**Week 2**

***PLSQL Exercises***

**Control Structures**

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

dob DATE,

balance NUMBER,

is\_vip CHAR(1) DEFAULT 'N'

);

INSERT INTO customers VALUES (1, 'Alice', DATE '1950-05-10', 12000, 'N');

INSERT INTO customers VALUES (2, 'Bob', DATE '1970-01-15', 8000, 'N');

ALTER TABLE customers ADD (interest\_rate NUMBER DEFAULT 5);

UPDATE customers SET interest\_rate = 5;

DECLARE

CURSOR c\_cust IS

SELECT customer\_id, dob, interest\_rate

FROM customers;

v\_age NUMBER;

v\_new\_rate NUMBER;

BEGIN

FOR rec IN c\_cust LOOP

SELECT TRUNC(MONTHS\_BETWEEN(SYSDATE, rec.dob) / 12)

INTO v\_age

FROM dual;

IF v\_age > 60 THEN

v\_new\_rate := rec.interest\_rate - 1;

UPDATE customers

SET interest\_rate = v\_new\_rate

WHERE customer\_id = rec.customer\_id;

DBMS\_OUTPUT.PUT\_LINE(

'Cust ' || rec.customer\_id

|| ' age ' || v\_age

|| ': new rate = ' || v\_new\_rate

);

END IF;

END LOOP;

COMMIT;

END;

/

BEGIN

FOR cust\_rec IN (

SELECT customer\_id

FROM customers

WHERE balance > 10000

) LOOP

UPDATE customers

SET is\_vip = 'Y'

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

DBMS\_OUTPUT.PUT\_LINE(

'Cust ' || cust\_rec.customer\_id

|| ' flagged as VIP'

);

END LOOP;

COMMIT;

END;

/

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER NOT NULL,

principal NUMBER(12,2),

interest\_rate NUMBER(5,2),

start\_date DATE,

due\_date DATE,

status VARCHAR2(20)

);

INSERT INTO loans VALUES (101, 1, 5000, 5.50, SYSDATE - 100, SYSDATE + 10, 'ACTIVE');

INSERT INTO loans VALUES (102, 2, 15000, 6.25, SYSDATE - 200, SYSDATE + 40, 'ACTIVE');

INSERT INTO loans VALUES (103, 3, 8000, 4.90, SYSDATE - 400, SYSDATE - 10, 'PAID');

BEGIN

FOR loan\_rec IN (

SELECT l.loan\_id, l.customer\_id, l.due\_date, c.name

FROM loans l

JOIN customers c

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

WHERE l.due\_date BETWEEN TRUNC(SYSDATE)

AND TRUNC(SYSDATE) + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Loan ' || loan\_rec.loan\_id

|| ' for Customer ' || loan\_rec.customer\_id

|| ' (' || loan\_rec.name || ') '

|| 'is due on ' || TO\_CHAR(loan\_rec.due\_date, 'DD-MON-YYYY')

);

END LOOP;

END;

/

**Stored Procedures**

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

customer\_id NUMBER NOT NULL,

AccountType VARCHAR2(20) NOT NULL,

Balance NUMBER(15,2) DEFAULT 0 CHECK (Balance >= 0),

LastModified DATE DEFAULT SYSDATE,

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

)

INSERT INTO Accounts (AccountID, customer\_id, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE)

INSERT INTO Accounts (AccountID, customer\_id, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE)

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE accounts

SET balance = ROUND(Balance \* 1.01, 2)

WHERE AccountType = 'SAVINGS';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' savings accounts updated with 1% interest.');

END ProcessMonthlyInterest;

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

INSERT INTO Employees VALUES (1, 'Alice Johnson', 'Branch Manager', 75000, 'Operations', DATE '2015-06-15');

INSERT INTO Employees VALUES (2, 'Bob Singh', 'Customer Support', 45000, 'Customer Service', DATE '2018-09-01');

INSERT INTO Employees VALUES (3, 'Carol Chen', 'Loan Officer', 55000, 'Lending', DATE '2019-11-20');

INSERT INTO Employees VALUES (4, 'David Kumar', 'IT Analyst', 65000, 'IT', DATE '2017-03-05');

INSERT INTO Employees VALUES (5, 'Eva Patel', 'HR Coordinator', 48000, 'Human Resources', DATE '2020-01-10');

