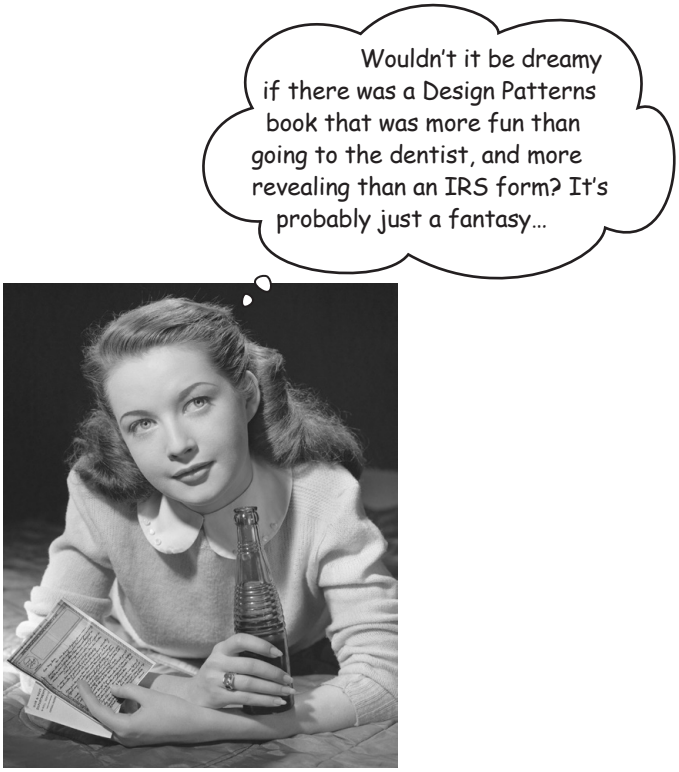
Head First
Design Patterns



Eric Freeman
Elisabeth Robson

O'REILLY®

Beijing • Boston • Farnham • Sebastopol • Tokyo

Table of Contents (summary)

	Intro	xxv
1	Welcome to Design Patterns: *intro to Design Patterns*	1
2	Keeping your Objects in the Know: *the Observer Pattern*	37
3	Decorating Objects: *the Decorator Pattern*	79
4	Baking with OO Goodness: *the Factory Pattern*	109
5	One-of-a-Kind Objects: *the Singleton Pattern*	169
6	Encapsulating Invocation: *the Command Pattern*	191
7	Being Adaptive: *the Adapter and Facade Patterns*	237
8	Encapsulating Algorithms: *theTemplate Method Pattern*	277
9	Well-Managed Collections: *the Iterator and Composite Patterns*	317
10	The State of Things: *the State Pattern*	381
11	Controlling Object Access: *the Proxy Pattern*	425
12	Patterns of Patterns: *compound patterns*	493
13	Patterns in the Real World: *better living with patterns*	563
14	Appendix: *Leftover Patterns*	597

Table of Contents (the real thing)

Intro

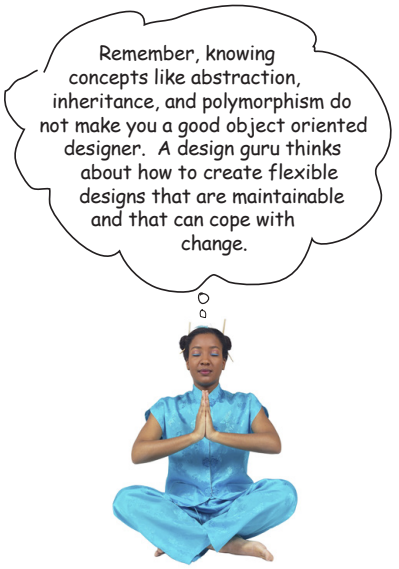
Your brain on Design Patterns. Here *you* are trying to *learn* something, while here your *brain* is doing you a favor by making sure the learning doesn't *stick*. Your brain's thinking, "Better leave room for more important things, like which wild animals to avoid and whether naked snowboarding is a bad idea." So how *do* you trick your brain into thinking that your life depends on knowing Design Patterns?

