***1)CONTROL STRUCTURES***

SET SERVEROUTPUT ON;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE loans';

EXECUTE IMMEDIATE 'DROP TABLE customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

age NUMBER,

balance NUMBER(10,2),

loan\_interest\_rate NUMBER(5,2),

IsVIP VARCHAR2(5)

);

CREATE TABLE loans (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER,

due\_date DATE,

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

);

INSERT INTO customers VALUES (101, 'John', 65, 15000, 7.5, 'FALSE');

INSERT INTO customers VALUES (102, 'Alice', 45, 8000, 8.0, 'FALSE');

INSERT INTO customers VALUES (103, 'Bob', 70, 11000, 6.5, 'FALSE');

INSERT INTO customers VALUES (104, 'Priya', 30, 5000, 9.0, 'FALSE');

INSERT INTO loans VALUES (201, 101, SYSDATE + 10); -- due in 10 days

INSERT INTO loans VALUES (202, 102, SYSDATE + 40); -- outside range

INSERT INTO loans VALUES (203, 103, SYSDATE + 5); -- due soon

INSERT INTO loans VALUES (204, 104, SYSDATE + 29); -- near limit

COMMIT;

BEGIN

-- Scenario 1: Apply 1% discount to loan interest rate for customers above 60

FOR rec1 IN (

SELECT customer\_id, age, loan\_interest\_rate

FROM customers

WHERE age > 60

) LOOP

UPDATE customers

SET loan\_interest\_rate = loan\_interest\_rate - 1

WHERE customer\_id = rec1.customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Discount applied: Customer ID ' || rec1.customer\_id ||

', Age: ' || rec1.age ||

', New Interest Rate: ' || (rec1.loan\_interest\_rate - 1));

END LOOP;

-- Scenario 2: Promote customers to VIP based on balance

FOR rec2 IN (

SELECT customer\_id, balance

FROM customers

WHERE balance > 10000

) LOOP

UPDATE customers

SET IsVIP = 'TRUE'

WHERE customer\_id = rec2.customer\_id;

DBMS\_OUTPUT.PUT\_LINE('VIP promoted: Customer ID ' || rec2.customer\_id ||

', Balance: ' || rec2.balance);

END LOOP;

