## **NAME: ONYEKAWA ONYINYECHI JOY**

## **REG NO: 2024/278364**

## **COURSE: 201-261**

## **GITHUB: https://github.com/bellia123/COS-261-201-TEST.git**

## **DATE: 9/05/2025**

## **Online Java Compiler: Programiz**

## **NOTE:** Comments are lines in codes that are ignored by the Java compiler. They are meant for us humans — to explain what the code does, why certain decisions were made, or to make it easier for developers and anyone else to understand later. All codes have comments added to them for better clarity.

## **Basics & Syntax**

### **1. Write a Java program to print "Hello, World!".**

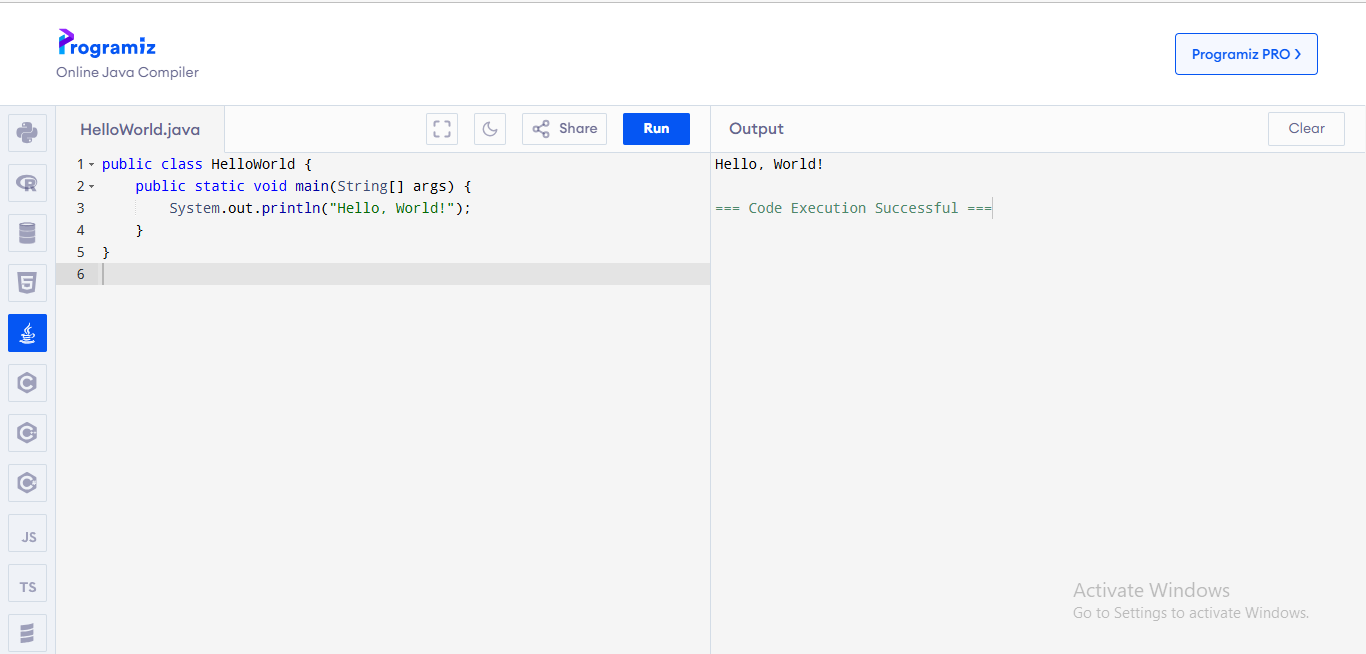
public class HelloWorld {

public static void main(String[] args) {

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

}

}



### **2. Explain the difference between == and .equals() in Java.**

In Java:

* == compares if two references point to the same object in memory
* .equals() compares the content or values of objects

Example:

public class EqualsVsEqual {

public static void main(String[] args) {

String s1 = new String("UNN");

String s2 = new String("UNN");

*// Using == (compares references)*

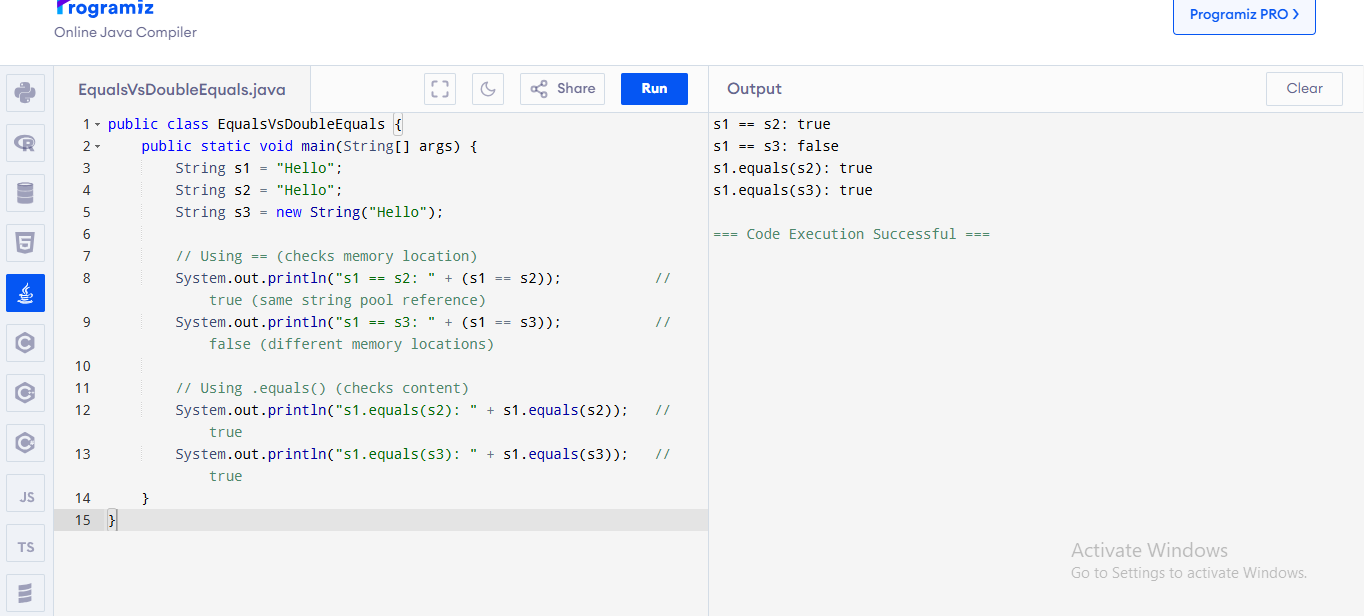
System.out.println("s1 == s2: " + (s1 == s2)); *// false*

*// Using equals() (compares values)*

System.out.println("s1.equals(s2): " + s1.equals(s2)); *// true*

}

}



### **3. What is the use of the main method in Java?**

The main method serves as the entry point for any Java application. When you run a Java program, the JVM (Java Virtual Machine) looks for the main method to start execution. It must have the exact signature:

public static void main(String[] args)

### **4. Write a Java program to add two numbers entered by the user.**

import java.util.Scanner;

public class AddNumbers {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

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

double num1 = input.nextDouble();

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

double num2 = input.nextDouble();

