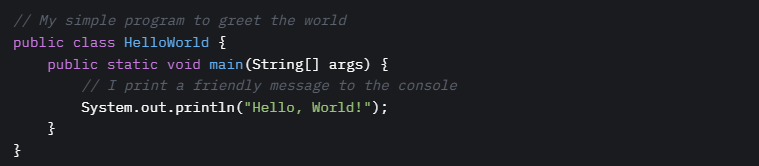
Name: chinweike kennedy

Reg no: 2020/245106

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



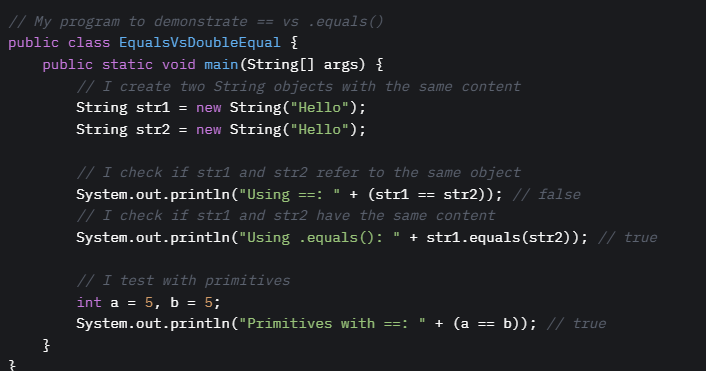
2. Explain the difference between `==` and `.equals()` in Java. Show with code examples and outputs.

Explanation:

- `==`: Compares object references (memory addresses) for equality. For primitives, it compares values.

- `.equals()`: Compares the content of objects (e.g., string values). Its behavior depends on the class’s implementation.

Code:



Output

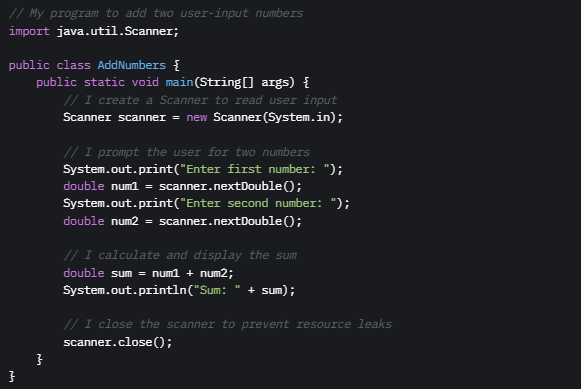


3. What is the use of the main method in Java?

The `main` method is the entry point of a Java program. When you run a Java application, the JVM looks for `public static void main(String[] args)` to start execution. It’s where I initialize my program’s logic and control flow.

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

Code:



Output:



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

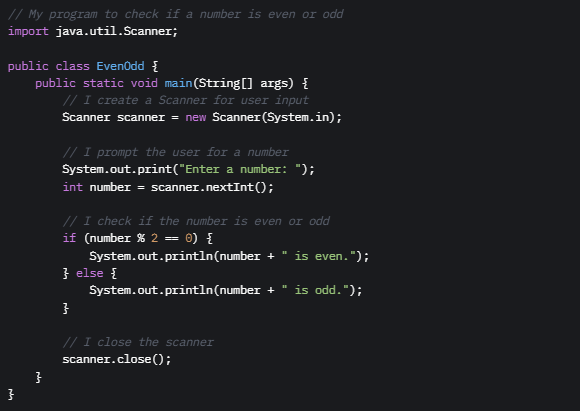
-int: A primitive data type for integers (e.g., 42). It’s not an object, so it has no methods and uses less memory.

-Integer: A wrapper class for `int`. It’s an object, supports methods (e.g., `Integer.parseInt()`), and can be used in collections.

- String: A class representing a sequence of characters (e.g., "Hello"). It’s immutable and has methods like `.length()`.

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

Code:

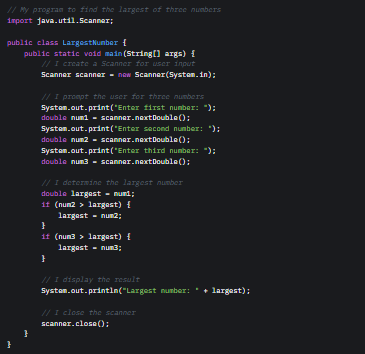


Output:



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

Code:



Output:



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

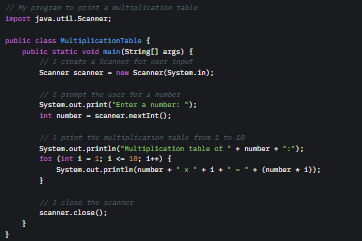
- while: I use it when I want to repeat a block of code as long as a condition is true. It checks the condition before executing.

- for: I use it for a fixed number of iterations, with initialization, condition, and update in one line. It’s compact.

- do-while: I use it when I need the loop to run at least once. It checks the condition after executing.