-- Scenario 3: Print loan due reminders within next 30 days

FOR rec3 IN (

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

FROM customers c

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

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

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec3.loan\_id ||

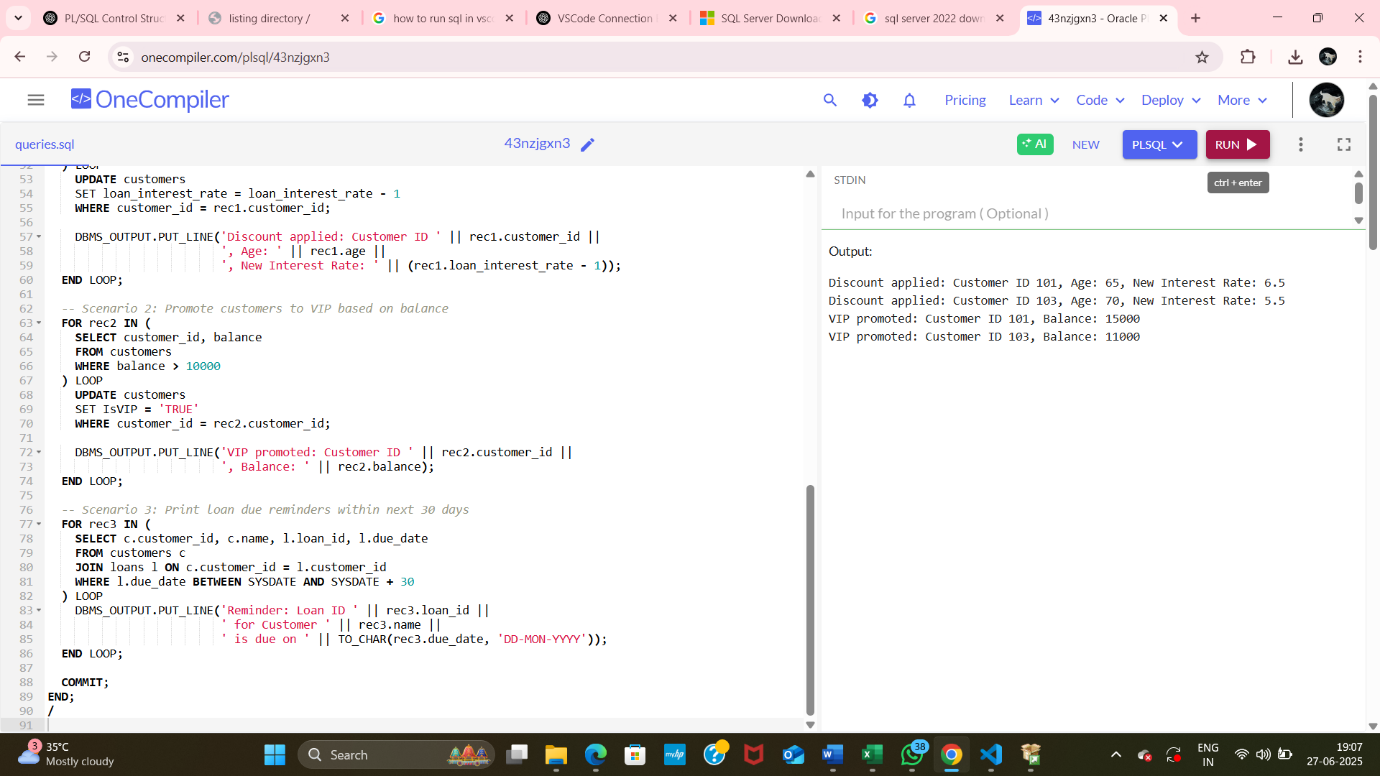
' for Customer ' || rec3.name ||

' is due on ' || TO\_CHAR(rec3.due\_date, 'DD-MON-YYYY'));

END LOOP;

COMMIT;

END;



***2)STORED PROCEDURES***

SET SERVEROUTPUT ON;

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE accounts';

EXECUTE IMMEDIATE 'DROP TABLE employees';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

CREATE TABLE accounts (

account\_id NUMBER PRIMARY KEY,

account\_type VARCHAR2(20),

balance NUMBER(10,2)

);

CREATE TABLE employees (

employee\_id NUMBER PRIMARY KEY,

department\_id NUMBER,

salary NUMBER(10,2)

);

INSERT INTO accounts VALUES (1001, 'SAVINGS', 10000);

INSERT INTO accounts VALUES (1002, 'SAVINGS', 15000);

INSERT INTO accounts VALUES (1003, 'CURRENT', 8000);

INSERT INTO accounts VALUES (1004, 'SAVINGS', 5000);

INSERT INTO employees VALUES (201, 101, 30000);

INSERT INTO employees VALUES (202, 101, 28000);

INSERT INTO employees VALUES (203, 102, 35000);

COMMIT;

-- Scenario 1: Process Monthly Interest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR acc IN (SELECT account\_id, balance FROM accounts WHERE account\_type = 'SAVINGS') LOOP

UPDATE accounts

SET balance = balance + (acc.balance \* 0.01)

WHERE account\_id = acc.account\_id;

DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID: ' || acc.account\_id);

END LOOP;

COMMIT;

END;

-- Scenario 2: Update Employee Bonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept\_id IN NUMBER,

p\_bonus\_percent IN NUMBER

) AS

BEGIN

FOR emp IN (SELECT employee\_id, salary FROM employees WHERE department\_id = p\_dept\_id) LOOP

UPDATE employees

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

WHERE employee\_id = emp.employee\_id;

DBMS\_OUTPUT.PUT\_LINE('Bonus applied to Employee ID: ' || emp.employee\_id);

END LOOP;

COMMIT;

END;

-- Scenario 3: Transfer Funds Between Accounts

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_acc IN NUMBER,

p\_to\_acc IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

-- Lock and check source balance

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = p\_from\_acc

FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in source account.');

END IF;

-- Deduct from source

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_acc;

-- Add to destination

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_acc;

DBMS\_OUTPUT.PUT\_LINE('Transferred ₹' || p\_amount ||

' from Account ' || p\_from\_acc ||

' to Account ' || p\_to\_acc);

COMMIT;

END;

-- Procedure Calls & Output

BEGIN

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

ProcessMonthlyInterest;

