

Index

A

- “Abandon (by) ship (date)!", 140
- Abstract class type, 21
- Abstract classes, 19, 22, 29
 - and common and variability analysis, 127–130
 - interfaces vs., 262–264
- Abstract Factory pattern, 193–213, 235, 289–290, 348, 401, 404
 - benefits of, 209
 - bridging analysis and design, 200
 - conceptual steps in, 200
 - configuration files, 210
 - consequences, 208
 - context of, 258
 - and contextual design, 404
 - decomposition by responsibility, 209
 - encapsulating variation in, 261–262
 - explanation of pattern name, 207
 - and families of objects, 209–210
 - field notes, 207–211
 - generic structure of, 208
 - how it works, 209
 - implementation, 201, 208
 - intent, 193, 208
 - key features, 208
 - learning, 194–202
 - participants/collaborators, 208
 - problem, 208
 - relating to the CAD/CAM problem, 211
 - roles of objects in, 204
 - solution, 208
 - using Java’s Class class, 210–211
- Abstraction, and switches, 196–197
- Abstraction class, 255, 401, 404
 - acquireReusable* method, 382
- Activity diagrams, 34
- Adapter pattern, 101–113, 120, 235
 - Class Adapter pattern, 109
 - comparing the Facade pattern with, 110–111
 - consequences, 108
 - context of, 260
 - field notes, 107–111
 - generic structure, 108
 - implementation, 108
 - intent, 101, 108
 - key features, 108
 - learning, 102–107
 - Object Adapter pattern, 109
 - participants/collaborators, 108
 - problem, 108
 - relating to the CAD/CAM problem, 111–112
 - solution, 108
 - types of, 109
 - using to help encapsulate variation, 262
- Adapter-Bridge relationship, 237–238
- Adapter-Facade relationship, 238
- AddrVerification class, 319
- Aggregation, 36, 89*fn*, 123, 141, 160, 169, 197
- Agile coding:
 - methods, 115
 - practices, and design patterns, 405–406
 - qualities of, 130–135
- Alexander, Christopher, 76–80, 116, 217–226, 249, 255–256, 404
- Analysis matrix, 279–295
 - applicability of, 292

420 Index

Analysis matrix, *continued*
 field notes, 291–294
 International E-Commerce System case study:
 expanding with new concepts, 285–286
 identify most important features, 282–284
 identifying design patterns by looking at columns, 289–290
 identifying design patterns by looking at rows, 288–289
 incompleteness/inconsistencies, 285
 using columns to identify implementation, 288
 using rows to identify rules, 287
 origins of, 282
 variations, 279–280
 International E-Commerce System case study, 280–291
 Animal class, 124–126
 AnimalMovement object, 125
 ApControl, 196–200, 202, 209
 Application programming interface (API), 69
 Architectural structures, comparison of, 77
 Attribute, 29

B

Base class, 19
 Beck, Kent, 131
begin() method, 387
 Behavioral patterns, 316, 349
 Bridge pattern, 159–192, 235, 401–402, 405
 Adapter pattern and, 184
 classic example of, 183–184
 classification of, 316
 combinatorial explosion, 167–168
 compound design patterns, 184

consequences, 185
 and contextual design, 403
 difficulty of, 160
 encapsulating variation in, 262
 generic structure, 185, 242
 implementation, 160, 180–183, 185
 in C++, 183
 instantiating the objects of, 184
 intent, 159, 185
 key features, 185
 learning, 160–168
 participants/collaborators, 185
 probes, 185
 and problem domain, 402–403
 solution, 185
 Bridge-Facade relationship, 238–239
 Builder pattern, 348

C

CADCAM interface, 274
 CAD/CAM problem, 47–59, 73
 describing, 52–53
 essential challenges/approaches, 55–58
 requirements, 229–230
 review of, 229–230
 solving with commonality and variability analysis (CVA), 270–276
 solving with patterns, 229–250
 vocabulary, 50–52
 cutout, 51
 dataset or model, 52
 features, 50
 geometry, 51
 hole, 51
 irregular, 51
 NC machine and NC set, 52
 part, 51
 slot, 51
 special, 51
 CAD/CAM system:
 CAD/CAM, defined, 49*fn*
 extracting information from, 49–50