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_pct IN NUMBER

) AS

BEGIN

UPDATE Employees

SET Salary = Salary \* (1 + p\_bonus\_pct/100)

WHERE Department = p\_department;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(

'Updated salaries in department "' || p\_department ||

'" by ' || p\_bonus\_pct || '%, rows affected: ' || SQL%ROWCOUNT

);

END UpdateEmployeeBonus;

/

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_acct IN NUMBER,

p\_to\_acct IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

-- Lock the source account row and check current balance

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_acct

FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in account ' || p\_from\_acct);

END IF;

-- Deduct from source account

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_acct;

-- Add to destination account

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_acct;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || TO\_CHAR(p\_amount) ||

' from account ' || p\_from\_acct ||

' to ' || p\_to\_acct);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

RAISE\_APPLICATION\_ERROR(-20002, 'One or both account IDs not found');

WHEN OTHERS THEN

ROLLBACK;

RAISE;

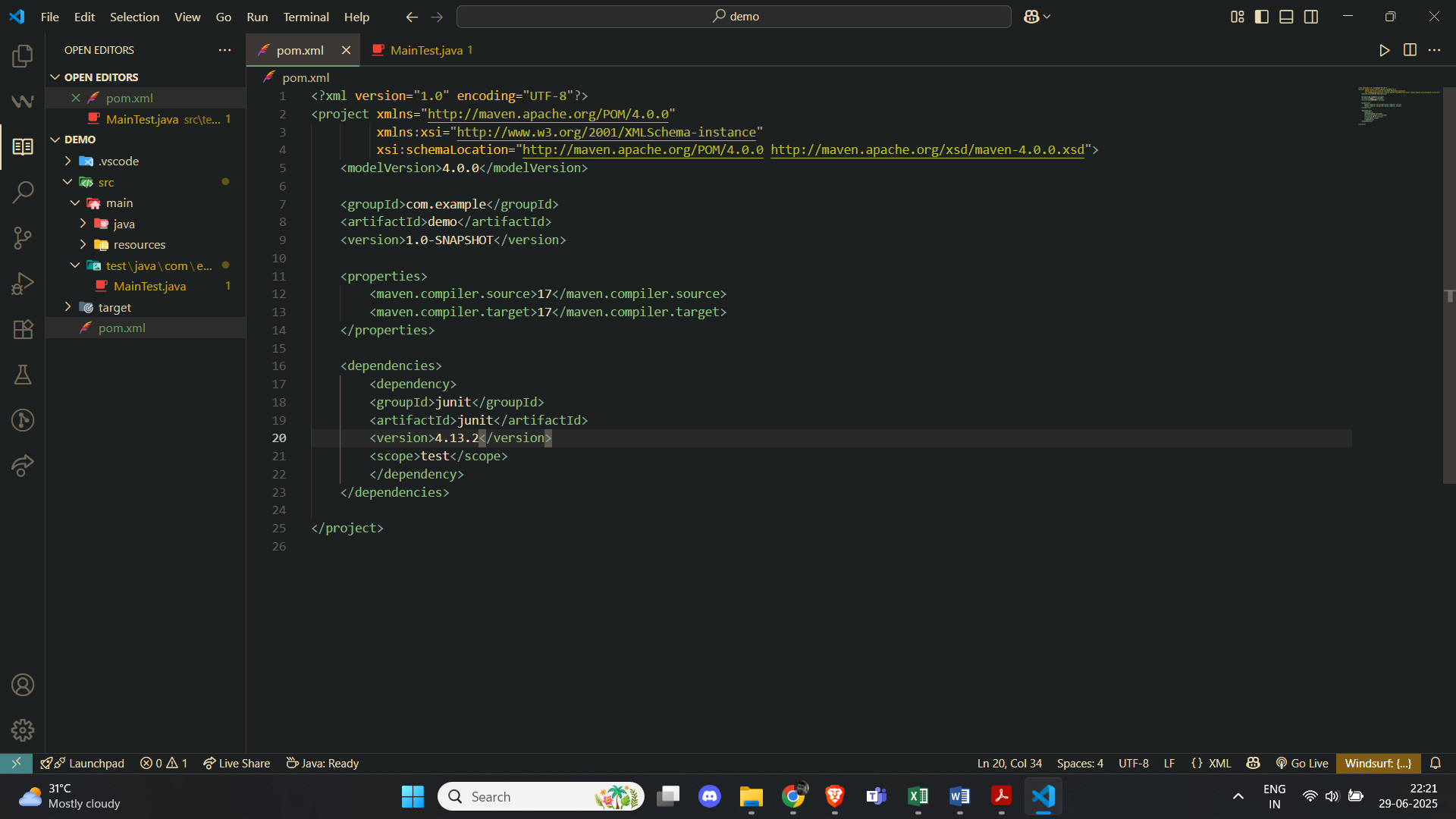
END TransferFunds;

