Extracted from:

Programming Kotlin

Creating Elegant, Expressive, and Performant JVM and Android Applications

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Programming Kotlin 1.3

Create Elegant, Expressive, and Performant JVM and Android Applications

Venkat Subramaniam

edited by Jacquelyn Carter

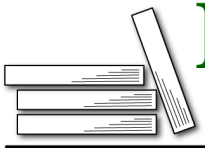
Programming Kotlin

Creating Elegant, Expressive, and Performant JVM and Android Applications

Venkat Subramaniam

The Pragmatic Bookshelf

Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

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

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

ISBN-13: 978-1-68050-635-8
Book version: B1.0—October 24, 2018

Ah, Kotlin—that's an island off St. Petersburg, Russia, but this book is about its namesake programming language. Programmers who use Kotlin don't just like the language—they say they love it. What are the reasons for such affection? That's the question we'll quickly start with. Then we'll be on our way to install the Kotlin Software Developer Kit (SDK), write some code, compile and execute it, so we can see it working.

Imagine taking the best of many different languages—C++, C#, Erlang, Groovy, Java, JavaScript, Python, Ruby, Scala, Smalltalk,—throwing them into a blender and turning it on; the resulting cocktail is Kotlin. The strength of Kotlin is in its diversity.

Andrey Breslav[1] and the team of developers behind the language at JetBrains[2] set out to create a fluent, expressive, pragmatic, and easy to use language that is less verbose than many mainstream languages. As programmers pick up Kotlin they quickly recognize good parts of their familiar languages and, at the same time, are intrigued by other awesome capabilities they've not been exposed to before. The familiar ideas in Kotlin makes programmers feel at home as they learn and adopt the language, yet the ideas that are new to them make them more productive compared to the languages they're used to. That's part of the reason why programmers are passionate about Kotlin.

The biggest uptick in interest for Kotlin came right after Google's announcement that Kotlin is an official language for Android development.[3] An endorsement from Google is certainly significant, but there are more reasons to be excited about Kotlin.

Kotlin is one of the few languages that can be used for server-side, mobile/Android, and front-end development. Code, written appropriately, can compile down to Java byte-code or may be transpiled (compiled from the source code of one language to the source code of another language) to JavaScript. Kotlin/Native supports targeting platforms including iOS, macOS, Linux, Windows, and WebAssembly, to compile your source code to native binaries. That makes Kotlin one of the few languages you can use for full-stack development.

As you journey through Kotlin, you may recognize a number of these features and trace their roots:

1. https://twitter.com/abreslav
2. https://www.jetbrains.com/
3. https://developer.android.com/kotlin

- Though syntactically different, Kotlin is semantically similar to Java, making it easy for Java programmers to adapt.
- Without inheriting from a class you may add your own domain specific convenience methods to classes. These methods, called extension functions, may be used just like methods that are part of the original class, with full Integrated Development Environment (IDE) support. That's like C# style extension methods in Kotlin, although Kotlin has much richer capabilities.
- Delegation is often a better design tool than inheritance to reuse code. Kotlin, inspired by languages like Groovy and Ruby, is quite versatile in the ways you can delegate method calls from one object to another, without compromising type safety.
- You can use the concise and elegant pattern matching syntax in Kotlin, which is similar to Erlang and Scala syntax, instead of the more verbose series of nested `if-else` statements.
- Extending existing functions and methods is easy in Kotlin (though it requires recompilation due to binary incompatibility), thanks to its default parameters capabilities, similar to JavaScript, Scala, Ruby, and Python.
- Named arguments, like in Groovy, Scala, and Ruby, makes the code highly expressive, easier to read, and less error prone.
- Where it makes sense, you may overload operators on your own classes or on third-party classes, much like in languages like C++ and Groovy.
- The elegance, fluency, and concise nature of Kotlin comes together to support creating internal DSLs, similar to languages like Groovy and Ruby, but with full support for static type checking.
- You can write C style procedures, Scala style scripts, Java like OO code, and Smalltalk/Erlang like functional style code in Kotlin.
- Kotlin is leading innovation in the area of asynchronous programming with coroutines and continuations.

