Introduction to Internal Iteration and Lazy Evaluation

Unlike the imperative style where external iterators are prominent (see Chapter 5, External Iteration and Argument Matching), in functional programming we use internal iterators. Internal iterators put iterations on autopilot; instead of focusing on iteration, you can keep your eyes on what to do for each element in a collection or a range. As well, internal iterators avoid explicit mutability and thus the iteration can be easily parallelized without the risk of race conditions.

Internal iterators are higher-order functions; that is, we pass lambdas to functions that perform various tasks related to iteration, like selecting particular values, transforming data, and so on. The higher-order functions provide common code for the iteration, and the lambdas tailor them for the problem at hand. For example, the `filter()` higher-order function will iterate over the elements in a collection and return selected values. The lambda passed to `filter()` decides the actual elements that get selected. For instance, given a list of languages, the `filter()` function can be used to return a sublist of only statically typed languages from among the original list. To accomplish this, the lambda that's passed to `filter()` can take a language as argument and return `true` if the language is statically typed and `false` otherwise.

Internal iterators in Kotlin are concise, expressive, and have reduced complexity compared to external iterators. But performance may occasionally be a drawback when using them. In some situations, as we'll explore later, internal iterators may end up performing some extra computations when compared to the corresponding external iterators. This may not have much impact when a collection is relatively small, such as a few hundred elements. On the other hand, if we're dealing with a very large collection of data, like thousands of elements, then the overhead will become an issue. This is where Kotlin sequences come in. Sequences are internal iterators and look almost the same as other internal iterators to the human eyes. Internally, though, they're implemented differently; they defer execution—hence they are lazy—and evaluate parts of the iteration only if necessary.

In this chapter, we'll first discuss the key differences between external and internal iterators, then explore different internal iterators built into the Kotlin standard library, and, finally, look at the benefits of lazy evaluations with

standard library, and, finally, look at the benefits of lazy evaluations with sequences.