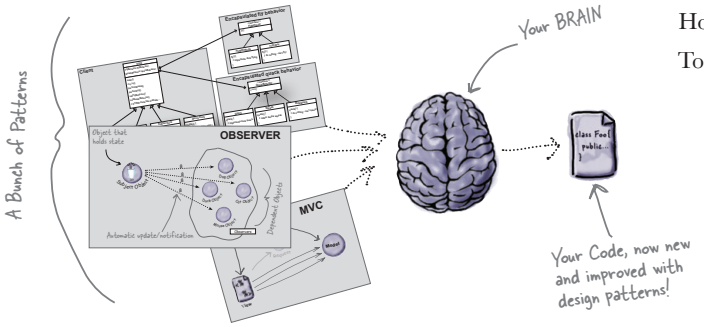
Who is this book for?	xxvi
We know what you're thinking.	xxvii
And we know what your brain is thinking.	xxvii
We think of a "Head First" reader as a learner.	xxviii
Metacognition: thinking about thinking	xxix
Here's what WE did	xxx
Here's what YOU can do to bend your brain into submission	xxxi
Read Me	xxxii
Tech Reviewers	xxxiv
Acknowledgments	xxxv

1 Welcome to Design Patterns

Someone has already solved your problems. In this chapter, you'll learn why (and how) you can exploit the wisdom and lessons learned by other developers who've been down the same design problem road and survived the trip. Before we're done, we'll look at the use and benefits of design patterns, look at some key object-oriented (OO) design principles, and walk through an example of how one pattern works. The best way to use patterns is to *load your brain* with them and then *recognize places* in your designs and existing applications where you can *apply them*. Instead of *code* reuse, with patterns you get *experience* reuse.

It started with a simple SimUDuck app 2
But now we need the ducks to FLY 3
But something went horribly wrong… 4
Joe thinks about inheritance… 5
How about an interface? 6
What would *you* do if you were Joe? 7
The one constant in software development 8
Zeroing in on the problem… 9
Separating what changes from what stays the same 10
Designing the Duck Behaviors 11
Implementing the Duck Behaviors 13
Integrating the Duck Behavior 15
Testing the Duck code 18
Setting behavior dynamically 20
The Big Picture on encapsulated behaviors 22
HAS-A can be better than IS-A 23
Speaking of Design Patterns… 24
Overheard at the local diner… 26
Overheard in the next cubicle… 27
The power of a shared pattern vocabulary 28
How do I use Design Patterns? 29
Tools for your Design Toolbox 32

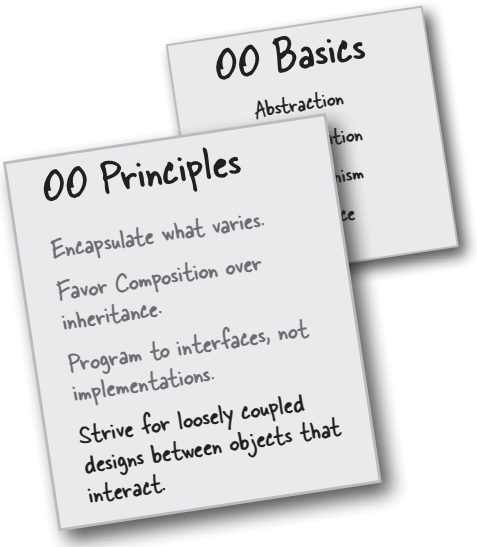




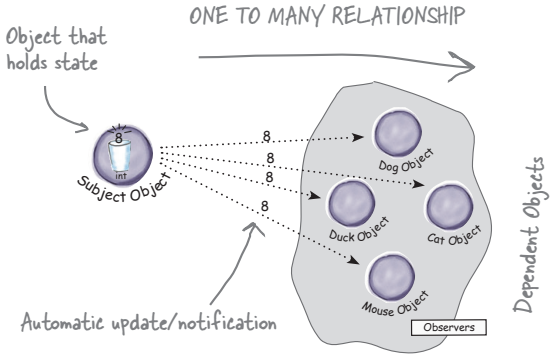
the Observer Pattern

2 Keeping your Objects in the Know

You don't want to miss out when something interesting happens, do you? We've got a pattern that keeps your objects *in the know* when something they *care about* happens. It's the Observer Pattern. It is one of the most commonly used design patterns, and it's incredibly useful. We're going to look at all kinds of interesting aspects of Observer, like its *one-to-many relationships* and *loose coupling*. And, with those concepts in mind, how can you help but be the life of the Patterns Party?



