UNIVERSITY OF NIGERIA, NSUKKA

FACULTY OF PHYSICAL SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

NAME: OKOYE PROMISE OLUEBUBE

REG NO: 2023/257044

DEPARTMENT: COMPUTER SCIENCE

COS 261 ONLINE TEST

* BASICS & SYNTAX

**1. Java Program to Print "Hello, World!"**

*// This is a simple Java program to print "Hello, World!"*

public class HelloWorld {

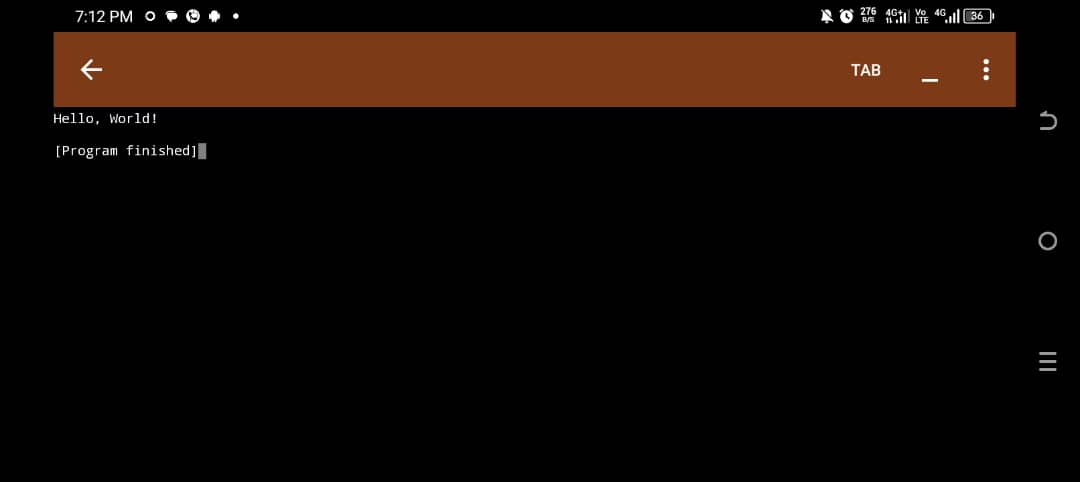
public static void main(String[] args) {

// Prints Hello, World! to the console

System.out.println("Hello, World!");

}

}



**2. Difference between == and .equals() in Java**

== checks reference equality (if both references point to the same object in memory).

.equals() checks value/content equality (if the values of objects are the same).

public class EqualityExample {

public static void main(String[] args) {

*// Using Strings for demonstration*

String a = new String("hello");

String b = new String("hello");

*// == checks if a and b reference the same object*

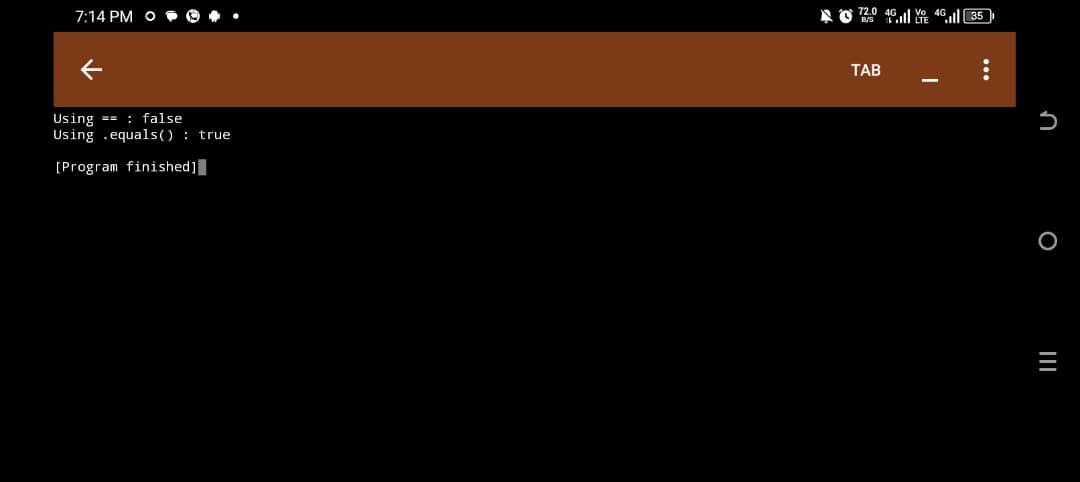
System.out.println("Using == : " + (a == b)); *// false*

*// .equals() checks if values of a and b are equal*

System.out.println("Using .equals() : " + a.equals(b)); *// true*

}

}



**3. Use of the main method in Java**

The main method is the entry point of any standalone Java application. It tells the JVM where to start executing the code.