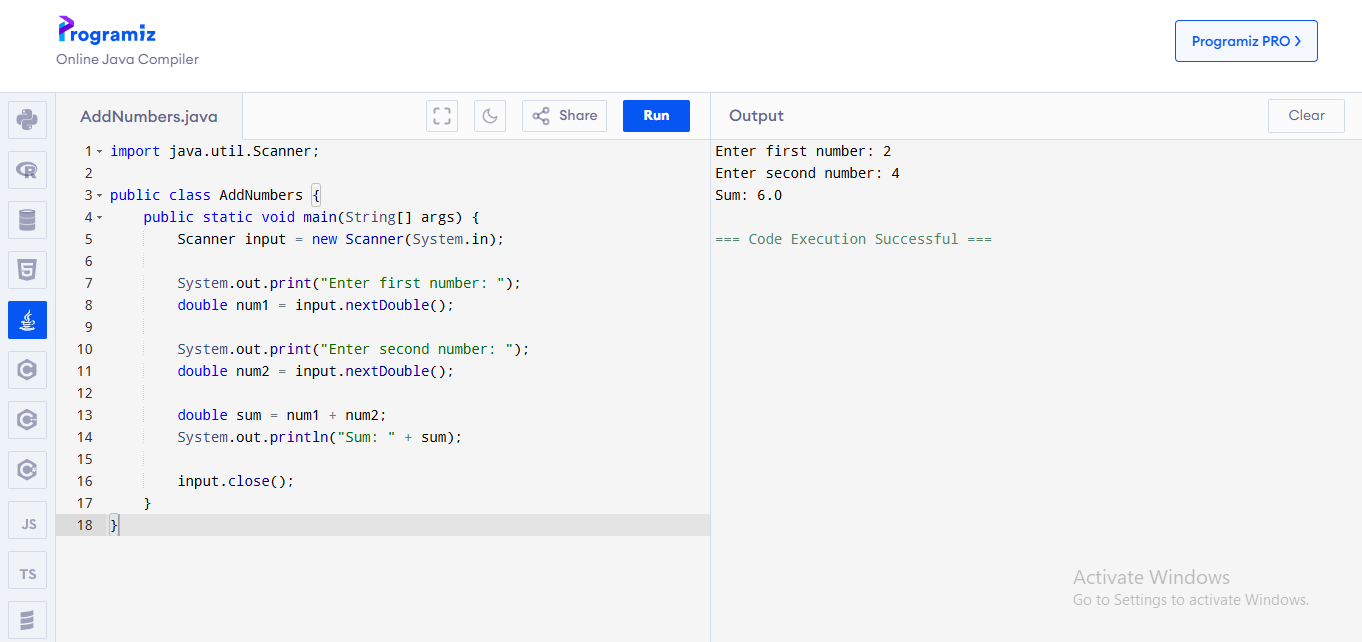
double sum = num1 + num2;

System.out.println("Sum: " + sum);

input.close();

}

}



### **5. What is the difference between int, Integer, and String?**

* int: A primitive data type that stores whole numbers (no decimals)
* Integer: A wrapper class for the primitive type int that provides methods to perform operations
* String: A class that represents a sequence of characters

## **Control Structures**

### **6. Write a program to check if a number is even or odd.**

import java.util.Scanner;

public class EvenOdd {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

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

int num = input.nextInt();

if (num % 2 == 0) {

System.out.println(num + " is even");

} else {

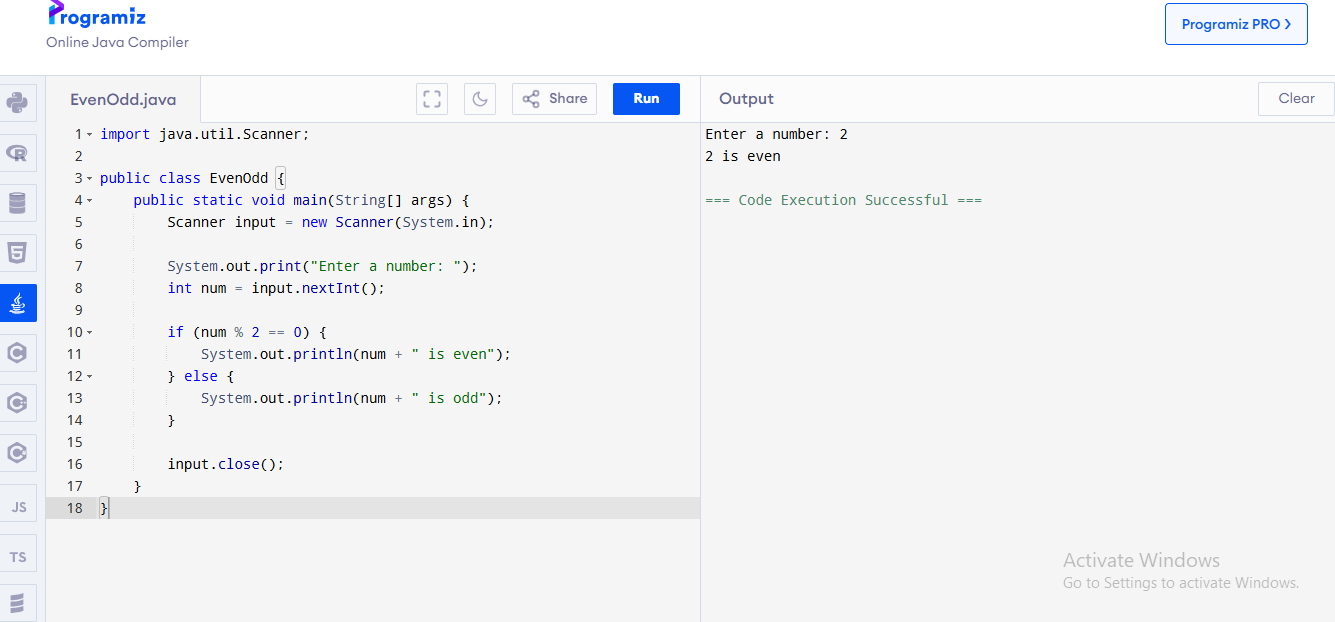
System.out.println(num + " is odd");

}

input.close();

}

}



### **7. Write a program to find the largest among three numbers.**

import java.util.Scanner;

public class LargestNumber {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

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

double num1 = input.nextDouble();

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

double num2 = input.nextDouble();

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

double num3 = input.nextDouble();

double largest = num1;

if (num2 > largest) {

largest = num2;

}

if (num3 > largest) {

largest = num3;

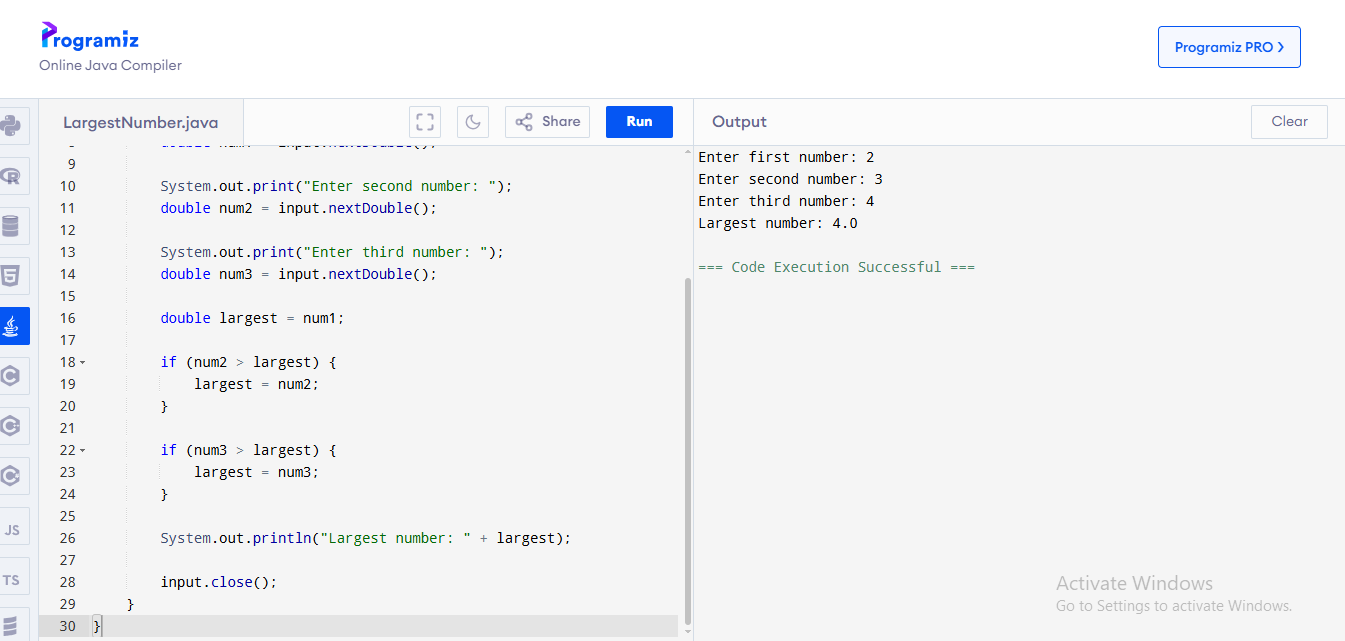
}

System.out.println("Largest number: " + largest);

input.close();

}

}



### **8. Explain the difference between while, for, and do-while loops in Java.**

* while loop: Executes a block of code as long as a condition is true. The condition is checked before executing the block.
* for loop: Used when you know how many times you want to execute a block of code.
* do-while loop: Similar to while loop, but the condition is checked after executing the block once (guarantees at least one execution).

