Table of Contents (summary)

Table of Contents (the real thing)

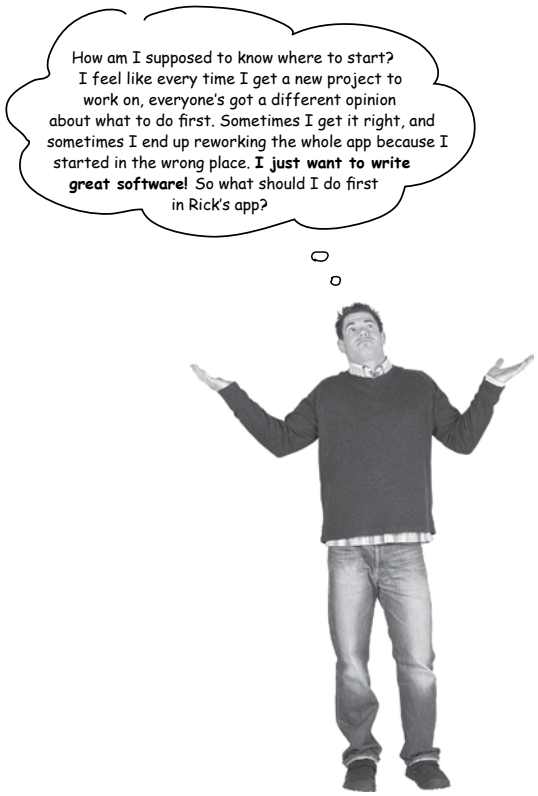
Intro

Your brain on OOA&D. 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 object-oriented analysis and design?

well-designed apps rock

1 Great Software Begins Here

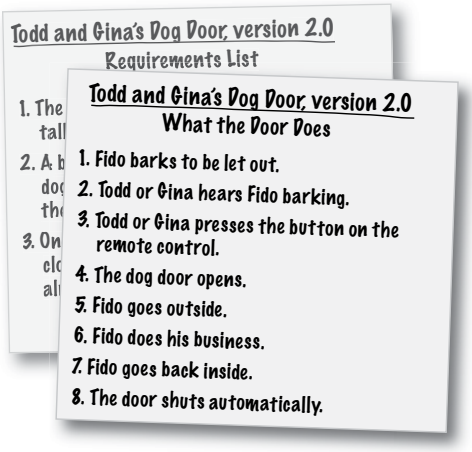
So how do you *really* write great software? It's never easy trying to figure out **where to start**. Does the application actually **do what it's supposed to**? And what about things like duplicate code—that can't be good, can it? It's usually pretty hard to know **what you should work on first**, and still make sure you don't screw everything else up in the process. No worries here, though. By the time you're done with this chapter, you'll **know how to write great software**, and be well on your way to improving the way you develop applications forever. Finally, you'll understand why **OOAD** is a four-letter word that your mother actually *wants* you to know about.

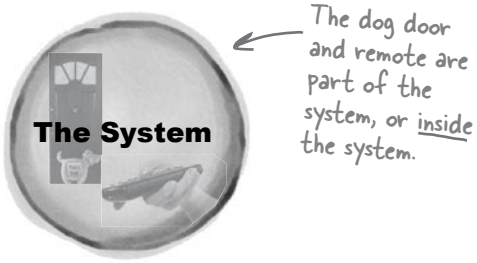


gathering requirements

2 Give Them What They Want

Everybody loves a satisfied customer. You already know that the first step in writing great software is making sure it does what the customer wants it to. But how do you figure out **what a customer really wants**? And how do you make sure that the customer even *knows* what they really want? That's where **good requirements** come in, and in this chapter, you're going to learn how to **satisfy your customer** by making sure what you deliver is actually what they asked for. By the time you're done, all of your projects will be "satisfaction guaranteed," and you'll be well on your way to writing great software, every time.