public class MainExample {

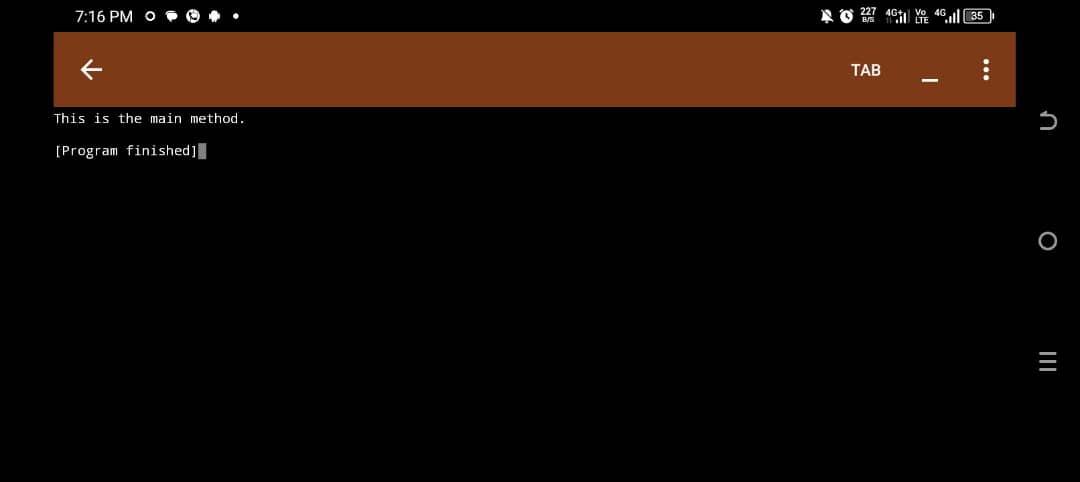
*// main method is where execution starts*

public static void main(String[] args) {

System.out.println("This is the main method.");

}

}



**4. Java Program to Add Two Numbers Entered by the User**

import java.util.Scanner;

public class AddNumbers {

public static void main(String[] args) {

*// Create Scanner object to take input from user*

Scanner scanner = new Scanner(System.in);

*// Prompt user to enter first number*

System.out.print("Enter first number: ");

int num1 = scanner.nextInt(); *// Read first number*

*// Prompt user to enter second number*

System.out.print("Enter second number: ");

int num2 = scanner.nextInt(); *// Read second number*

*// Calculate the sum*

int sum = num1 + num2;

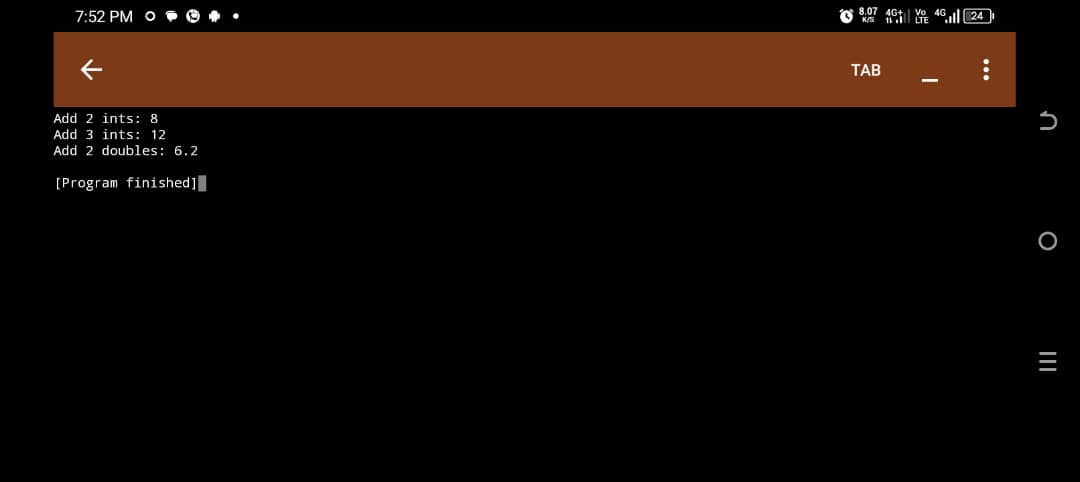
*// Display the result*

System.out.println("The sum is: " + sum);

scanner.close(); // Close the scanner

}

}



**5. Difference between int, Integer, and String in Java**

public class DataTypeExample {

public static void main(String[] args) {

*// int is a primitive data type*

int a = 5;

*// Integer is a wrapper class (used in collections or as an object)*

Integer b = 10;

*// String stores a sequence of characters*

String text = "Java is fun!";

*// Printing all values*

System.out.println("int: " + a);

System.out.println("Integer: " + b);

System.out.println("String: " + text);

