Scala for Java Developers

A Practical Primer

Toby Weston

Apress®

Scala for Java Developers

A Practical Primer

Toby Weston

Apress®

Scala for Java Developers: A Practical Primer

Toby Weston
London, United Kingdom

ISBN-13 (pbk): 978-1-4842-3107-4 ISBN-13 (electronic): 978-1-4842-3108-1
https://doi.org/10.1007/978-1-4842-3108-1

Library of Congress Control Number: 2017963118

Cover image by Freepik (www.freepik.com)

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Steve Anglin
Development Editor: Matthew Moodie
Technical Reviewer: Jeff Friesen
Coordinating Editor: Mark Powers
Copy Editor: Francesca Louise White

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484231074. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

In memory of Félix Javier García López

Table of Contents

About the Author

Toby Weston is an independent software developer based in London. He specializes in Java and Scala development, working in agile environments. He's a keen blogger and writer, having written for JAXenter and authored the books *Essential Acceptance Testing* (Leanpub) and *Learning Java Lambdas* (Packt).

About the Technical Reviewer

Jeff Friesen is a freelance teacher and software developer with an emphasis on Java. In addition to authoring *Java I/O, NIO and NIO.2* (Apress) and *Java Threads and the Concurrency Utilities* (Apress), Jeff has written numerous articles on Java and other technologies (such as Android) for JavaWorld (`JavaWorld.com`), informIT (`informIT.com`), Java.net, SitePoint (`SitePoint.com`), and other websites. Jeff can be contacted via his website at `JavaJeff.ca`. or via his LinkedIn profile (`www.linkedin.com/in/javajeff`).

Acknowledgments

Thanks go out to James Maggs, Alex Luker, Rhys Keepence and Xuemin Guan for their feedback on early drafts and Lee Benfield for building the excellent CFR decompiler and sharing it with the community.

Additionally, thank you to Amy Brown for providing an early copyedit of this book.

Preface

Audience

This book is for Java developers looking to transition to programming in Scala.

The Structure of the Book

The book is split into four parts: a tour of Scala, a comparison between Java and Scala, a closer look at Scala-specific features and functional programming idioms, and finally a discussion about adopting Scala into existing Java teams.

In Part I, we're going to take a high-level tour of Scala. You'll get a feel for the language's constructs and how Scala is similar in a lot of ways to Java, yet very different in others. We'll take a look at installing Scala and using the interactive interpreter and we'll go through some basic syntax examples.

Part II talks about key differences between Java and Scala. We'll look at what's missing in Scala compared to Java, vice versa, and how concepts translate from one language to another.

Then in Part III, we'll talk about some of the language features that Scala offers that aren't found in Java. This part also talks a little about functional programming idioms.

Finally, we'll talk about adopting Scala into legacy Java projects and teams. It's not always an easy transition, so we'll look at why you would want to, and some of the challenges you might face.

Compiling Code Fragments

Later in the book, I introduce the Scala REPL: an interactive tool for working with Scala and the Scala version of Java's JShell. You'll see REPL sessions prefixed with `scala>`.

When you do so, you can expect to be able to type in the code following `scala>` in the REPL verbatim, hit enter, and see the results. An example follows.

```
// an example REPL session
scala> val x = 6 * 9
x: Int = 54
```

If you don't see the `scala>` prefix, assume the fragment may depend on previous code examples. I've tried to introduce these logically, balancing the need to show complete listings with trying to avoid pages and pages of dry code listings.

If things don't make sense, always refer to the full source code. In short, you may find it useful to consult the full source while you read.

Larger Fragments in the REPL

If you'd like to transpose some of the larger code fragments into the REPL, you may notice compiler errors on hitting enter. The REPL is geared up to evaluate a line at a time. Pasting larger fragments or typing in long examples requires you to be in *paste mode*.

Typing `:paste` enters paste mode, allowing you to type multiple lines. Pressing `Ctrl + D` exits paste mode and compiles the code.

```
scala> :paste
// Entering paste mode (ctrl-D to finish)
val x = 4
val y = 34
x + y * 2
// press Ctrl + D
res1: Int = 72
```

Infrequently, you may notice an ellipsis (`...`) or triple question marks (`???`) in code fragments. When you see this, it indicates that the fragment is incomplete and will usually be followed by additional code to fill in the blanks. It probably won't compile. It's used when I've felt that additional code would be uninteresting, distracting, or when I'm building up examples.

