Table of Contents for Programming Scala, 2nd Edition

safaribooksonline.com/library/view/programming-scala-2nd/9781491950135/pr03.html

Preface

Programming Scala introduces an exciting and powerful language that offers all the benefits of a modern object model, *functional programming* (FP), and an advanced type system, while leveraging the industry's investment in the Java Virtual Machine (JVM). Packed with code examples, this comprehensive book teaches you how to be productive with Scala quickly, and explains what makes this language ideal for today's scalable, distributed, component-based applications that support concurrency and distribution. You'll also learn how Scala takes advantage of the advanced JVM as a platform for programming languages.

Learn more at http://programming-scala.org or at the book's catalog page.

Welcome to Programming Scala, Second Edition

Programming Scala, First Edition was published five years ago, in the fall of 2009. At the time, it was only the third book dedicated to Scala, and it just missed being the second by a few months. Scala version 2.7.5 was the official release, with version 2.8.0 nearing completion.

A lot has changed since then. At the time of this writing, the Scala version is 2.11.2. Martin Odersky, the creator of Scala, and Jonas Bonér, the creator of Akka, an actor-based concurrency framework, cofounded Typesafe to promote the language and tools built on it.

There are also a lot more books about Scala. So, do we really need a second edition of this book? Many excellent beginner's guides to Scala are now available. A few advanced books have emerged. The encyclopedic reference remains *Programming in Scala*, Second Edition, by Odersky et al. (Artima Press).

Yet, I believe *Programming Scala, Second Edition* remains unique because it is a *comprehensive* guide to the Scala language and ecosystem, a guide for beginners to advanced users, and it retains the focus on the pragmatic concerns of working professionals. These characteristics made the first edition popular.

Scala is now used by many more organizations than in 2009 and most Java developers have now heard of Scala. Several persistent questions have emerged. Isn't Scala complex? Since Java 8 added significant new features found in Scala, why should I switch to Scala?

I'll tackle these and other, real-world concerns. I have often said that I was *seduced by Scala*, warts and all. I hope you'll feel the same way after reading *Programming Scala, Second Edition*.

How to Read This Book

Because this is a comprehensive book, beginning readers don't need to read the whole thing to be productive with Scala. The first three chapters, *Zero to Sixty: Introducing Scala*, *Type Less, Do More*, and *Rounding Out the Basics*, provide a quick summary of core language features. The fourth and fifth chapters, *Pattern Matching* and *Implicits*, begin the more in-depth coverage with two fundamental tools that you'll use every day in your Scala code.

If you're new to functional programming (FP), Chapter 6 provides an introduction to this important approach to software development, as implemented in Scala. Next is Chapter 7, which explains Scala's extensions to the venerable `for` loop and how it provides a succinct syntax for sophisticated, idiomatic functional code.

Then we turn to Scala's support for *object-oriented programming* (OOP) in Chapter 8. I put the FP chapter before the OOP chapters to emphasize the importance of FP in solving many software development problems of our time. It would be easy to use Scala as a "better object-oriented Java," but that would ignore its most powerful tools! Most of this chapter will be conceptually easy to understand, as you learn how Scala defines classes, constructors, etc. that are familiar in Java.

Chapter 9 explores Scala's ability to compose behaviors using *traits*. Java 8 adds a subset of this functionality through its enhancements to interfaces, partially inspired by Scala traits. Even experienced Java programmers will need to understand this material.

The next four chapters, 10 through 13, *The Scala Object System, Part I*, *The Scala Object System, Part II*, *The Scala Collections Library*, and *Visibility Rules*, walk through Scala's object model and library types in detail. You should read Chapter 10 carefully, because it contains essential information to master early on. However, Chapter 11 goes into less critical information, the details of properly implementing nontrivial type hierarchies. You might skim that chapter the first time through the book. Chapter 12 discusses the design of the collections and some useful information about using them wisely. Again, skim this chapter if you're new to Scala and come back to it when you're trying to master the details of the collections API. Finally, Chapter 13 explains in detail Scala's fine-grained extensions to Java's notions of public, protected, and private visibility. Skim this chapter.

Next we move into more advanced territory, starting with Chapter 14 and Chapter 15, which cover Scala's sophisticated type system. I've divided the coverage into two chapters: the first chapter covers concepts that new Scala programmers will need to learn relatively quickly, while the second chapter covers more advanced material that can be deferred until later.