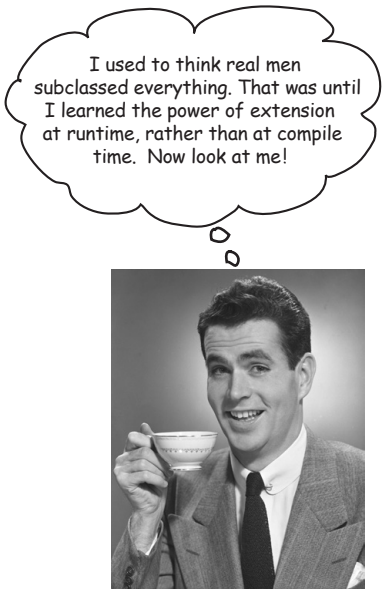
The Weather Monitoring application overview	39
Meet the Observer Pattern	44
Publishers + Subscribers = Observer Pattern	45
The Observer Pattern defined	51
The Power of Loose Coupling	54
Designing the Weather Station	57
Implementing the Weather Station	58
Power up the Weather Station	61
Looking for the Observer Pattern in the Wild	65
Coding the life-changing application	66
Meanwhile, back at Weather-O-Rama	69
Test Drive the new code	71
Tools for your Design Toolbox	72
Design Principle Challenge	73



the Decorator Pattern

3 Decorating Objects

Just call this chapter "Design Eye for the Inheritance Guy." We'll re-examine the typical overuse of inheritance and you'll learn how to decorate your classes at runtime using a form of object composition. Why? Once you know the techniques of decorating, you'll be able to give your (or someone else's) objects new responsibilities *without making any code changes to the underlying classes.*



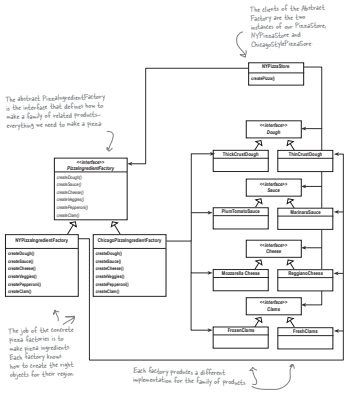
Welcome to Starbuzz Coffee	80
The Open-Closed Principle	86
Meet the Decorator Pattern	88
Constructing a drink order with Decorators	89
The Decorator Pattern defined	91
Decorating our Beverages	92
Writing the Starbuzz code	95
Coding beverages	96
Coding condiments	97
Serving some coffees	98
Real-World Decorators: Java I/O	100
Decorating the java.io classes	101
Writing your own Java I/O Decorator	102
Test out your new Java I/O Decorator	103
Tools for your Design Toolbox	105

the Factory Pattern

4 Baking with OO Goodness

Get ready to bake some loosely coupled OO designs.

There is more to making objects than just using the **new** operator. You'll learn that instantiation is an activity that shouldn't always be done in public and can often lead to *coupling problems*. And we don't want *that*, do we? Find out how Factory Patterns can help save you from embarrassing dependencies.