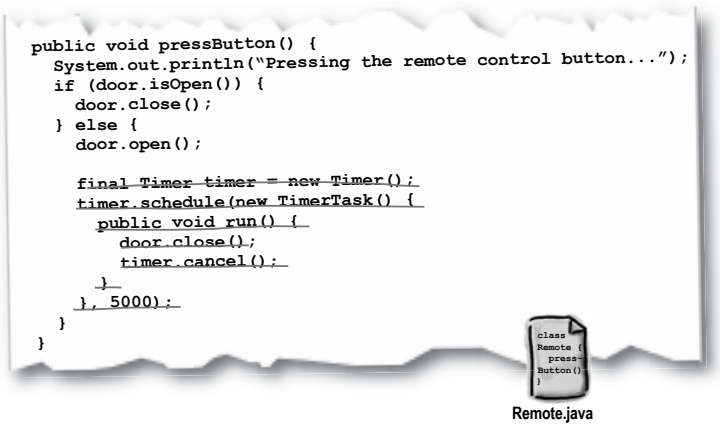


requirements change

3 I Love You, You're Perfect... Now Change

Think you've got just what the customer wanted?

Not so fast... So you've talked to your customer, gathered requirements, written out your use cases, and delivered a killer application. It's time for a nice relaxing cocktail, right? Right... until your customer decides that they really wanted something **different than what they told you**. They love what you've done, really, but it's **not quite good enough anymore**. In the real world, **requirements are always changing**, and it's up to you to roll with these changes and keep your customer satisfied.

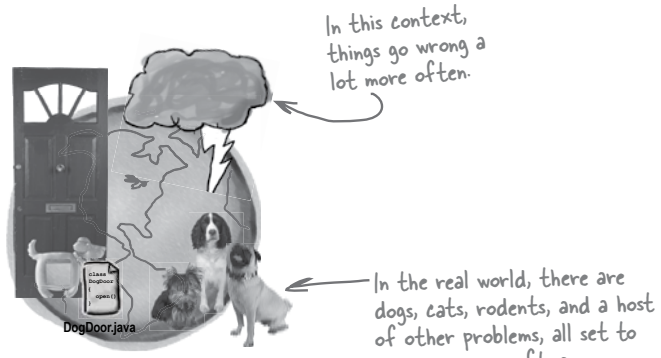


analysis

4 Taking Your Software into the Real World

It's time to graduate to real-world applications.

Your application has to do more than work on your own personal development machine, finely tuned and perfectly setup; your apps have to work when **real people use them**. This chapter is all about making sure that your software works in a **real-world context**. You'll learn how **textual analysis** can take that use case you've been working on and turn it into classes and methods that you know are what your customers want. And when you're done, you too can say: "I did it! My software is **ready for the real world**!"



The Real World

good design = flexible software

5 (part 1)

Nothing Ever Stays the Same

Change is inevitable. No matter how much you like your software right now, it's probably going to **change** tomorrow. And the harder you make it for your software to change, the more difficult it's going to be to respond to your **customer's changing needs**. In this chapter, we're going to revisit an old friend, try and improve an existing software project, and see how **small changes can turn into big problems**. In fact, we're going to uncover a problem so big that it will take a TWO-PART chapter to solve it!

5 (interlude)

good design = flexible software

5 (part 2)

Give Your Software a 30-minute Workout

Ever wished you were just a bit more flexible?

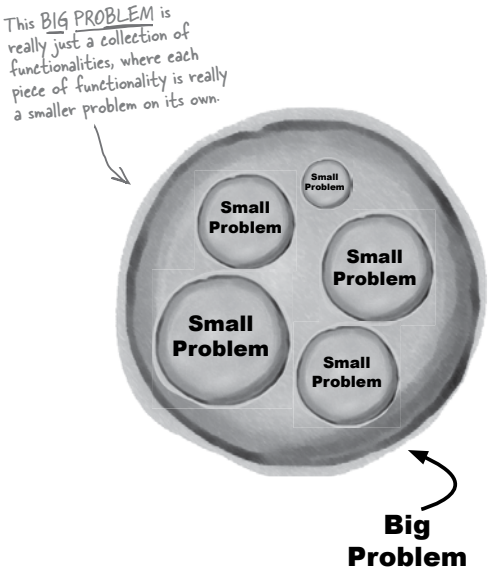
When you run into problems making changes to your application, it probably means that your software needs to be **more flexible and resilient**. To help stretch your application out, you're going to do some analysis, a whole lot of design, and learn how OO principles can really **loosen up your application**. And for the grand finale, you'll see how **higher cohesion can really help your coupling**. Sound interesting? Turn the page, and let's get back to fixing that inflexible application.

solving really big problems

6 "My Name is Art Vandelay... I am an Architect"

It's time to build something REALLY BIG. Are you ready?

