Introduction to Unit Testing in Kotlin

Code always does what we type and not what we meant, and that's true in statically typed languages as much as in dynamically typed languages. The Kotlin compiler's rigorous verification will substantially reduce errors that may occur in code. But, as the application evolves, it's our responsibility to verify that code continues to work as intended.

Manually running the code to verify if everything works as expected is expensive, time consuming, and in itself error prone. Automated testing is one of those steps that takes time but, in turn, results in saving significant time in the long run. Unit testing is a part of automated testing and, in this chapter, we'll look at how to write unit tests for code written using Kotlin.

We'll first look at how to write empirical tests for functions with no side effects—functions that will result in fast, predictable, deterministic results. Then we'll explore writing interaction tests for code with dependencies—that is, code that isn't idempotent and, depending on the state of the dependencies, may yield different results for each call. Finally, we'll look at writing tests for code that uses coroutines to make asynchronous calls.

You may pick from a variety of tools to unit test Kotlin code. In this chapter, we'll use KotlinTest for running the tests and Mockk for mocking out dependencies. We'll also measure the code coverage using Jacoco. Along the way, we'll discuss the reasons for choosing these tools.

At the end of this chapter, you'll know how to write automated tests for classes written in Kotlin and for top-level functions, how to mock out classes and extension functions, how to write tests for coroutines/asynchronous calls, and how to measure code coverage.

Let's get started with writing automated tests.