- versions, 55
 - CalcTax class, 298, 361
 - Canonical form, 242
 - Cardinality, 41–42
 - Case diagrams, 34
 - Changing requirements, 7, 28
 - dealing with, using functional decomposition, 8–10
 - Child classes, 22
 - Circle class, 163–165
 - display* method, 24
 - Class, 22, 29
 - defined, 17–18
 - Class Adapter pattern, 109
 - Class diagrams, 34, 35–42
 - aggregation, 39–40
 - cardinality, 41–42
 - composition, 40
 - and uses relationship, 40
 - has-a relationship, 39
 - high-level, 56–57
 - is-a relationship, 38–39
 - and relationships between classes, 40–41
 - variations, 36
 - Class notation, objects, 45
 - Client, 379, 382
 - Code Complete: A Practical Handbook of Software Construction* (McConnell), 9
 - Code fragments, and design, 63
 - Code, quality of, 141
 - Cohesion, 68, 195–196, 351
 - Cohesive code, 132, 134
 - Collaborators, patterns, 83
 - Collection class, *display* method, 24
 - Combinatorial explosion, 197
 - Commonality and Variability Analysis
 - Table, 271, 293–294
 - translating into classes, 273
 - Commonality and variability analysis (CVA), 231–232, 256, 269–277, 354
 - and abstract classes, 127–130
 - and application design, 269–270
 - defined, 128
 - and design patterns, 401–402
 - solving the CAD/CAM problem with, 270–276
 - Communication, at multiple levels, 14
 - Complexification, 221–222, 256
 - Complexity, and functional decomposition, 4
 - Component, 300–301
 - Composite pattern, 401
 - Composition, 36, 141*fn*, 247
 - Conceptual level, software development process, 14–16
 - Concrete classes, 19, 22
 - ConcreteComponent, 300–301, 306, 403
 - ConcreteStrategy, 401
 - CONNECT command, 333
 - Consequences/forces, design patterns, 82–83
 - Constructors, 27, 29
 - Context, 234–236
 - rule to use when considering, 236
 - Context first rule, 224
 - Contextual design, 403–404
 - Control program, 13
 - Coplien, Jim, 116, 127
 - Coupling, 9, 195–196, 351
 - tight, 68, 195–196
 - Courtyard pattern, 79, 223
 - Creational patterns, 316
 - Cultural anthropology, 76
 - Currency class, 126–127
 - Customer class, 318–319
 - Cutout features, 247–248
- D**
- Data Access Patterns* (Nock), 383
 - Data elements, 18
 - Data hiding, 1, 21, 22, 119
 - Decomposition by responsibility, 209
 - Decorator, 300–302, 305, 403
 - Decorator chain, 300

422 Index

- Decorator pattern, 297–311, 355, 401
 - applying to the case study, 301–305
 - as chain of objects, 300–301
 - classification of, 316
 - consequences, 308
 - and contextual design, 403
 - decomposing by responsibilities, 304–305
 - essence of, 309–310
 - field notes, 307
 - generic structure of, 308
 - how it works, 300
 - implementation, 308, 309–310
 - intent, 308
 - key features, 308
 - participants/collaborators, 308
 - problem, 308
 - and problem domain, 403
 - solution, 308
 - stream I/O, 305–307
 - using, 307–308
 - Decoupling patterns, 317, 351
 - Delegation, 5
 - Dependency inversion principle, 222, 254, 256
 - Dependency relationship, 40
 - Dependents, 319
 - Deployment diagrams, 34
 - Derived classes, 22, 29
 - Design, 217–227
 - adding preformed parts, 219
 - building by adding distinctions, 217–225
 - building by fitting things together, 218–219
 - complexification, 221–222
 - keeping the big picture in mind, 220
 - modularity, 220
 - process, 221–223
 - Design decisions, making, 259
 - Design patterns, 73–92
 - and complex problems, 80
 - consequences/forces, 82–83
 - defined, 78
 - description, components required of, 80
 - key features of, 83
 - learning alternatives to large inheritance hierarchies, 89
 - software, 81–82
 - sources of, 76–80
 - studying, 83–89, 253
 - Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma/Helm/Johnson/Vlissides), 81–82, 123
 - Designing from context, 253–261, 276
 - “Design-with-change-in-mind” approach
 - case study, 141–142
 - Destructors, 27, 29
 - Display class, 24
 - display* method:
 - Collection class, 24
 - Shape class, 24
 - Square method, 24
 - DisplayDriver, 198
 - doQuery* method, 333–334
 - Double-checked locking:
 - defined, 365–366
 - features of, 365–366
 - Double-Checked Locking pattern, 97, 364–369
 - defined, 359
 - field notes, 368
 - and Java, 366–367
 - draw* method, `VIRectangle`, 165
 - draw_a_line* method, 162
 - drawCircle* method, 164–165
 - Display class, 24
 - drawline* method, 165–166
 - Display class, 24
- E**
- Electronic magazine, 410
 - Encapsulated code, 134
 - Encapsulated variation, Bridge pattern as example of, 261

