Introduction to Type Safety in Kotlin

The more statically typed a language is, the more readily it should ensure type safety, but without the need to excessively specify types. Kotlin works hard to make your code more type safe and less error prone with enhanced `null` checks, smart type casting, and fluent type checking. In this chapter, you'll learn about a few basic types in Kotlin and the effective type checking capabilities built in to the compiler. We'll also look at how Kotlin fails fast at compile time to prevent many errors from sneaking into runtimes—this will make your programming efforts more productive.

Can you imagine turning on the television and getting a `NullPointerException`? That happened to my friend Brian Sletten after he turned on his TV to watch his favorite show, and saw that error instead—with programs like that who needs horror movies? Kotlin will help you prevent `NullPointerExceptions` at *compile time*.

With Kotlin's design-by-contract approach, you clearly express if and when a function or a method may receive or return a `null` reference. If a reference may possibly be `null`, then you're forced to perform a `null` check before you can access any useful methods or properties of the object that's referenced. With this facility, Kotlin makes the code safe, which can save the day from debugging and embarrassing blowups in production. Also, Kotlin provides a number of operators to work with `null` types, which reduces the noise in code when dealing with references that may be `null`. What's even more exciting about this capability is that these checks are purely at compile time and don't add anything to the bytecode.

Much like Java's `Object`, all classes in Kotlin inherit from the `Any` class. This class brings under one fold a few common methods that are useful on instances of just about any class in Kotlin. When working with multiple types, if a need to cast between types arises, the smart cast feature of Kotlin will take care of automatically casting so you don't have to write mundane code that's obvious to both you and the compiler. This not only saves keystrokes but also greatly reduces the verbosity in code, leading to code that is easier to maintain. And, in case you need to perform an explicit type cast, Kotlin has elegant syntax for that too, as

need to perform an explicit type cast, Kotlin has elegant syntax for that too, as you'll soon see.

One of the more advanced concepts in Kotlin is the nice support for working with covariance and contravariance of generics parametric types. That sounds complicated—and it is—but this chapter will demystify those concepts so you can use generics more effectively in Kotlin than in Java. We'll also explore reified type parameters that reduce clutter and error when creating and calling generic functions that require type information at runtime.

In this chapter you'll learn about the `Any` and `Nothing` classes, about the nullable references and related operators, and the benefits of smart casts. You'll also learn how to safely perform type casts and how to create extensible generic functions that are type safe. This will help you to design code that is less error prone and easier to maintain.

Let's start then!
