

Clean Code

A Handbook of Agile Software Craftsmanship

The Object Mentors:

Robert C. Martin

Michael C. Feathers Timothy R. Ottinger
Jeffrey J. Langr Brett L. Schuchert
James W. Grenning Kevin Dean Wampler
Object Mentor Inc.

*Writing clean code is what you must do in order to call yourself a professional.
There is no reasonable excuse for doing anything less than your best.*

Technische Universität Darmstadt	
FACHBEREICH INFORMATIK	
BIBLIOTHEK	
Inventar-Nr.:	1711-00074
Sachgebiete:	Software
Standort:	Do 2 / MA



Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Contents

Foreword.....	xix
Introduction	xxv
On the Cover.....	xxix
Chapter 1: Clean Code.....	1
There Will Be Code	2
Bad Code.....	3
The Total Cost of Owning a Mess.....	4
The Grand Redesign in the Sky.....	5
Attitude.....	5
The Primal Conundrum.....	6
The Art of Clean Code?.....	6
What Is Clean Code?.....	7
Schools of Thought.....	12
We Are Authors.....	13
The Boy Scout Rule.....	14
Prequel and Principles.....	15
Conclusion.....	15
Bibliography.....	15
Chapter 2: Meaningful Names.....	17
Introduction.....	17
Use Intention-Revealing Names.....	18
Avoid Disinformation.....	19
Make Meaningful Distinctions.....	20
Use Pronounceable Names.....	21
Use Searchable Names.....	22

Avoid Encodings	23
Hungarian Notation	23
Member Prefixes.....	24
Interfaces and Implementations	24
Avoid Mental Mapping	25
Class Names	25
Method Names	25
Don't Be Cute	26
Pick One Word per Concept	26
Don't Pun	26
Use Solution Domain Names	27
Use Problem Domain Names	27
Add Meaningful Context	27
Don't Add Gratuitous Context	29
Final Words	30
Chapter 3: Functions	31
Small!	34
Blocks and Indenting.....	35
Do One Thing	35
Sections within Functions	36
One Level of Abstraction per Function	36
Reading Code from Top to Bottom: <i>The Steardown Rule</i>	37
Switch Statements	37
Use Descriptive Names	39
Function Arguments	40
Common Monadic Forms.....	41
Flag Arguments	41
Dyadic Functions.....	42
Triads.....	42
Argument Objects.....	43
Argument Lists	43
Verbs and Keywords	43
Have No Side Effects	44
Output Arguments	45
Command Query Separation	45

Prefer Exceptions to Returning Error Codes	46
Extract Try/Catch Blocks	46
Error Handling Is One Thing.....	47
The Error.java Dependency Magnet	47
Don't Repeat Yourself	48
Structured Programming	48
How Do You Write Functions Like This?	49
Conclusion	49
SetupTeardownIncluder	50
Bibliography	52
Chapter 4: Comments	53
Comments Do Not Make Up for Bad Code	55
Explain Yourself in Code	55
Good Comments	55
Legal Comments.....	55
Informative Comments.....	56
Explanation of Intent.....	56
Clarification.....	57
Warning of Consequences.....	58
TODO Comments.....	58
Amplification.....	59
Javadocs in Public APIs.....	59
Bad Comments	59
Mumbling	59
Redundant Comments	60
Misleading Comments.....	63
Mandated Comments.....	63
Journal Comments.....	63
Noise Comments	64
Scary Noise	66
Don't Use a Comment When You Can Use a Function or a Variable.....	67
Position Markers.....	67
Closing Brace Comments.....	67
Attributions and Bylines.....	68

Commented-Out Code.....	68
HTML Comments	69
Nonlocal Information	69
Too Much Information	70
Inobvious Connection.....	70
Function Headers.....	70
Javadocs in Nonpublic Code	71
Example.....	71
Bibliography.....	74
Chapter 5: Formatting	75
The Purpose of Formatting	76
Vertical Formatting	76
The Newspaper Metaphor	77
Vertical Openness Between Concepts	78
Vertical Density	79
Vertical Distance	80
Vertical Ordering.....	84
Horizontal Formatting.....	85
Horizontal Openness and Density.....	86
Horizontal Alignment.....	87
Indentation.....	88
Dummy Scopes.....	90
Team Rules.....	90
Uncle Bob’s Formatting Rules.....	90
Chapter 6: Objects and Data Structures	93
Data Abstraction.....	93
Data/Object Anti-Symmetry	95
The Law of Demeter.....	97
Train Wrecks	98
Hybrids	99
Hiding Structure	99
Data Transfer Objects.....	100
Active Record.....	101
Conclusion.....	101
Bibliography.....	101