/

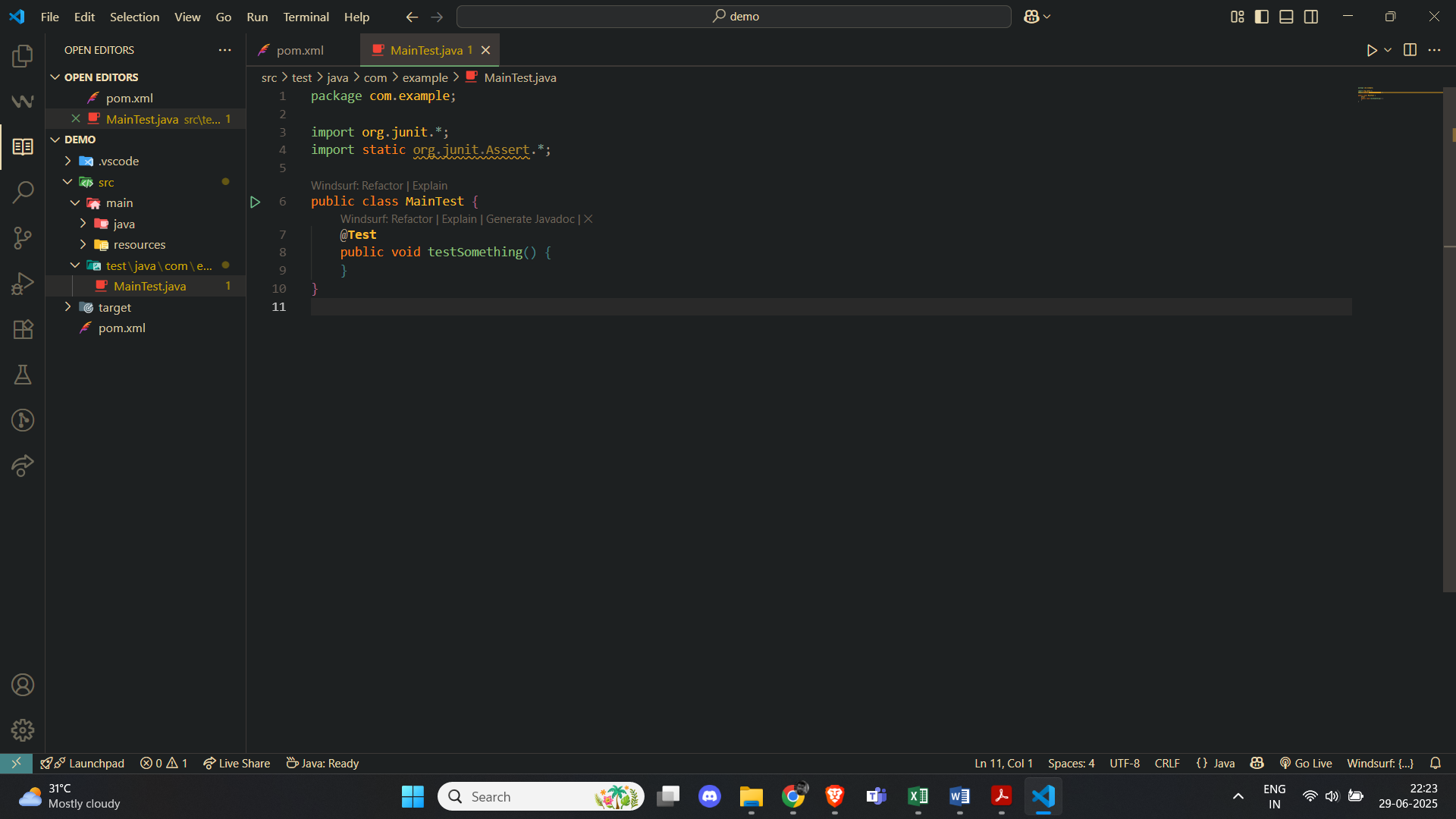
***JUnit Basic Testing Exercises***

**Setting Up JUnit**

The pom.xml file

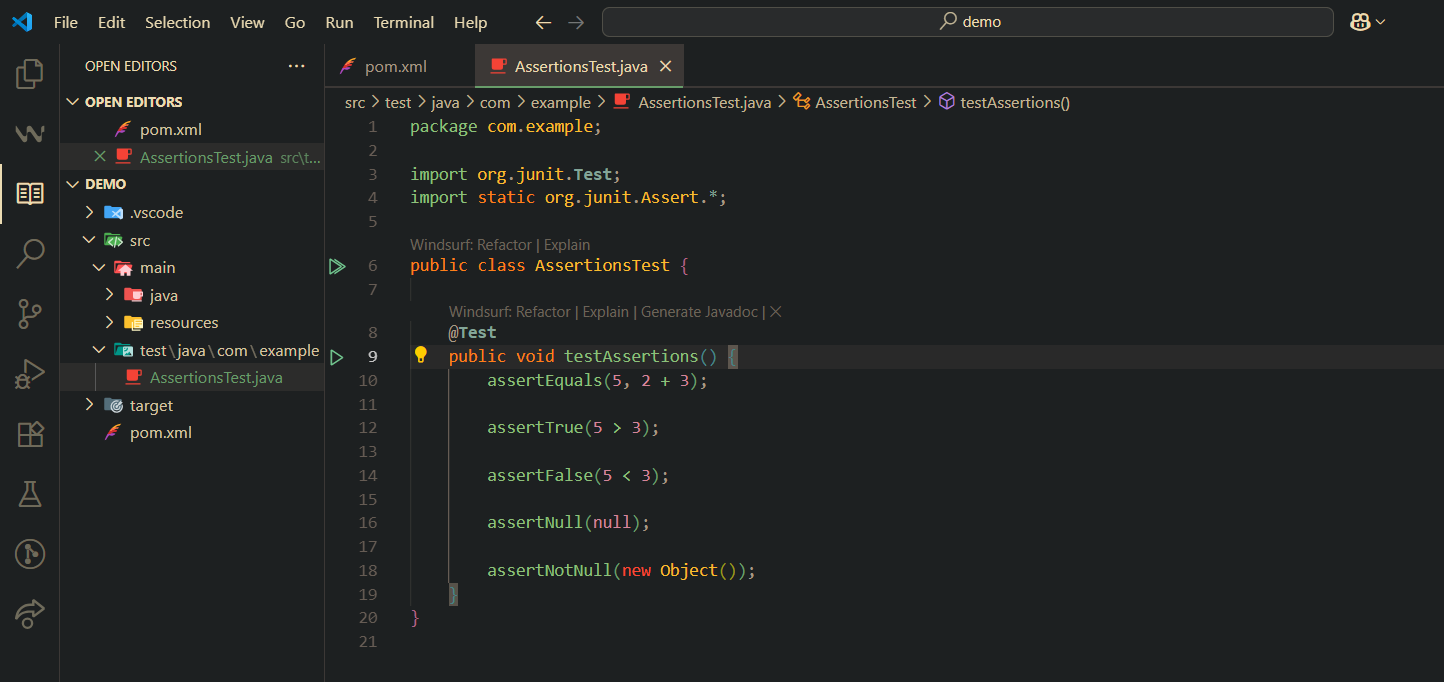


Test class in my project

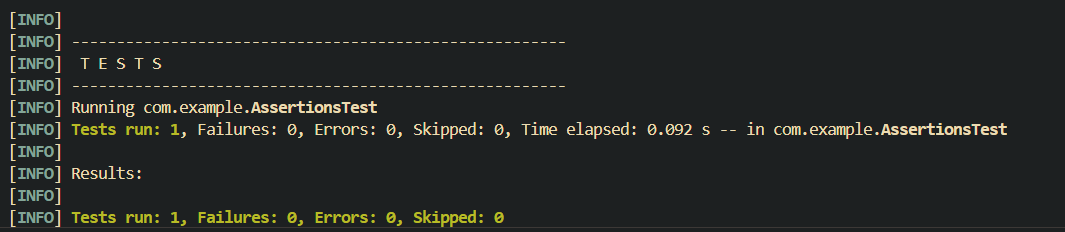