Identifying the aspects that vary	112
Encapsulating object creation	114
Building a simple pizza factory	115
The Simple Factory defined	117
A framework for the pizza store	120
Allowing the subclasses to decide	121
Declaring a factory method	125
It's finally time to meet the Factory Method Pattern	131
View Creators and Products in Parallel	132
Factory Method Pattern defined	134
Looking at object dependencies	138
The Dependency Inversion Principle	139
Applying the Principle	140
Families of ingredients…	145
Building the ingredient factories	146
Reworking the pizzas…	149
Revisiting our pizza stores	152
What have we done?	153
Abstract Factory Pattern defined	156
Factory Method and Abstract Factory compared	160
Tools for your Design Toolbox	162

the Singleton Pattern

5 One-of-a-Kind Objects

Our next stop is the Singleton Pattern, our ticket to creating one-of-a-kind objects for which there is only one instance, ever. You might be happy to know that of all patterns, the Singleton is the simplest in terms of its class diagram; in fact, the diagram holds just a single class! But don't get too comfortable; despite its simplicity from a class design perspective, it's going to require some deep object-oriented thinking in its implementation. So put on that thinking cap, and let's get going.

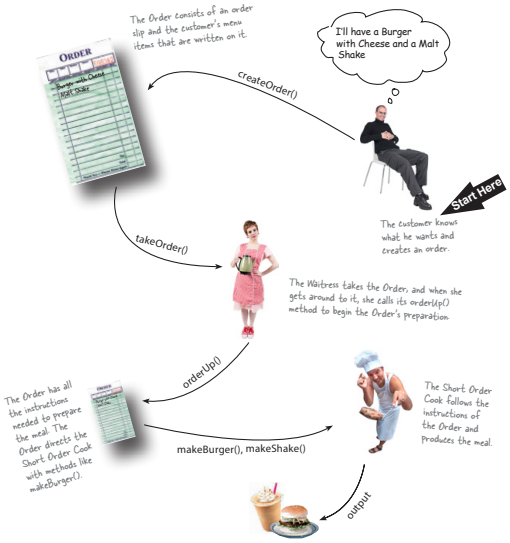
Dissecting the classic Singleton Pattern implementation 173
The Chocolate Factory 175
Singleton Pattern defined 177
~~Houston~~ Hershey, PA, we have a problem 178
Dealing with multithreading 180
Can we improve multithreading? 181
Meanwhile, back at the Chocolate Factory… 183
Tools for your Design Toolbox 186

the Command Pattern

6 Encapsulating Invocation

In this chapter, we take encapsulation to a whole new level: we're going to encapsulate method invocation.

That's right—by encapsulating method invocation, we can crystallize pieces of computation so that the object invoking the computation doesn't need to worry about how to do things, it just uses our crystallized method to get it done. We can also do some wickedly smart things with these encapsulated method invocations, like save them away for logging or reuse them to implement undo functionality in our code.

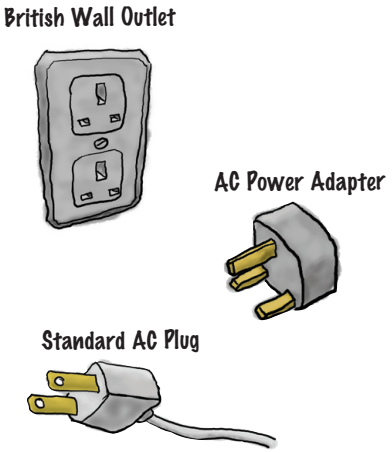


Home Automation or Bust	192
Taking a look at the vendor classes	194
A brief introduction to the Command Pattern	197
From the Diner to the Command Pattern	201
Our first command object	203
Using the command object	204
Assigning Commands to slots	209
Implementing the Remote Control	210
Implementing the Commands	211
Putting the Remote Control through its paces	212
Time to write that documentation…	215
What are we doing?	217
Time to QA that Undo button!	220
Using state to implement Undo	221
Adding Undo to the Ceiling Fan commands	222
Every remote needs a Party Mode!	225
Using a macro command	226
More uses of the Command Pattern: queuing requests	229
More uses of the Command Pattern: logging requests	230
Command Pattern in the Real World	231
Tools for your Design Toolbox	233