}

}



* CONTROL STRUCTURES

**6. Program to Check if a Number is Even or Odd**

import java.util.Scanner;

public class EvenOdd {

public static void main(String[] args) {

*// Create scanner object to take input*

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a number: ");

int number = scanner.nextInt(); // Read number from user

*// Check if number is divisible by 2*

if (number % 2 == 0) {

System.out.println(number + " is Even.");

}

else {

System.out.println(number + " is Odd.");

}

scanner.close(); // Close the scanner

}

}



**7. Program to Find the Largest Among Three Numbers**

import java.util.Scanner;

public class LargestOfThree {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

*// Prompting the user to input three numbers*

System.out.print("Enter first number: ");

int a = scanner.nextInt();

System.out.print("Enter second number: ");

int b = scanner.nextInt();

System.out.print("Enter third number: ");

int c = scanner.nextInt();

*// Compare the three numbers using if-else*

if (a >= b && a >= c) {

System.out.println("The largest number is: " + a);

} else if (b >= a && b >= c) {

System.out.println("The largest number is: " + b);

} else {

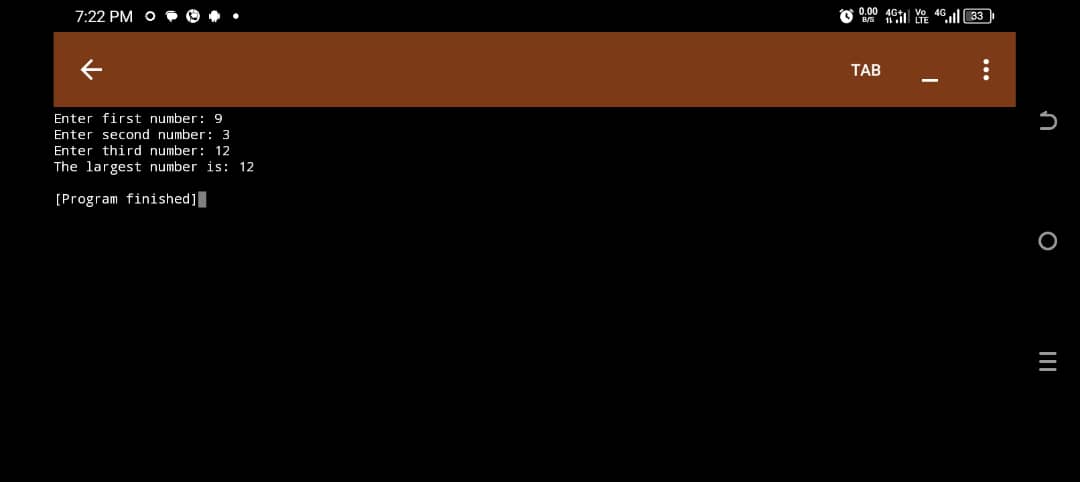
System.out.println("The largest number is: " + c);

}

scanner.close();

}

}



**8. Difference between while, for, and do-while Loops in Java**

**Examples:**

public class LoopExamples {

public static void main(String[] args) {

int i = 1;

*// while loop: checks condition first*

System.out.println("Using while loop:");

while (i <= 3) {

System.out.println("i = " + i);

i++;

}

*// for loop: initialization, condition, and update in one place*

System.out.println("\nUsing for loop:");

for (int j = 1; j <= 3; j++)

System.out.println("j = " + j);

}

*// do-while loop: runs at least once*

System.out.println("\nUsing do-while loop:");

int k = 1;

do {

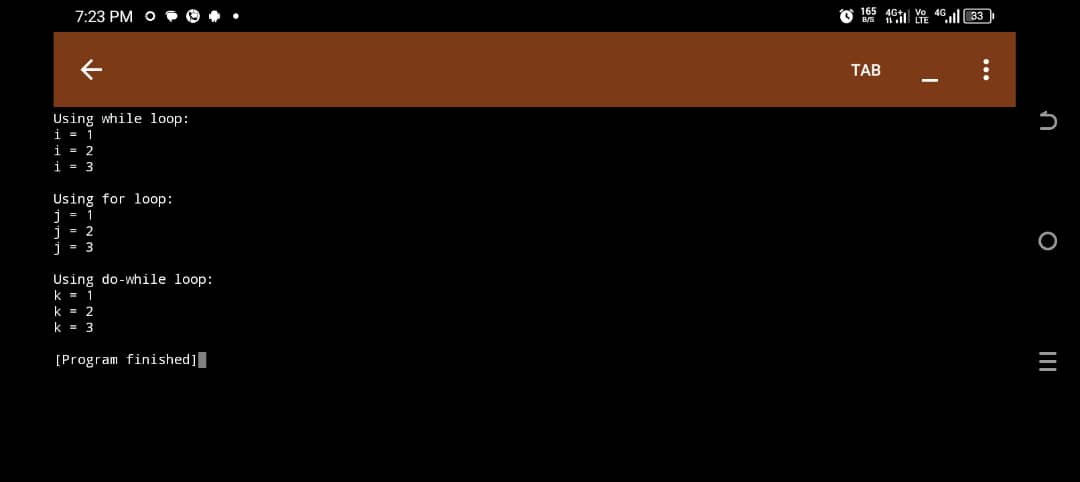
System.out.println("k = " + k);

k++;

