Introduction to Asynchronous Programming in Kotlin

Coroutines are a great way to implement non-blocking calls. You can configure coroutines to run your tasks concurrently or in parallel by using different coroutine contexts. We explored the fundamentals of coroutines in the previous chapter. In this chapter, we'll build on those concepts to create robust asynchronous programs. We'll start by looking at how coroutines help you comprehend the complexities of asynchronous programs. Then we'll discuss that gnarly issue of exception handling and how coroutines deal with them. We'll then talk about cancelling coroutines that have been started and how cancellation and exceptions interplay. We'll also look into supervisory jobs and how they manage cancellations.

Coroutines are great for long-running tasks, but we don't want a task to unintentionally run forever. We'll use timeout to force-fail coroutines that have outrun their permissible time.

As we walk through the concepts in this chapter, along the way we'll use `async()` and `await()` functions to create practical asynchronous programs that will illustrate the power of coroutines.
