Index

Numbers

3DBoard, solving without inheritance 405

A

abstract base class 200–203, 212
abstracting behavior 205
abstraction 243
actors 294, 299, 300, 302
aggregation 208, 412–417, 414, 558
versus composition 413
Airplane class 577
alternate paths 70, 85–87
dog door 123
dog door requirement changes 120
questions and answers 125
analysis xiii–xxii, 145–196, 283
identifying problems 148
planning solutions 149
textual 169
use cases and 151
Analysis and Design Puzzle 510, 536–538
anonymous class 82
anti patterns 563
architecturally significant features 331
architecture
defined 326–328
purpose of 351
three questions 332
Architecture Puzzle 340
revisited 345
solution 346–347
array of arrays 347
association 194
attributes 194

B

Bark object 158, 159
multiple 165
BarkRecognizer.java 131, 132, 135
Don’t Repeat Yourself Principle 382
BarkRecognizer class 132, 135, 159
debate 161
questions and answers 132
recognize() method 160, 189
base classes, modifying 381
behavior
of subclasses 241
reusing from other classes 414
Be the Browser 333, 334, 365, 437, 438
big problems
breaking into smaller pieces 281, 309, 310–314
looking at 281
versus little problems 280
Board class 346, 348, 349, 364
boolean property versus numStrings property 39
Bullet Points 106, 142, 191, 320, 372, 417, 475, 586

C

cancel() method (TimerTask anonymous class) 82
case diagrams, versus use cases 77
change 228
constant in software development 115
changing requirements 115
checked exception 470
class diagrams 19, 194
   abstract classes 206
   annotating guitar application 39
dissected 182–183
Guitar 29
limitations 187
subclasses 206
UML and 577
what’s missing 188, 190
classes, protecting 523
clear value 74, 75, 98, 100
closed for modification 377–381, 417, 418
code, robust 18
coding standards 572
cohesion 269–276
collections 251
commonality 355–361, 367, 441–450
   commonality-focused solution 450
   emphasizing 446–447
commonality analysis 361, 367
common behavior 205
composition 408–417, 414, 558
   assembling behaviors from other classes 408
   ownership and 411
   versus aggregation 413
   when to use 409
Connection class 514, 517
constant in software analysis & design 115
context 147
contract, programming by 461–463
coordinating movement 337, 366, 367
CRC cards 564–565
customer’s language 306
customer goal 72
customers
   listening to 63
   satisfying 12
   understanding customer’s requirements 65
dead value 74, 75, 98, 100
defensive programming 464–465
delimitation 43, 45, 162, 414, 504
3DBoard 406
BarkRecognizer class 160
DogDoor class (Sam’s) 160
versus inheritance 407
when to use 407
design
   death of a 246
   preliminary 485
design lifecycle 273
design patterns 8, 10, 22, 34, 50, 52, 313, 315, 316, 319, 563
design principles 376, 377, 380, 400, 415
   writing your own 141
Dessert interface 230
DessertService class 230
dog door
   alternate paths 123
   automatically closing 82
   barking focus 172
   coding changes in 131
   final test drive 140
   main path 123
   planning for things going wrong 68–71
   single scenario 124
   Todd and Gina’s Dog Door 387, 389
   updating after analysis 152
   where code should go for closing door 138
DogDoor.java 57, 59, 131, 135, 139, 140
   Don’t Repeat Yourself Principle 383
   gathering requirements 61
test drive 59
   second 83

592  index
DogDoor class 59, 135
  allowedBarks attribute 183
  updating 139
DogDoor class (Sam’s) 158
  delegation 160
Dog Door Class Diagram (Maria’s) 180
DogDoorSimulator.java 59, 83, 88–90, 131, 134, 135, 192
  code magnets 88, 90
  test drive 89
DogDoorSimulator class 59, 135
  updating 134
domain analysis 306, 313, 317, 320, 485
Don’t Repeat Yourself Principle (see DRY) 382
double encapsulation 249
DRY (Don’t Repeat Yourself Principle) 382–388
  main idea 384

E
encapsulate what varies 115–116
encapsulation 28, 34, 52, 226, 231, 243, 245, 282, 504, 582, 585
  Bullet Points 50
double 249
  emphasizing 448–449
  guitar specifications 41
  OCP and 381
  questions and answers 31
enumerated types 16
enums 16–18
equals()
  Java’s default 521
  overriding 516
  RouteFinder 521
essence of a system 332, 335
external initiator 74, 75, 93, 94, 98, 100

