Clean Code

“Any fool can write code that a computer can understand.
Good programmers write code that humans can understand.”
— Martin Fowler

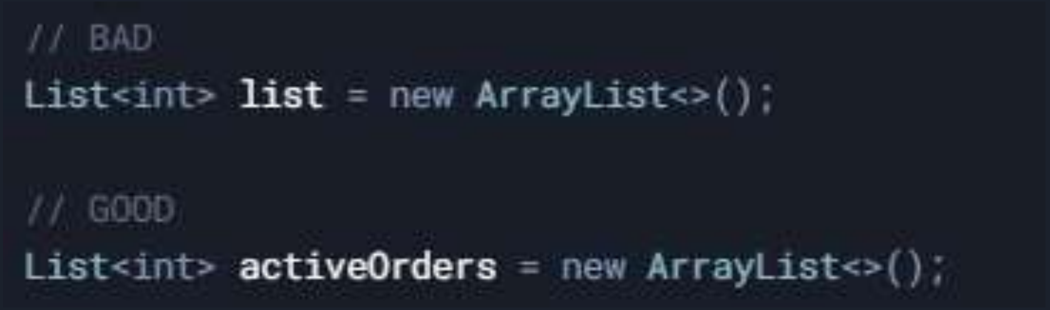
Prepared by: Ali Khalaf

What is Clean Code?

- **Myth**: “If it works, it’s clean.” → **False**.
- **Truth**:
    - Readable, maintainable, and self-explanatory.
    - Written for **people**, not just machines.
- **Why Care?**
    - Developers spend 80% of time **reading** code
    - Critical for team collaboration, debugging, and future updates.
- You are not only a programmer you are an AUTHOR!!

Naming Principles

- **Meaningful Names**:
  - Replace "`u`" with "`user`", "`v`" with "`isValid`".
  - Avoid "noise words" like "`data`", "`info`" (e.g., `userData` → `user`).
- **Consistency**:
  - Use the same term across the codebase (e.g., `fetch` vs. `retrieve`).
- **Avoid Redundancy**:
  - Class names should not repeat the context (e.g., `Car.carSpeed` → `Car.speed`).



```
// BAD
List<int> list = new ArrayList<>();

// GOOD
List<int> activeOrders = new ArrayList<>();
```

Naming Anti-Patterns & Best Practices

Anti-Patterns

Cryptic Abbreviations: ordSts → orderStatus

Misleading Names: accountsList (if not List) → accounts

Magic Numbers: if (status == 2) → ORDER_COMPLETED

Best Practices

Pronounceable Names: genymdhms → generationTimestamp

Searchable Names: MAX_ORDERS_PER_DAY instead of 100

Naming is technical communication. A good name answers:

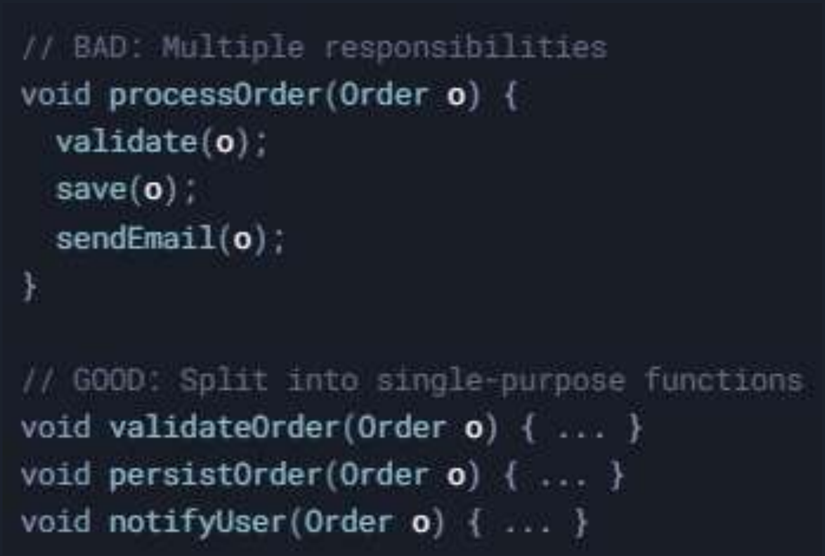
- *Why it exists?*
- *What it does?*
- *How it's used?*

Functions

Do One Thing: Single responsibility.

Small: ≤ 20 lines.

DRY: Eliminate duplication.



```
// BAD: Multiple responsibilities
void processOrder(Order o) {
  validate(o);
  save(o);
  sendEmail(o);
}

// GOOD: Split into single-purpose functions
void validateOrder(Order o) { ... }
void persistOrder(Order o) { ... }
void notifyUser(Order o) { ... }
```

Comments

Comments to Avoid:

- **Redundant**: `// increment count` → `count++;`
- **Outdated**: `// TODO: Fix in 2020` (still there in 2025).

