Wrapping Up

Kotlin’s support to build a hierarchy of classes goes beyond the support provided in Java. Interfaces are almost the same in both languages, except in the way they are defined—Kotlin has no `default` keyword, and `static` methods go into companion objects even for interfaces. Kotlin’s nested and inner classes also differ from the same concepts in Java—unlike Java, Kotlin makes a clear distinction between nested and inner classes, to make the intent clear. Classes are `final` by default, but not all open classes are widely inheritable—you can control inheritance using `sealed` classes if you like. Kotlin places restrictions around inheritance so classes don’t accidentally serve as a base class. This forces you to make the intent explicit and makes code safer to use. Finally, the instances of `enum` classes are created as `static` members, and where specialization is needed, they turn into anonymous inner classes.

Good design guidelines often suggest preferring delegation over inheritance, but many OO languages offer little direct support for delegation. Kotlin is keen on helping you make the right design choices.

In the next chapter, we’ll learn how to use the delegation feature of Kotlin.