Similarly, Chapter 16, *Advanced Functional Programming*, covers more advanced mathematical concepts that the average Scala developer won't need initially, such as *Monad* and *Functor* from *Category Theory*.

Chapter 17, *Tools for Concurrency*, will be useful for developers of large-scale services that require concurrency for resiliency and scalability (most of us, actually). It discusses Akka, a rich actor-based concurrency model, and library types such as `Futures` for writing asynchronous code.

Chapter 18, *Scala for Big Data*, makes the case that a *killer app* for Scala, and functional programming in general, is *Big Data*, or any data-centric computation.

Chapters 19 and 20, *Dynamic Invocation in Scala* and *Domain-Specific Languages in Scala*, go together. They are somewhat advanced topics, discussing tools for construction of rich *domain-specific languages*.

Chapter 21, *Scala Tools and Libraries*, discusses tools like IDEs and third-party libraries. If you're new to Scala, read about IDE and editor support, and the section on *SBT*, the de facto build tool for Scala projects. Use the library lists for reference later on. Chapter 22, *Java Interoperability*, will be useful for teams that need to interoperate between Java and Scala code.

I wrote Chapter 23, *Application Design*, for architects and software leads to share my thoughts about good application design. I believe the traditional model of relatively fat JAR files with complex object graphs is a broken model and needs to go.

Finally, the most advanced topic in the book is covered in Chapter 24, *Metaprogramming: Macros and Reflection*. You can definitely skip this chapter if you're a beginner.

The book concludes with Appendix A for further reading.

What Isn't Covered?

A focus of the latest 2.11 release is modularizing the library to decompose it into smaller JAR files, so it's easier to exclude unneeded code from deployment in space-sensitive environments (e.g., mobile devices). Some previously deprecated packages and types of the library were also removed. Other parts are deprecated in the 2.11 release, often because they are no longer maintained and there are better, third-party alternatives.

Hence, we won't discuss the following packages that are deprecated in 2.11:

The following were deprecated in Scala 2.10 and removed in 2.11:

`scala.util.automata`
: For building deterministic, finite automatons (DFAs) from regular expressions.

`scala.util.grammar`
: Part of a parsing library.

`scala.util.logging`
: The recommendation is to use one of the many third-party, actively maintained logging libraries for the JVM.

`scala.util.regexp`
: Regular expression parsing. The `scala.util.matching` package with regular expression support has been enhanced instead.

The .NET compiler backend
: For a while, the Scala team worked on a compiler backend and library for the .NET runtime environment, but interest in this port has waned, so it was discontinued.

We won't discuss *every* package and type in the library. Here is a partial list of omissions for space and other reasons:

`scala.swing`
: Wrapper around the Java Swing library. While still maintained, it is rarely used.

`scala.util.continuations`
: Compiler plug-in for continuation-passing style (CPS) code generation. It is a specialized tool with limited adoption.

The `App` and `DelayedInit` traits
: This pair of types was meant to conveniently implement `main` (entry-point) types, the analog of `static main` methods in Java classes. However, they sometimes cause surprising behavior, so I don't recommend using them. Instead, I'll write `main` routines in the normal, idiomatic Scala way.

`scala.ref`
: Wrappers around Java types such as `WeakReference`, which corresponds to `java.lang.ref.WeakReference`.

`scala.runtime`
: Types used as part of the library implementation.

`scala.util.hashing`
: Hashing algorithms.

Welcome to Programming Scala, First Edition

Programming languages become popular for many reasons. Sometimes, programmers on a given platform prefer a particular language, or one is institutionalized by a vendor. Most Mac OS programmers use Objective-C. Most Windows programmers use C++ and .NET languages. Most embedded-systems developers use C and C++.

Sometimes, popularity derived from technical merit gives way to fashion and fanaticism. C++, Java, and Ruby have been the objects of fanatical devotion among programmers.

Sometimes, a language becomes popular because it fits the needs of its era. Java was initially seen as a perfect fit for browser-based, rich client applications. Smalltalk captured the essence of object-oriented programming as that model of programming entered the mainstream.

