Introduction to Objects and Classes

> Kotlin supports the object-oriented paradigm, but with a few twists. The compiler does a healthy dose of code generation to remove boilerplate code and also provides a few special types of classes not available in Java. In this part, you'll learn the benefits of data classes, sealed classes, companion objects, singletons, and more. From the design point of view, you'll learn when to use inheritance vs. delegation, both of which are directly supported by Kotlin. It's time for your **OO** code to enjoy a serious makeover.

Kotlin is determined to make your code fluent, concise, and expressive, whether you're writing procedural, functional, or object-oriented code. Kotlin doesn't require classes, and you can work with top-level functions when they suffice. When object-oriented programming is the right design choice for your applications, Kotlin will get you moving fast in that direction without insisting on verbose boilerplate code to create classes. You can write classes with less effort, fewer lines of code, and with greater speed and ease.

Kotlin's facility to create and work with classes and objects is more akin to the features in Scala than in Java. But Kotlin takes the low-ceremony approach further than Scala in a few ways. You invoke class constructors like functions—there's no `new` keyword in Kotlin. You don't waste your time and effort to define fields—that's an implementation detail that Kotlin takes care of. Instead, you define properties, and Kotlin proceeds to generate the backing fields where necessary. If your focus is on representing data rather than behavior, you can achieve that using data classes, for which Kotlin generates a few conventional methods.

In this chapter you'll learn to create classes with constructors, properties, methods, companion objects, and data classes. You'll see how to define instance methods, and how defining static methods in Kotlin is different than in Java. You'll also learn how to create generic classes with constraints of parameterized types. But first we'll start with absolutely zero ceremony—working just with objects in the next lesson.