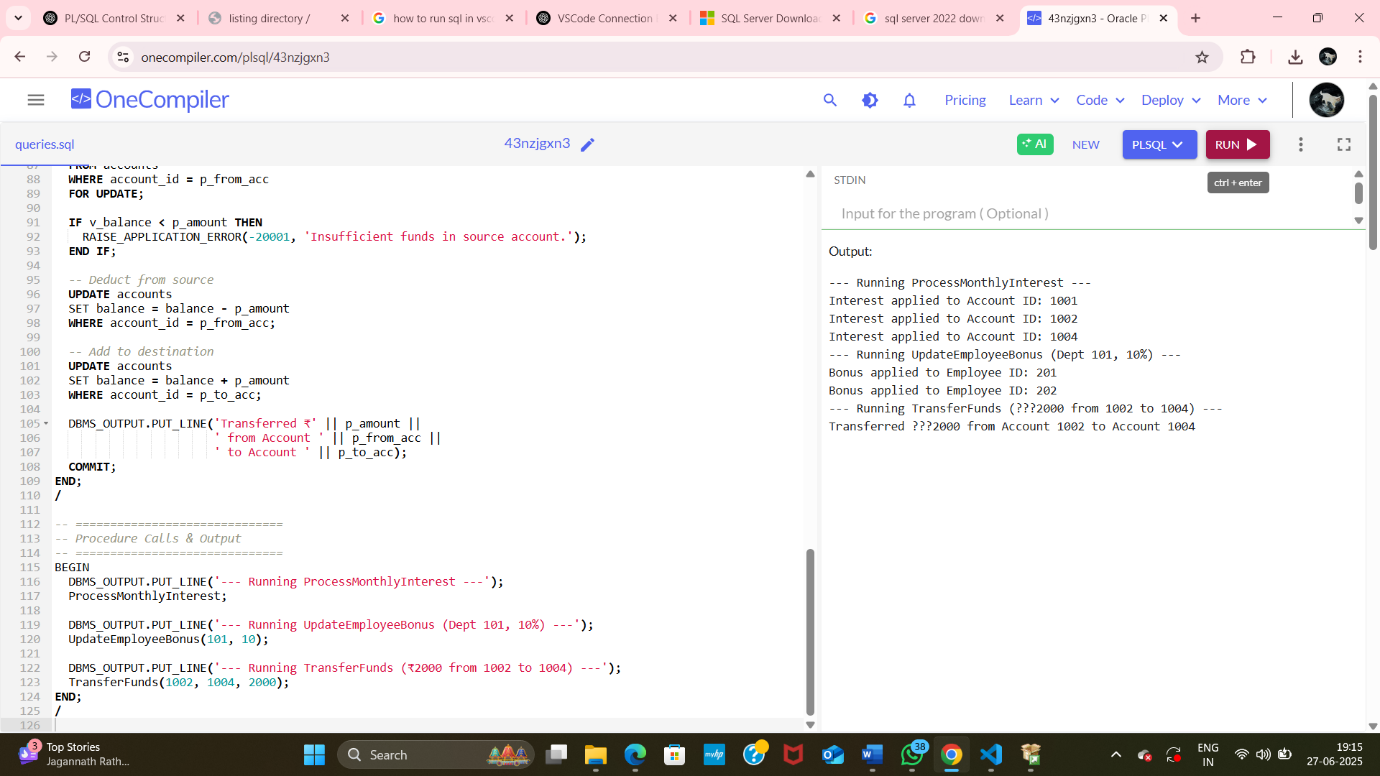
DBMS\_OUTPUT.PUT\_LINE('--- Running UpdateEmployeeBonus (Dept 101, 10%) ---');

UpdateEmployeeBonus(101, 10);

DBMS\_OUTPUT.PUT\_LINE('--- Running TransferFunds (₹2000 from 1002 to 1004) ---');

TransferFunds(1002, 1004, 2000);

END;



***3)SETTING UP JUNIT***

class Calculator {

public int add(int a, int b) {

return a + b;

}

}

public class Main {

public static void testAdd() {

Calculator calc = new Calculator();

if (calc.add(5, 3) != 8) {

System.out.println(" Test 1 Failed: 5 + 3 != 8");

} else {

System.out.println(" Test 1 Passed: 5 + 3 = 8");

}

if (calc.add(-2, 2) != 0) {

System.out.println(" Test 2 Failed: -2 + 2 != 0");

} else {

System.out.println(" Test 2 Passed: -2 + 2 = 0");

}

if (calc.add(0, 0) != 0) {

System.out.println(" Test 3 Failed: 0 + 0 != 0");

} else {

System.out.println(" Test 3 Passed: 0 + 0 = 0");