Today, concurrency, heterogeneity, always-on services, and ever-shrinking development schedules are driving interest in functional programming. It appears that the dominance of object-oriented programming may be over. Mixing paradigms is becoming popular, even necessary.

We gravitated to Scala from other languages because Scala embodies many of the optimal qualities we want in a general-purpose programming language for the kinds of applications we build today: reliable, high-performance, highly concurrent Internet and enterprise applications.

Scala is a multiparadigm language, supporting both object-oriented and functional programming approaches. Scala is scalable, suitable for everything from short scripts up to large-scale, component-based applications. Scala is sophisticated, incorporating state-of-the-art ideas from the halls of computer science departments worldwide. Yet Scala is practical. Its creator, Martin Odersky, participated in the development of Java for years and understands the needs of professional developers.

Both of us were seduced by Scala, by its concise, elegant, and expressive syntax and by the breadth of tools it put at our disposal. In this book, we strive to demonstrate why all these qualities make Scala a compelling and indispensable programming language.

If you are an experienced developer who wants a fast, thorough introduction to Scala, this book is for you. You may be evaluating Scala as a replacement for or complement to your current languages. Maybe you have already decided to use Scala, and you need to learn its features and how to use it well. Either way, we hope to illuminate this powerful language for you in an accessible way.

We assume that you are well versed in object-oriented programming, but we don't assume that you have prior exposure to functional programming. We assume that you are experienced in one or more other programming languages. We draw parallels to features in Java, C#, Ruby, and other languages. If you know any of these languages, we'll point out similar features in Scala, as well as many features that are new.

Whether you come from an object-oriented or functional programming background, you will see how Scala elegantly combines both paradigms, demonstrating their complementary nature. Based on many examples, you will understand how and when to apply OOP and FP techniques to many different design problems.

In the end, we hope that you too will be seduced by Scala. Even if Scala does not end up becoming your day-to-day language, we hope you will gain insights that you can apply regardless of which language you are using.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

Tip

This element signifies a tip or suggestion.

Note

This element signifies a general note.

Warning

This element indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Programming Scala, Second Edition* by Dean Wampler and Alex Payne. Copyright 2015 Kevin Dean Wampler and Alex Payne, 978-1-491-94985-6."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

Getting the Code Examples

You can download the code examples from GitHub. Unzip the files to a convenient location. See the *README* file in the distribution for instructions on building and using the examples. (I'll summarize those instructions in the first chapter.)

Some of the example files can be run as scripts using the `scala` command. Others must be compiled into class files. Some files contain deliberate errors and won't compile. I have adopted a filenaming convention to indicate each of these cases, although as you learn Scala it should become obvious from the contents of the files, in most cases:

**.scala*
: The standard Scala file extension is *.scala*, but that doesn't distinguish between source files that must be compiled using `scalac`, script files you run directly with `scala`, or deliberately invalid code files used at times in this book. So, in the example code, any file with the *.scala* extension must be compiled separately, like you would compile Java code.

**.sc*
: Files that end in *.sc* can be run as scripts on a command line using `scala`, e.g., `scala foo.sc`. You can also start `scala` in the interpreter mode and load any script file in the interpreter using the `:load filename` command. Note that this naming convention is *not* a standard convention in the Scala community, but it's used here because the *SBT* build will ignore these files. Also, this file extension is used by the new IDE *worksheet* feature we will discuss in the next chapter. So, it's a convenient hack. To be clear, you will normally

use *.scala* as the extension of scripts and code files alike.

**.scalaX* and **.scX*
: Some example files contain deliberate errors that will cause compilation errors. Rather than break the build for the examples, those files will use the extension *.scalaX* for code files or *.scX* for scripts. Again, this is not an industry convention. These files will also have embedded comments to explain what's wrong with them.

Safari® Books Online

Note

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at http://bit.ly/programmingScala_2E.

To comment or ask technical questions about this book, send email to bookquestions@oreilly.com.

For more information about our books, courses, conferences, and news, see our website at http://www.oreilly.com.

Find us on Facebook: http://facebook.com/oreilly

Follow us on Twitter: http://twitter.com/oreillymedia

Watch us on YouTube: http://www.youtube.com/oreillymedia

Acknowledgments for the Second Edition

As I, Dean Wampler, worked on this edition of the book, I continued to enjoy the mentoring and feedback from many of my Typesafe colleagues, plus the valuable feedback from people who reviewed the early-access releases. I’m especially grateful to Ramnivas Laddad, Kevin Kilroy, Lutz Huehnken, and Thomas Lockney, who reviewed drafts of the manuscript. Thanks to my long-time colleague and friend, Jonas Bonér, for writing an updated Foreword for the book.