Chapter 7: Error Handling	103
Use Exceptions Rather Than Return Codes	104
Write Your Try-Catch-Finally Statement First	105
Use Unchecked Exceptions	106
Provide Context with Exceptions	107
Define Exception Classes in Terms of a Caller's Needs	107
Define the Normal Flow	109
Don't Return Null	110
Don't Pass Null	111
Conclusion	112
Bibliography	112
Chapter 8: Boundaries	113
Using Third-Party Code	114
Exploring and Learning Boundaries	116
Learning log4j	116
Learning Tests Are Better Than Free	118
Using Code That Does Not Yet Exist	118
Clean Boundaries	120
Bibliography	120
Chapter 9: Unit Tests	121
The Three Laws of TDD	122
Keeping Tests Clean	123
Tests Enable the -ilities	124
Clean Tests	124
Domain-Specific Testing Language	127
A Dual Standard	127
One Assert per Test	130
Single Concept per Test	131
F.I.R.S.T.	132
Conclusion	133
Bibliography	133
Chapter 10: Classes	135
Class Organization	136
Encapsulation	136

Classes Should Be Small!	136
The Single Responsibility Principle.....	138
Cohesion.....	140
Maintaining Cohesion Results in Many Small Classes.....	141
Organizing for Change	147
Isolating from Change.....	149
Bibliography	151
Chapter 11: Systems	153
How Would You Build a City?	154
Separate Constructing a System from Using It	154
Separation of Main.....	155
Factories	155
Dependency Injection.....	157
Scaling Up	157
Cross-Cutting Concerns	160
Java Proxies	161
Pure Java AOP Frameworks	163
AspectJ Aspects	166
Test Drive the System Architecture	166
Optimize Decision Making	167
Use Standards Wisely, When They Add <i>Demonstrable Value</i>	168
Systems Need Domain-Specific Languages	168
Conclusion	169
Bibliography	169
Chapter 12: Emergence	171
Getting Clean via Emergent Design	171
Simple Design Rule 1: Runs All the Tests	172
Simple Design Rules 2–4: Refactoring	172
No Duplication	173
Expressive	175
Minimal Classes and Methods	176
Conclusion	176
Bibliography	176
Chapter 13: Concurrency	177
Why Concurrency?	178
Myths and Misconceptions.....	179

Challenges	180
Concurrency Defense Principles	180
Single Responsibility Principle	181
Corollary: Limit the Scope of Data	181
Corollary: Use Copies of Data	181
Corollary: Threads Should Be as Independent as Possible	182
Know Your Library	182
Thread-Safe Collections	182
Know Your Execution Models	183
Producer-Consumer	184
Readers-Writers	184
Dining Philosophers	184
Beware Dependencies Between Synchronized Methods	185
Keep Synchronized Sections Small	185
Writing Correct Shut-Down Code Is Hard	186
Testing Threaded Code	186
Treat Spurious Failures as Candidate Threading Issues	187
Get Your Nonthreaded Code Working First	187
Make Your Threaded Code Pluggable	187
Make Your Threaded Code Tunable	187
Run with More Threads Than Processors	188
Run on Different Platforms	188
Instrument Your Code to Try and Force Failures	188
Hand-Coded	189
Automated	189
Conclusion	190
Bibliography	191
Chapter 14: Successive Refinement	193
Args Implementation	194
How Did I Do This?	200
Args: The Rough Draft	201
So I Stopped	212
On Incrementalism	212
String Arguments	214
Conclusion	250

Chapter 15: JUnit Internals	251
The JUnit Framework	252
Conclusion	265
Chapter 16: Refactoring SerialDate	267
First, Make It Work	268
Then Make It Right	270
Conclusion	284
Bibliography	284
Chapter 17: Smells and Heuristics	285
Comments	286
C1: <i>Inappropriate Information</i>	286
C2: <i>Obsolete Comment</i>	286
C3: <i>Redundant Comment</i>	286
C4: <i>Poorly Written Comment</i>	287
C5: <i>Commented-Out Code</i>	287
Environment	287
E1: <i>Build Requires More Than One Step</i>	287
E2: <i>Tests Require More Than One Step</i>	287
Functions	288
F1: <i>Too Many Arguments</i>	288
F2: <i>Output Arguments</i>	288
F3: <i>Flag Arguments</i>	288
F4: <i>Dead Function</i>	288
General	288
G1: <i>Multiple Languages in One Source File</i>	288
G2: <i>Obvious Behavior Is Unimplemented</i>	288
G3: <i>Incorrect Behavior at the Boundaries</i>	289
G4: <i>Overridden Safeties</i>	289
G5: <i>Duplication</i>	289
G6: <i>Code at Wrong Level of Abstraction</i>	290
G7: <i>Base Classes Depending on Their Derivatives</i>	291
G8: <i>Too Much Information</i>	291
G9: <i>Dead Code</i>	292
G10: <i>Vertical Separation</i>	292
G11: <i>Inconsistency</i>	292
G12: <i>Clutter</i>	293