}

}

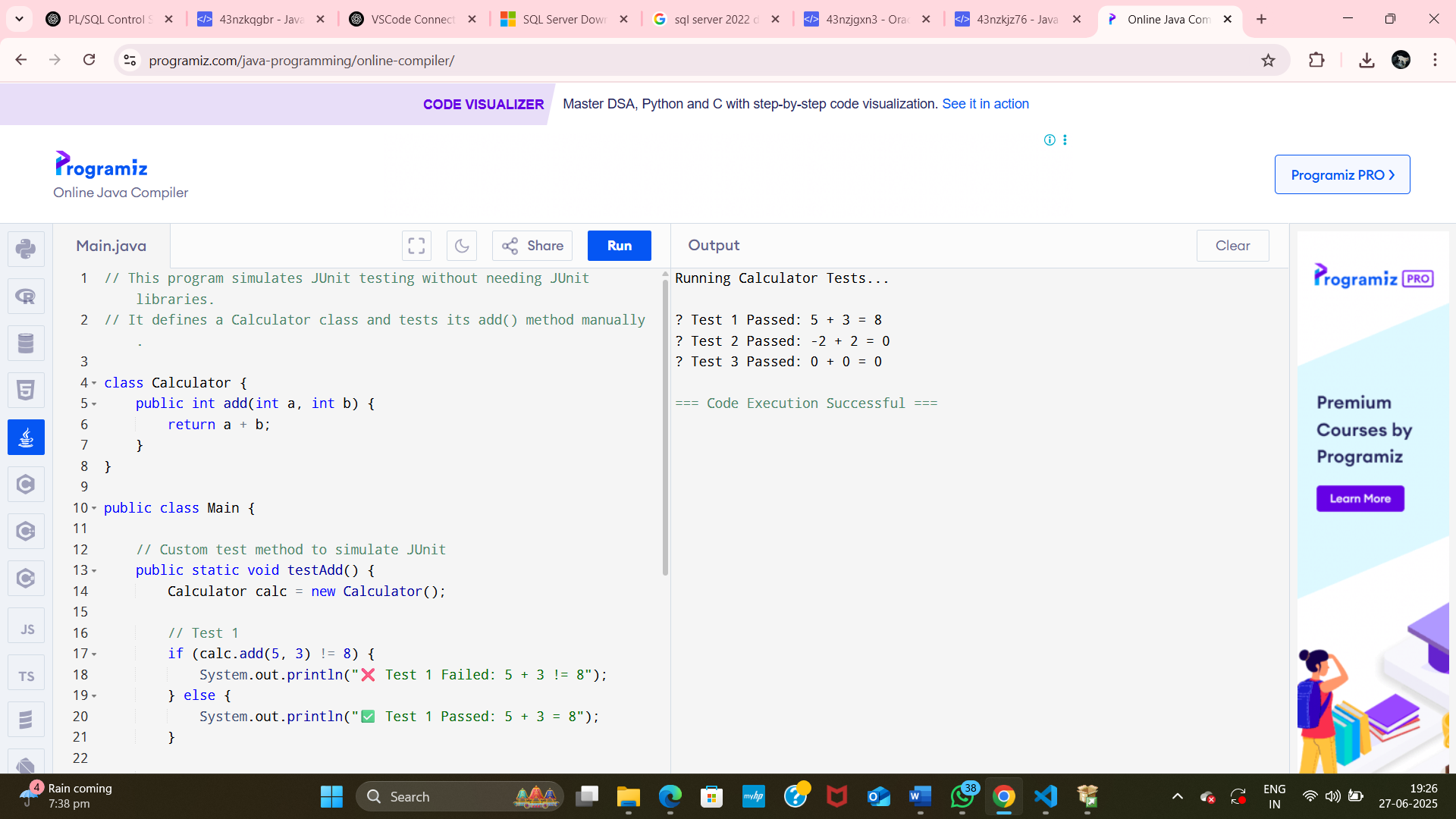
public static void main(String[] args) {

System.out.println("Running Calculator Tests...\n");

testAdd();

}

}



***4)ASSERTIONS IN JUNIT***

public class Main {

public static void main(String[] args) {

System.out.println("Running simulated JUnit assertions...\n");

if (2 + 3 == 5) {

System.out.println(" assertEquals passed: 2 + 3 == 5");

} else {

System.out.println(" assertEquals failed: 2 + 3 != 5");

}

if (5 > 3) {

System.out.println(" assertTrue passed: 5 > 3");

} else {

System.out.println(" assertTrue failed: 5 is not greater than 3");

}

if (!(5 < 3)) {

System.out.println(" assertFalse passed: 5 < 3 is false");

} else {

System.out.println(" assertFalse failed: 5 < 3 is true");

}

Object obj1 = null;

if (obj1 == null) {

System.out.println(" assertNull passed: object is null");

} else {

System.out.println(" assertNull failed: object is not null");

}

Object obj2 = new Object();

if (obj2 != null) {

System.out.println(" assertNotNull passed: object is not null");