### **9. Write a Java program to print the multiplication table of any number.**

import java.util.Scanner;

public class MultiplicationTable {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

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

int num = input.nextInt();

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

for (int i = 1; i <= 12; i++) {

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

}

input.close();

}

}



## **OOP Concepts**

### **10. Explain the four pillars of OOP in Java.**

1. **Encapsulation**: Bundling data and methods that operate on the data within a single unit (class) and restricting direct access to some components.
2. **Inheritance**: The ability of a class to derive properties and characteristics from another class.
3. **Polymorphism**: The ability to present the same interface for different underlying forms or data types (method overloading and overriding).
4. **Abstraction**: Hiding complex implementation details and showing only the necessary features of an object.

### **11. Create a class Student with properties name, matricNo, and score, and add methods to display the student's info.**

public class Student {

private String name;

private String matricNo;

private double score;

*// Constructor*

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

this.name = name;

this.matricNo = matricNo;

this.score = score;

}

*// Method to display student information*

public void displayInfo() {

System.out.println("Student Information:");

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

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

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

}

*// Main method for testing*

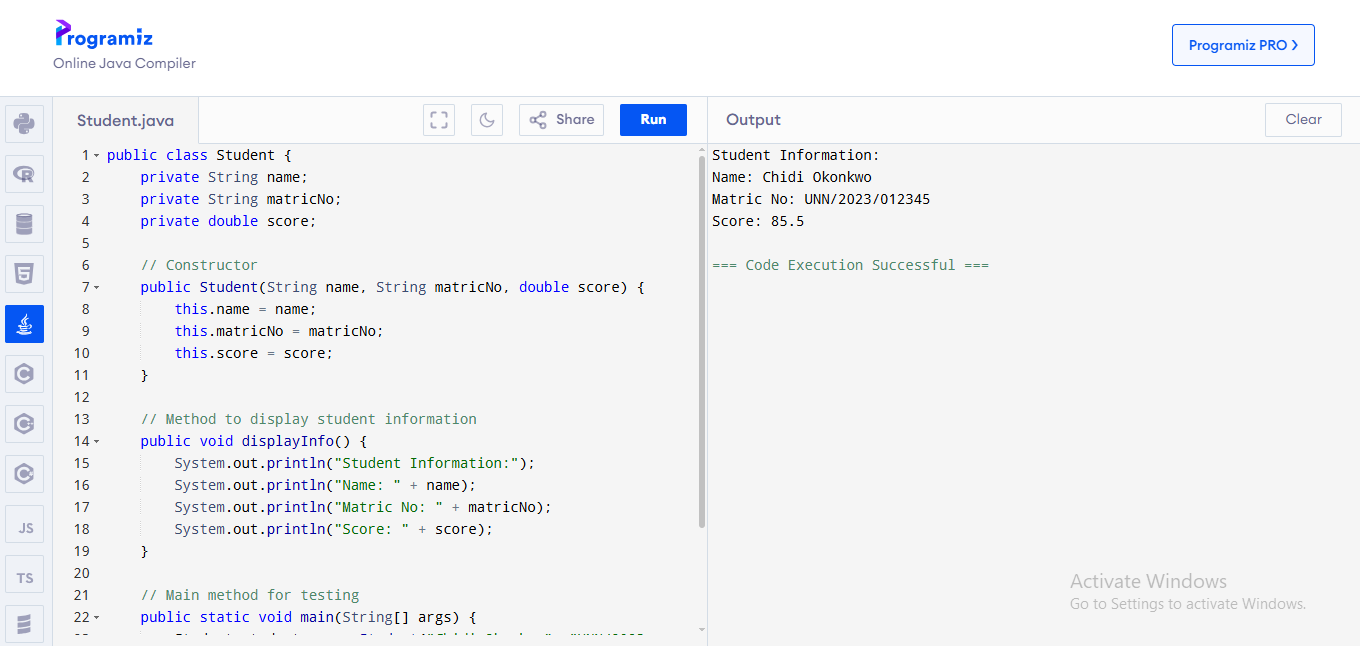
public static void main(String[] args) {

Student student = new Student("Chidi Okonkwo", "UNN/2023/012345", 85.5);

student.displayInfo();

}

}



### **12. What is method overloading? Give a code example.**

Method overloading is when multiple methods in the same class have the same name but different parameters (different number or types of parameters).

public class MethodOverloading {

*// Method with two parameters*

public int add(int a, int b) {

return a + b;

}

*// Method with three parameters*

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

return a + b + c;

}

*// Method with different parameter types*

public double add(double a, double b) {

return a + b;

}

public static void main(String[] args) {

MethodOverloading calc = new MethodOverloading();

System.out.println("Sum of 5 and 10: " + calc.add(5, 10));

System.out.println("Sum of 5, 10, and 15: " + calc.add(5, 10, 15));

System.out.println("Sum of 5.5 and 10.5: " + calc.add(5.5, 10.5));

}

}



### **13. What is inheritance? Create a base class Person and a subclass Teacher.**

Inheritance is an OOP concept where a class (subclass) can inherit properties and methods from another class (superclass).

*// Base class*

public class Person {

protected String name;

protected int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public void displayInfo() {

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

System.out.println("Age: " + age);

}

}

*// Subclass*

public class Teacher extends Person {

private String subject;

public Teacher(String name, int age, String subject) {

super(name, age); *// Call the constructor of the superclass*

this.subject = subject;

}

*// Override the displayInfo method*

@Override

public void displayInfo() {

super.displayInfo(); *// Call the displayInfo method of the superclass*

System.out.println("Subject: " + subject);

}

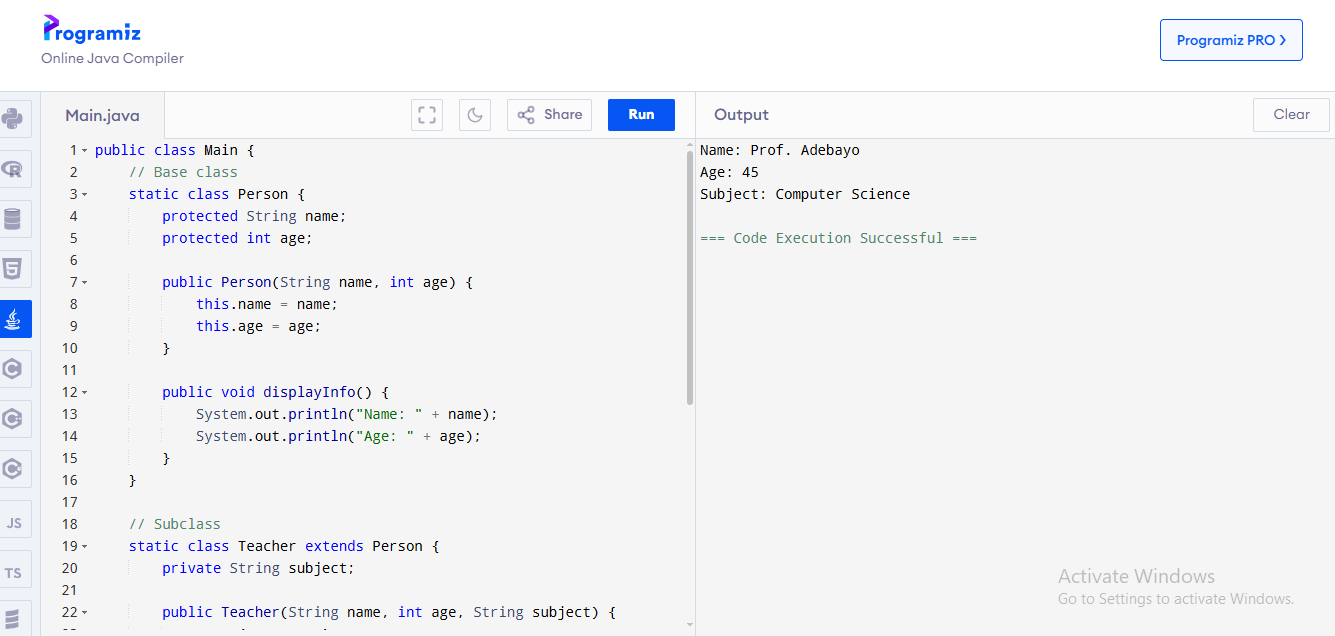
public static void main(String[] args) {