**Assertions in JUnit**

AssertionTest.java

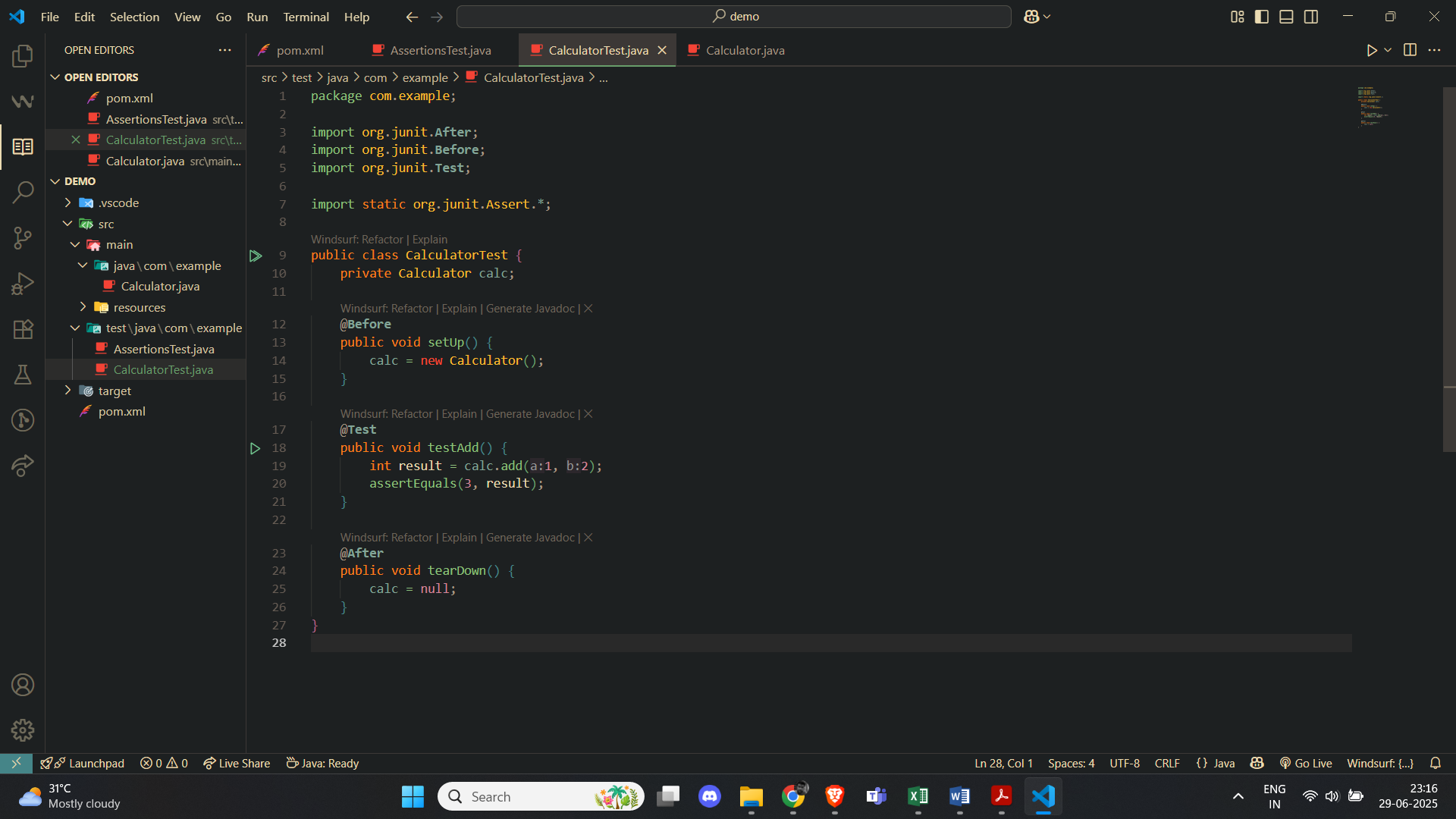


Output

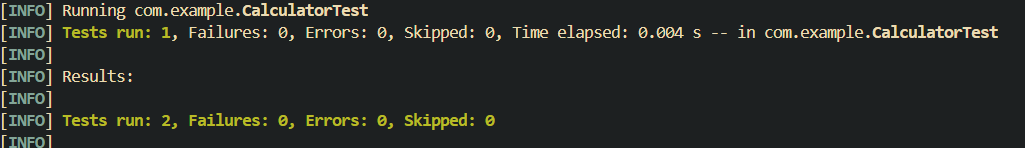


**Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and teardown Methods in JUnit**

CalculatorTest.java



Output



***Mockito exercises***

**Mocking and Stubbing**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

class MyServiceTest {

@Test

void testExternalApiReturnsMockData() {

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);

