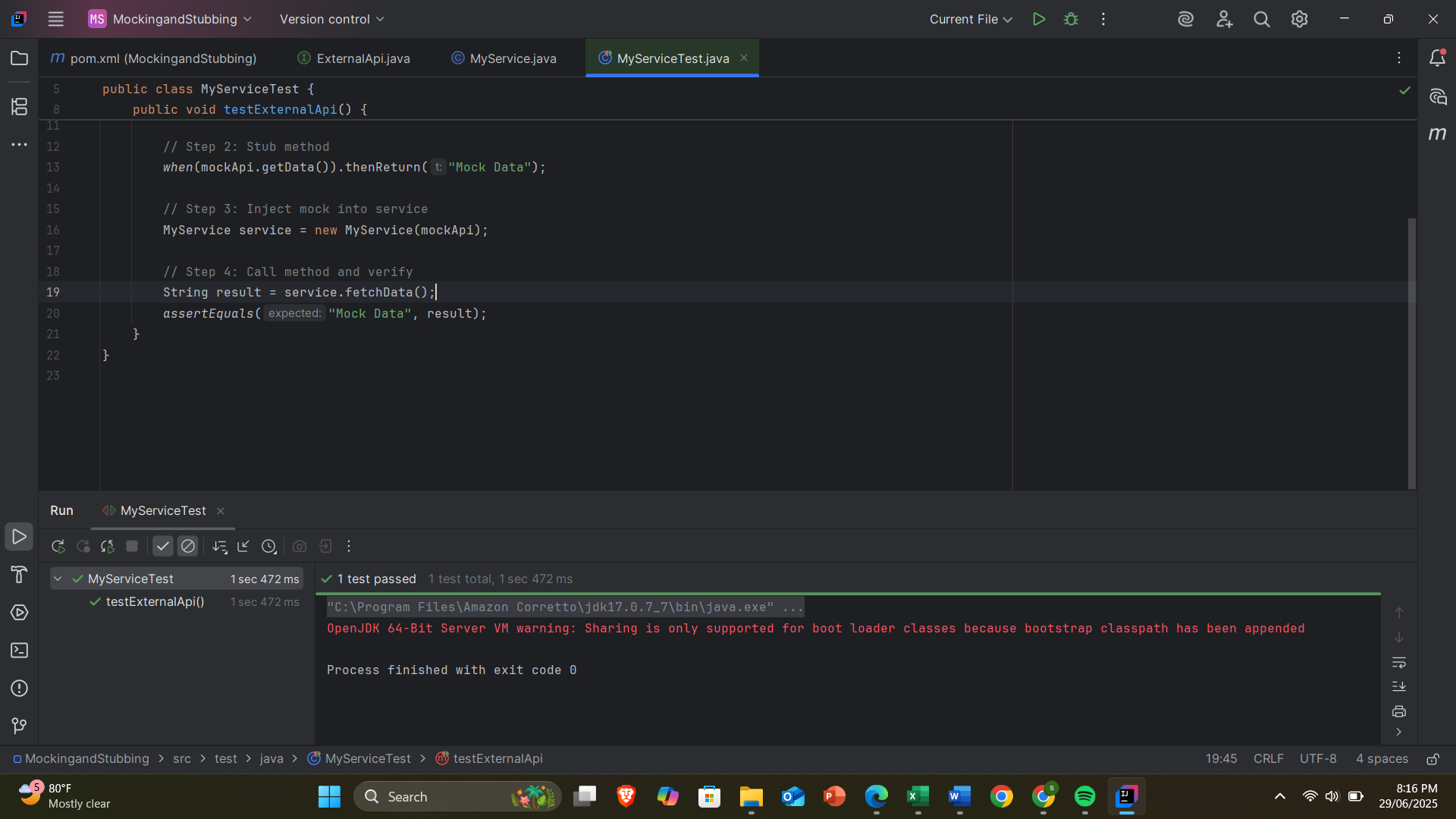
public interface ExternalApi {  
 String getData();  
}

**src/main/java/MyService**

public class MyService {  
 private final ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData();