} else {

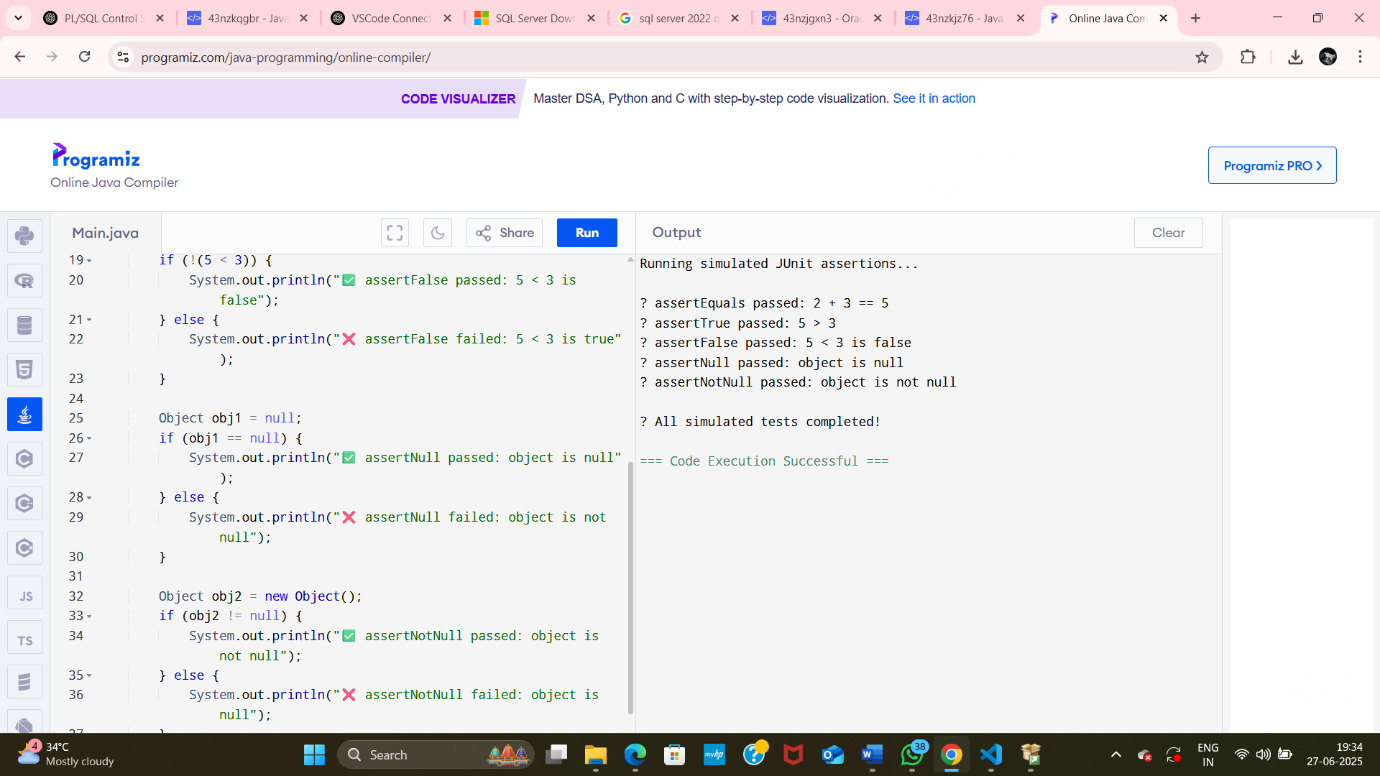
System.out.println(" assertNotNull failed: object is null");

}

System.out.println("\n All simulated tests completed!");

}

}



***5)AAA PATTERN***

public class Main {

static class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

private Calculator calculator;

public void setUp() {

System.out.println(" Setup: Creating Calculator instance");

calculator = new Calculator();

}

public void tearDown() {

System.out.println(" Teardown: Clearing Calculator instance\n");

calculator = null;

}

public void testAddition() {

setUp();

int a = 10;

int b = 5;

int result = calculator.add(a, b);

if (result == 15) {

System.out.println(" testAddition passed");

} else {

System.out.println(" testAddition failed: Expected 15, Got " + result);

}

tearDown(); }

public void testSubtraction() {

setUp();

int a = 20;

int b = 8;

int result = calculator.subtract(a, b);

if (result == 12) {

System.out.println(" testSubtraction passed");

} else {

System.out.println(" testSubtraction failed: Expected 12, Got " + result);

}

tearDown();

}

public static void main(String[] args) {

Main testRunner = new Main();

System.out.println(" Running tests with AAA pattern and setup/teardown:\n");

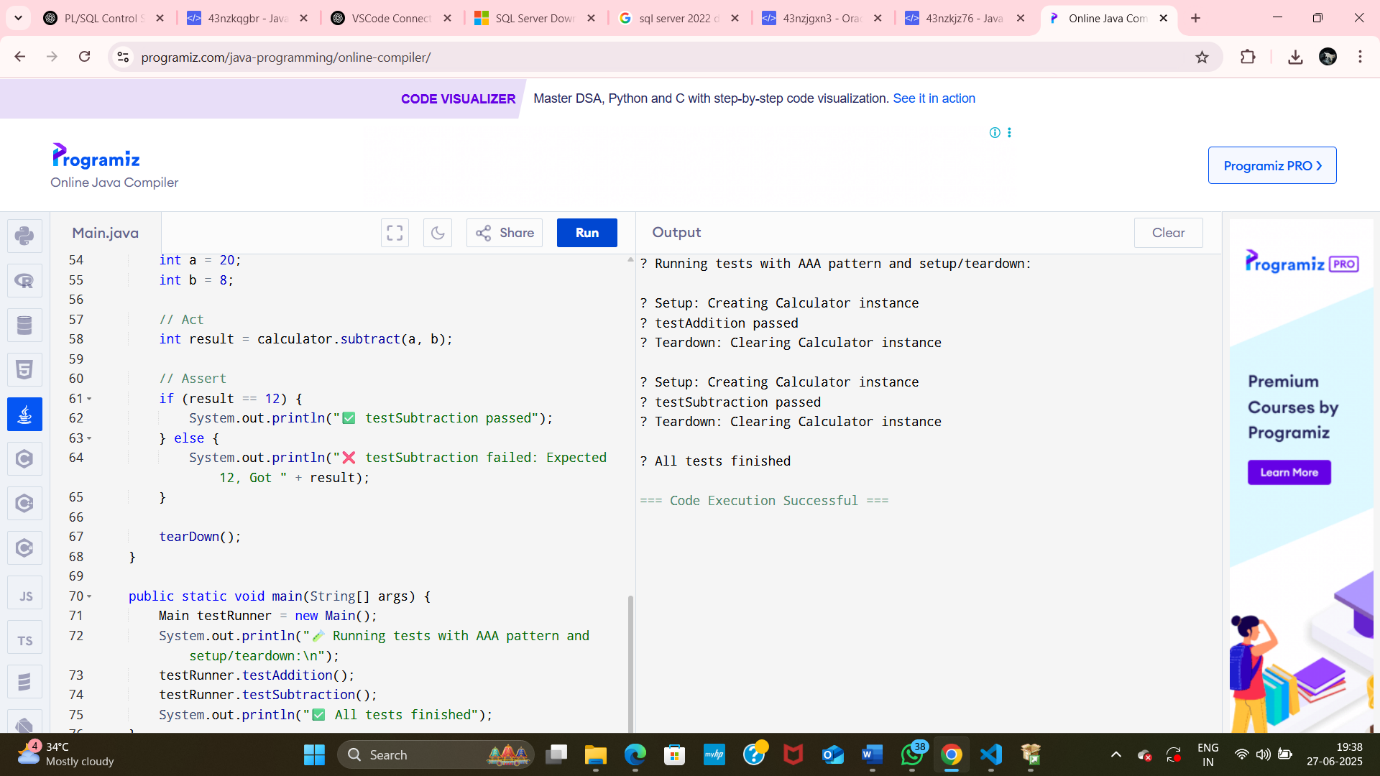
testRunner.testAddition();

