Introduction to Lambdas in Kotlin

> The functional style is less complex compared to the imperative style of programming. Code reads like a problem statement in the functional style and so is easier to understand. With the ability to pass functions to functions, we can use functional decomposition in addition to object decomposition in Kotlin.
>
> In this part, we'll focus on the functional programming capabilities of Kotlin. You'll learn how to create lambda expressions, why and where to use them, and how to ensure we don't compromise performance to gain fluency. You'll also learn to use internal iterators, sequences, and design patterns that can benefit from using lambda expressions.

Functional-style code is declarative—you focus on what to do and assign the details of how to do it to underlying libraries of functions. In contrast, in the imperative style of programming you have to deal with both the whats and hows. As applications become complicated, having to deal with the hows makes the code highly complex. The functional style is inherently less complex—you can get more accomplished with not only fewer lines of code but also code that is expressive, easier to understand, and a breeze to maintain.

The imperative style of programming has been the mainstream for a few decades now. Languages like Java have offered a combination of imperative style and object-oriented programming. Object-oriented programming is helpful to abstract and encapsulate concepts. The goal of functional programming is not to replace OO, which greatly reduces complexity; the real concern is the imperative style. That's one of the reasons why Java has moved forward to accommodate the functional style of programming in recent years. Now we can do both imperative and functional styles of programming along with object-oriented programming in Java. While Java has evolved, Kotlin was born that way, as a mixed paradigm language that supports the imperative, functional, and object-oriented paradigms.

In this chapter you'll learn the benefits of the functional style, how to create and use lambda expressions, or lambdas for short, and the dos and don'ts with use lambda expressions, or lambdas for short, and the dos and don'ts with

lambdas. You'll also learn about lexical scoping, passing methods where lambdas are expected, and when to create anonymous functions instead of lambdas.

Sometimes appearances can be deceiving, and the functional style may result in poor performance if we're not careful. We'll also see how to keep the fluency in code without compromising performance.

So, let's begin…!
