Introduction to Delegation

Both inheritance and delegation, which are distinctive design tools in **OO** programming, promote reuse by way of extending one class from another. We have to wisely choose between them. Languages' capabilities often limit our choices, and this is where Kotlin's ability to support both comes into play.

While powerful and often used, inheritance—where a class derives properties, methods, and implementation from another class—leads to tight coupling and is inflexible. Most languages that provide inheritance don't permit a class to choose between different base classes. Whatever a class inherits from, you're stuck with it, like biological parents—whether you like them or not, you don't have a choice.

Delegation is more flexible than inheritance. An object may delegate or pass some of its responsibilities to an instance of another class. Different instances of a class may delegate to instances of different classes. It's like the way siblings, with the same parents, can and do choose different friends.

Design books like Design Patterns Elements of Reusable Object-Oriented Software advise us to prefer delegation over inheritance where possible. Effective Java, Third Edition makes a strong recommendation for Java programs as well. Yet we see reuse via inheritance more often than delegation in Java, because the language provides good support for inheritance and little for delegation. Kotlin takes the design recommendations of those books to heart and provides facilities for delegation.

In this chapter, you'll learn when to use delegation instead of inheritance, and how to use it. We'll also look at a few built-in delegates in Kotlin. Since we don't delegate fun things like learning to someone else, let's get moving.