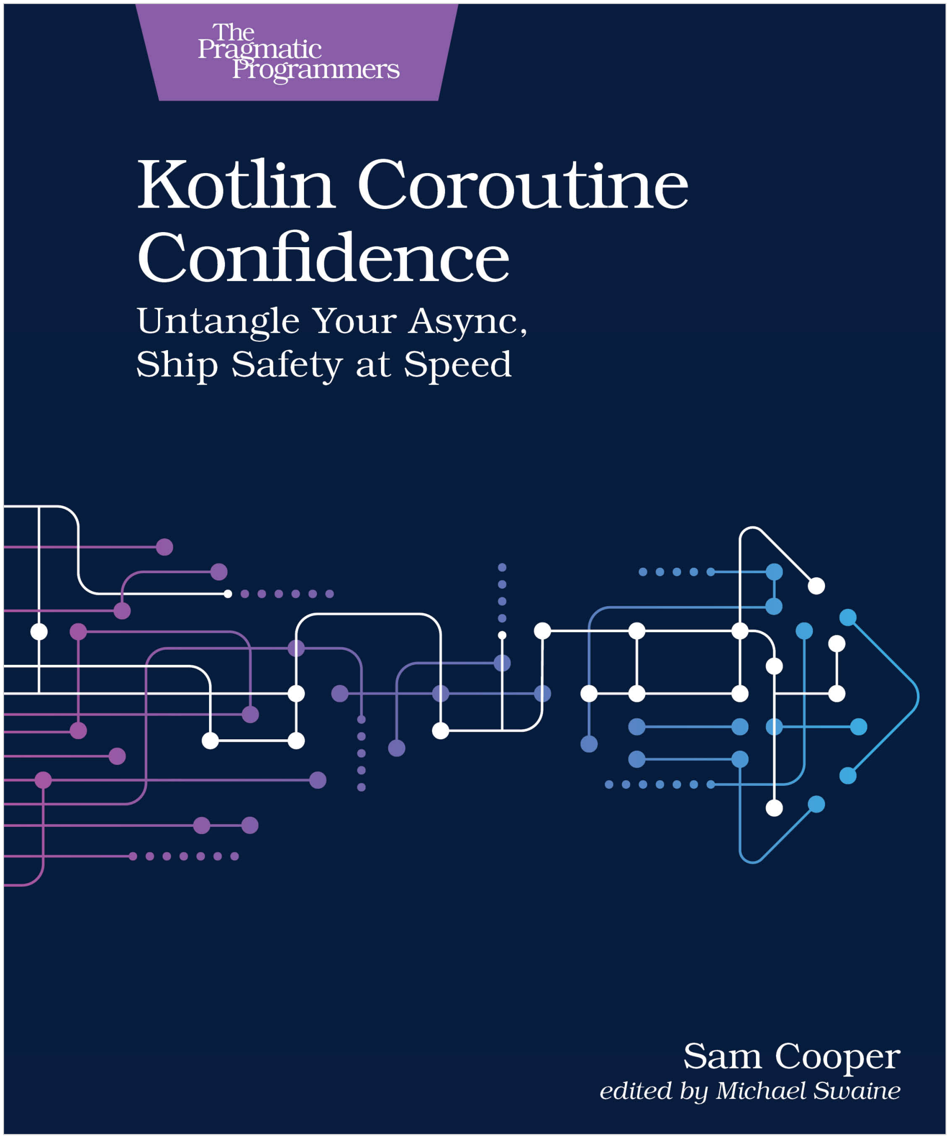

The Pragmatic Programmers

Kotlin Coroutine Confidence

Untangle Your Async, Ship Safety at Speed

Sam Cooper

edited by Michael Swaine

Introduction

For as long as I've been writing code, programming has been a tale of two styles.

One is the undisputed heavyweight champion—the original programming paradigm we all know and love. It's sequential. It's synchronous. It's structured. It lays out its statements in a clear and logical parade of loops, function calls, and conditionals.

In the other corner is the lightweight challenger. Zipping from thread to thread with freedom and agility, the callbacks and continuations of asynchronous code weave a dancing web that juggles inputs, outputs, and user interactions with unmatched efficiency.

Asynchronous programming has long held the advantage when it comes to responsive user interfaces and high throughput. But its stilted stitched-together style often falls far short of the ergonomic control flow, resource management, and error handling that have earned synchronous programming its unassailable spot at the top.

So for years, the two have been locked in a stalemate.

Today, old-fashioned synchronous programming styles are staging a comeback. More and more languages have begun introducing features and frameworks that bridge the gap, letting us write clear and sequential programs that execute with the speed and efficiency of asynchronous code.

Kotlin's coroutines are part of this renaissance. And with structured concurrency, they have a secret weapon that could turn the tide and solve some of asynchronous programming's greatest flaws.