} while (k <= 3);

}

}



**9. Java Program to Print the Multiplication Table of Any Number**

import java.util.Scanner;

public class MultiplicationTable {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

*// Ask user for the number*

System.out.print("Enter a number to print its multiplication table: ");

int number = scanner.nextInt();

*// Print table from 1 to 10*

System.out.println("Multiplication table of " + number + ":");

for (int i = 1; i <= 10; i++) {

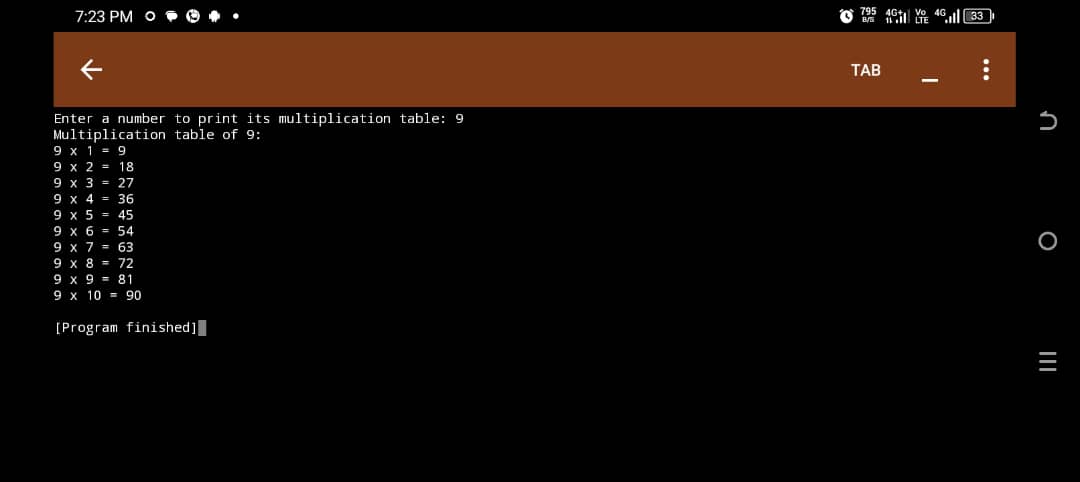
System.out.println(number + " x " + i + " = " + (number \* i));

}

scanner.close();

}

}



* INTERMEDIATE-LEVEL QUESTIONS

OOP CONCEPTS.

**10. Four Pillars of OOP in Java**

1. **Encapsulation**: Bundling data (variables) and methods into a single unit (class). Helps protect data from direct access.
2. **Abstraction**: Hiding complex implementation details and showing only the essential features.
3. **Inheritance**: Allows one class to inherit fields and methods from another.
4. **Polymorphism**: Ability of an object to take many forms (like method overloading/overriding).

**11. Student Class with name, matricNo, score**

*// Student class with encapsulated properties and a method to display student info*

public class Student {

*// Properties*

String name;

String matricNo;

int score;

*// Constructor to initialize student object*

public Student(String name, String matricNo, int score) {

this.name = name;

this.matricNo = matricNo;

this.score = score;

}

*// Method to display student info*

public void displayInfo() {

System.out.println("Name: " + name);

System.out.println("Matric No: " + matricNo);

System.out.println("Score: " + score);

}

*// Main method for testing*

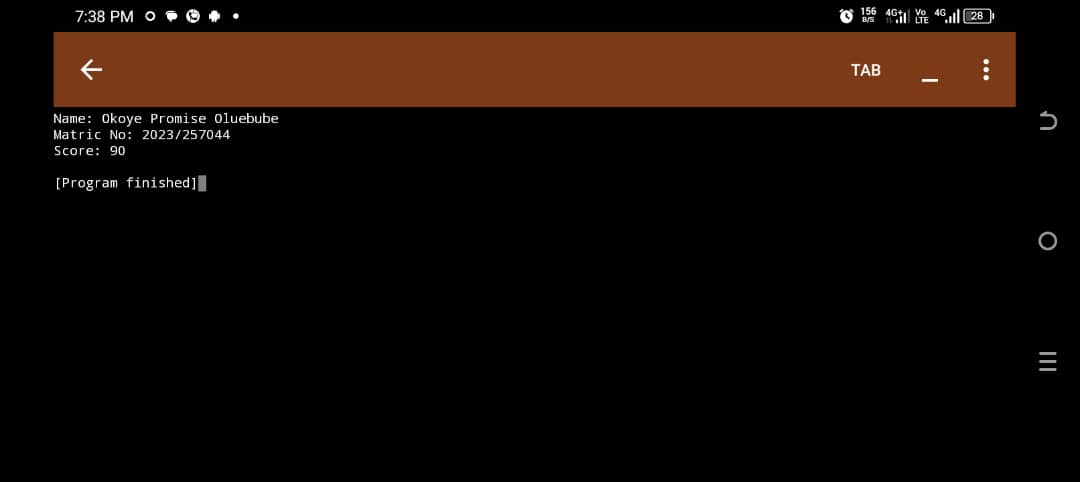
public static void main(String[] args) {