}

}

**Verifying Interactions**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

class MyServiceTest {

@Test

void testVerifyApiCall() {

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

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi, times(1)).getData();

verifyNoMoreInteractions(mockApi);

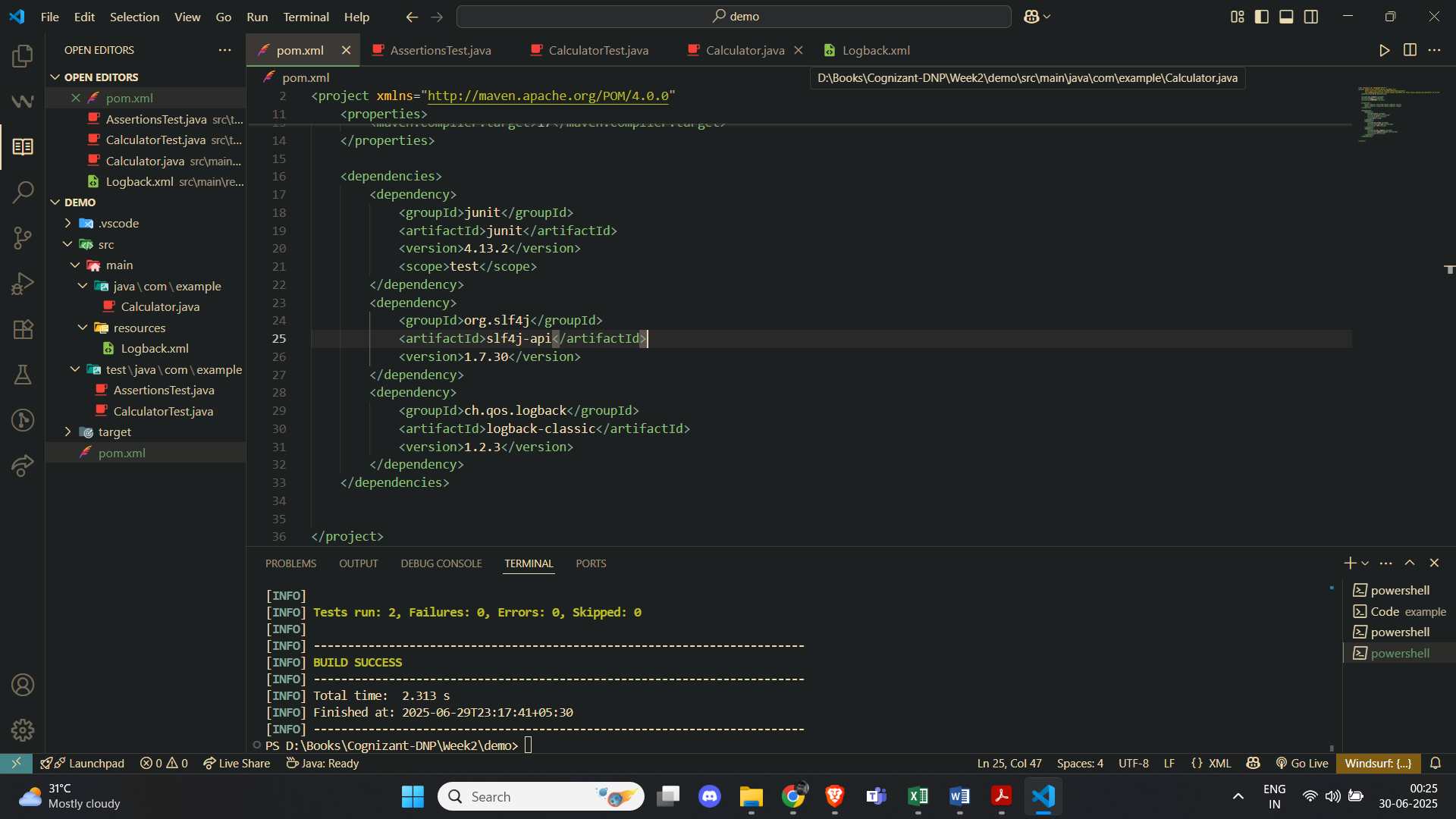
}

}

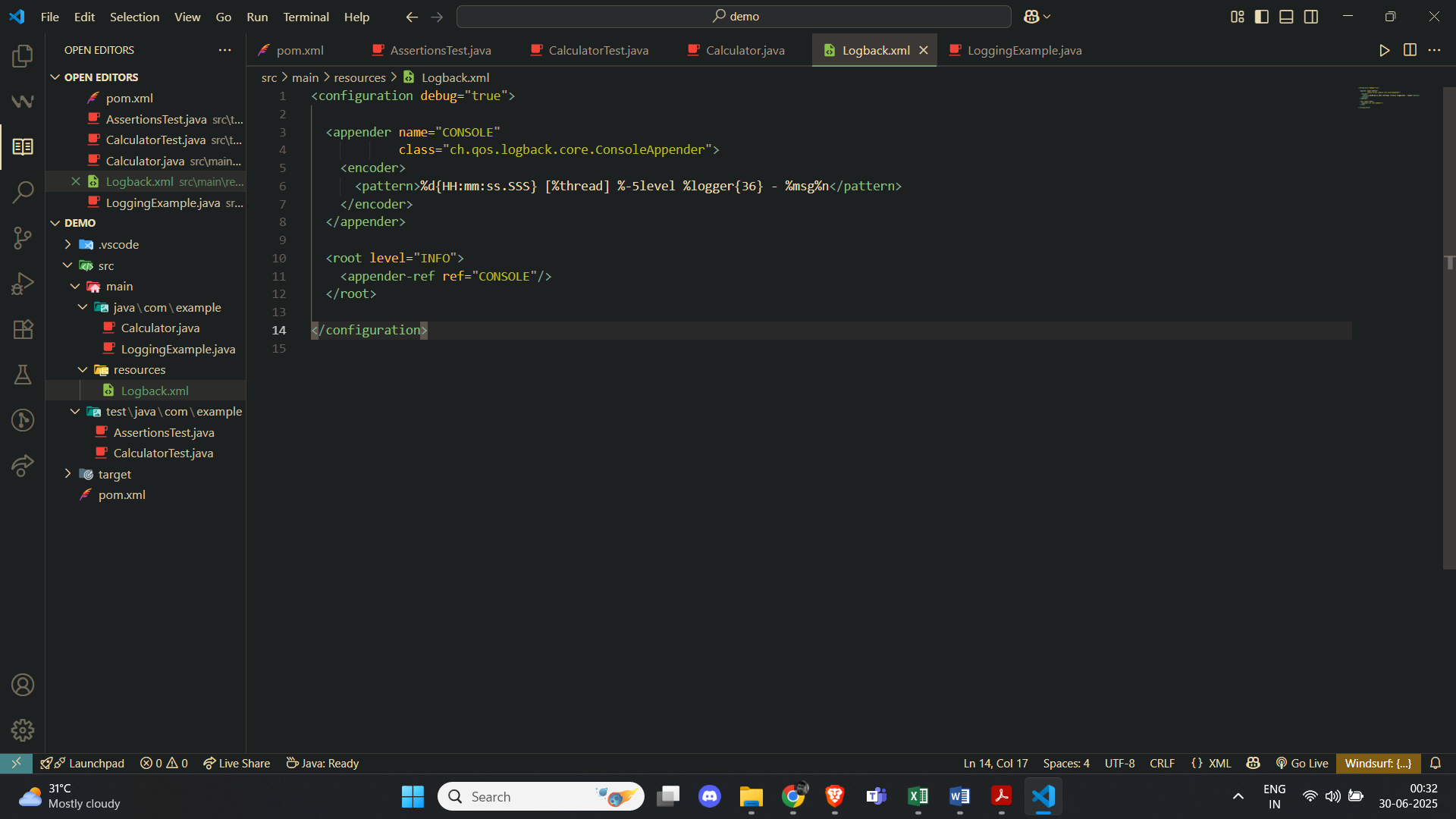
***SL4J Logging exercises***

**Logging Error Messages and Warning Levels**

pom.xml



logback.xml



LoggingExample.java

