Clean Code
Clean Code: A Handbook of Agile Software Craftsmanship

Table of Contents

Cover
Half Title
Title Page
Copyright Page
Contents
Foreword
Introduction
On the Cover
Chapter 1: Clean Code
  There Will Be Code
  Bad Code
  The Total Cost of Owning a Mess
    The Grand Redesign in the Sky
    Attitude
    The Primal Conundrum
  The Art of Clean Code?
    What Is Clean Code?
Schools of Thought
We Are Authors
The Boy Scout Rule
Prequel and Principles
Conclusion
Table of Contents

Bibliography

Chapter 2: Meaningful Names
  Introduction
  Use Intention-Revealing Names
  Avoid Disinformation
  Make Meaningful Distinctions
  Use Pronounceable Names
  Use Searchable Names
  Avoid Encodings
    Hungarian Notation
    Member Prefixes
    Interfaces and Implementations
  Avoid Mental Mapping
  Class Names
  Method Names
  Don't Be Cute
  Pick One Word per Concept
  Don't Pun
  Use Solution Domain Names
  Use Problem Domain Names
  Add Meaningful Context
  Don't Add Gratuitous Context
  Final Words

Chapter 3: Functions
  Small!
    Blocks and Indenting
  Do One Thing
Table of Contents

Sections within Functions

One Level of Abstraction per Function
  Reading Code from Top to Bottom: The Stepdown Rule

Switch Statements

Use Descriptive Names

Function Arguments
  Common Monadic Forms
  Flag Arguments
  Dyadic Functions
  Triads
  Argument Objects
  Argument Lists
  Verbs and Keywords

Have No Side Effects
  Output Arguments

Command Query Separation

Prefer Exceptions to Returning Error Codes
  Extract Try/Catch Blocks
  Error Handling Is One Thing
  The Error.java Dependency Magnet

Don't Repeat Yourself

Structured Programming

How Do You Write Functions Like This?

Conclusion

SetupTeardownIncluder

Bibliography

Chapter 4: Comments
  Comments Do Not Make Up for Bad Code
Table of Contents

Explain Yourself in Code

Good Comments
- Legal Comments
- Informative Comments
- Explanation of Intent
- Clarification
- Warning of Consequences
- TODO Comments
- Amplification
- Javadoc in Public APIs

Bad Comments
- Mumbling
- Redundant Comments
- Misleading Comments
- Mandated Comments
- Journal Comments
- Noise Comments
- Scary Noise
- Don't Use a Comment When You Can Use a Function or a Variable
- Position Markers
- Closing Brace Comments
- Attributions and Bylines
- Commented-Out Code
- HTML Comments
- Nonlocal Information
- Too Much Information
- Inobvious Connection
- Function Headers
- Javadoc in Nonpublic Code
- Example
Table of Contents

Bibliography

Chapter 5: Formatting
The Purpose of Formatting
Vertical Formatting
   The Newspaper Metaphor
   Vertical Openness Between Concepts
   Vertical Density
   Vertical Distance
   Vertical Ordering
Horizontal Formatting
   Horizontal Openness and Density
   Horizontal Alignment
   Indentation
   Dummy Scopes

Team Rules
Uncle Bob's Formatting Rules

Chapter 6: Objects and Data Structures
Data Abstraction
Data/Object Anti-Symmetry
The Law of Demeter
   Train Wrecks
   Hybrids
   Hiding Structure
Data Transfer Objects
   Active Record

Conclusion
Bibliography

Chapter 7: Error Handling
Table of Contents

Use Exceptions Rather Than Return Codes
Write Your Try-Catch-Finally Statement First
Use Unchecked Exceptions
Provide Context with Exceptions
Define Exception Classes in Terms of a Caller's Needs
Define the Normal Flow
Don't Return Null
Don't Pass Null
Conclusion
Bibliography

Chapter 8: Boundaries
   Using Third-Party Code
   Exploring and Learning Boundaries
   Learning log4j
   Learning Tests Are Better Than Free
   Using Code That Does Not Yet Exist
   Clean Boundaries
   Bibliography

Chapter 9: Unit Tests
   The Three Laws of TDD
   Keeping Tests Clean
      Tests Enable the -ilities
   Clean Tests
      Domain-Specific Testing Language
      A Dual Standard
   One Assert per Test
      Single Concept per Test
Table of Contents

F.I.R.S.T.
Conclusion
Bibliography

Chapter 10: Classes
Class Organization
   Encapsulation
Classes Should Be Small!
   The Single Responsibility Principle
   Cohesion
   Maintaining Cohesion Results in Many Small Classes
Organizing for Change
   Isolating from Change
Bibliography

Chapter 11: Systems
How Would You Build a City?
Separate Constructing a System from Using It
   Separation of Main
   Factories
   Dependency Injection
Scaling Up
   Cross-Cutting Concerns
Java Proxies
Pure Java AOP Frameworks
AspectJ Aspects
Test Drive the System Architecture
Optimize Decision Making
Use Standards Wisely, When They Add Demonstrable Value
Systems Need Domain-Specific Languages
Table of Contents

Conclusion
Bibliography

Chapter 12: Emergence
Getting Clean via Emergent Design
Simple Design Rule 1: Runs All the Tests
Simple Design Rules 24: Refactoring
No Duplication
Expressive
Minimal Classes and Methods
Conclusion
Bibliography