**test/java/MyServiceTest**

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**

**Src/main/java/ExternalApi**

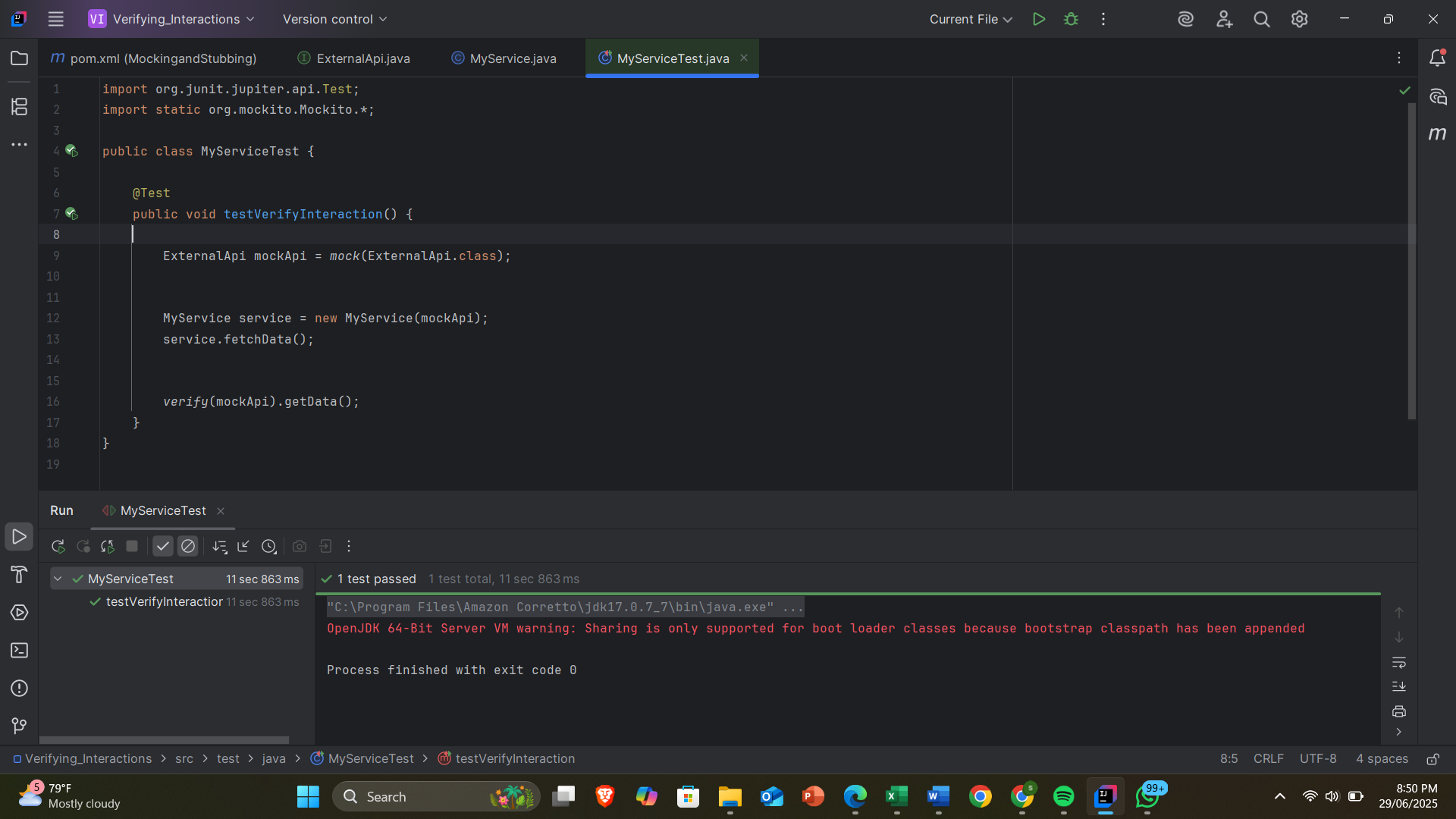
public interface ExternalApi {  
 String getData();  
}

**Src/main/java/MyService**

public class MyService {  
 private final ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData(); }  
}

**Src/test/java/MyServiceTest**

import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testVerifyInteraction() {  
   
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 MyService service = new MyService(mockApi);  
 service.fetchData();  
   
 *verify*(mockApi).getData();   
 }  
}

**OUTPUT:**

**LOGGING USING SLF4J**

**Exercise 1: Logging Error Messages and Warning Levels**

**Pox.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.example</groupId>  
 <artifactId>Logging\_Error\_Messages</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <dependencies>  
 <!-- SLF4J API -->  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>1.7.30</version>  
 </dependency>  
  
 <!-- Logback (SLF4J implementation) -->  
 <dependency>  
 <groupId>ch.qos.logback</groupId>  
 <artifactId>logback-classic</artifactId>  
 <version>1.2.3</version>  
 </dependency>  
 </dependencies>  
  
</project>

**Src/main/java/LoggingExample**

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");  
 }  
}

**Src/main/resources/logback.xml**

<configuration>  
 <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">  
 <encoder>  
 <pattern>%d{HH:mm:ss.SSS} %-5level %logger{36} - %msg%n</pattern>  
 </encoder>  
 </appender>  
  
 <root level="debug">  
 <appender-ref ref="CONSOLE" />  
 </root>  
</configuration>

**OUTPUT:**