These are just a few of the significant features that are prominent in Kotlin.

Reasons to Love Kotlin

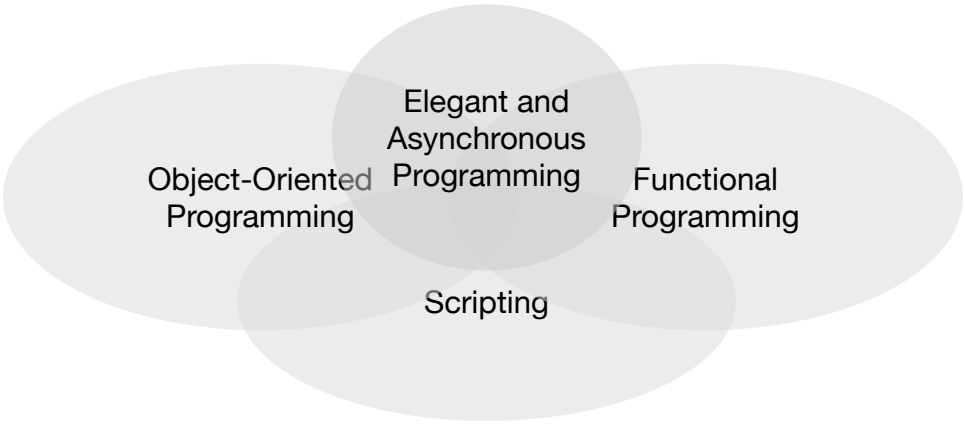
Once you dig in, Kotlin feels more like a Swiss Army Knife than a cocktail—there's so much you can do with this language with so little code. The language supports multiple paradigms. It's statically typed with a healthy dose of strong type inference. It may be compiled to Java byte-code, transpiled

to JavaScript, or it may target native binaries using Kotlin/Native. It is highly fluent and elegant, and it's a charm to work with. Let's further explore the reasons to adopt Kotlin.

Multi-paradigm Programming

Kotlin treats us like an adult, it offers choices and let's us pick the approach that's best suited for the problem at hand. The language has few ceremonies; you're not required to write everything in classes nor are you required to compile every piece of code. The language is largely un-opinionated and offers different programming paradigms for you to choose or even intermix.

You can see the different programming paradigms supported in Kotlin in the following figure.



You can write procedural code, that is code and functions directly in a file, like in JavaScript or C, and you can run it as a script, without any extra compilation steps, but with the exceptional type safety you expect from a statically typed language. The benefit is you can do rapid prototyping of your ideas or illustrate how a particular design pattern may be used, but without being drowned in the ceremonies that other languages often impose. That gives you the shortest time from idea to demo.

Much like in Java, you can create classes and write object-oriented code in Kotlin, but without much boiler plated code. Thus, it takes less code to achieve the same results as in Java. Kotlin guides you along to create your hierarchy of classes intentionally rather than accidentally. Classes are final by default and if you intend a class to serve as a base class, you must specify that explicitly. Furthermore, delegation is a language level syntax, so we can select prudently between inheritance and delegation.

Though the mainstream world has predominantly used the imperative style of programming, code written using the functional style is less complex, more expressive, concise, elegant, and fluent. Kotlin provides exceptional support for both the imperative and functional style of programming. You can readily benefit from the key functional capabilities you're used to from other languages that support the paradigm.

You can make immediate use of the elegance and low ceremony of Kotlin syntax to create internal domain specific languages (DSLs). In addition to creating your own fluent APIs, you can also benefit from fluency in a number of different libraries, for example the Kotlin API for the Spring framework.[4]

In addition to programming concurrency using the Java Developer Kit (JDK), you may also write asynchronous programs using Kotlin's coroutines. This feature is highly critical for applications that make use of cloud services or are deployed as microservices; it allows you to interact efficiently with other services to exchange data asynchronously.