Encapsulating the system, 98
 Encapsulating variation principle, 254, 261–262
 Encapsulation, 1, 21, 22, 26, 29, 119–126, 400
 of data, 121
 how to think about, 120
 of methods, 121
 of objects, 121
 traditional view vs. new view, 119–123
 of type, 121
 types of, 120–121
end() method, 387
 ESPRIT, 81*fn*
Extract method, 339
 eXtreme programming (XP), 115, 341
 YAGNI, 378

F

Facade pattern, 93–100, 235
 context of, 260
 defined, 95
 encapsulating variation in, 262
 field notes, 97–98
 general approach, setting of, 98
 generic structure of, 96
 intent, 93
 key features, 96
 learning, 94–95
 Relating to the CAD/CAM Problem, 99
 Facade-Adapter relationship, 238
 Factories, 345–396
 defined, 348
 encapsulation of business rules for creating objects, 347
 field notes, 355–356
 and integration, 354
 parallels in XP practices and, 395
 roles of, 355
 summary of, 393–396
 uses of, 381–382
 vectors of change, limiting, 353–354

Factory Method pattern, 348, 385–391
 consequences, 389
 defined, 386
 field notes, 388–390
 generic structure, 389
 implementation, 389
 intent, 386, 389
 and object-oriented languages, 387
 participants/collaborators, 389
 problem, 389
 QueryTemplate class, 385–387
 requirements for the case study, 385–386
 solution, 389
 Factory patterns, 349
 Feature class, 240–241, 243, 247–248, 273
 Finalizers, 27
 Fowler, Martin, 133
 Functional decomposition, 3–5, 28, 29
 problems with, 4–5

G

Gang of Four, 81–82, 89, 93, 101, 116, 121, 123, 141, 152, 159, 193, 256, 315, 319, 332
 categories of patterns, 315–317
 behavioral patterns, 316
 creational patterns, 316
 decoupling patterns, 317
 structural patterns, 316
 and conceptual motivations of patterns, 348–349
 General student, defined, 17–18
 Generalized classes, 121
 Generic structure:
 of Abstract Factory pattern, 208
 of Adapter pattern, 108
 of Bridge pattern, 185, 242
 of Decorator pattern, 308
 of Facade pattern, 96
 of Factory Method pattern, 389

424 Index

Generic structure, *continued*
 of Object Pool pattern, 383
 of Observer pattern, 326
 patterns, 83
 of Singleton pattern, 363
 of Strategy pattern, 154
 of Template Method pattern, 342
 getCollection(), ShapeDataBase class,
 24
 getEdgeType, 243
 getEnumerator, 387
 getLength, 243, 247
 getLocation, 247
 getX, 243
 Shape class, 24
 getY, 243
 Shape class, 24
 God objects, 9
 gotoNextClassroom() method, 17
 Grand, Mark, 383

H

Has-a relationship, 35, 247
 Hiding data, 1, 21, 22, 119
 High-level class diagram, 56–57
 HighResFactory, 202–203
 High-resolution family, 194
 High-resolution print driver (HRPD), 194
 Hole features, 247–248

I

IEnumerable interface, 387
 ImpFeature, 243
 Implementation:
 hiding, 120
 patterns, 83
 Implementation class, 255
 Implementation interface, 401
 Implementation level, software develop-
 ment process, 14–16

Individual learning, and design pattern
 study, 88
 Inheritance, 19, 22, 29, 141, 197, 247,
 400
 and classification of variations in be-
 haviors, 123
 designing with, 164
 overuse of, 169
 proper use of, 162
 Instance, 18, 23, 30
 Instantiation, 18, 23, 30
 Intent, patterns, 83
 Intention-revealing name, 133
 Interaction diagrams, 34, 42–45
 sequence diagram, 42–44
 Interfaces, 19*fn*, 23
 International E-Commerce System case
 study, 142–154, 156
 analysis matrix:
 expanding with new concepts,
 285–286
 filling out, 283–286
 identify most important features,
 282–284
 identifying design patterns by
 looking at columns, 289–290
 identifying design patterns by
 looking at rows, 288–289
 incompleteness/inconsistencies,
 285
 using columns to identify imple-
 mentation, 288
 using rows to identify rules, 287
 customers, 286
 expanding with new concepts,
 285–286
 high-level application design, 290
 identify most important features,
 282–284
 incompleteness/inconsistencies, 285
 organizing features in a matrix,
 282–283
 requirements, 317–318
 handling, 143–152

Singleton pattern, applying to the case study, 361–363
 using columns to identify implementation, 288
 using rows to identify rules, 287
 variations, handling, 280–291