Student s1 = twohunderedlevel Student("Okoye Promise Oluebube", "2023/257044", 90);

s1.displayInfo();

}

}



**12. Method Overloading**

Method overloading is when two or more methods in the same class have the same name but different parameters (type or number).

public class Calculator {

*// Method to add two integers*

public int add(int a, int b) {

return a + b;

}

*// Overloaded method to add three integers*

public int add(int a, int b, int c) {

return a + b + c;

}

*// Overloaded method to add two doubles*

public double add(double a, double b) {

return a + b;

}

public static void main(String[] args) {

Calculator calc = new Calculator();

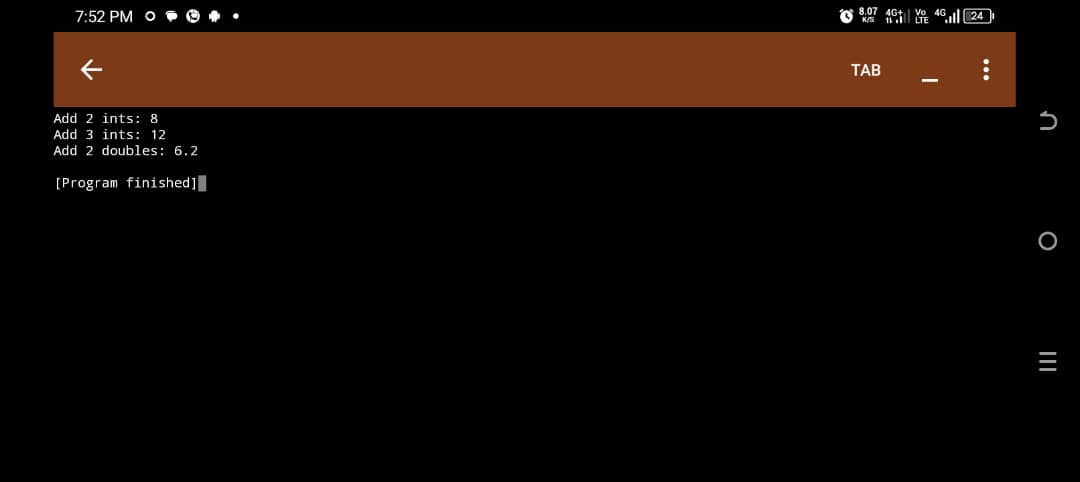
System.out.println("Add 2 ints: " + calc.add(5, 3));

System.out.println("Add 3 ints: " + calc.add(2, 4, 6));

System.out.println("Add 2 doubles: " + calc.add(2.5, 3.7));

}

}



**13. Inheritance: Person Base Class and Teacher Subclass**

*// Base class*

class Person {

String name;

public void speak() {

System.out.println("I am a person.");

}

}

*// Subclass*

class Teacher extends Person {

String subject;

public void teach() {

System.out.println(name + " teaches " + subject);

}

public static void main(String[] args) {

Teacher t1 = new Teacher();

t1.name = "Mrs. Aminat Atanda";

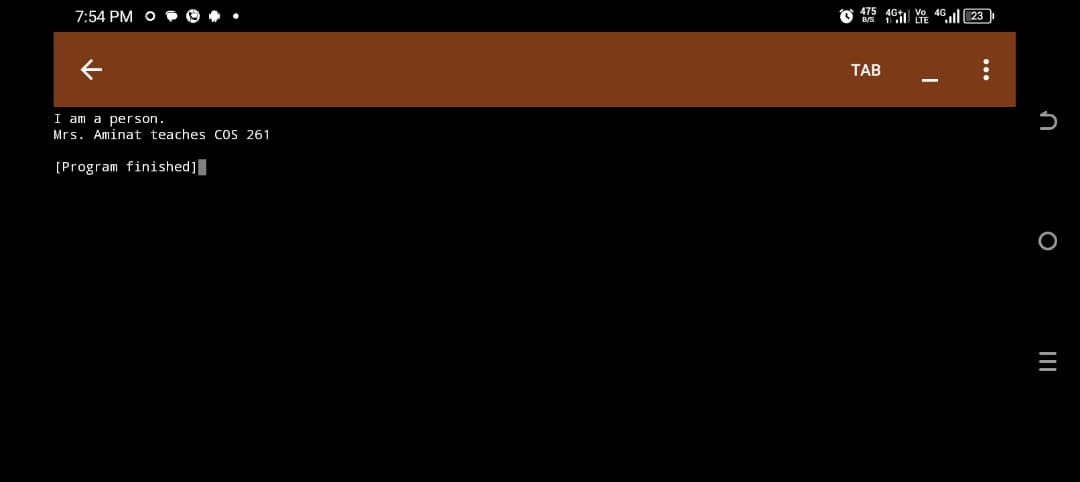
t1.subject = "Cos 261";