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

Code:



Output:



10. Explain the four pillars of OOP in Java.

1. Encapsulation: I bundle data and methods in a class and control access using access modifiers (e.g., `private`, `public`). It protects data integrity.

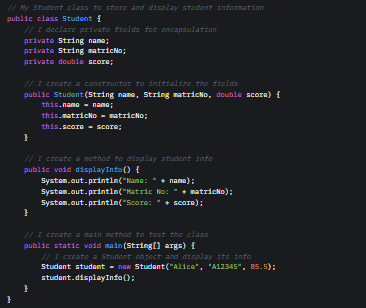
2. Inheritance: I create a new class that inherits properties and methods from an existing class, promoting code reuse (e.g., `extends`).

3. Polymorphism: I allow objects to take multiple forms, either through method overriding (runtime) or overloading (compile-time).

4. Abstraction: I hide complex implementation details and expose only necessary features using abstract classes or interfaces.

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

Code:



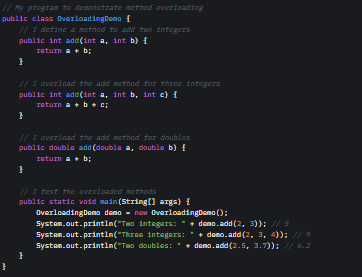
Output:



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

Explanation: Method overloading is when I define multiple methods with the same name in a class but with different parameter lists (number, type, or order). Java decides which method to call based on the arguments.

Code:



Output:



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

Explanation: Inheritance allows a class (subclass) to inherit fields and methods from another class (superclass) using `extends`. It promotes code reuse.

Code:



Output:



14. What does it mean to write “clean code”? Give 3 practices that make code clean and maintainable.

Clean Code: Code that is easy to read, understand, and maintain.

Practices:

1. Meaningful Naming: I use descriptive names for variables and methods (e.g., `calculateTotalScore` instead of `calc`).

2. Consistent Formatting: I follow consistent indentation and spacing (e.g., using 4-space tabs).

3. Single Responsibility: I ensure each method or class has one purpose (e.g., a `Student` class only manages student data).

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

Long methods are hard to read, test, and maintain. I break them into smaller methods to:

- Improve readability.

- Make debugging easier.

- Promote reuse.

- Follow the single responsibility principle.

16. What naming conventions should be followed in Java for: Classes, Variables, Methods. Give examples with screenshot of code and output.

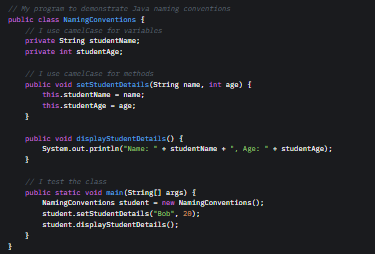
Conventions:

- Classes: PascalCase (e.g., `StudentRecord`).

- Variables: camelCase (e.g., `studentName`).

- Methods: camelCase (e.g., `calculateAverage`).

Code:



Output:



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

Breaking a program into methods:

- Improves readability by organizing code logically.

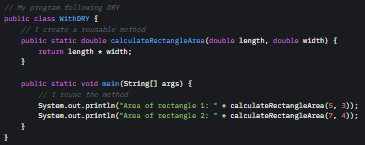
- Makes debugging easier by isolating issues.

- Promotes reuse across the program.

- Enhances maintainability by keeping methods focused.

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

Code:



Output:



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

- Modularity: I organize code into classes for better structure.

- Reusability: I can reuse classes and objects across programs.

- Encapsulation: I protect data using access modifiers.

- Maintainability: Classes make code easier to update and debug.

- Scalability: Objects support complex systems with clear relationships.

20. Why is testing important during program development?

Testing ensures my code works as expected, catches bugs early, and prevents regressions. It improves reliability and user trust.

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

- Syntax Error: I make a mistake in code structure (e.g., missing `;`). Caught during compilation.

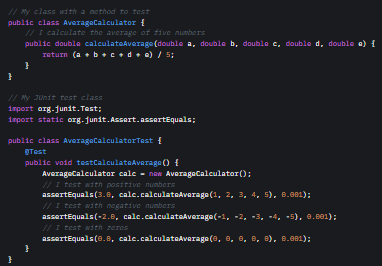
- Runtime Error: My program crashes during execution (e.g., `NullPointerException`). Occurs at runtime.

- Logic Error: My code runs but produces incorrect results (e.g., wrong calculation). Hardest to detect.

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

I’d use unit testing (e.g., JUnit) to verify the method with different inputs.

Code:



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

Comments make my code easier to understand, maintain, and collaborate on. They explain intent, logic, or complex sections for future readers (including myself).

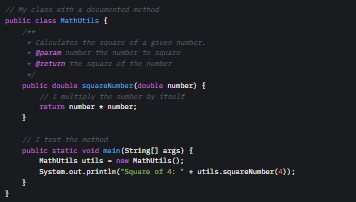
24. What are JavaDoc comments and how are they different from regular comments?