testRunner.testSubtraction();

System.out.println(" All tests finished");

}

}



***6)MOCKING AND STUBBING***

public class Main {

interface ExternalApi {

String getData();

}

static class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

public static void main(String[] args) {

System.out.println(" Simulating mocking and stubbing...\n");

ExternalApi mockApi = new ExternalApi() {

@Override

public String getData() {

return "Mock Data"; }

};

MyService service = new MyService(mockApi);

String result = service.fetchData();

if ("Mock Data".equals(result)) {

System.out.println(" Test Passed: fetchData() returned: " + result);

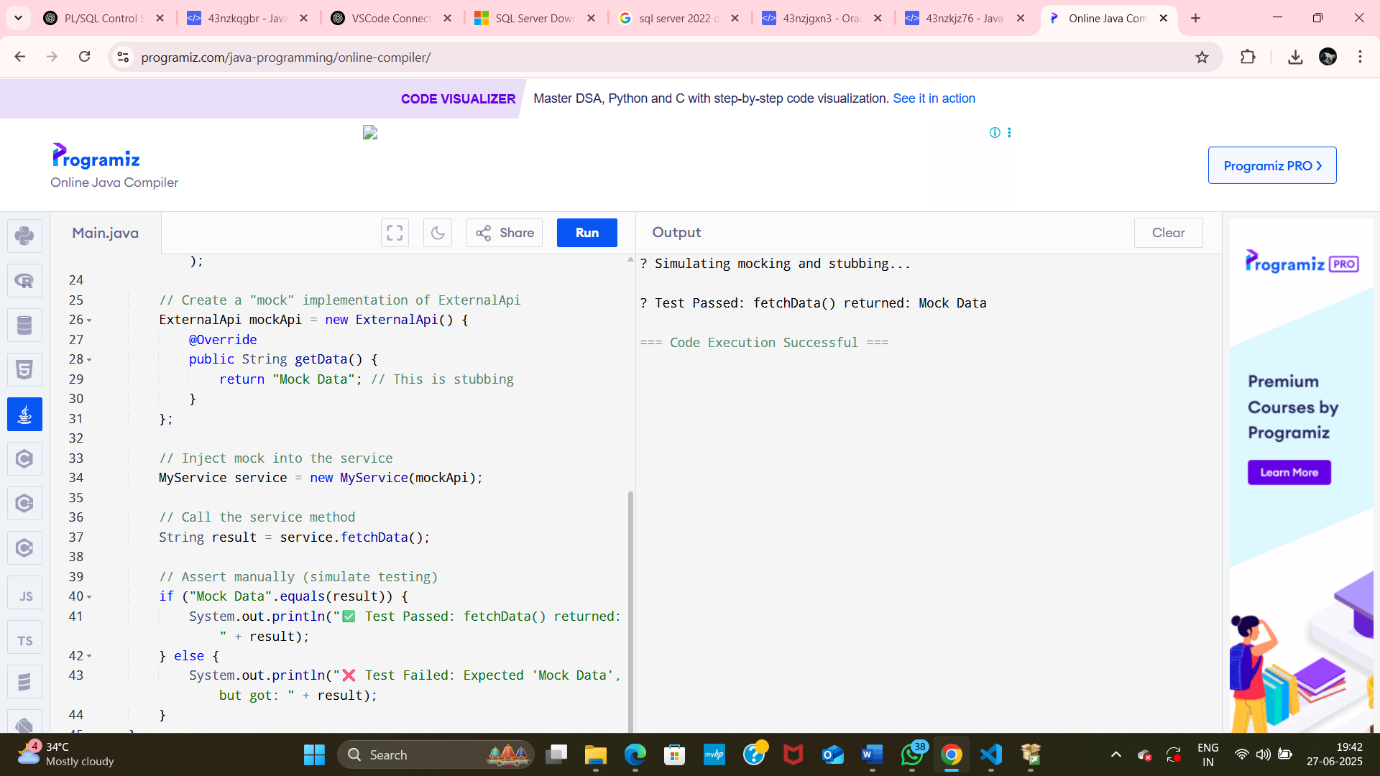
} else {

System.out.println(" Test Failed: Expected 'Mock Data', but got: " + result);

}

}

}



***7)VERIFYING INSTRUCTIONS***

public class Main {

interface ExternalApi {

String getData(String input);

}

static class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData("hello");

}

}

static class MockExternalApi implements ExternalApi {

private boolean wasCalled = false;

private String receivedArgument = null;

@Override

public String getData(String input) {

wasCalled = true;

receivedArgument = input;

return "Mocked Result";

}

public boolean wasGetDataCalled() {

return wasCalled;

}

public String getReceivedArgument() {

return receivedArgument;

}

}

public static void main(String[] args) {

System.out.println(" Verifying method interaction with specific argument...\n");

MockExternalApi mockApi = new MockExternalApi();

MyService service = new MyService(mockApi);

String result = service.fetchData();

if (mockApi.wasGetDataCalled()) {

System.out.println(" getData() was called on ExternalApi");

} else {

System.out.println(" getData() was NOT called");

}

if ("hello".equals(mockApi.getReceivedArgument())) {

System.out.println(" Correct argument passed: " + mockApi.getReceivedArgument());

} else {

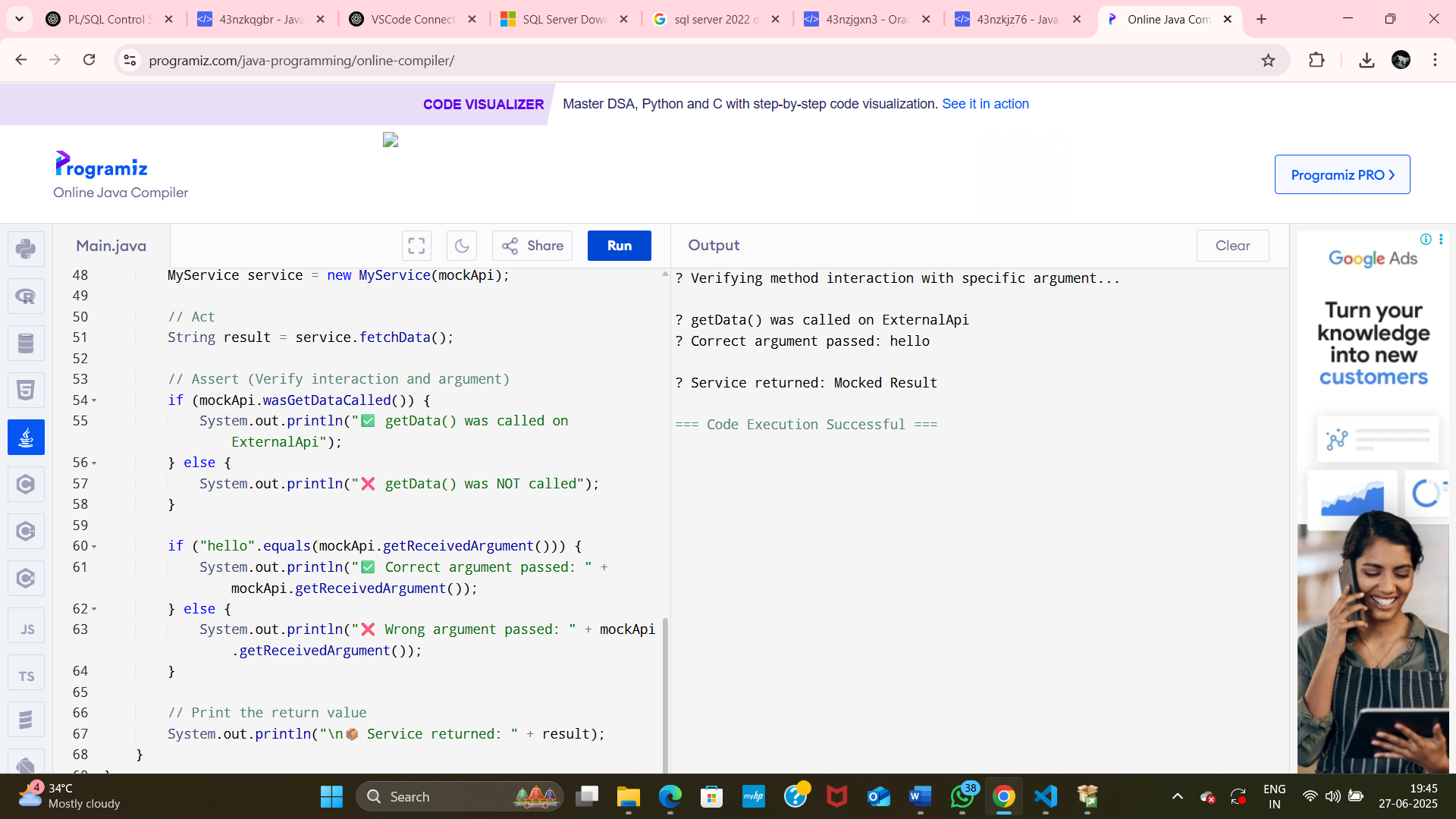
System.out.println(" Wrong argument passed: " + mockApi.getReceivedArgument());