t1.speak(); *// Inherited method*

t1.teach(); *// Teacher's own method*

}

}



* GENERAL PRACTICES.

**14.** **What Does It Mean to Write “Clean Code”?**

Clean code is easy to read, understand, and maintain. It follows good practices and avoids clutter.

**3 Practices for Clean Code:**

1. Use meaningful variable and method names (e.g., calculateTotal() instead of ct()).

2. Keep methods short and focused – each method should do one thing only.

3. Use comments only when necessary, and write self-explanatory code that doesn't rely heavily on them.

**15. Why Should You Avoid Writing Very Long Methods in Java?**

Hard to read and understand – developers may struggle to follow the logic.

Difficult to debug and maintain – fixing bugs becomes harder in tangled logic.

Violates the Single Responsibility Principle – one method should do one job.

**Best Practice:** Break long methods into smaller, reusable ones with clear purposes.

**16. Java Naming Conventions**

public class StudentProfile { // Class name in PascalCase

int studentScore;

*// Variable name in camelCase*

*// Method name in camelCase*

public void calculateAverage() {

System.out.println("Calculating average...");

}

*// Entry point for the program*

public static void main(String[] args) {

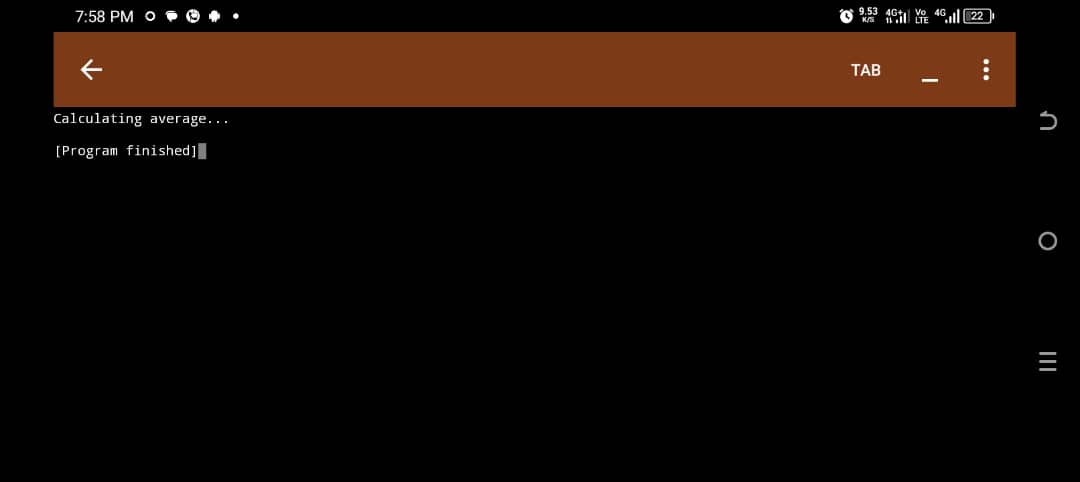
StudentProfile sp = new StudentProfile(); *// create an object*

sp.calculateAverage(); *// call method*

}

}

}



**17. Importance of Breaking Java Programs into Methods**

Improves code readability and reuse

Easier to test and debug

Follows Single Responsibility Principle

**18. DRY (Don't Repeat Yourself) Concept**

public class DRYExample {

*// DRY principle: reuse the method instead of repeating logic*

public static int square(int number) {

return number \* number;