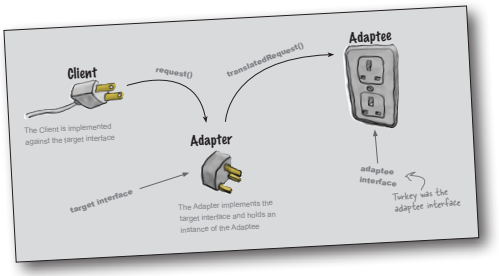
the Adapter and Facade Patterns

7 Being Adaptive

In this chapter we're going to attempt such impossible feats as putting a square peg in a round hole. Sound impossible? Not when we have Design Patterns. Remember the Decorator Pattern? We **wrapped objects** to give them new responsibilities. Now we're going to wrap some objects with a different purpose: to make their interfaces look like something they're not. Why would we do that? So we can adapt a design expecting one interface to a class that implements a different interface. That's not all; while we're at it, we're going to look at another pattern that wraps objects to simplify their interface.



Adapters all around us	238
Object-oriented adapters	239
If it walks like a duck and quacks like a duck, then it ~~must~~ might be a ~~duck~~ turkey wrapped with a duck adapter…	240
Test drive the adapter	242
The Adapter Pattern explained	243
Adapter Pattern defined	245
Object and class adapters	246
Real-world adapters	250
Adapting an Enumeration to an Iterator	251
Home Sweet Home Theater	257
Watching a movie (the hard way)	258
Lights, Camera, Facade!	260
Constructing your home theater facade	263
Implementing the simplified interface	264
Time to watch a movie (the easy way)	265
Facade Pattern defined	266
The Principle of Least Knowledge	267
How NOT to Win Friends and Influence Objects	268
The Facade Pattern and the Principle of Least Knowledge	271
Tools for your Design Toolbox	272



the Template Method Pattern

8 Encapsulating Algorithms

We've encapsulated object creation, method invocation, complex interfaces, ducks, pizzas…what could be next?

We're going to get down to encapsulating pieces of algorithms so that subclasses can hook themselves right into a computation anytime they want. We're even going to learn about a design principle inspired by Hollywood. Let's get started…

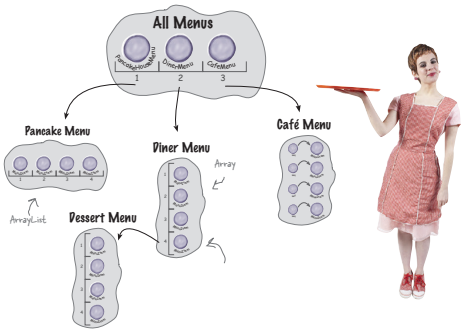
It's time for some more caffeine 278
Whipping up some coffee and tea classes (in Java) 279
Let's abstract that Coffee and Tea 282
Taking the design further… 283
Abstracting prepareRecipe() 284
What have we done? 287
Meet the Template Method 288
What did the Template Method get us? 290
Template Method Pattern defined 291
Hooked on Template Method… 294
Using the hook 295
The Hollywood Principle and Template Method 299
Template Methods in the Wild 301
Sorting with Template Method 302
We've got some ducks to sort… 303
What is compareTo()? 303
Comparing Ducks and Ducks 304
Let's sort some Ducks 305
The making of the sorting duck machine 306
Swingin' with Frames 308
Custom Lists with AbstractList 309
Tools for your Design Toolbox 313

the Iterator and Composite Patterns

9 Well-Managed Collections

There are lots of ways to stuff objects into a collection.

Put them into an Array, a Stack, a List, a hash map—take your pick. Each has its own advantages and tradeoffs. But at some point your clients are going to want to iterate over those objects, and when they do, are you going to show them your implementation? We certainly hope not! That just wouldn't be professional. Well, you don't have to risk your career; in this chapter you're going to see how you can allow your clients to iterate through your objects without ever getting a peek at how you store your objects. You're also going to learn how to create some super collections of objects that can leap over some impressive data structures in a single bound. And if that's not enough, you're also going to learn a thing or two about object responsibility.