Chapter 13: Concurrency
Why Concurrency?
   Myths and Misconceptions
Challenges
Concurrency Defense Principles
   Single Responsibility Principle
   Corollary: Limit the Scope of Data
   Corollary: Use Copies of Data
   Corollary: Threads Should Be as Independent as Possible
Know Your Library
   Thread-Safe Collections
Know Your Execution Models
   Producer-Consumer
   Readers-Writers
   Dining Philosophers
Beware Dependencies Between Synchronized Methods
Keep Synchronized Sections Small
Table of Contents

Writing Correct Shut-Down Code Is Hard
Testing Threaded Code
   Treat Spurious Failures as Candidate Threading Issues
   Get Your Nonthreaded Code Working First
   Make Your Threaded Code Pluggable
   Make Your Threaded Code Tunable
   Run with More Threads Than Processors
   Run on Different Platforms
   Instrument Your Code to Try and Force Failures
   Hand-Coded
   Automated
Conclusion
Bibliography

Chapter 14: Successive Refinement
   Args Implementation
      How Did I Do This?
   Args: The Rough Draft
      So I Stopped
      On Incrementalism
   String Arguments
Conclusion

Chapter 15: JUnit Internals
   The JUnit Framework
Conclusion

Chapter 16: Refactoring SerialDate
   First, Make It Work
   Then Make It Right
Conclusion
Table of Contents

Bibliography

Chapter 17: Smells and Heuristics

Comments
   C1: Inappropriate Information
   C2: Obsolete Comment
   C3: Redundant Comment
   C4: Poorly Written Comment
   C5: Commented-Out Code

Environment
   E1: Build Requires More Than One Step
   E2: Tests Require More Than One Step

Functions
   F1: Too Many Arguments
   F2: Output Arguments
   F3: Flag Arguments
   F4: Dead Function

General
   G1: Multiple Languages in One Source File
   G2: Obvious Behavior Is Unimplemented
   G3: Incorrect Behavior at the Boundaries
   G4: Overridden Safeties
   G5: Duplication
   G6: Code at Wrong Level of Abstraction
   G7: Base Classes Depending on Their Derivatives
   G8: Too Much Information
   G9: Dead Code
   G10: Vertical Separation
   G11: Inconsistency
   G12: Clutter
# Table of Contents

G13: Artificial Coupling  
G14: Feature Envy  
G15: Selector Arguments  
G16: Obscured Intent  
G17: Misplaced Responsibility  
G18: Inappropriate Static  
G19: Use Explanatory Variables  
G20: Function Names Should Say What They Do  
G21: Understand the Algorithm  
G22: Make Logical Dependencies Physical  
G23: Prefer Polymorphism to If/Else or Switch/Case  
G24: Follow Standard Conventions  
G25: Replace Magic Numbers with Named Constants  
G26: Be Precise  
G27: Structure over Convention  
G28: Encapsulate Conditionals  
G29: Avoid Negative Conditionals  
G30: Functions Should Do One Thing  
G31: Hidden Temporal Couplings  
G32: Don't Be Arbitrary  
G33: Encapsulate Boundary Conditions  
G34: Functions Should Descend Only One Level of Abstraction  
G35: Keep Congurable Data at High Levels  
G36: Avoid Transitive Navigation  

**Java**  
J1: Avoid Long Import Lists by Using Wildcards  
J2: Don't Inherit Constants  
J3: Constants versus Enums  

**Names**  
N1: Choose Descriptive Names
# Table of Contents

N2: Choose Names at the Appropriate Level of Abstraction  
N3: Use Standard Nomenclature Where Possible  
N4: Unambiguous Names  
N5: Use Long Names for Long Scopes  
N6: Avoid Encodings  
N7: Names Should Describe Side-Effects  

Tests  
T1: Insufficient Tests  
T2: Use a Coverage Tool!  
T3: Don't Skip Trivial Tests  
T4: An Ignored Test Is a Question about an Ambiguity  
T5: Test Boundary Conditions  
T6: Exhaustively Test Near Bugs  
T7: Patterns of Failure Are Revealing  
T8: Test Coverage Patterns Can Be Revealing  
T9: Tests Should Be Fast  

Conclusion  
Bibliography  

Appendix A: Concurrency II  
Client/Server Example  
   The Server  
      Adding Threading  
      Server Observations  
      Conclusion  
Possible Paths of Execution  
   Number of Paths  
   Digging Deeper  
   Conclusion  
Knowing Your Library
Table of Contents

Executor Framework
Nonblocking Solutions
Nonthread-Safe Classes

Dependencies Between Methods Can Break Concurrent Code
  Tolerate the Failure
  Client-Based Locking
  Server-Based Locking

Increasing Throughput
  Single-Thread Calculation of Throughput
  Multithread Calculation of Throughput

Deadlock
  Mutual Exclusion
  Lock & Wait
  No Preemption
  Circular Wait
  Breaking Mutual Exclusion
  Breaking Lock & Wait
  Breaking Preemption
  Breaking Circular Wait

Testing Multithreaded Code

Tool Support for Testing Thread-Based Code

Conclusion

Tutorial: Full Code Examples
  Client/Server Nonthreaded
  Client/Server Using Threads

Appendix B: org.jfree.date.SerialDate
Appendix C: Cross References of Heuristics
Epilogue
Index
Table of Contents