Extracted from:

Programming DSLs in Kotlin

Design Expressive and Robust Special Purpose Code

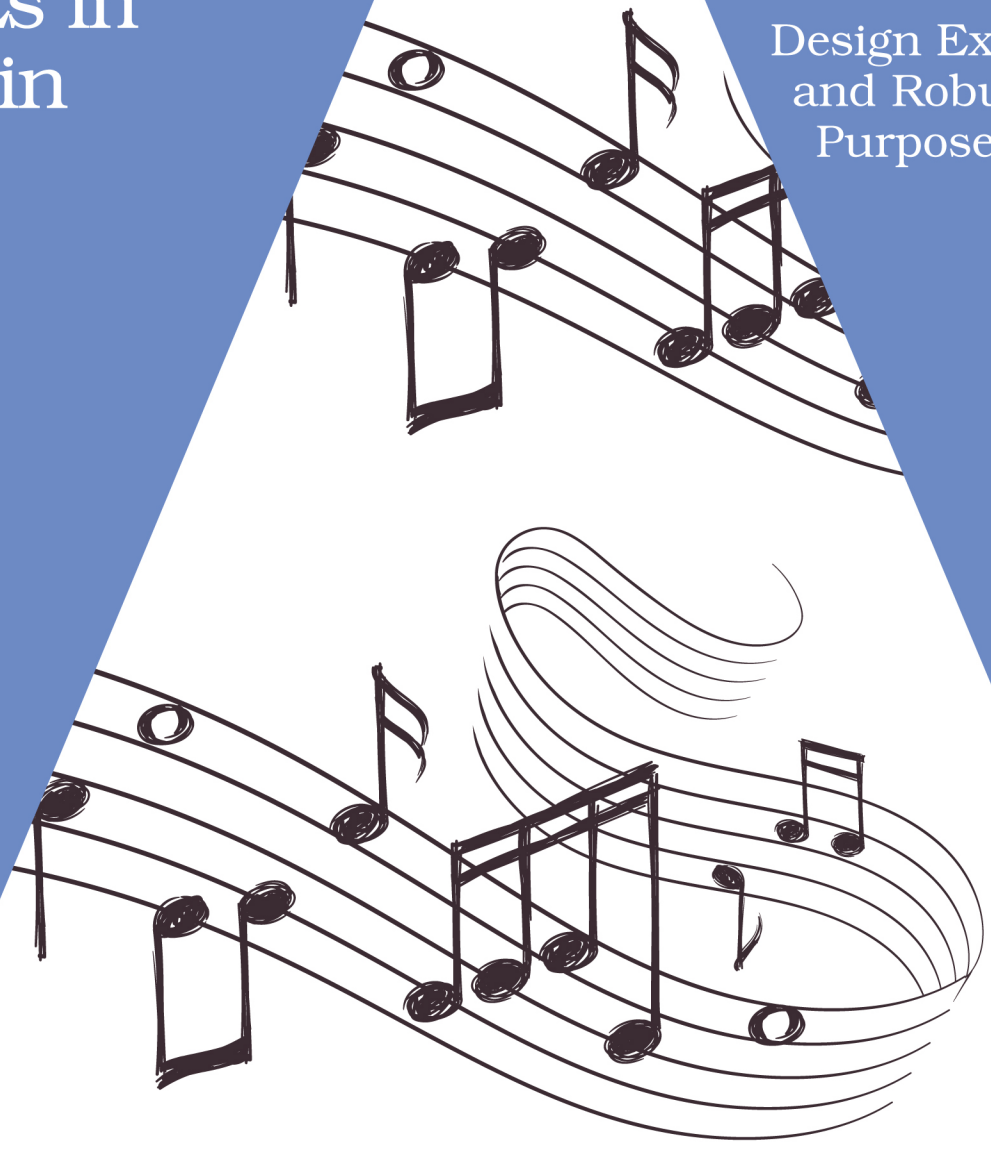
The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Programming DSLs in Kotlin

Design Expressive and Robust Special Purpose Code



Venkat Subramaniam

edited by Jacquelyn Carter

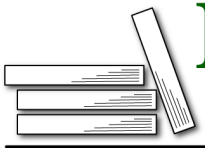
Programming DSLs in Kotlin

Design Expressive and Robust Special Purpose Code

Venkat Subramaniam

The Pragmatic Bookshelf

Raleigh, North Carolina



For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit *https://pragprog.com*.

The team that produced this book includes:

CEO: Dave Rankin
COO: Janet Furlow
Managing Editor: Tammy Coron
Development Editor: Jacquelyn Carter
Copy Editor: L. Sakhi MacMillan
Founders: Andy Hunt and Dave Thomas

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-793-5
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—March 2021

Both in life and programming, it's not just what we say but how we say it that matters. The "how" in domain-specific languages (DSLs) is intriguing. It elevates the task of designing DSLs into an art form, where we can focus on fluency, conciseness, and esthetics of the syntax without compromising the engineering concerns of semantics, correctness, completeness, robustness, and resilience.

Design should be functional and serve its essential purpose. A good design is also easy, intuitive, convenient, and pleasant to use. Kotlin is a wonderful language that can meet those goals and, at the same time, reduce the effort it takes to implement DSLs.

In this book, you'll learn how to create your own DSLs in Kotlin that are easy and intuitive for the users. You'll learn how to build fluency and how to extend the language so you can easily add domain-specific properties and functions. We'll quickly move to the techniques of adding an execution context for the DSLs, so as to integrate the code in the DSL into the larger scope of the application in which they're used. Finally, we'll look at ways to make the code robust and handle errors, both ahead of execution and at runtime.

Let's start with an example of a DSL. The following Cascading Style Sheets (CSS) syntax is intriguing:

```
a.active {
  color: #FF0000;
}
```

It's simple, easy to both write and read, and we don't have to be a programmer to work with it. Once we know the context and a few nuances of the syntax, we can be on our way to use CSS to tailor the appearances of websites. That's a pretty darn good domain-specific language, albeit an external DSL. It's called *external* since it's not written on top of any other language; it stands on its own.

Now, consider the following:

```
Robot operate {
  it turns left
  it runs fast
  it turns right
}
```

That too is easy to both write and read, and we don't have to be a programmer to use it—that's our own DSL. It's an *internal* DSL since it's written on top of a host general purpose language, Kotlin in this case.

Internal DSLs, which are the focus of this book, remove the need to deal with tokenizers and parsers but bring in a set of their own challenges—we have to work within the limitations of the host language. Curious?

To create internal DSLs, you need:

- A good knowledge of the host language to exploit its capabilities
- Some tricks to work around any limitations
- A strong caffeinated beverage to perk you up as you tackle the challenges

In this book, you'll find sufficient details for the first and substantial portions of the second, but you're expected to bring along the third.

Focus on the Key Characteristics

When designing DSLs, we must focus on two main characteristics:

- Fluency
- Implicit context

Let's quickly take a look at what these two things are about.

Fluency

The following feels like code written in a C-like language:

```
Robot.operate()
```