F
feature analysis 434–435
feature driven development 427, 428, 433, 475, 478
  versus use case driven development 430–432
feature lists 484, 495
Feature Magnets
  RouteFinder feature list 497
Feature Puzzle 474
  solutions 476–477
features 290
  architecturally significant 331
  interpreting 366
  mapping to use case diagrams 298, 300
  versus requirements 292
  versus use cases 499, 500
Final CATASTROPHE! 230
FindGuitar.java
  updating 42
FindGuitar class 33
FindInstrument.java 259, 260
  initializeInventory() method 261
test driving 261, 264
flexibility 34, 49, 52, 380
  interfaces and 224
  versus functionality 535
fragile 18
Friedman, Dan 301
functionality 34, 52, 324, 329, 330, 342, 345, 364, 371, 374, 499
  breaking applications into smaller chunks of 473
  versus flexibility 535

G
game-specific units 352–353
game system framework (GSF) 284
Gary’s Games
customer conversations 288–289
features 290
  vision statement 284
Gary’s Game System Framework
  coordinating game movement 365
  feature list 305, 324, 329
    determining significance 333, 334
    key features 336, 350, 363, 371
    risk 338
getNumStrings() method (GuitarSpec.java) 42, 44
getHashCode() method (GuitarSpec.java) 516
Guitar.java
  adding class for mandolins 208
  updating 252
GuitarSpec.java
  adding new property 44
  getNumStrings() method 42, 44
  numStrings property 42, 44
GuitarSpec class 29, 210
  adding property to 40
  encapsulating properties of 45
  questions and answers 31

H
HAS-A 558–559
hashCode(), overriding 516
Head First Labs website 17
Head First learning principles xxvi

I
implementation 485
inheritance 243, 380, 558, 579
  alternatives 405, 414
  hidden structure problems 402
misusing 401
  versus delegation 407
  (see also LSP)
instrument-specific subclasses
  killing 247
Instrument.java 208
Instrument abstract base class 201
instrument application
  ease-of-change challenge 266–269
Instrument class
  adding new subclasses 218
  completed class diagram 201
  extending 208
  updated 252
InstrumentSpec 203–220
  creating abstract class for 209
  making concrete class 237–238
  matches() method 378
InstrumentSpec.java 209
InstrumentSpec class
  updated 252
InstrumentType.java 256
InstrumentType enum 256
interface 224, 226, 230, 232
  coding to an 282
Inventory.java 212, 256
  making sure is well designed 37
  search() method 37, 42, 45, 237
Inventory class 32
  addInstrument() method 241
  updating 256
  updating to work with multiple instrument types 212
Inventory Roundup 260
  solutions 262–263
IS-A 558–559
iterating over all choices (guitar application) 23
iterative design 503, 530, 539
iterative development 475, 485
J
Java’s default equals() 521

L
Line class 514
Liskov Substitution Principle (see LSP) 141
LoadTester class 529
loosely coupled 162
LSP (Liskov Substitution Principle) 400–405, 412–418
hidden structure problems 402
violating 404

M
main path 92, 99, 101
dog door 123
Mandolin.java 208
MandolinSpec.java 211
MandolinSpec class 200, 202, 205, 211
Map for storing properties 253, 254
Method Magnets 525
solutions 526
metrics 566
mismatched object type 27
Model-View-Controller pattern (see MVC pattern) 48, 50
multiplicity 165, 183, 194
MVC pattern 314

N
noun analysis 194
nouns in use cases 167, 175
numStrings property (GuitarSpec.java) 42, 44
numStrings property versus boolean property 39

O
Object-Oriented Analysis & Design (see OOA&D) 48, 49, 50
Object Oriented Design Principles 141
objects
concepts 27
forwarding operation to another object 43
naming 27
ObjectvilleSubway.txt file 527
Objectville Subway Map 490–491
OCP (Open-Closed Principle) 49, 377–381, 400, 417, 418
step-by-step 379
OOA&D (Object-Oriented Analysis & Design) 48, 49, 50
power of 318–319
process overview 484
OOA&D Cross 51, 107, 143, 195, 277, 321, 373, 419, 479, 552
OOA&D Magnets 487, 549
solutions 550
OOA&D Toolbox 276, 320, 418, 478
more tools for 142
tools 106
OO CATASTROPHE! 223–232
OO Cross 587
OO Principles 232
Open-Closed Principle (see OCP) 48, 49, 50
open for extension 377, 378, 381, 417, 418
operation 194
optional paths 120