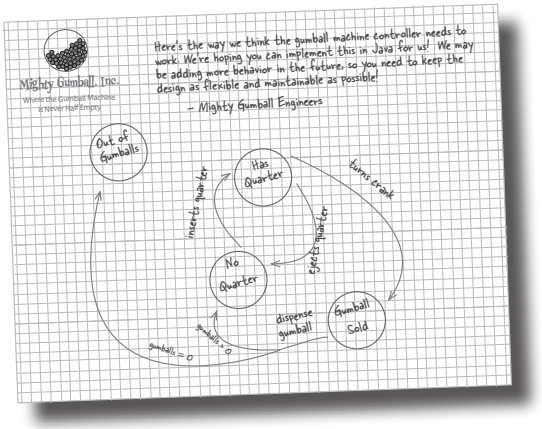


Breaking News: Objectville Diner and Objectville Pancake House Merge 318
Check out the Menu Items 319
Implementing the spec: our first attempt 323
Can we encapsulate the iteration? 325
Meet the Iterator Pattern 327
Adding an Iterator to DinerMenu 328
Reworking the DinerMenu with Iterator 329
Fixing up the Waitress code 330
Testing our code 331
Reviewing our current design… 333
Cleaning things up with java.util.Iterator 335
Iterator Pattern defined 338
The Iterator Pattern Structure 339
The Single Responsibility Principle 340
Meet Java's Iterable interface 343
Java's enhanced for loop 344
Taking a look at the Café Menu 347
Iterators and Collections 353
Is the Waitress ready for prime time? 355
The Composite Pattern defined 360
Designing Menus with Composite 363
Implementing MenuComponent 364
Implementing the MenuItem 365
Implementing the Composite Menu 366
Now for the test drive… 369
Tools for your Design Toolbox 376

the State Pattern

10 The State of Things

A little-known fact: the Strategy and State Patterns were twins separated at birth. You'd think they'd live similar lives, but the Strategy Pattern went on to create a wildly successful business around interchangeable algorithms, while State took the perhaps more noble path of helping objects to control their behavior by changing their internal state. As different as their paths became, however, underneath you'll find almost precisely the same design. How can that be? As you'll see, Strategy and State have very different intents. First, let's dig in and see what the State Pattern is all about, and then we'll return to explore their relationship at the end of the chapter.



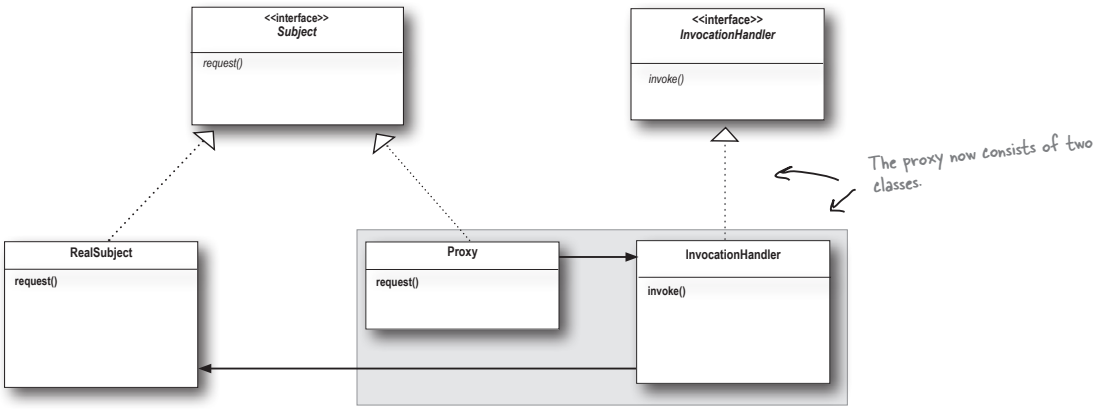
Java Breakers	382
State machines 101	384
Writing the code	386
In-house testing	388
You knew it was coming…a change request!	390
The messy STATE of things…	392
The new design	394
Defining the State interfaces and classes	395
Reworking the Gumball Machine	398
Now, let's look at the complete GumballMachine class…	399
Implementing more states	400
The State Pattern defined	406
We still need to finish the Gumball 1 in 10 game	409
Finishing the game	410
Demo for the CEO of Mighty Gumball, Inc.	411
Sanity check…	413
We almost forgot!	416
Tools for your Design Toolbox	419

the Proxy Pattern

11 Controlling Object Access

Ever play good cop, bad cop? You're the good cop and you provide all your services in a nice and friendly manner, but you don't want everyone asking you for services, so you have the bad cop control access to you. That's what proxies do: control and manage access. As you're going to see, there are lots of ways in which proxies stand in for the objects they proxy. Proxies have been known to haul entire method calls over the internet for their proxied objects; they've also been known to patiently stand in for some pretty lazy objects.