G13: <i>Artificial Coupling</i>	293
G14: <i>Feature Envy</i>	293
G15: <i>Selector Arguments</i>	294
G16: <i>Obscured Intent</i>	295
G17: <i>Misplaced Responsibility</i>	295
G18: <i>Inappropriate Static</i>	296
G19: <i>Use Explanatory Variables</i>	296
G20: <i>Function Names Should Say What They Do</i>	297
G21: <i>Understand the Algorithm</i>	297
G22: <i>Make Logical Dependencies Physical</i>	298
G23: <i>Prefer Polymorphism to If/Else or Switch/Case</i>	299
G24: <i>Follow Standard Conventions</i>	299
G25: <i>Replace Magic Numbers with Named Constants</i>	300
G26: <i>Be Precise</i>	301
G27: <i>Structure over Convention</i>	301
G28: <i>Encapsulate Conditionals</i>	301
G29: <i>Avoid Negative Conditionals</i>	302
G30: <i>Functions Should Do One Thing</i>	302
G31: <i>Hidden Temporal Couplings</i>	302
G32: <i>Don't Be Arbitrary</i>	303
G33: <i>Encapsulate Boundary Conditions</i>	304
G34: <i>Functions Should Descend Only</i> <i>One Level of Abstraction</i>	304
G35: <i>Keep Configurable Data at High Levels</i>	306
G36: <i>Avoid Transitive Navigation</i>	306
Java	307
J1: <i>Avoid Long Import Lists by Using Wildcards</i>	307
J2: <i>Don't Inherit Constants</i>	307
J3: <i>Constants versus Enums</i>	308
Names	309
N1: <i>Choose Descriptive Names</i>	309
N2: <i>Choose Names at the Appropriate Level of Abstraction</i>	311
N3: <i>Use Standard Nomenclature Where Possible</i>	311
N4: <i>Unambiguous Names</i>	312
N5: <i>Use Long Names for Long Scopes</i>	312
N6: <i>Avoid Encodings</i>	312
N7: <i>Names Should Describe Side-Effects</i>	313

Tests	313
T1: <i>Insufficient Tests</i>	313
T2: <i>Use a Coverage Tool!</i>	313
T3: <i>Don't Skip Trivial Tests</i>	313
T4: <i>An Ignored Test Is a Question about an Ambiguity</i>	313
T5: <i>Test Boundary Conditions</i>	314
T6: <i>Exhaustively Test Near Bugs</i>	314
T7: <i>Patterns of Failure Are Revealing</i>	314
T8: <i>Test Coverage Patterns Can Be Revealing</i>	314
T9: <i>Tests Should Be Fast</i>	314
Conclusion	314
Bibliography	315
Appendix A: Concurrency II	317
Client/Server Example	317
The Server	317
Adding Threading	319
Server Observations	319
Conclusion	321
Possible Paths of Execution	321
Number of Paths	322
Digging Deeper	323
Conclusion	326
Knowing Your Library	326
Executor Framework	326
Nonblocking Solutions	327
Nonthread-Safe Classes	328
Dependencies Between Methods	
Can Break Concurrent Code	329
Tolerate the Failure	330
Client-Based Locking	330
Server-Based Locking	332
Increasing Throughput	333
Single-Thread Calculation of Throughput	334
Multithread Calculation of Throughput	335
Deadlock	335
Mutual Exclusion	336
Lock & Wait	337

No Preemption.....	337
Circular Wait	337
Breaking Mutual Exclusion.....	337
Breaking Lock & Wait.....	338
Breaking Preemption.....	338
Breaking Circular Wait.....	338
Testing Multithreaded Code.....	339
Tool Support for Testing Thread-Based Code	342
Conclusion.....	342
Tutorial: Full Code Examples	343
Client/Server Nonthreaded.....	343
Client/Server Using Threads	346
 Appendix B: org.jfree.date.SerialDate	 349
 Appendix C: Cross References of Heuristics.....	 409
 Epilogue.....	 411
 Index	 413