Irregular features, 247–248

Is-a relationship, 19, 35, 247

iterator method, 387

Iterator pattern, 401

J

Jeffries, Ron, 132–133

K

Knowledge Analysis and Design Support (KADS), 81*fn*

L

Liskov, Barbara, 257

Loosely coupled code, 134

LowResFactory, 202–203

Low-resolution display driver (LRDD), 194

Low-resolution family, 194

M

Maguire, Steve, 378

Maintainability of code, and design pattern study, 88–89

McConnell, Steve, 9

Member, 30

Meta-level, 87–88

Methods, 15, 30

Model class, 240–241, 246

Models of programs, 33

Modifiability, and design pattern study, 88–89

Modularity, 220

Money class, 126–127

Multithreaded application:
 and Double-Checked Locking pattern, 364
 and Singleton pattern, 359*fn*

N

Name, patterns, 83

“New” perspective, 116

Nock, Clifton, 383

Numerically controlled (NC) machine, 52

O

Object Pool pattern, 371–384
 consequences, 382
 defined, 381
 generic structure of, 383
 implementation, 383
 intent, 382
 key features, 382
 participants/collaborators, 382
 problem, 382
 problem requiring management of objects, 372–383
 client code, 377–378
 data members, 376–377
 duel checking, 380–381
 getInstanceOfPort(), 377
 methods, 377
 returnInstanceOfPort(), 377
 reference, 383
 solution, 382
 uses of, 381

Object wrappers, 110

Object-oriented design, limitations of, 47–59

Object-oriented geometry (OOG), 58

426 Index

- Object-oriented languages, 126, 309, 387
 - Object-oriented paradigm, 3–32
 - defined, 15
 - origin of, 3–32
 - Object-oriented principles, summary of, 400
 - Object-oriented programming, 3, 23–26, 347–348
 - Object-Oriented Software Construction* (Meyers), 16
 - Object-oriented solution, 61–71
 - solving with special cases, 61–70
 - Objects, 1, 29, 30, 117–118, 400
 - class notation, 45
 - common problems in creating, 355–356
 - creation/management, 352–353
 - as instances of classes, 18
 - responsibilities of, 15
 - sending messages, 43
 - traditional view vs. new view, 117–118
 - Observable class, 325
 - Observer pattern, 315–329
 - applying to the case study, 319–325
 - consequences, 326
 - field notes, 325–327
 - generic structure, 326
 - implementation, 326
 - with the Adapter pattern, 324
 - intent, 319, 326
 - key features, 326
 - participants/collaborators, 326
 - problem, 326
 - solution, 326
 - using, 325–327
 - Once and Only Once rule, 131–132, 400
 - One at a time rule, 223
 - OOGFeature objects, 111–112, 247–248
 - OOGHole, 248
 - OOGSlot, 247
 - Open-closed principle (OCP), 253, 254
 - OracleQT, 334
 - Overanalysis/overdesign, 140
- P**
- Paired programming, 35*fn*
 - “Paralysis by analysis”, 140
 - Participants, patterns, 83
 - Pattern Language, A* (Alexander), 260
 - Pattern languages, 225
 - Pattern-based analysis, 264
 - Patterns, 116, *See also* Design patterns
 - and agile coding practices, 405–406
 - benefits of, 244, 262
 - and commonality and variability analysis (CVA), 401–402
 - and contextual design, 403–404
 - field notes, 406–407
 - Gang of Four categories, 315–316
 - as multidimensional descriptions, 405
 - parts of (diagram), 405
 - and refactoring, 340
 - relationships within, 404–405
 - solving CAD/CAM problem with, 229–250
 - studying, 253
 - thinking in, 215, 231–247
 - identify the patterns, 233
 - process, 232
 - work through the patterns by context, 233–239
 - Patterns in Java, Volume 1* (Grand), 383
 - Pentagon class, 122–123
 - PentagonSpecialBorder, 122
 - Perspectives, 23
 - Polymorphism, 21–23, 30, 56, 103
 - Port, 376–382
 - PortManager, 376, 379–380
 - Principles and strategies, 253–267, 403
 - abstract classes vs. interfaces, 262–264
 - dependency inversion principle, 254
 - designing from context, 253–261
 - encapsulating variation principle, 254, 261–262
 - open-closed principle (OCP), 253
 - principle of healthy skepticism, 264–265

PrintDriver, 198
 Private accessibility, 21, 37
 Problem domain, decomposing into responsibilities, 402–403
 Programming by intention, 133, 400
 Protected accessibility, 21, 37
 Prototype pattern, 348
 Proxy pattern, 401
 Public accessibility, 21, 37
 Public interface, 21
 Publish-Subscribe, 319

