Wrapping Up

The fluency and the power of Kotlin makes it a great language to create internal DSLs. And the ability to assign a context object for the execution of lambdas gives Kotlin a distinctive advantage as a host language for DSLs, compared to other statically typed languages. Designing a DSL with good fluency can be challenging, but with a good understanding of the various techniques to bring forward fluency and a good dose of patience, you can find your way to design for the fluency you desire.

Since it's a statically typed language, Kotlin offers type safety when it comes to using DSLs. This has the benefit of failing fast. Also, using DSL marker annotations, you can improve error checking by restricting fluent calls to immediate receiver objects, instead of automatically routing them to the parent receiver. You can use the techniques you've learned in this chapter to design your own DSLs and also to design your APIs in a way that's fluent for users.

In the next chapter, we'll see another highly expressive power of Kotlin—the optimization it provides for tail recursions.