Teacher teacher = new Teacher("Prof. Adebayo", 45, "Computer Science");

teacher.displayInfo();

}

}



## **General Practices**

### **14. What does it mean to write "clean code"? Give 3 practices that make code clean and maintainable.**

Clean code is code that is easy to read, understand, and maintain. Three practices for clean code:

1. **Meaningful names**: Use descriptive names for variables, methods, and classes that clearly indicate their purpose.
2. **Small functions/methods**: Keep methods short and focused on a single task to make them easier to understand and test.
3. **Consistent formatting**: Follow consistent indentation, spacing, and naming conventions throughout the codebase.

### **15. Why should you avoid writing very long methods in Java programs?**

Long methods should be avoided because:

* They are hard to read and understand
* They often perform multiple tasks, violating the Single Responsibility Principle
* They are difficult to test, debug, and maintain
* They often contain duplicate code and high complexity

### **16. What naming conventions should be followed in Java for: Classes, Variables, Methods.**

In Java:

* **Classes**: Start with uppercase letter, use CamelCase (e.g., StudentRecord, BankAccount)
* **Variables**: Start with lowercase letter, use camelCase (e.g., firstName, totalAmount)
* **Methods**: Start with lowercase letter, use camelCase (e.g., calculateTotal(), getName())

java code

public class BankAccount { *// Class name starts with uppercase*

private double accountBalance; *// Variable starts with lowercase*

private String accountHolder;

public double getBalance() { *// Method starts with lowercase*

return accountBalance;

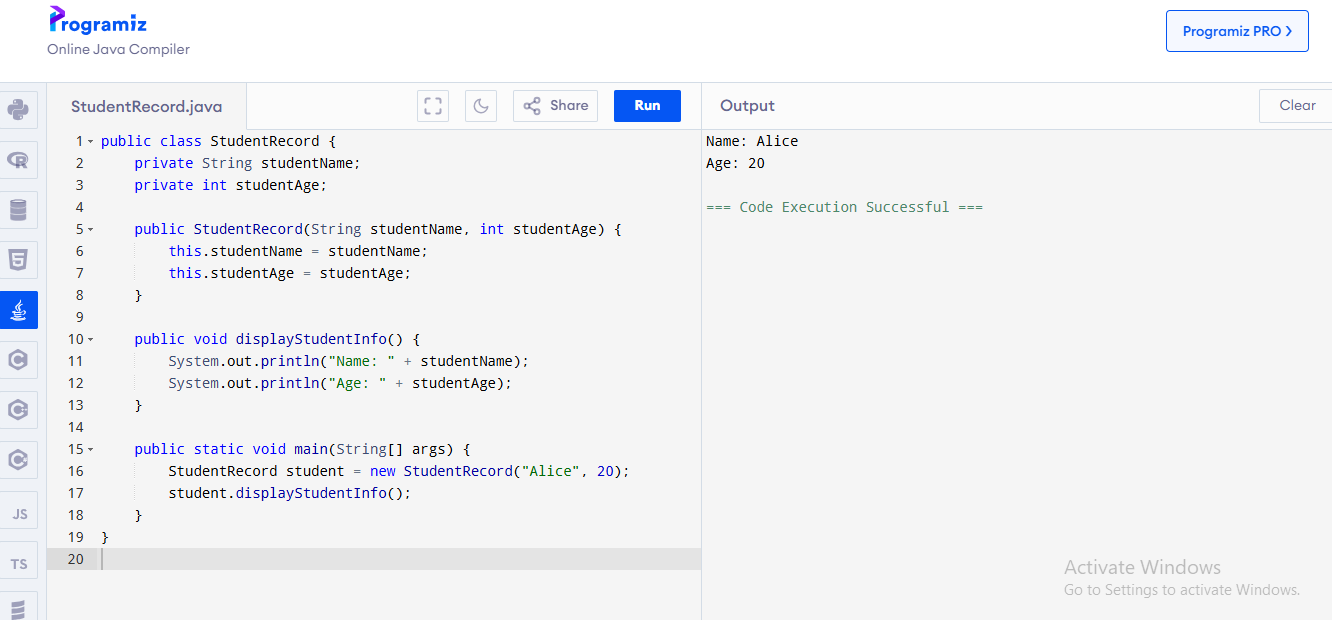
}

public void deposit(double amount) {

accountBalance += amount;

}

}



### **17. What is the importance of breaking your Java program into methods?**

Breaking programs into methods is important because:

* Improves code organization and readability
* Enables code reuse and reduces duplication
* Makes code easier to test and debug
* Follows the Single Responsibility Principle (each method does one thing well)
* Makes maintaining and modifying the program easier

### **18. Explain the concept of DRY (Don't Repeat Yourself) with a Java code example.**

DRY means avoiding code duplication by creating reusable components.

Example with code duplication:

javacode

public class NotDRY {

public static void main(String[] args) {

*// Calculate area of first rectangle*

double length1 = 5.0;

double width1 = 3.0;

double area1 = length1 \* width1;

System.out.println("Rectangle 1 area: " + area1);

*// Calculate area of second rectangle*

double length2 = 7.0;

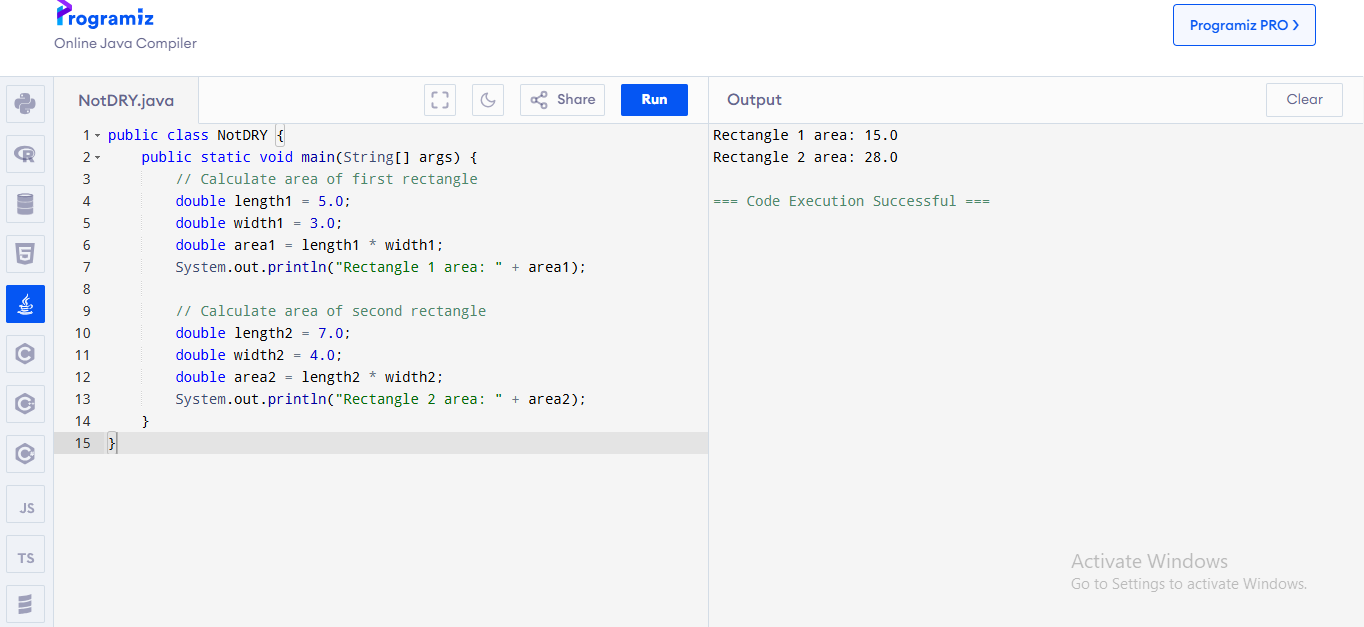
double width2 = 4.0;