Q

QueryTemplate class, 333–334, 385–387

R

Readability, 131–133
 Readable code, 134
 Rectangle class, 163–164
 Redundancy, 68, 131
 Redundant code, 134
 Refactoring, 340
 Requirements:
 and change, 7
 dealing with, 8–10
 problem of, 6–7
 ResFactory, 199–200, 202
 Responsibilities, reorganization of, 12–13
 ReusablePool, 382

S

SalesOrder, 297–299
 SalesOrder class, 349, 355, 362
 SalesReceipt class, 127
 SalesTicket object, 297–299
 Scaling systems, 395–396
 SELECT command, 333

Seniormost patterns, 237
 Sequence diagram, 34
 reading, 167
 Shape class, 24, 163–164
 ShapeDataBase class, 24
 Shift of responsibility, 12
 Singleton pattern, 97, 348, 359–363
 applying to the case study, 361–363
 collaborating objects, 360
 consequences, 363
 defined, 359
 field notes, 368
 generic structure of, 363
 how it works, 360
 implementation, 363
 instantiating Stateless Facades with a singleton, 368
 intent, 360, 363
 key features, 363
 participants/collaborators, 363
 problem, 363
 purpose of, 361
 solution, 363
 Stateful Singletons, 368
 synchronizing the creation of, 365
 Slot features, 247
 SlotFeature class, 61, 246–247
 Software design, applying Alexander's approach to, 224–225
 Software development process:
 perspectives in, 14
 steps in, 393–394
 Solution, patterns, 83
someMethod method, 339
 Source and behavior objects, 307–308
 Special features, 247–248
 Special object methods, 27
 Specialized classes, 22, 121
 Specification level, software development process, 14–16
 Square class, display method, 24
 Standard object-oriented solution, 61–71
 State diagrams, 34
 State pattern, 401

428 Index

Strategy pattern, 139–157, 401–402, 405
 ConcreteStrategies, 153
 consequences, 153
 Context, 153–154
 coupling between context and strategies, 155
 eliminating class explosions with, 156
 encapsulating business rules, 155
 field notes, 154–156
 generic structure, 154
 handling new requirements, 139–142
 common excuses, 140
 implementation, 154
 implementing, 289–290
 intent, 152–153
 key features, 153
 motivations of, 153
 new cases and normalization, handling, 152
 participants/collaborators, 153
 problem, 153
 solution, 153
 Strategy, 153–154
 and unit testing, 155

Structural patterns, 316, 349

Student object, 17

Subclasses, 19, 22

Sub-matrices, 293

Superclass, 19, 30

Swapping out systems, 98

System usage, tracking, 98

T

Tax class, 361–362

Tax object, 349

Team communications, and design pattern study, 88

Template Method pattern, 331–343, 385, 401
 consequences, 342
doquery method, 386
 and elimination of duplication, 338–339

field notes, 340–341
 generic structure, 342
 implementation, 342
 intent, 332, 342

International E-Commerce case study, 331–332
 applying Template Method to, 333–334
 participants/collaborators, 342
 problem, 342
 solution, 342
 using, 340–341
 using to reduce redundancy, 334–340

Testability, 131, 133–134

Testable code, 134

Test-driven development, 115, 135

Tight coupling, 68, 195–196

Timeless Way of Building, The (Alexander), 78–80, 218, 226, 404

Tracking system usage, 98

Type encapsulation, 123

U

UML Distilled (Fowler), 13

Unified Modeling Language (UML), 30
 defined, 33–34
 notation for access, 37
 notation for relationships, 38
 reasons for using, 34–35
 UML diagrams and their purpose, 34

Universal context for software development, 232–249, 349–351

Unwanted side effect, 9–10

Uses relationship, 40

Uses-a relationship, 35

USTax class, 361–362, 366

V

VI features, instantiating, 64–65

VI methods, implementation of, 65

VIFacade, 244–245, 247
VIHoles, 99
VIImp, 244–247
VIRectangle implementation, 162
V1Slots, 62–63, 99
V1System, 99
V2 features, instantiating, 65–66
V2 methods, implementation of, 67
V2Rectangle implementation, 162
V2Slots, 63
Variability analysis, and abstract classes,
127–130
Variations, handling, 280–291
Visibility, 21
Visitor pattern, 401

W

Weak cohesion, 68, 195–196
Web site companion, 410
WelcomeLetter class, 319
Wrappers, 110
Writing Solid Code (Maguire), 378

X

XP. *See* eXtreme programming (XP)

Y

YAGNI, 378



