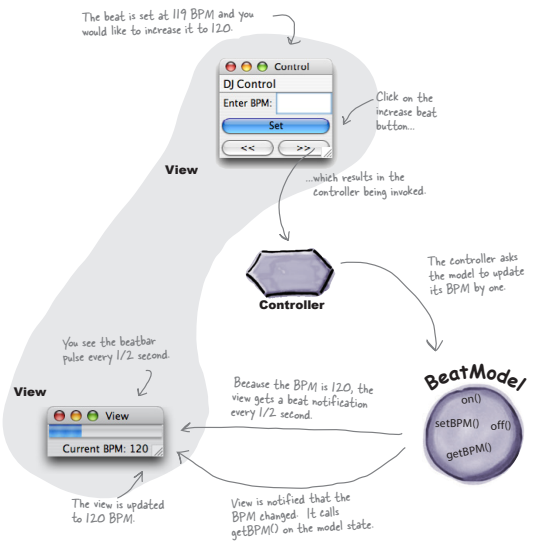
Coding the Monitor 427
Testing the Monitor 428
Remote methods 101 433
Getting the GumballMachine ready to be a remote service 446
Registering with the RMI registry… 448
The Proxy Pattern defined 455
Get ready for the Virtual Proxy 457
Designing the Album Cover Virtual Proxy 459
Writing the Image Proxy 460
Using the Java API's Proxy to create a protection proxy 469
Geeky Matchmaking in Objectville 470
The Person implementation 471
Five-minute drama: protecting subjects 473
Big Picture: creating a Dynamic Proxy for the Person 474
The Proxy Zoo 482
Tools for your Design Toolbox 485
The code for the Album Cover Viewer 489



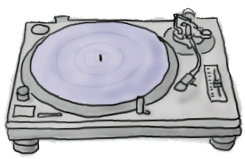
compound patterns

12 Patterns of Patterns

Who would have ever guessed that Patterns could work together? You've already witnessed the acrimonious Fireside Chats (and you haven't even seen the Pattern Death Match pages that the editor forced us to remove from the book), so who would have thought patterns can actually get along well together? Well, believe it or not, some of the most powerful OO designs use several patterns together. Get ready to take your pattern skills to the next level; it's time for compound patterns.



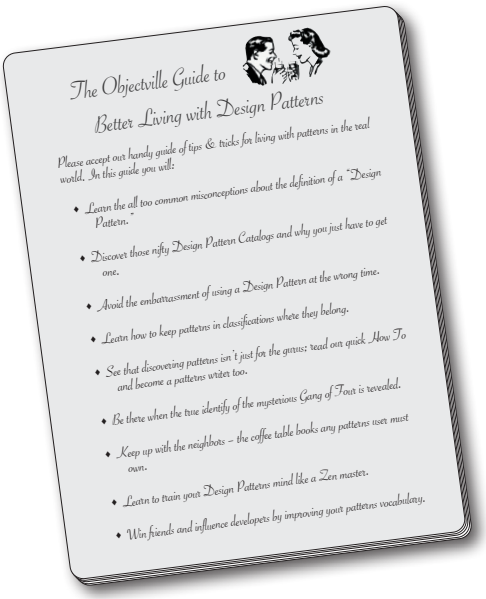
Working together	494
Duck reunion	495
What did we do?	517
A ~~bird's~~ duck's-eye view: the class diagram	518
The King of Compound Patterns	520
Meet Model-View-Controller	523
A closer look…	524
Understanding MVC as a set of Patterns	526
Using MVC to control the beat…	528
Building the pieces	531
Now let's have a look at the concrete BeatModel class	532
The View	533
Implementing the View	534
Now for the Controller	536
Putting it all together…	538
Exploring Strategy	539
Adapting the Model	540
And now for a test run…	542
Tools for your Design Toolbox	545