}

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

}

}



***8)LOGGING ERROR MESSAGES AND WARNING LEVELS***

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

public class Main {

public static void log(String level, String message) {

String timestamp = LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));

System.out.println(timestamp + " [" + level + "] " + message);

}

public static void main(String[] args) {

System.out.println(" Simulated Logging Output:\n");

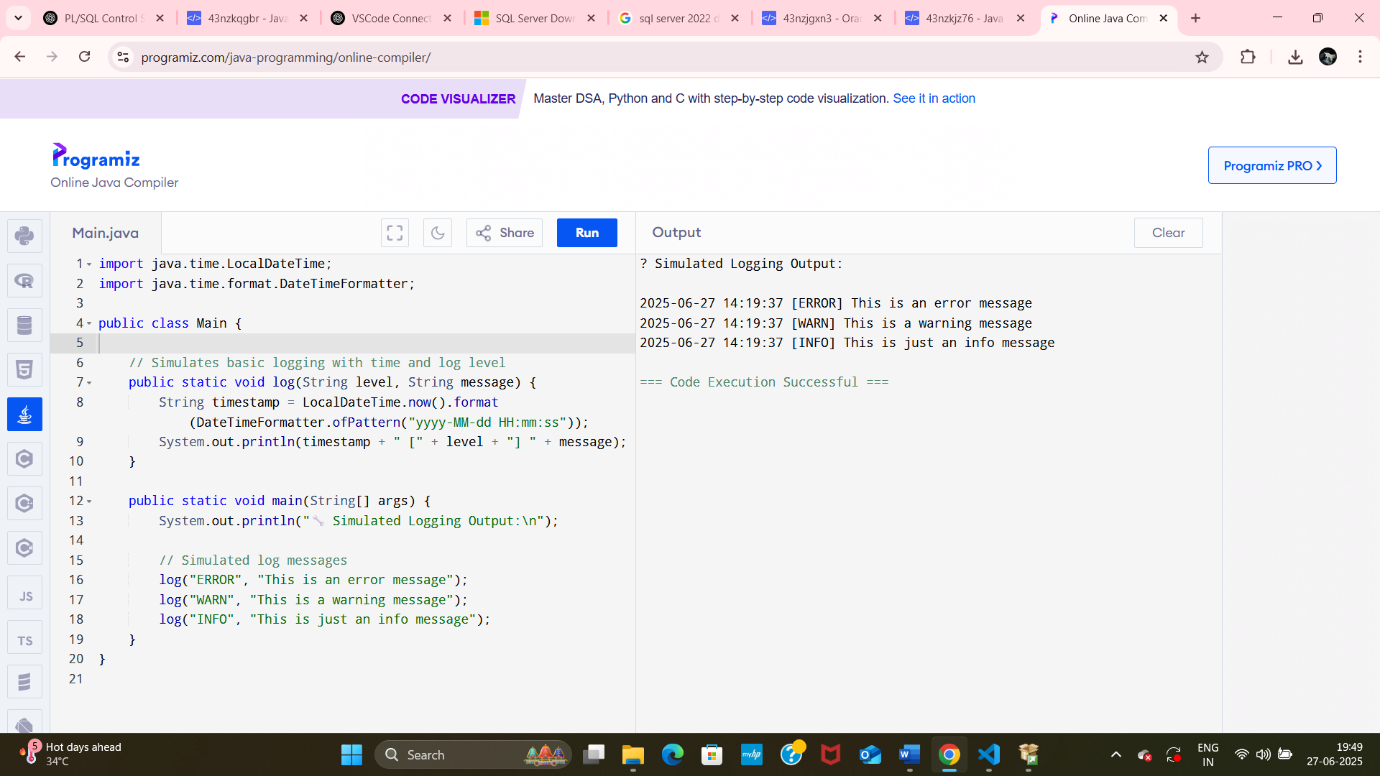
log("ERROR", "This is an error message");

log("WARN", "This is a warning message");

log("INFO", "This is just an info message");

}

}



**THANKYOU**