And special thanks to Ann who allowed me to consume so much of our personal time with this project. I love you!

Acknowledgments for the First Edition

As we developed this book, many people read early drafts and suggested numerous improvements to the text, for which we are eternally grateful. We are especially grateful to Steve Jensen, Ramnivas Laddad, Marcel Molina, Bill Venners, and Jonas Bonér for their extensive feedback.

Much of the feedback we received came through the Safari Rough Cuts releases and the online edition available at http://programmingscala.com. We are grateful for the feedback provided by (in no particular order) Iulian Dragos, Nikolaj Lindberg, Matt Hellige, David Vydra, Ricky Clarkson, Alex Cruise, Josh Cronemeyer, Tyler Jennings, Alan Supynuk, Tony Hillerson, Roger Vaughn, Arbi Sookazian, Bruce Leidl, Daniel Sobral, Eder Andres Avila, Marek Kubica, Henrik Huttunen, Bhaskar Maddala, Ged Byrne, Derek Mahar, Geoffrey Wiseman, Peter Rawsthorne, Geoffrey Wiseman, Joe Bowbeer, Alexander Battisti, Rob Dickens, Tim MacEachern, Jason Harris, Steven Grady, Bob Follek, Ariel Ortiz, Parth Malwankar, Reid Hochstedler, Jason Zaugg, Jon Hanson, Mario Gleichmann, David Gates, Zef Hemel, Michael Yee, Marius Kreis, Martin Süsskraut, Javier Vegas, Tobias Hauth, Francesco Bochicchio, Stephen Duncan Jr., Patrik Dudits, Jan Niehusmann, Bill Burdick, David Holbrook, Shalom Deitch, Jesper Nordenberg, Esa Laine, Gleb Frank, Simon Andersson, Patrik Dudits, Chris Lewis, Julian Howarth, Dirk Kuzemczak, Henri Gerrits, John Heintz, Stuart Roebuck, and Jungho Kim. Many other readers for whom we only have usernames also provided feedback. We wish to thank Zack, JoshG, ewilligers, abcoates, brad, teto, pjcj, mkleint, dandoyon, Arek, rue, acangiano, vkelman, bryanl, Jeff, mbaxter, pjb3, kxen, hipertracker, ctran, Ram R., cody, Nolan, Joshua, Ajay, Joe, and anonymous contributors. We apologize if we have overlooked anyone!

Our editor, Mike Loukides, knows how to push and prod gently. He’s been a great help throughout this crazy process. Many other people at O’Reilly were always there to answer our questions and help us move forward.

We thank Jonas Bonér for writing the Foreword for the book. Jonas is a longtime friend and collaborator from the aspect-oriented programming (AOP) community. For years, he has done pioneering work in the Java community. Now he is applying his energies to promoting Scala and growing that community.

Bill Venners graciously provided the quote on the back cover. The first published book on Scala, *Programming in Scala* (Artima), that he cowrote with Martin Odersky and Lex Spoon, is indispensable for the Scala developer. Bill has also created the wonderful ScalaTest library.

We have learned a lot from fellow developers around the world. Besides Jonas and Bill, Debasish Ghosh, James Iry, Daniel Spiewak, David Pollack, Paul Snively, Ola Bini, Daniel Sobral, Josh Suereth, Robey Pointer, Nathan Hamblen, Jorge Ortiz, and others have illuminated dark corners with their blog entries, forum discussions, and personal conversations.

Dean thanks his colleagues at Object Mentor and several developers at client sites for many stimulating discussions on languages, software design, and the pragmatic issues facing developers in industry. The members of the Chicago Area Scala Enthusiasts (CASE) group have also been a source of valuable feedback and inspiration.

Alex thanks his colleagues at Twitter for their encouragement and superb work in demonstrating Scala’s effectiveness as a language. He also thanks the Bay Area Scala Enthusiasts (BASE) for their motivation and community.

Most of all, we thank Martin Odersky and his team for creating Scala.