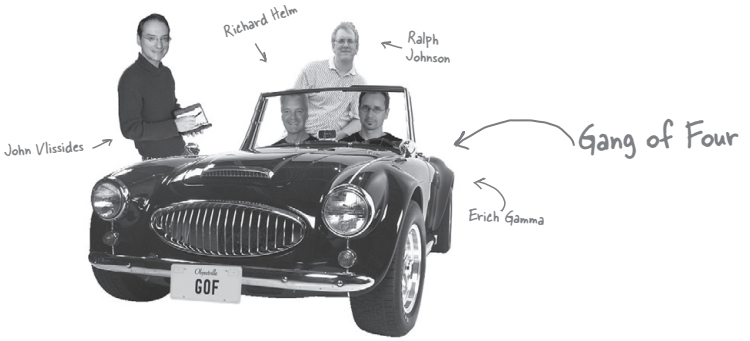
better living with patterns

13 Patterns in the Real World

Ahhhh, now you're ready for a bright new world filled with Design Patterns. But, before you go opening all those new doors of opportunity, we need to cover a few details that you'll encounter out in the real world—that's right, things get a little more complex than they are here in Objectville. Come along, we've got a nice guide to help you through the transition…

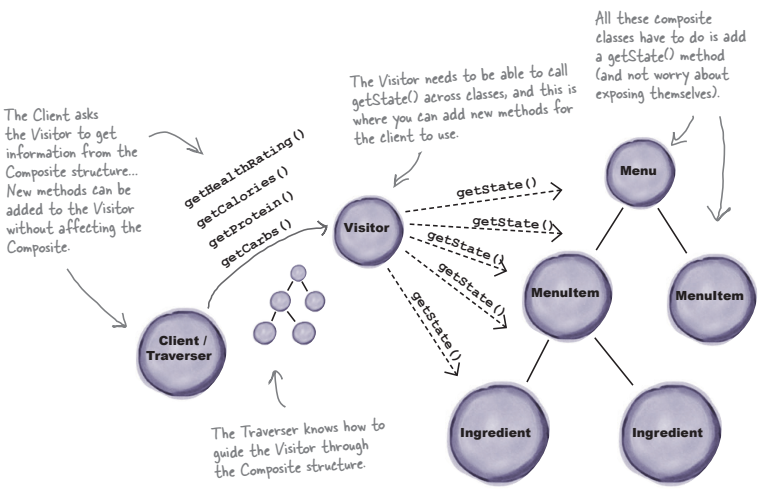


Design Pattern defined	565
Looking more closely at the Design Pattern definition	567
May the force be with you	568
So you wanna be a Design Patterns writer	573
Organizing Design Patterns	575
Thinking in Patterns	580
Your Mind on Patterns	583
Don't forget the power of the shared vocabulary	585
Cruisin' Objectville with the Gang of Four	587
Your journey has just begun…	588
The Patterns Zoo	590
Annihilating evil with Anti-Patterns	592
Tools for your Design Toolbox	594
Leaving Objectville	595



14 Appendix: Leftover Patterns

Not everyone can be the most popular. A lot has changed in the last 25+ years. Since *Design Patterns: Elements of Reusable Object-Oriented Software* first came out, developers have applied these patterns thousands of times. The patterns we summarize in this appendix are full-fledged, card-carrying, official GoF patterns, but aren't used as often as the patterns we've explored so far. But these patterns are awesome in their own right, and if your situation calls for them, you should apply them with your head held high. Our goal in this appendix is to give you a high-level idea of what these patterns are all about.



Bridge	598
Builder	600
Chain of Responsibility	602
Flyweight	604
Interpreter	606
Mediator	608
Memento	610
Prototype	612
Visitor	614

Index 617