double area2 = length2 \* width2;

System.out.println("Rectangle 2 area: " + area2);

}

}



DRY version with a reusable method:

javacode

public class DRY {

*// Reusable method to calculate rectangle area*

public static double calculateRectangleArea(double length, double width) {

return length \* width;

}

public static void main(String[] args) {

*// Calculate area of first rectangle*

double area1 = calculateRectangleArea(5.0, 3.0);

System.out.println("Rectangle 1 area: " + area1);

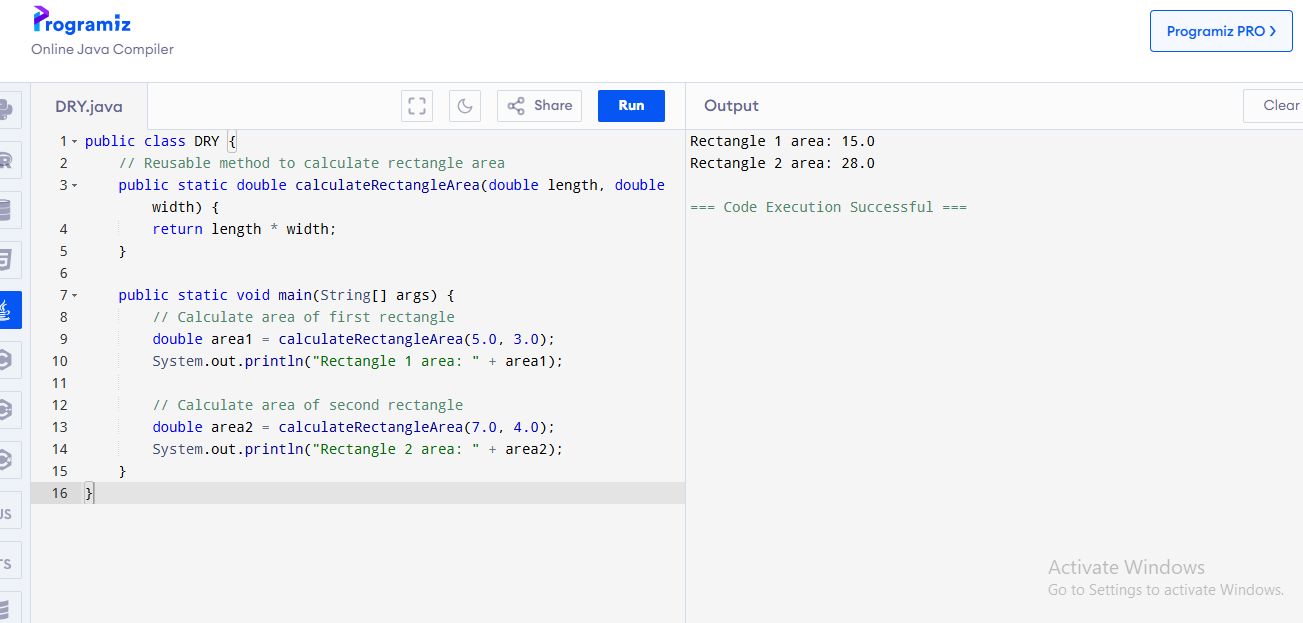
*// Calculate area of second rectangle*

double area2 = calculateRectangleArea(7.0, 4.0);

System.out.println("Rectangle 2 area: " + area2);

}

}



### **19. What are the benefits of using classes and objects instead of writing all logic in the main method?**

Benefits include:

* Better organization and structure of code
* Reusability of code across different parts of the program
* Encapsulation of data and operations together
* Makes large programs more manageable
* Promotes modular design and separation of concerns
* Easier to maintain and extend the functionality

## **Testing & Debugging**

### **20. Why is testing important during program development?**

Testing is important because:

* It helps identify bugs and issues early in development
* Ensures that the application works as expected
* Provides confidence in the reliability of the code
* Facilitates refactoring without introducing new bugs
* Serves as documentation for how the code should work
* Reduces the cost of fixing bugs compared to finding them in production

### **21. What is the difference between syntax error, runtime error, and logic error?**

* **Syntax Error**: Error in the program's code structure (grammar) that prevents compilation. Example: missing semicolon, unmatched brackets.
* **Runtime Error**: Occurs during program execution and causes the program to terminate abnormally. Example: division by zero, null pointer exception.
* **Logic Error**: The program runs without crashing but produces incorrect results due to flawed logic. Example: using < instead of >, incorrect algorithm implementation.

### **22. How would you test a method that calculates the average of five numbers?**

javacode

public class AverageCalculator {

*// Method to calculate average*

public static double calculateAverage(double[] numbers) {

if (numbers.length == 0) {

return 0.0;

}

double sum = 0.0;

for (double num : numbers) {

sum += num;

}

return sum / numbers.length;

}

*// Test method*

public static void testCalculateAverage() {

*// Test case 1: Regular numbers*

double[] test1 = {10.0, 20.0, 30.0, 40.0, 50.0};

double expected1 = 30.0;

double result1 = calculateAverage(test1);

System.out.println("Test 1: " + (Math.abs(result1 - expected1) < 0.0001 ? "PASSED" : "FAILED"));

*// Test case 2: Negative numbers*

double[] test2 = {-10.0, -20.0, -30.0, -40.0, -50.0};

double expected2 = -30.0;

double result2 = calculateAverage(test2);

System.out.println("Test 2: " + (Math.abs(result2 - expected2) < 0.0001 ? "PASSED" : "FAILED"));

*// Test case 3: Mixed numbers*

double[] test3 = {-10.0, 0.0, 10.0, 20.0, 30.0};

double expected3 = 10.0;

double result3 = calculateAverage(test3);

System.out.println("Test 3: " + (Math.abs(result3 - expected3) < 0.0001 ? "PASSED" : "FAILED"));

*// Test case 4: Empty array*

double[] test4 = {};

double expected4 = 0.0;

double result4 = calculateAverage(test4);

System.out.println("Test 4: " + (Math.abs(result4 - expected4) < 0.0001 ? "PASSED" : "FAILED"));