Good Comments:

- **Legal**: // Copyright 2023, Company X.
- **Warnings**: // WARNING: Costs $0.01 per call.
- **Complex Logic:** // Uses SHA-256 for secure hashing.

Formatting

- **Vertical Formatting**:
  - Group related code (e.g., variable declarations near usage).
  - Separate concepts with blank lines.
  - Functions should be short enough to fit on one screen (~20 lines).
- **Horizontal Formatting**:
  - Limit line length to **80–120 characters** (improves readability).
  - Use indentation consistently (e.g., 2/4 spaces).
- **Team Consistency**:
  - Agree on formatting rules (tabs vs. spaces, brace placement).
  - Use IDE auto-formatters (e.g., Prettier, IntelliJ).

Formatting

```java
// BAD: No vertical grouping, cramped
public class Order {
private int id;
private String status;
public Order(int id,String status){this.id=id;this.status=status;}
}

// GOOD: Organized and spaced
public class Order {
    private int id;
    private String status;

    public Order(int id, String status) {
        this.id = id;
        this.status = status;
    }
}
```

The Boy Scout Rule

"Leave the code cleaner than you found it."

How to Apply:

- Fix small issues:
    a. Rename a confusing variable.
    b. Break up a long function.
    c. Delete a redundant comment.

Impact:

- Prevents "broken windows" (minor issues → chaos).
- Gradual improvement without massive rewrites.

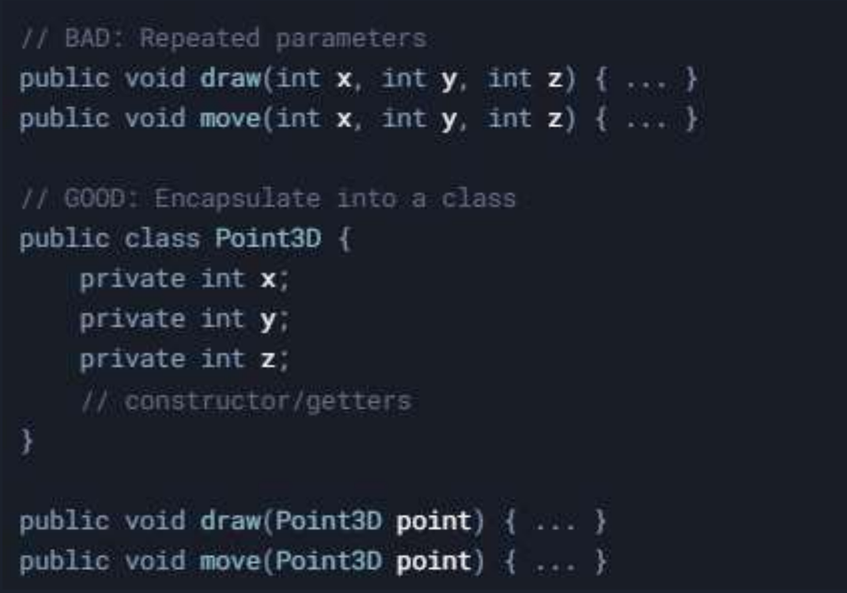
Naming Case Types

Case Type	Example	Used In	Typical Usage
`snake_case`	`is_valid`	Python	Variables, Functions, Methods
`camelCase`	`isValid`	Java, JavaScript	Variables, Functions, Methods
`PascalCase`	`AdminRole`	Python, Java, JS	Classes, Types
`kebab-case`	`<side-drawer>`	HTML	Custom HTML/Web Components

Code Smells & Refactoring

Common Smells:

1. **Long Methods**:
   - Functions > 20 lines → Break into smaller methods.
2. **Primitive Obsession**:
   - Overusing primitives (e.g., `String phone`) → Replace with objects (`PhoneNumber` class).
3. **Data Clumps**:
   - Groups of variables passed together (e.g., `x, y, z`) → Encapsulate into a class (`Point3D`).



```java
// BAD: Repeated parameters
public void draw(int x, int y, int z) { ... }
public void move(int x, int y, int z) { ... }

// GOOD: Encapsulate into a class
public class Point3D {
    private int x;
    private int y;
    private int z;
    // constructor/getters
}

public void draw(Point3D point) { ... }
public void move(Point3D point) { ... }
```

Key Principles of Clean Code

1. Agile Craftsmanship

   Clean code enables agility

2. Relentless Refactoring

   Continuous improvement

3. Readability

   Prioritize clarity over complexity

4. Future-Proof

   Maintainable, understandable code

Reference: *Clean Code* by Robert C. Martin