P
Painter class 226
Patterns Exposed 92, 126, 341, 343
placeholders 201
planning for things going wrong 68–71
polymorphism 243, 581
pressButton() xii, 136, 139
programming
  by contract 461–463, 471
  versus defensive programming 466–468
defensively 464–465, 471
properties
  storing 250, 251, 253
  that vary across objects 251
  unused 27
protecting classes 523

R

readable code 572–573
refactoring 574
relationships 326, 331, 343, 351, 353
Remote.java 59, 135, 140
  Don’t Repeat Yourself Principle 382
Remote class 58, 59, 135
  automatically closing door 82
code magnet solutions 108
remote control
  simplifying 139
requirements 93, 94, 484
  changes xii–xxii, 111–144
  changes in use cases 129
  checking against use cases 78
  creating list of 64, 66
defined 62
gathering xi–xxii, 55–110
good 67, 142, 282
list 81
  list, checking 133
  list for changes to dog door 130
  questions and answers 67
  versus features 292
requirements list 286
reusable 49
reusable design 36

Rick’s Search Tool 3–54
  download 33
capsulation 28
iterating over all choices 23
looking deeper for problems 25
making well-designed and flexible 234–235
mismatched object type 27
problems with 6
  revisiting problems 14
  String comparisons 16
search() method 21, 22
  analyzing 26
test drive 23, 46
risk 338, 339
  reducing 349, 362, 371
  use cases and 341
robust code 18
RouteFinder 488–556
  back to requirements phase 533
  breaking into modules 501, 502
  closer look at representing a subway 505
coding
  Station class 516
Connection class 514, 517
different output sources 551
feature list 532
getDirections() method 544
iterative design 503, 530, 539
Line class 514
LoadTester class 529
more extensible loading 551
statement of work 488
Station class 514
Subway class 514
coding 519
  equals() 520
  hashCode() 520
subway file 506
SubwayLoader class 524
SubwayPrinter class 545
SubwayTester class 547
textual analysis 510–513
RuntimeException 470
satisfying customers 12
scenarios 73
  Board interface 342, 344
  questions and answers 125
  risk and 343
  single, for dog door 124
search() method (guitar application) 21, 22
  analyzing 26
search() method (Inventory.java) 37, 42, 45
seeing the big picture 294–295
sequence diagrams 567
Sharpen your pencil
  adding support for mandolins to search tool 199
    answers 200
  analysis 149
    answers 150
  annotating guitar application class diagram 39, 40
  breaking RouteFinder into modules 502
Connection class 517
  answers 518
  designing test cases 455
    answers 456
  design principle 141
Dog class, not including 174
  answers 179
DogDoor class
  allowedBarks attribute 183, 185
dog door requirements 65
drawing class diagram for updated instrument application 257
game-specific units 353
  answers 354
Gary’s Games, things to do first 285
Gary’s Games features 291, 303
  answers 290, 304
Gary’s Game System Framework
  most important features 330
great software 11
guitar application 7, 19
GuitarSpec object 29, 30
instrument-specific subclasses 239, 240
Inventory.java 37
letting GuitarSpec handle comparisons 53
OCP, finding in your project 381
OO principles 243
  answers 244
potential Unit properties 442–443
  answers 444–445
power of use cases 104
reasons requirements might change 115
requirements for dog door 79
  answers 80
risk 339
rodents getting into kitchen 60
RouteFinder feature list 493
  answers 494
scenarios in use cases 127
  answers 128
scenario testing 135
  answers 136
skeleton for Airplane class 577
  answers 578
SRP, applying to automobile class 393
  answers 394
structure for RouteFinder code 496
  answers 498
Unit class programming contract 469
  answers 470
use case for Get directions 533
use case for loading subway lines 503, 507
  answers 504, 508
use case nouns 167
  answers 168
use cases 95
  real power of 102–103
  real power of, solution 104–105
  writing more 96–97
use case to store a bark 153
  answers 153
value of encapsulating your data 583
  answers 584
writing code based on class diagram 157
  answers 158
software
  good enough 274–275
  great (see great software)
  well-designed 217
software, great 283
software release 485
solutions, questions and answers 151
SRP (Single Responsibility Principle) 49, 390–399, 504
  analysis 392
  going from multiple to single 395
SRP Analysis
  CRC cards and 565
SRP Sightings 396–397
  answers 398–399
start condition 93, 94
statechart diagram 568–569
state machine diagrams 568–569
Station class 514
  coding 516
String comparisons 16
Style.java 211
subclasses
  adding new instruments 218
  behavior of 241
  behavior versus properties 242
  instrument-specific
    killing 247
    misusing 401
    standing in for superclass 581
    versus abstract base classes 201
subtype must be substitutable for its base type 403
Subway.java
  implementation 540–542
Subway class 514
  coding 519
  equals() 520
  hashCode() 520
SubwayLoader class 524
  making sure correctly loads in all stations and
  connections 527
SubwayPrinter class 545–546
SubwayTester class 547
system boundary 296

