You’re now almost at the end of your journey. By now you should have a sound understanding of the overall principles of how to decompile and how to make some attempts at protecting your code. Having said that, I’ve found from working with clients and colleagues that even if you understand what decompilation and obfuscation really mean, it doesn’t help you figure out what practical measures you can take to protect your code. A little knowledge can often create more questions than answers.

As the Competency Centre for Java (JCC) says on its deCaf website FAQ:

*Is it true that no one will ever be able to decompile my deCaf protected application? NO. deCaf does not make decompilation impossible. It makes it difficult. Making decompilation impossible is impossible.*

The goal of this book is to help raise the bar and make it more difficult for anyone to decompile your code. Currently in the Android world there seems to be a “hear no evil, see no evil” approach to decompilation, but sooner or later that will change. After reading this book you should be forewarned and, more important, forearmed about the best practical approach to safeguard your code, given your specific circumstances.

This chapter examines a case study to help overcome this conundrum. Almost everyone who tries to protect their code does so using some sort of obfuscation
tool. The case study looks at this approach in more detail to help you come to a conclusion about how to best protect your code. It has the following format:

- Problem description
- Myths
- Proposed solutions: ProGuard and DashO

Obfuscation Case Study

For many people, the fear of someone decompiling their Android application is nowhere near the top of the list of things they’re worrying about. It ranks way below installing the latest version of Maven or Ant. Sure, they’d like to protect against decompilation, but nobody has the time—and doesn’t ProGuard take care of that anyway?

There are two simple options in this scenario: use obfuscation to protect the application, or ignore decompilation as if it’s not a problem. The latter, of course, isn’t a recommended choice for obvious reasons.

Myths

Over the years, I’ve heard many different arguments about whether it makes sense to protect your code. The most common one today is that if you create a good Android application and continue to improve it, that will safeguard you against anyone decompiling your code. It’s a common belief that if you write good applications, the source will protect itself—that upgrades and good support are much better ways of protecting your code than using obfuscation or any of the other techniques discussed in this book.

Other arguments are that software development is about how you apply your knowledge, not getting access to someone else’s applications. The original code these days may come from a well-described design pattern, so nobody cares if it’s hacked. And all developers (the good ones, anyway) can always think of a better way of doing something after it’s completed, so why worry? Chances are that if someone is so unimaginative that they have to resort to stealing your code, they won’t be capable of building on the code and turning it into something useful. And it’s impossible for you to read your own code six months after it’s developed, so how would anyone else make sense of it?

Obfuscated code can also be very difficult to debug. Error reports from the field need to be traced back to the correct method so the developer can debug and