}

public static void main(String[] args) {

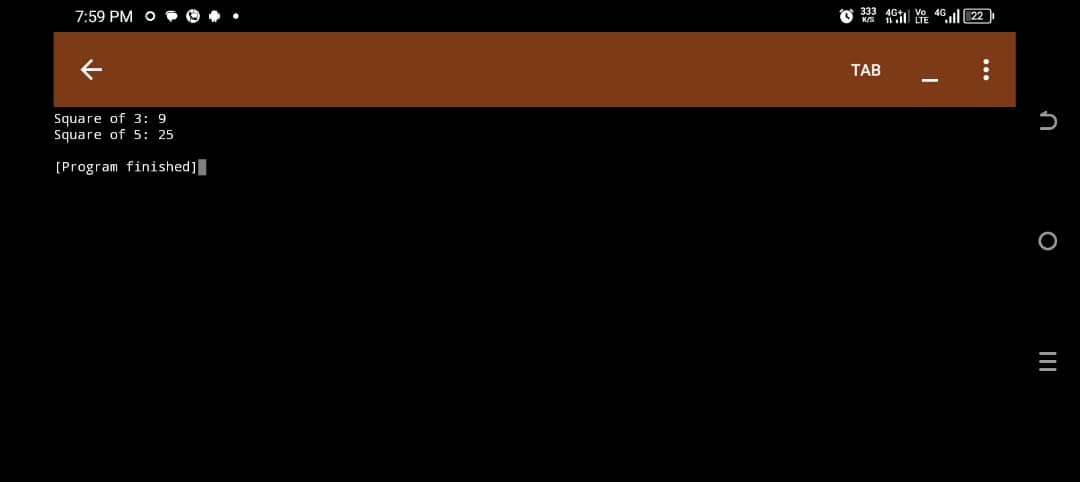
System.out.println("Square of 3: " + square(3));

System.out.println("Square of 5: " + square(5));

}

}

Instead of duplicating number \* number, we create a reusable method.



**19. Benefits of Using Classes and Objects**

Better organization of code

Code reuse and modularity

Easier testing and debugging

Supports real-world modeling via OOP

* TESTING & DEBUGGING

**20. Why Testing Is Important**

Catches errors early

Ensures correctness of logic

Saves time and cost in the long run

Boosts confidence in changes

**21. Errors in Java**

**1. Syntax Error**

What it is: A mistake in the rules of the language.

When it happens: During compilation.

Example: Missing semicolon, misspelling a keyword, or using the wrong brackets.

Java response: The program won’t compile.

int number = 5 *// Syntax error: missing semicolon*

System.out.println("Hello") *// Error here too*

**2. Runtime Error**

What it is: An error that occurs while the program is running.

When it happens: After the code compiles successfully but fails during execution.

Example: Dividing by zero, accessing an invalid array index.

Java response: The program crashes with an exception.

int result = 10 / 0; *// Runtime error: ArithmeticException*

**3. Logic Error**

**What it is:** The program runs without crashing, but the output is incorrect because the logic is wrong.

When it happens: While running.

**Example:** Using + instead of -, or putting the wrong condition in a loop.

Java response: No error message, but wrong result.

int a = 5;

int b = 3;

System.out.println("Sum is: " + (a - b)); *// Logic error: should be a + b*

**22. Testing a Method to Calculate Average**

public class Average {

public static double getAverage(int a, int b, int c, int d, int e) {

return (a + b + c + d + e) / 5.0;

}

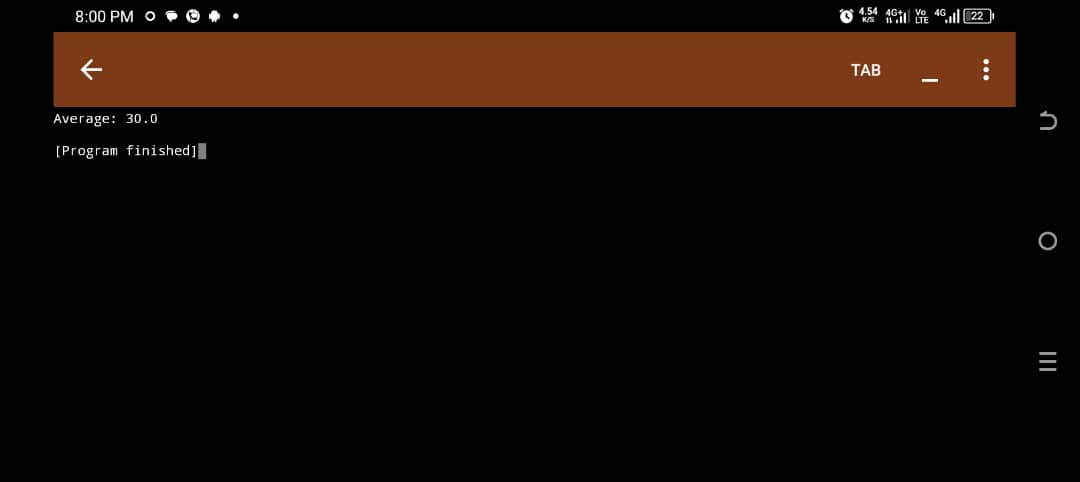
public static void main(String[] args) {

System.out.println("Average: " + getAverage(10, 20, 30, 40, 50)); *// Should print 30.0*

}

}

Test case: Input: 10, 20, 30, 40, 50 → Expected Output: 30.0



* DOCUMENTATION & COMMENTS

**23. Why Write Comments in Code**

Makes code understandable to others

Aids future maintenance

Helps explain complex logic

**24. JavaDoc vs Regular Comments**

JavaDoc comments are special comments in Java used to generate documentation automatically using the javadoc tool. They begin with /\*\* and end with \*/.