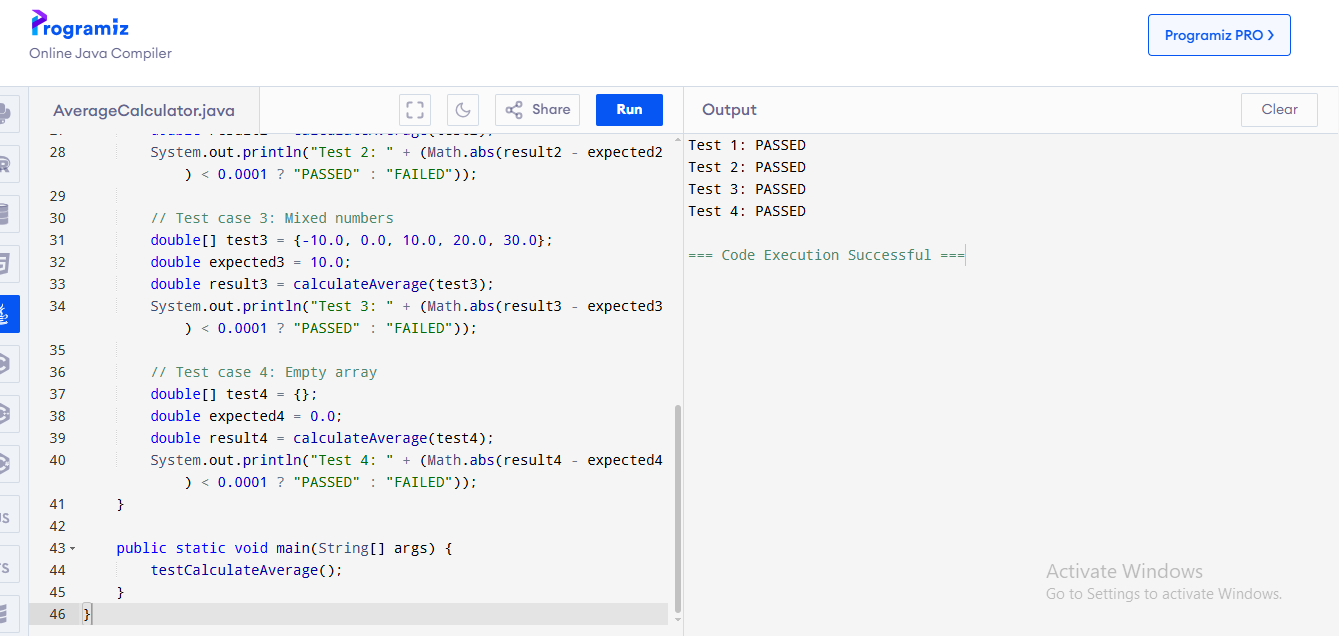
}

public static void main(String[] args) {

testCalculateAverage();

}

}



## **Documentation & Comments**

### **23. Why should Java developers write comments in their code?**

Java developers should write comments to:

* Explain complex logic or algorithms
* Provide context that is not obvious from the code
* Help other developers understand the code's purpose and usage
* Document assumptions, limitations, or special considerations
* Make future maintenance easier, even for the original developer

### **24. What are JavaDoc comments and how are they different from regular comments?**

JavaDoc comments are special comments used to generate API documentation automatically:

* They start with /\*\* and end with \*/
* They support special tags like @param, @return, @throws
* They are associated with classes, methods, and fields
* They are processed by the JavaDoc tool to generate HTML documentation

Regular comments (// or /\* \*/) are ignored by the compiler and not used for documentation generation.

### **25. Write a sample Java method with JavaDoc comments.**

javacode

*/\*\**

\* Calculates the area of a rectangle.

\*

*\* @param length The length of the rectangle in meters.*

*\* @param width The width of the rectangle in meters.*

*\* @return The area of the rectangle in square meters.*

*\* @throws IllegalArgumentException If either length or width is negative.*

*\*/*

public double calculateRectangleArea(double length, double width) {

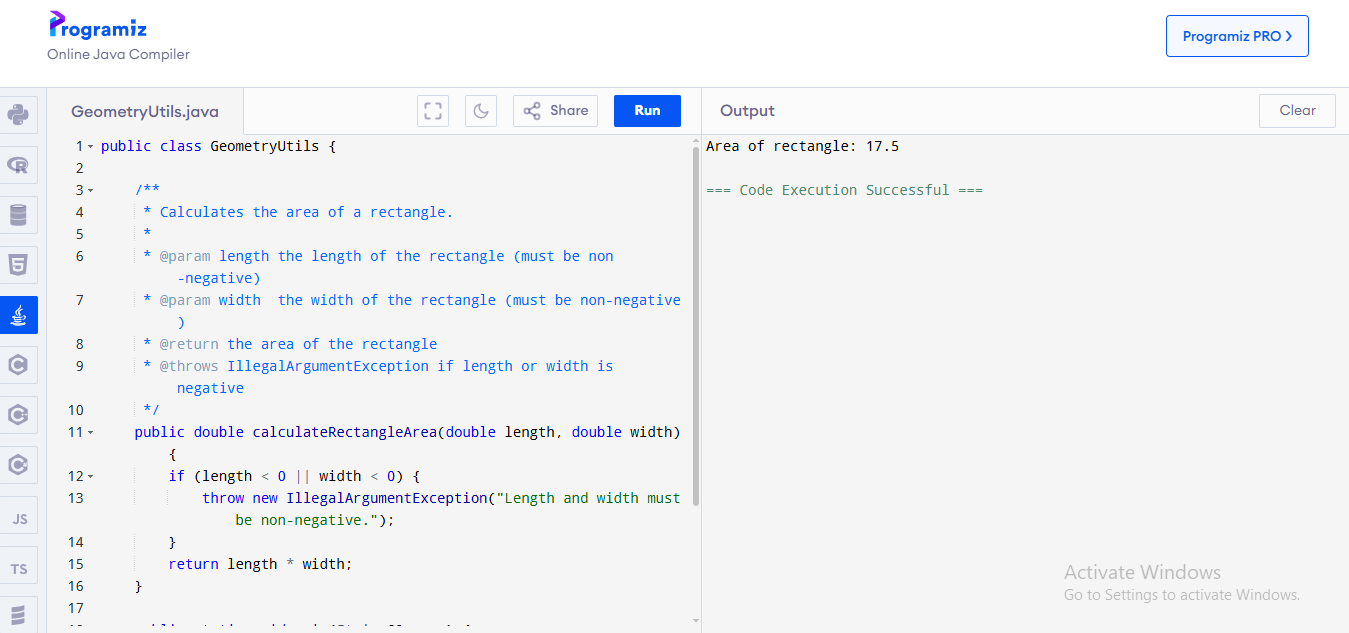
if (length < 0 || width < 0) {

throw new IllegalArgumentException("Length and width must be non-negative");

}

return length \* width;

}



## **Versioning & Collaboration**

### **26. What is version control and why is it important in team projects?**

Version control is a system that records changes to files over time, allowing developers to recall specific versions later. It's important in team projects because:

* It enables multiple developers to work on the same codebase simultaneously
* It tracks changes, showing who made what changes and why
* It provides backup and history of all changes
* It facilitates code review and quality control
* It allows for branching and merging of features
* It simplifies deployment and rollback processes

### **27. How would you explain the concept of "code refactoring" to a junior developer?**

Code refactoring is the process of restructuring existing code without changing its external behavior. It's like renovating a house - you're improving its structure and organization without changing its appearance from the outside.

To a junior developer, I would explain:

* Refactoring makes code cleaner, more efficient, and easier to maintain
* It should be done incrementally, with testing after each change
* Common refactoring techniques include: extracting methods, renaming variables for clarity, removing duplicate code
* It's not about adding new features, but improving how existing features are implemented
* Refactoring should be a regular part of development, not just a one-time activity

### **28. What tools can Java developers use to collaborate on large projects?**

Three examples of collaboration tools for Java developers:

1. **Git/GitHub**: Version control system and platform for code hosting, review, and collaboration
2. **Maven/Gradle**: Build automation tools that manage dependencies and project structure
3. **Jenkins**: Continuous Integration/Continuous Deployment (CI/CD) tool for automated testing and deployment

## **Good Practices Summary**

### **29. Mention 5 best practices you follow when developing a Java program.**

1. **Follow naming conventions**: Use meaningful, clear names for variables, methods, and classes
2. **Write modular code**: Break down complex functionality into smaller, reusable methods
3. **Handle exceptions properly**: Use try-catch blocks and create custom exceptions when appropriate
4. **Write unit tests**: Test individual components to ensure they work correctly
5. **Document your code**: Add comments and JavaDoc to explain complex logic and API usage

### **30. What is code readability, and why is it more important than "smart" code?**

Code readability refers to how easily developers can understand code just by reading it. It's more important than "smart" (clever but complex) code because:

* Readable code is easier to maintain and debug
* Most code is read many more times than it is written
* Teams can collaborate more effectively with readable code
* Readable code reduces the time needed for new developers to understand the codebase
* "Smart" code often uses tricks that are difficult for others to understand and maintain

## **Mini Projects / Logic Building**

### **31. Build a command-line application that keeps track of student grades.**

Student Grade Management System

Javacode

import java.util.ArrayList;

import java.util.Scanner;

/\*\*

\* A command-line application that manages student records and grades.