Who Is This Book For?

If you write Kotlin code, this book is for you.

Coroutines have something to offer for almost every program. Whether you build mobile apps, web pages, backend servers, or something else altogether, you can use the tools and techniques in this book to improve your code and enhance your development experience.

For those new to coroutines, the book will take you on a step-by-step journey. We'll jump right in with a real-world example of an asynchronous program, and we'll gradually build on that foundation with a new topic in each chapter. You'll learn how coroutines can upgrade your asynchronous code, and how they compare with other tools you might have used, both in Kotlin and in other languages.

Or perhaps you've already started using coroutines, and you're here to learn more about them. Yes, we'll answer all those frustrating questions. Why can't I launch a coroutine there? Why does that function work that way? And what's this exception doing here? You'll learn about the unique challenges that come with asynchronous programming and concurrency, and understand how each tool in your coroutine toolbox has been carefully designed to help solve these problems.

Online Resources

You'll find a dedicated page[1] for this book on the Pragmatic Bookshelf website. From there, you can download the complete source code for all the examples you'll be working with. If you're reading the book in digital form, you'll also find a handy link above each code snippet that'll bring you directly to the source file.

Got feedback or questions? Follow the links on the book's webpage to find the DevTalk forum, where you can report a mistake, make a suggestion, or just discuss what you've learned with other readers.

What You'll Need

Before you start, you should already be familiar with the Kotlin programming language. We'll be looking closely at how coroutines work with common control-flow structures—things like functions, loops, and try–catch–finally blocks—so it'll help if you have a clear understanding of how those basics work in everyday code.

As we introduce each topic, we'll work through a code example where we use coroutines to solve a real-life problem. If you'd like to follow along with the

1. http://pragprog.com/titles/sckotlin

code examples in the book, you'll also need to be comfortable using an IDE like IntelliJ IDEA or Android Studio, setting up and executing a simple project, and adding dependencies. If you'd rather just read the code without running it, that's fine too!

For simplicity, we'll run our code examples in a desktop Java environment. They've been written and tested using Kotlin 2.1 and Java 21, but they should work fine on any recent version. We won't include any code examples for other targets, aside from one or two illustrative Android snippets, but when there are changes required to translate the code from platform to platform, we'll talk about what those are.

What's in This Book?

We'll begin our journey with a simple program that uses coroutines to wait for outside events without blocking its thread. To do so, we'll introduce suspending functions, which are the building blocks that give coroutines their power. We'll compare them to ordinary functions, and to other kinds of asynchronous programming. Why use suspending functions instead of callbacks or blocked threads? We'll build a photo-gallery app that demonstrates how coroutines can give us the best of both worlds, keeping our single-threaded user-interface code clean and responsive.

But why do suspending functions come with extra rules and syntax—restrictions that don't seem to apply to other solutions like callbacks and futures? As we continue to improve our photo-gallery app, we'll add a coroutine scope, and use its coroutine builder functions to add new coroutines to an existing application. We'll compare this to the ways we might make an asynchronous call in other languages or frameworks, and we'll see how Kotlin's approach is protecting us from a memory leak minefield.

Coroutines aren't just for user-interface code, and we'll use them to harness the multitasking power of any asynchronous operation. As we do so, we'll also learn about the rules that link our tasks together into a safe, structured hierarchy. We'll make sure errors don't go missing, and we'll use cooperative cancellation to stop our coroutines becoming resource-stealing zombies.

Next, we'll stage a contest to pit our coroutines against old-fashioned threads. We'll learn why coroutines are sometimes called *lightweight threads*, and we'll use a new coroutine dispatcher to give our coroutines an even bigger speed boost with parallel execution.

Coroutines can interoperate with almost any other asynchronous programming style you can think of. They can even call blocking functions, if you use the right dispatcher. We'll learn some built-in functions and facilities to convert between threads, coroutines, callbacks, and futures, and we'll even try our hand at creating our own suspension point from scratch. As we near the end of our journey, we'll introduce flows, using them to encapsulate asynchronous procedures and integrate with Reactive Streams.

Since no software is complete without tests, we'll wrap up with some handy tools and techniques for testing the coroutines we've written.

Ready to get started?

Let's dive into the world of coroutines, and write our first suspending function!

Part I

Best of Both Worlds

Familiar sequential control flow, or responsive user interfaces? Powerful parallel processing, or low-cost lightweight execution? Safe, predictable outcomes, or efficient asynchronous multitasking? It's a world full of compromises—but it doesn't have to be.

In the first part of this book, you'll learn to speak the language of coroutines. We'll unpick all of those difficult choices, and you'll assemble the coroutine tools and techniques to make them each a thing of the past.