The CLEAN Code Toolkit

Write Maintainable, Scalable, and Readable Code

What's Inside?

This toolkit is designed to help you write better software, refactor with confidence, and improve your code reviews. Inside, you'll find:

✓ CLEAN Code Cheatsheet – A quick reference guide to instantly improve your code.

✓ CLEAN Code Qualities Checklist – How to spot and apply key qualities of great code.

✓ CLEAN Code Refactoring Guide – Step-by-step strategies with before-and-after examples.

✓ CLEAN Code Code Review Checklist – A structured way to ensure high-quality code before merging.

1. CLEAN Code Cheatsheet

CLEAN Code Principles at a Glance

Principle	What It Means	How to Apply It	What Happens When Missing
Cohesion	Functions/classes should do one thing well	Follow the Single Responsibility Principle (SRP)	Code becomes confusing, hard to maintain
Loose Coupling	Minimize dependencies between components	Use dependency injection & interfaces	Changes ripple across the system, making updates risky
Encapsulation	Hide internal details, expose only necessary functionality	Keep fields private, provide controlled access	Implementation details leak, making refactoring difficult
Assertiveness	Objects manage their own state rather than querying others	Push behavior into the object that owns the data	Code becomes procedural, violating object-oriented principles
Non-Redundancy	Avoid duplicated logic and unnecessary repetition	Apply DRY (Don't Repeat Yourself)	Codebase becomes bloated and inconsistent

(c) 2025 The Passionate Programmer - https://passprog.com

CLEAN Code Best Practices

• Keep functions small and focused on one task.

• Use meaningful names that clearly express intent.

• Favor composition over inheritance to reduce tight coupling.

• Keep state private, exposing only necessary data.

• Remove unused code to keep the codebase clean.

Code Smells to Watch For

• Long methods – Break them into smaller, reusable functions.

• Tightly coupled code – Introduce interfaces or dependency injection.

• Excessive getters/setters – Move behavior into the object.

• Duplicated code – Extract common logic into functions/classes.

• Confusing naming – Use descriptive, domain-relevant names.

2. CLEAN Code Qualities Checklist

Quick-Check Questions

• Does each function/class serve only one purpose?

• Can a module be changed without breaking others?

• Are implementation details hidden behind interfaces?

• Do objects manage their own state, rather than querying others?

• Is code duplicated unnecessarily?

How to Improve CLEAN Code

• Refactor aggressively – Remove duplication and extract functions.

• Apply design patterns – Use Dependency Injection, Strategy Pattern, etc.

• Write automated tests – Ensure encapsulation and loose coupling.

• Use meaningful names – Make code self-explanatory.

• Eliminate dead code – Keep your codebase lean and relevant.

3. CLEAN Code Refactoring Guide

Why Refactor?

Refactoring improves readability, maintainability, and performance. Applying CLEAN Code principles ensures your codebase remains scalable and easy to work with.

Step 1: Identify Code Smells

Code Smell	What It Looks Like	Why It's a Problem	Solution
Long Methods	Functions with 50+ lines of code	Hard to understand, reuse, and test	Extract smaller functions
Large Classes	Classes with too many responsibilities	Violates SRP, harder to maintain	Split into smaller classes
Tight Coupling	Objects depend too much on others	Changes ripple across system	Use dependency injection
Duplicated Code	The same logic repeated in multiple places	Harder to maintain, higher risk of bugs	Extract into reusable functions
Poor Naming	Variables named x, data, tmp	Makes code unclear	Use meaningful names

Step 2: Apply CLEAN Code Principles

• Cohesion – Ensure each function/class has a single, well-defined purpose.

• Loose Coupling – Use dependency injection and interfaces to minimize dependencies.

• Encapsulation – Hide internal details, exposing only necessary functionality.

(c) 2025 The Passionate Programmer - https://passprog.com

• Assertiveness – Objects manage their own state, not rely on others.

• Non-Redundancy – Apply DRY (Don't Repeat Yourself) to eliminate duplication.

Step 3: Refactoring in Action

Before (Messy Code):

```
void processOrder(Order order) {
    if (order.items.isEmpty()) {
        System.out.println("Order is empty.");
        return;
    }
    double total = 0;
    for (Item item : order.items) {
        total += item.price;
    }
    System.out.println("Total: " + total);
}
```

After (Refactored Code):

```
void processOrder(Order order) {
    if (isEmpty(order)) return;
    double total = calculateTotal(order);
    printTotal(total);
}

private boolean isEmpty(Order order) {
    return order.items.isEmpty();
}

private double calculateTotal(Order order) {
    return order.items.stream().mapToDouble(item -> item.price).sum();
}

private void printTotal(double total) {
    System.out.println("Total: " + total);
}
```

✓ Improved: Readability, maintainability, and testability.

Step 4. CLEAN Code Code Review Checklist

General Code Quality Checks

• Does the code follow the Single Responsibility Principle (SRP)?

(c) 2025 The Passionate Programmer - https://passprog.com

• Are function and class names meaningful and self-explanatory?

• Is there any duplicated code that should be refactored?

• Are there unnecessary comments explaining “what” instead of “why”?

Common Code Smells to Watch For

• Long methods & large classes – Break them down into smaller units.

• Hardcoded values – Use constants or configuration settings.

• Deeply nested conditionals – Use guard clauses or strategy patterns.

• Overuse of static methods – Can lead to tight coupling.

• Magic numbers & strings – Use descriptive constants.

Final Check Before Merging

✓ Code aligns with team coding standards.

✓ Dependencies are well-managed and justified.

✓ Performance implications are considered and optimized.

✓ Code has been peer-reviewed and approved.

By following these principles, you’ll transform your code into something that’s scalable, maintainable, and easy to work with.

(c) 2025 The Passionate Programmer - https://passprog.com