\* Allows adding, updating, and viewing student records.

\*/

public class StudentGradeSystem {

// ArrayList to store student records

private static ArrayList<Student> students = new ArrayList<>();

private static Scanner scanner = new Scanner(System.in);

public static void main(String[] args) {

boolean running = true;

System.out.println("===== STUDENT GRADE MANAGEMENT SYSTEM =====");

while (running) {

displayMenu();

int choice = getChoice();

switch (choice) {

case 1:

addStudent();

break;

case 2:

updateStudent();

break;

case 3:

viewAllStudents();

break;

case 4:

searchStudent();

break;

case 5:

running = false;

System.out.println("Thank you for using the Student Grade System. Goodbye!");

break;

default:

System.out.println("Invalid choice. Please try again.");

}

}

scanner.close();

}

/\*\*

\* Displays the main menu options

\*/

private static void displayMenu() {

System.out.println("\nPlease select an option:");

System.out.println("1. Add a new student");

System.out.println("2. Update student grade");

System.out.println("3. View all students");

System.out.println("4. Search for a student");

System.out.println("5. Exit");

System.out.print("Enter your choice (1-5): ");

}

/\*\*

\* Gets the user's menu choice and validates it

\* @return The validated menu choice

\*/

private static int getChoice() {

while (!scanner.hasNextInt()) {

System.out.print("Please enter a number (1-5): ");

scanner.next(); // Consume invalid input

}

return scanner.nextInt();

}

/\*\*

\* Adds a new student record to the system

\*/

private static void addStudent() {

scanner.nextLine(); // Consume newline

System.out.println("\n----- Add a New Student -----");

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

String name = scanner.nextLine();

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

String matricNo = scanner.nextLine();

// Check if matric number already exists

for (Student s : students) {

if (s.getMatricNo().equals(matricNo)) {

System.out.println("Error: A student with this matric number already exists!");

return;

}

}

System.out.print("Enter grade (0-100): ");

double grade = validateGrade();

students.add(new Student(name, matricNo, grade));

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

}

/\*\*

\* Updates an existing student's grade

\*/

private static void updateStudent() {

if (students.isEmpty()) {

System.out.println("No students in the system yet!");

return;

}

scanner.nextLine(); // Consume newline

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

System.out.print("Enter student matric number: ");

String matricNo = scanner.nextLine();

for (Student s : students) {

if (s.getMatricNo().equals(matricNo)) {

System.out.print("Enter new grade (0-100): ");

double newGrade = validateGrade();

s.setGrade(newGrade);

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

return;

}

}

System.out.println("Student with matric number " + matricNo + " not found!");

}

/\*\*

\* Displays all student records

\*/

private static void viewAllStudents() {

if (students.isEmpty()) {

System.out.println("No students in the system yet!");

return;

}

System.out.println("\n----- All Students -----");

System.out.printf("%-20s %-15s %-10s %-10s%n", "NAME", "MATRIC NO", "GRADE", "GRADE POINT");

System.out.println("------------------------------------------------------");

for (Student s : students) {

System.out.printf("%-20s %-15s %-10.2f %-10s%n",

s.getName(),

s.getMatricNo(),

s.getGrade(),

s.getLetterGrade());

}

}

/\*\*

\* Searches for a student by matric number

\*/

private static void searchStudent() {

if (students.isEmpty()) {

System.out.println("No students in the system yet!");

return;

}

scanner.nextLine(); // Consume newline

System.out.println("\n----- Search for a Student -----");

System.out.print("Enter student matric number: ");

String matricNo = scanner.nextLine();

for (Student s : students) {

if (s.getMatricNo().equals(matricNo)) {

System.out.println("\nStudent found:");

System.out.println("Name: " + s.getName());

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

System.out.println("Grade: " + s.getGrade());

System.out.println("Grade Point: " + s.getLetterGrade());

return;

}

}

System.out.println("Student with matric number " + matricNo + " not found!");

}

/\*\*

\* Validates grade input to ensure it's between 0 and 100

\* @return A valid grade value

\*/

private static double validateGrade() {

double grade = -1;

boolean valid = false;

while (!valid) {

while (!scanner.hasNextDouble()) {

System.out.print("Please enter a valid number: ");

scanner.next(); // Consume invalid input

}

grade = scanner.nextDouble();

if (grade >= 0 && grade <= 100) {

valid = true;

} else {

System.out.print("Grade must be between 0 and 100. Try again: ");

}

}

return grade;

}

/\*\*

\* Student class to represent a student record

\*/

static class Student {

private String name;

private String matricNo;

private double grade;

public Student(String name, String matricNo, double grade) {

this.name = name;

this.matricNo = matricNo;

this.grade = grade;

}

public String getName() {

return name;

}

public String getMatricNo() {

return matricNo;

}

public double getGrade() {

return grade;

}

public void setGrade(double grade) {

this.grade = grade;

}

/\*\*

\* Converts numeric grade to letter grade

\* @return The letter grade corresponding to the numeric grade

\*/

public String getLetterGrade() {

if (grade >= 70) {

return "A";

} else if (grade >= 60) {

return "B";

} else if (grade >= 50) {

return "C";

} else if (grade >= 45) {

return "D";

} else if (grade >= 40) {

return "E";

} else {

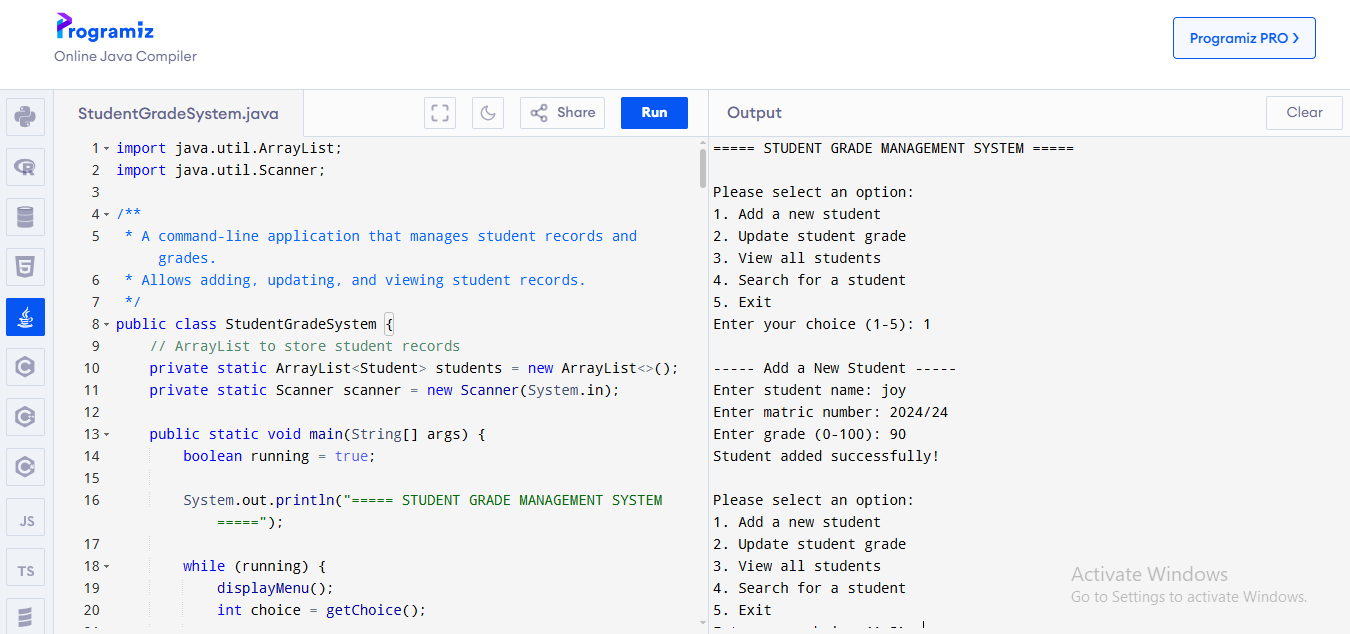
return "F";