- JavaDoc Comments: I use `/ ... /` to document classes, methods, or fields. They generate API documentation and include tags like `@param` or `@return`.

- Regular Comments: I use `//` or `/ ... /` for internal notes. They don’t generate documentation.

25. Write a sample Java method with JavaDoc comments.

Code:



Output:



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

Version Control: Tracks changes to code, allowing multiple developers to collaborate. I use tools like Git to manage versions.

Importance:

- Tracks who made what changes.

- Enables reverting to previous versions.

- Supports parallel development via branches.

- Prevents conflicts in team projects.

27. How would you explain the concept of “code refactoring” to a junior developer?

I’d say: “Refactoring is like tidying up your room without changing what it’s used for. You improve the code’s structure—making it cleaner, simpler, or more efficient—without altering its behavior. For example, I might rename variables for clarity or split a long method into smaller ones.”

28. What tools can Java developers use to collaborate on large projects? Attach screenshots of 3 examples.

Tools:

1. GitHub: I use it for version control, pull requests, and code reviews.

2. IntelliJ IDEA: I collaborate using its Git integration and code-sharing features.

3. Slack: I communicate with my team and share code snippets.

Screenshot Note: To visualize:

- GitHub: Visit a repository (e.g., the one I’ll share). You’ll see the code, commits, and pull requests.

- IntelliJ IDEA: Open a Java project, and the Git tab shows commit and branch options.

- Slack: Check a channel with code snippets shared in backticks (```java ... ```).

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

1. Use Meaningful Names: I name variables and methods clearly (e.g., `calculateTotal`).

2. Write Comments: I document my code for clarity.

3. Follow DRY: I avoid duplicating logic by reusing methods.

4. Test Thoroughly: I write unit tests to verify functionality.

5. Keep Methods Short: I break long methods into smaller, focused ones.

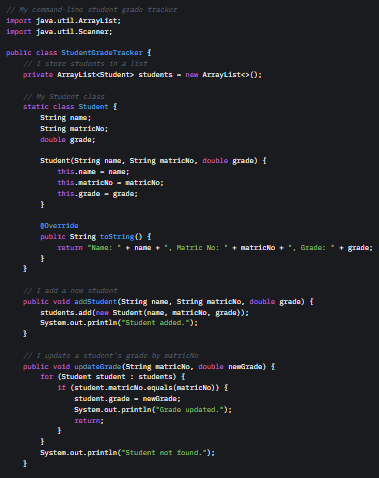
30. What is code readability, and why is it more important than “smart” code?

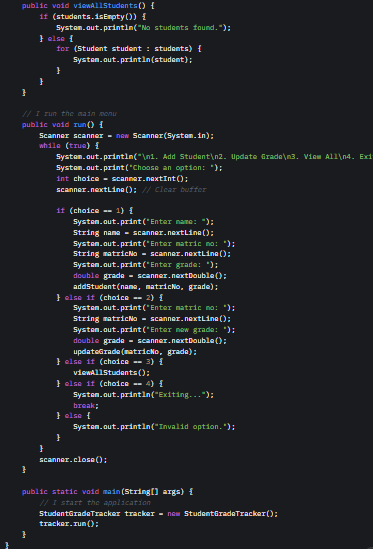
Code Readability: Code that’s easy to understand at a glance. I use clear names, consistent formatting, and simple logic.

Why It’s More Important: Smart code (e.g., overly clever one-liners) can be hard to maintain or debug. Readable code saves time for me and my team, reduces errors, and improves collaboration.

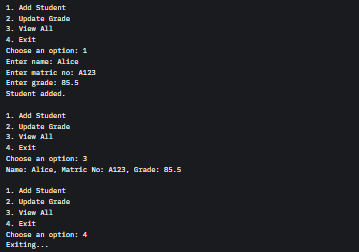
31. Build a command-line application that keeps track of student grades and allows adding, updating, and viewing records.

Code:



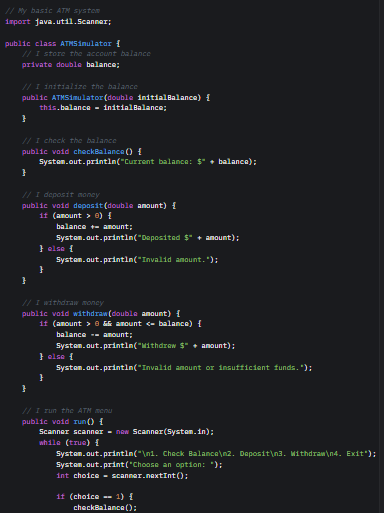


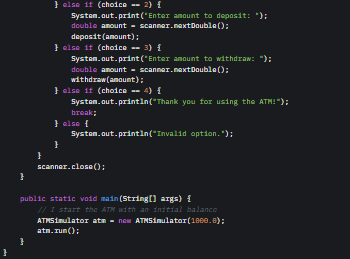
Output:



32. Write a program that simulates a basic ATM system (check balance, deposit, withdraw).

Code:





Output:

