Introduction to Fluency in Kotlin

> *Programmers often say that Kotlin code is beautiful, and there are good reasons for that. Kotlin code is concise, has less ceremony, and the language provides hygienic syntax to extend the capabilities of existing classes to add domain-specific methods and properties. You'll see the fluency of Kotlin shine in this part, and you'll learn to apply the capabilities of the language to implement your own fluent APIs and internal domain-specific languages. You'll also learn about the optimization Kotlin provides for recursion so you can make use of this intriguing programming construct for large input parameters.*

The famous words of Blaise Pascal, "If I had more time I would have written a shorter letter," apply as much to programming as writing. In this chapter I urge you to take the time to learn some useful ways to write shorter yet readable code.

Code is written just once, but is read, refactored, enhanced, and maintained continuously through the lifetime of the application. Verbose, poor-quality, hard-to-understand code can turn even the politest person on the planet into a cusser. By increasing quality you can lower the cost of maintaining code.

Fluent code is concise, easy to read and understand, and is a gift we give to ourselves and to the rest of the team. It's pleasing to read, makes us productive, and can serve as a great motivator to develop better-quality software.

Fluent code has a direct economic impact on developing software. Most teams are in a perpetual state of urgency and rarely have the luxury to fix bugs when they have plenty of time and are relaxed. Cluttered code that is verbose and hard to read greatly increases the stress level, which in turn makes it harder to see through and resolve issues. It takes less effort and time to parse through code that is fluent than code that is long, noisy, and verbose. While some programmers may be able to write more fluent code in any language, a language that offers greater fluency lets all of us create more fluent code with less effort. When fluency becomes a natural part of coding, everyone benefits—the developers and the business.

In this chapter, we'll focus on how to make Kotlin code fluent, expressive, and concise. We'll start with the controversial operator overloading capability, then we'll cover how to inject methods and properties into existing third- party classes, to extend functions, to make function calls fluent, and to reduce code when using any object. We'll wrap up the chapter by looking at an advanced feature that allows attaching a receiver to lambda expressions. The techniques you learn here will be a nice segue into creating DSLs in the next chapter.