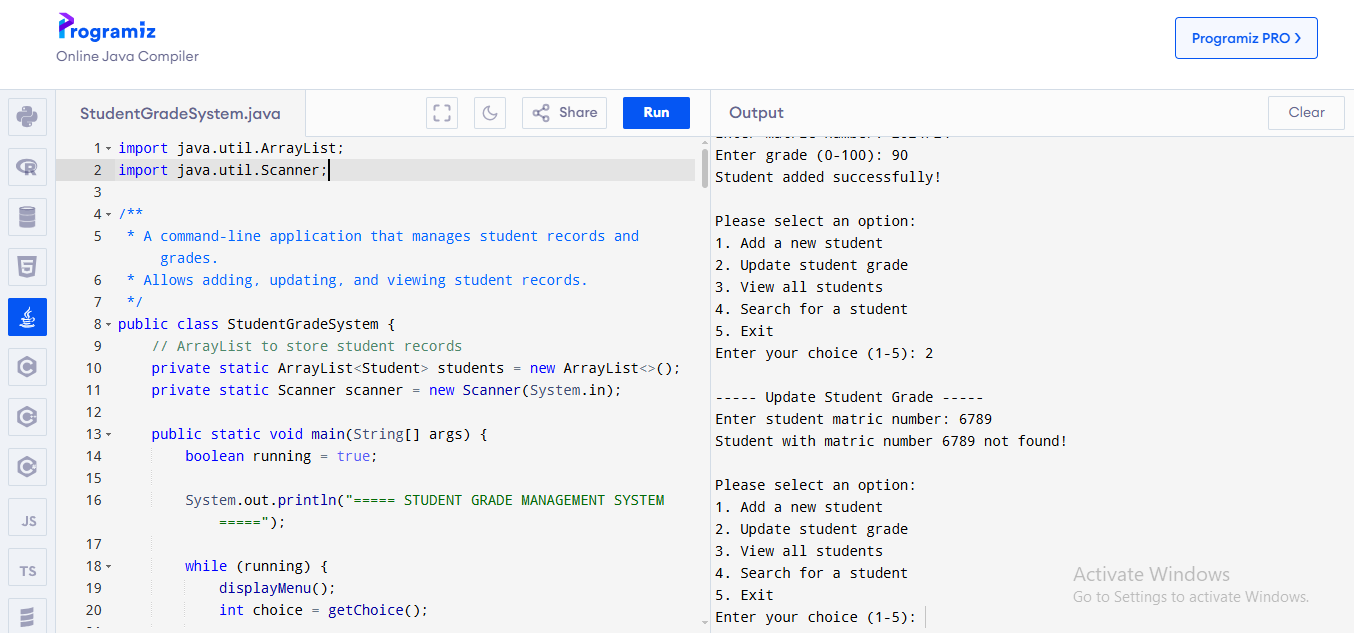
}

}

}

}





### **32. Write a program that simulates a basic ATM system.**

ATM System Simulation

Java Code

import java.util.Scanner;

/\*\*

\* A Java program that simulates a basic ATM system.

\* Features include: check balance, deposit, withdraw

\*/

public class ATMSystem {

private static Scanner scanner = new Scanner(System.in);

private static double balance = 10000.00; // Initial balance

private static final int PIN = 1234; // Default PIN

public static void main(String[] args) {

System.out.println("===== WELCOME TO UNN BANK ATM =====");

if (authenticateUser()) {

boolean running = true;

while (running) {

displayMenu();

int choice = getMenuChoice();

switch (choice) {

case 1: // Check Balance

checkBalance();

break;

case 2: // Deposit

deposit();

break;

case 3: // Withdraw

withdraw();

break;

case 4: // Exit

System.out.println("\nThank you for using UNN Bank ATM. Goodbye!");

running = false;

break;

default:

System.out.println("Invalid choice. Please try again.");

}

// Prompt to continue or exit after each operation

if (running && choice >= 1 && choice <= 3) {

System.out.print("\nPress Enter to continue...");

scanner.nextLine(); // Wait for Enter key

}

}

} else {

System.out.println("Too many incorrect PIN attempts. Your card has been blocked.");

}

scanner.close();

}

/\*\*

\* Authenticates the user by validating PIN

\* @return true if authentication successful, false otherwise

\*/

private static boolean authenticateUser() {

int attempts = 0;

final int MAX\_ATTEMPTS = 3;

while (attempts < MAX\_ATTEMPTS) {

System.out.print("\nPlease enter your PIN: ");

// Validate input is a number

if (!scanner.hasNextInt()) {

System.out.println("Invalid PIN format. Please enter digits only.");

scanner.next(); // Consume invalid input

attempts++;

continue;

}

int enteredPin = scanner.nextInt();

scanner.nextLine(); // Consume newline

if (enteredPin == PIN) {

System.out.println("PIN accepted. Welcome!");

return true;

} else {

attempts++;

int remainingAttempts = MAX\_ATTEMPTS - attempts;

if (remainingAttempts > 0) {

System.out.println("Incorrect PIN. You have " + remainingAttempts +

" attempt" + (remainingAttempts == 1 ? "" : "s") + " remaining.");

}

}

}

return false;

}

/\*\*

\* Displays the main menu options

\*/

private static void displayMenu() {

System.out.println("\n===== ATM MENU =====");

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

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

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

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

System.out.print("Enter your choice (1-4): ");

}

/\*\*

\* Gets and validates user menu choice

\* @return The validated menu choice

\*/

private static int getMenuChoice() {

while (!scanner.hasNextInt()) {

System.out.print("Please enter a number (1-4): ");

scanner.next(); // Consume invalid input

}

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline

return choice;

}

/\*\*

\* Displays current account balance

\*/

private static void checkBalance() {

System.out.println("\n----- ACCOUNT BALANCE -----");

System.out.printf("Your current balance is: ₦%.2f\n", balance);

}

/\*\*

\* Handles deposit operation

\*/

private static void deposit() {

System.out.println("\n----- DEPOSIT -----");

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

double amount = getValidAmount();

if (amount <= 0) {

System.out.println("Invalid amount. Deposit amount must be positive.");

return;

}

balance += amount;

System.out.printf("₦%.2f has been deposited to your account.\n", amount);

System.out.printf("Your new balance is: ₦%.2f\n", balance);

}

/\*\*

\* Handles withdrawal operation

\*/

private static void withdraw() {

System.out.println("\n----- WITHDRAWAL -----");

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

double amount = getValidAmount();

if (amount <= 0) {

System.out.println("Invalid amount. Withdrawal amount must be positive.");

return;

}

if (amount > balance) {

System.out.println("Insufficient funds. Your withdrawal cannot be processed.");

System.out.printf("Your current balance is: ₦%.2f\n", balance);

return;

}

balance -= amount;

System.out.printf("₦%.2f has been withdrawn from your account.\n", amount);

System.out.printf("Your new balance is: ₦%.2f\n", balance);

}

/\*\*

\* Gets and validates a monetary amount input

\* @return A valid monetary amount

\*/

private static double getValidAmount() {

while (!scanner.hasNextDouble()) {

System.out.print("Please enter a valid amount: ₦");

scanner.next(); // Consume invalid input

}

double amount = scanner.nextDouble();

scanner.nextLine(); // Consume newline

return amount;

}