Statically Typed with Type Inference

Statically typed languages offer compile time type safety, but Kotlin walks a few extra miles to prevent common errors that are likely in other statically typed languages. For instance, the Kotlin type system distinguishes nullable types from non-nullable types. It also has very strong type inference, in the same vein of languages like Scala, F#,and Haskell. You don't have to spend your time keying in type details that are obvious to everyone looking at the code. At the same time, when the type may not be 100% clear, Kotlin requires that you specify it. It's not overly zealous—it supports type inference to the right measure, so we can be productive and at the same time the code can be type safe.

One Language for Full Stack Development

Just like `javac` compiles Java source code to byte-code to run on the Java Virtual Machine (JVM), `kotlinc-jvm` compiles the Kotlin code to byte-code to run on virtual machines. You can write your server side code and Android applications using Kotlin, and target the specific version of the virtual machine that you'd like to use for deployment. Thus your Spring code on the backend and your Android or iOS native code on the devices all may be written using the same language. Where necessary, you may also intermix Kotlin code with Java code—no legacy code has to be left behind.

4. https://spring.io/blog/2017/08/01/spring-framework-5-kotlin-apis-the-functional-way

Kotlin also transpiles to JavaScript. You can write Kotlin code that may transform to JavaScript and run in Node.js on the server side, or in browsers on the Web frontend.

Furthermore, using Kotlin/Native you may compile code to native binary to run on targeted platforms and to WebAssembly to run within browsers.

Fluent and Elegant

Some languages impose high ceremony and force you to create boiler plated code. Some developers argue that IDEs remove the burden of having to write that code manually. True, but even if the IDEs were to vomit that boiler plate code, your team has to spend the time and effort maintaining that code each day. Languages like Scala, Groovy, and Ruby synthesize code that programmers will have to otherwise write. Likewise, Kotlin creates a few things for you, like fields, getters, and setters, from convention. Less effort, better results.

Kotlin makes a few things optional. For example, a semicolon is optional. Not having to place the ; symbol leads to a more fluent syntax—a must for creating easy to read internal DSLs. Furthermore, Kotlin provides an infix annotation that we can use, making the dot and parenthesis optional. With these capabilities you can write fluent and elegant code like:

```
operate robot {
  turn left
  turn right
  move forward
}
```

Yes, it's not some fiction, that's real Kotlin code—you'll learn to create your own fluent code like this later in the book, without the need for any parsers or external tools.

Why Should You Choose Kotlin?

There are many reasons why Kotlin may be a right choice for your current project or the next one:

- Kotlin delivers on the "less is more" promise, you write less boiler plated code. The less code you write, the less your team has to maintain, and there are fewer errors to deal with.
- Kotlin gives you the freedom to mix the imperative and functional styles of programming; you pick what's best for the problem at hand. Also, you may write it in one way and refactor it later as you desire—make it work, then make it better real soon.

- Kotlin offers lot more compile time safety when compared to a lot of other statically typed languages. The code you write will fail less and fail fast—during compilation rather than runtime. This is one of the reasons why the Spring[5] team decided to embrace Kotlin.
- Kotlin coroutines makes it a lot easier to create high performance asynchronous code compared to what's available in the Java Development Kit (JDK).
- Some of the features that are scheduled to appear in future versions of Java are already in Kotlin—you can experience and benefit from future Java right now by using Kotlin.
- You may intermix Kotlin and Java code in your projects—using Kotlin is not an all-or-nothing proposition.
- You can not only use fluent DSL-like syntax to interact with APIs, like in the Spring Kotlin API, but also design your own APIs to be fluent and expressive for programmers who use your code.
- You can reduce duplication among parts of your system with Kotlin. For example, the same business rules that check users' input may be compiled to Java bytecode for backend validations, transpiled to JavaScript for frontend validation, compiled to native binaries to run on targeted platforms like iOS and Android, or to WebAssembly to run within browsers.
- Finally, Kotlin is a great choice for Android development since it's an official language for that platform.

You'll see more about why Kotlin is exciting in the many pages of this book. Buckle up, it's going to be a fun ride. Let's start by getting the SDK installed so we can start writing Kotlin.

5. https://spring.io/blog/2017/01/04/introducing-kotlin-support-in-spring-framework-5-0