T

terrain types 309
test cases 454, 477, 571
test driven development 440
Test Puzzle 457, 527
  solution 528–530
  solutions 458–459
test scenarios
  Unit class 437–439
textual analysis 169, 173, 174, 189, 510
  code magnets 177
    solution 178
The Little Lisper 301
There are no Dumb Questions
  advanced OO techniques 222
    aggregation 415
    alternate paths 125
    array of arrays 347
    BarkRecognizer class 132
    class diagrams 19, 572
    cohesion 271
    commonality 445
    commonality versus variability 369
    composition 415
    deciding between design choices 451
    defensive programming 465
    delegation 43, 415
    DessertService class 230
    Don’t Repeat Yourself Principle 385
    encapsulation 31, 34, 585
    enumerated types 16
    equals() 521
    essence of a system 335
features versus requirements 293, 494
fewer classes and flexibility 255
FindInstrument.java, getting different results 264
hasConnection() method 528
holding off writing code 364
inheritance 415
Instrument class 213
making Instrument abstract 203
matching up every feature to a use case 500
multiple solutions 151
nouns and verbs in use cases 179
OCP 381
options 550
polymorphism 581
programming by contract 471
reducing risk 362
repeated steps in use case 515
requirements 67
RuntimeException 470
scenarios 125
Single Responsibility Principle 391
SRP analysis 395
steps in coding 22
subclassing 415
test cases 456
test driven development 440
textual context 173
Tile and Unit classes 349
Timer class 82
UML 207
unit testing 570
use case diagrams 299
use cases 77
use cases for storing a bark 154
uses cases and risk 345
validation 508
Thread class, using to close door 82
Tile.java 348
Tile class 348, 349
tiles 309
time periods 309
Timer class
using to close door 82
TimerTask anonymous class
cancel() method 82
UML and class diagrams 577
UML Cheat Sheet 207
UML diagrams 181, 184
Unit.java 348
Unit class 348, 349, 435–437
class diagram 441
matching tests to design 452
programming by contract 462–463
test scenarios 437–439
writing 453
UnitGroup class 476–477
units 309
finding what is common between 355
game-specific (see game-specific units)
moving 472
unit testing 570–571
upgradable 49
use case diagrams 296–297, 305, 484, 495
mapping features to 298
use case driven development 427, 429, 475, 478
versus driven development 430–432
use cases 71–86, 286
analysis and 151
changes and requirements 129
checking against requirements 78
clear value 74
definite starting and stopping point 74
exposed 92, 126
external initiator 74
formats 560–562
Get directions 533
answers 534
identifying classes to use 170
use cases (continued)
  loading subway lines 507
  checking flow 509
Load network of subway lines 527
magnets 75, 98–99
  solution 76, 100
nouns 167, 175
questions and answers 77
real power of 102
risk and 341
seeing the big picture 294
storing a bark 153
three parts 74
updating after analysis 152
verbs 176
versus case diagrams 77
versus features 499
writing so not confusing 122–123

V
validation 508
variability 356, 358, 360, 442, 444, 445
verb analysis 194
verbs in use cases 176
vision statement 284, 286

W
Weapon interface 408–410
well-designed software 217
Who Am I? 416, 468
  solutions 420, 481