These comments are written above classes, methods, or fields and include tags like:

@param – describes a method parameter

@return – describes the return value

@author, @version, @throws, etc.

Regular comments (// and /\* ... \*/) are only for the developer to read inside the code. They are ignored by documentation tools.

JavaDoc comments (/\*\* ... \*/) are meant for generating official documentation, and can include tags like @param, @return, and @author.

**25. Sample Java Method with JavaDoc**

*/\*\**

*\* Calculates the sum of two integers.*

*\* @param a First number*

*\* @param b Second number*

*\* @return The sum of a and b*

*\*/*

public int add(int a, int b) {

return a + b;

}

* VERSIONING & COLLABORATION

**26. What Is Version Control**

Version control systems (like Git) help developers:

Track changes to code

Collaborate safely

Revert to previous versions

**27. Explaining Code Refactoring**

"Refactoring means improving the structure or readability of code without changing its functionality. For example, breaking a long method into smaller helper methods."

**28. Collaboration Tools for Java Projects**

1. GitHub – For version control and collaboration.

2. GitLab – Like GitHub, with built-in CI/CD.

3. Bitbucket – Another Git platform, integrates with Jira.

* GOOD PRACTICES SUMMARY

**29. 5 Java Best Practices**

1. Use meaningful variable/method names.

2. Keep methods short and single-purpose.

3. Write unit tests for important methods.

4. Follow Java naming conventions.

5. Avoid magic numbers; use constants.

**30. What Is Code Readability and Why It’s Important**

**Readable code is:**

Easy to understand

Easier to maintain and debug

Better for teamwork

Readable code > Smart (but complex) code because code is read more often than it’s written.

* ADVANCED-LEVEL QUESTIONS

MINI PROJECTS/LOGIC BUILDING

**31. Student Grade Tracker**

import java.util.HashMap;

import java.util.Scanner;

public class StudentGrades {

*// HashMap to store student names and their grades*

static HashMap<String, Integer> studentRecords = new HashMap<>();

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int choice;

*// Menu loop*

do {

System.out.println("\n--- Student Grade Tracker ---");

System.out.println("1. Add Student");

System.out.println("2. Update Grade");

System.out.println("3. View All Records");

System.out.println("4. Exit");

System.out.print("Enter your choice: ");

choice = scanner.nextInt();

scanner.nextLine(); *// consume newline*

switch (choice) {

case 1 -> {

System.out.print("Enter student name: ");

String name = scanner.nextLine();

System.out.print("Enter grade: ");

int grade = scanner.nextInt();

studentRecords.put(name, grade);

System.out.println("Student added successfully.");

}

case 2 -> {

System.out.print("Enter student name to update: ");

String name = scanner.nextLine();

if (studentRecords.containsKey(name)) {

System.out.print("Enter new grade: ");

int newGrade = scanner.nextInt();

studentRecords.put(name, newGrade);

System.out.println("Grade updated.");

} else {

System.out.println("Student not found.");

}

}

case 3 -> {

System.out.println("\n--- Student Records ---");

for (String name : studentRecords.keySet()) {

System.out.println(name + " : " + studentRecords.get(name));

}

}

case 4 -> System.out.println("Exiting...");

default -> System.out.println("Invalid choice.");

}

} while (choice != 4);

scanner.close();

}

}



**32. Basic ATM Simulator**

import java.util.Scanner;

public class ATM {

*// Sample balance*

static double balance = 1000.00;

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int choice;

*// Menu loop*

do {

System.out.println("\n--- ATM Menu ---");

System.out.println("1. Check Balance");

System.out.println("2. Deposit Money");

System.out.println("3. Withdraw Money");

System.out.println("4. Exit");

System.out.print("Choose an option: ");

choice = scanner.nextInt();

switch (choice) {

case 1 -> System.out.println("Your current balance is: ₦" + balance);

case 2 -> {

System.out.print("Enter amount to deposit: ");

double deposit = scanner.nextDouble();

if (deposit > 0) {

balance += deposit;

System.out.println("Deposit successful.");

} else {

System.out.println("Invalid deposit amount.");

}

}

case 3 -> {

System.out.print("Enter amount to withdraw: ");

double withdraw = scanner.nextDouble();

if (withdraw > 0 && withdraw <= balance) {

balance -= withdraw;

System.out.println("Withdrawal successful.");

} else {

System.out.println("Insufficient funds or invalid amount.");

}

}

case 4 -> System.out.println("Thank you for using the ATM.");

default -> System.out.println("Invalid option.");

}

} while (choice != 4);

scanner.close();

}

}



This is the link to my git hub repository https://github.com/Promiseokoye1/COS-261-Test..git