You've got a ton of tools in your OOA&D toolbox, but how do you use those tools when you have to build something **really big**? Well, you may not realize it, but **you've got everything you need** to handle big problems. We'll learn about some new tools, like **domain analysis** and **use case diagrams**, but even these new tools are based on things you already know about—like listening to the customer and understanding what you're going to build before you start writing code. Get ready... it's time to start playing the architect.



architecture

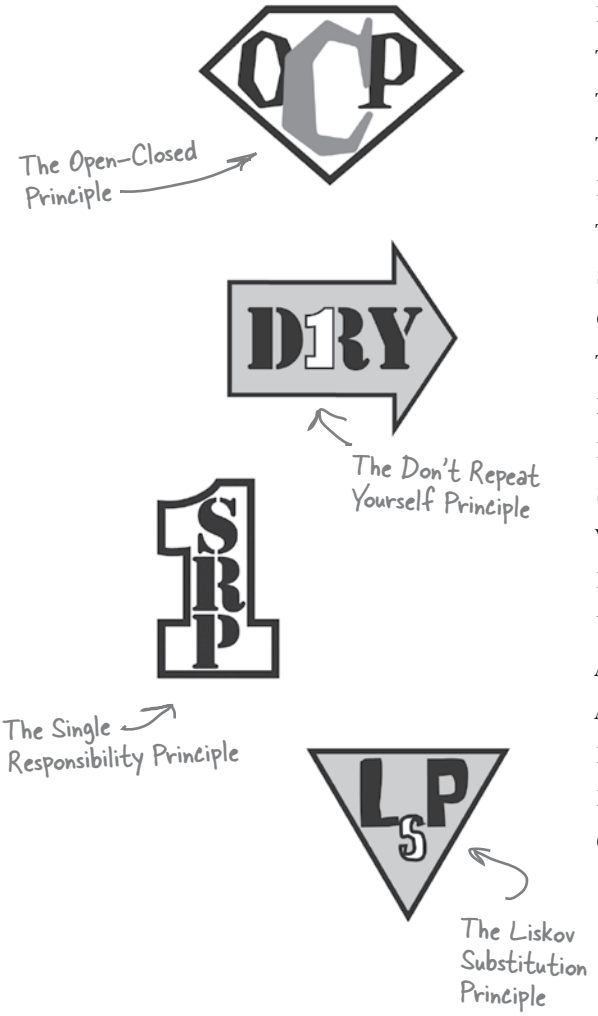
7 Bringing Order to Chaos

You have to start somewhere, but you better pick the *right* somewhere! You know how to break your application up into lots of small problems, but all that means is that you have **LOTS** of small problems. In this chapter, we're going to help you figure out **where to start**, and make sure that you don't waste any time working on the wrong things. It's time to take all those **little pieces** laying around your workspace, and figure out how to turn them into a **well-ordered, well-designed application**. Along the way, you'll learn about the all-important **3 Qs of architecture**, and how **Risk** is a lot more than just a cool war game from the '80s.

design principles

8 Originality is Overrated

Imitation is the sincerest form of not being stupid. There's nothing as satisfying as coming up with a completely new and original solution to a problem that's been troubling you for days—until you find out someone else **solved the same problem**, long before you did, and did an even better job than you did! In this chapter, we're going to look at some **design principles** that people have come up with over the years, and how they can make you a better programmer. Lay aside your thoughts of "doing it your way"; this chapter is about **doing it the smarter, faster way**.

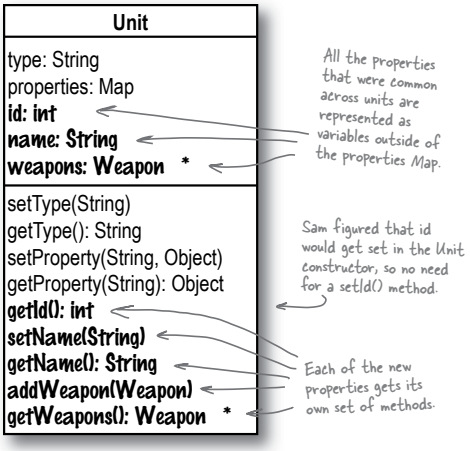


iterating and testing

9 The Software is Still for the Customer

It's time to show the customer how much you really care.

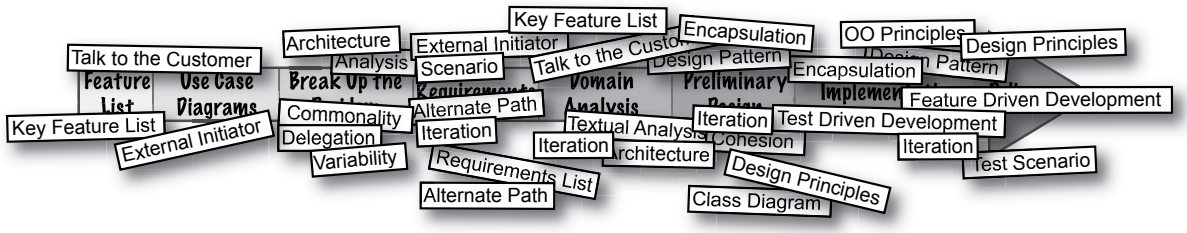
Nagging bosses? Worried clients? Stakeholders that keep asking, "Will it be done on time?" No amount of well-designed code will please your customers; you've got to **show them something working**. And now that you've got a solid OO programming toolkit, it's time to learn how you can **prove to the customer** that your software works. In this chapter, we learn about two ways to **dive deeper** into your software's functionality, and give the customer that warm feeling in their chest that makes them say, ***Yes, you're definitely the right developer for this job!***



the ooa&d lifecycle

10 Putting It All Together

Are we there yet? We've been working on lots of individual ways to improve your software, but now it's time to **put it all together**. This is it, what you've been waiting for: we're going to take **everything** you've been learning, and show you how it's all really part of **a single process** that you can use over and over again to **write great software**.



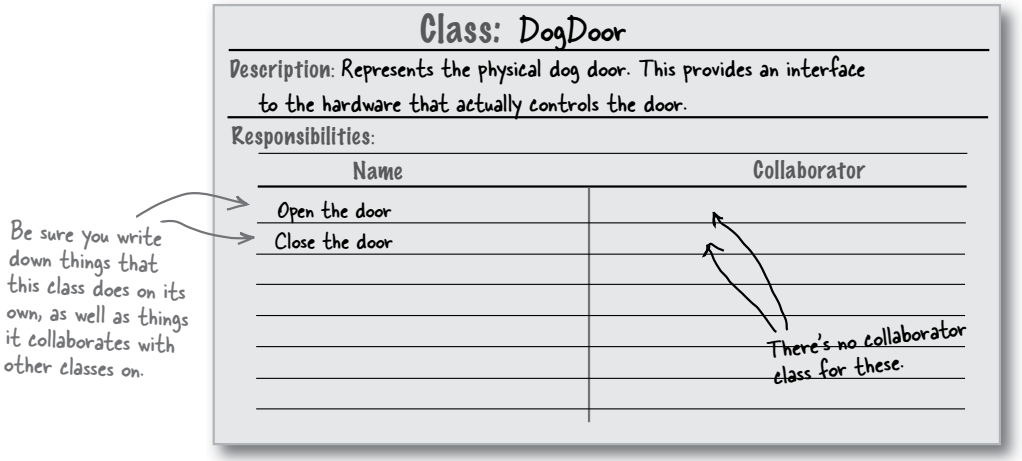
appendix i: leftovers

The Top Ten Topics (we didn't cover)

Believe it or not, there's still more. Yes, with over 550 pages under your belt, there are still things we couldn't cram in. Even though these last ten topics don't deserve more than a mention, we didn't want to let you out of Objectville without a little more information on each one of them. But hey, now you've got just a little bit more to talk about during commercials of CATASTROPHE... and who doesn't love some stimulating OOA&D talk every now and then?

Anti Patterns

Anti-patterns are the reverse of design patterns: they are common BAD solutions to problems. These dangerous pitfalls should be recognized and avoided.



appendix ii: welcome to objectville

Speaking the Language of OO

Get ready to take a trip to a foreign country. It's time to visit Objectville, a land where **objects do just what they're supposed to**, applications are all **well-encapsulated** (you'll find out exactly what that means shortly), and designs are easy to **reuse and extend**. But before we can get going, there are a few things you need to know first, and a little bit of **language skills** you're going to have to learn. Don't worry, though, it won't take long, and before you know it, you'll be speaking the language of OO like you've been living in the well-designed areas of Objectville for years.