Source Code

The source code for this book is available at GitHub: `https://github.com/tobyweston/learn-scala-java-devs`. You can clone the repository or download an archive directly from the site.

The source code is licensed under Apache 2.0 open source license.

Source Code Appendix

The book often includes partial code fragments in an attempt to avoid reams of distracting "scaffolding" code. Code may refer to previous fragments and this may not be immediately obvious. Try to read the code as if each example builds on what's gone before.

If this style isn't for you, I've also included a code listing appendix. This offers complete listings for the more complex code, in case you want to see all the code in one place. It's not there to pad out the book. Honest.

PART I

Scala Tour

Welcome to *Scala for Java Developers: A Practical Primer*. This book will help you transition from programming in Java to programming in Scala. It's designed to help Java developers get started with Scala without necessarily adopting all of the more advanced functional programming idioms.

Scala is both an object-oriented language and a functional language and, although I do talk about some of the advantages of functional programming, this book is more about being productive with imperative Scala than getting to grips with functional programming. If you're already familiar with Scala but are looking to make the leap to pure functional programming, this probably isn't the book for you. Check out the excellent *Functional Programming in Scala*[1] by Paul Chiusano and Rúnar Bjarnason instead.

The book often compares "like-for-like" between Java and Scala. So, if you're familiar with doing something a particular way in Java, I'll show how you might do the same thing in Scala. Along the way, I'll introduce the Scala language syntax.

[1] http://amzn.to/1Aegnwj

CHAPTER 1

The Scala Language

Scala is both a functional programming language *and* an object-oriented programming language. As a Java programmer, you'll be comfortable with the object-oriented definition: Scala has classes, objects, inheritance, composition, polymorphism—all the things you're used to in Java.

In fact, Scala goes somewhat further than Java. There are no "non"-objects. Everything is an object, so there are no primitive types like `int` and no static methods or fields. Functions are objects and even *values* are objects.

Scala can be accurately described as a functional programming language because it allows and promotes the use of techniques important in functional programming. It provides language level features for things like immutability and programming functions without side effects.

Traditional functional programming languages like Lisp or Haskel *only* allow you to program using these techniques. These are often referred to as *pure* functional programming languages. Scala is not *pure* in this sense; it's a hybrid. For example, you can still work with mutable data, leverage the language to work with immutable data, or do both. This is great for flexibility and easy adoption but not too great for consistency and uniformity of design.

As a Functional Programming Language

In general, functional programming languages support:

1. First-class and higher-order functions.
2. Anonymous functions (lambdas).
3. Pure functions and immutable data.

It can be argued that Java supports these characteristics and certainly Java has been trying to provide better support. However, any movement in this direction has felt like an after thought and has generally resulted in verbose syntax or tension with existing idioms and language APIs.

It is unlikely that people will ever describe Java as a functional programming language despite it's advancements. Partly because it's clunky to use in this style and partly because of it's long history as an object-oriented language.

Scala on the other hand was designed as a functional programming language from day one. It has better language constructs and library support so it feels more natural when coding in a functional style. For example, it has keywords to define immutable values and the library collection classes are all immutable by default.

The Past

Scala started life in 2001 as a research project at EPFL in Switzerland. It was released publicly in 2004[2] after an internal release in 2003. The project was headed by Martin Odersky, who'd previously worked on Java generics and the Java compiler for Sun Microsystems.

It's quite rare for an academic language to cross over into industry, but Odersky and others launched Typesafe Inc. (later renamed Lightbend Inc.), a commercial enterprise built around Scala. Since then, Scala has moved firmly into the mainstream as a development language.

Scala offers a more concise syntax than Java but runs on the JVM. Running on the JVM should (in theory) mean an easy migration to production environments; if you already have the Oracle JVM installed in your production environment, it makes no difference if the bytecode was generated from the Java or Scala compiler.

It also means that Scala has Java interoperability built in, which in turn means that Scala can use any Java library. One of Java's strengths over its competitors was always the huge number of open source libraries and tools available. These are pretty much all available to Scala too. The Scala community has that same open source mentality, and so there's a growing number of excellent Scala libraries out there.

[2]See Odersky, "A Brief History of Scala" on Artima and wikipedia for more background.

The Future

Scala has definitely moved into the mainstream as a popular language. It has been adopted by lots of big companies including Twitter, eBay, Yahoo, HSBC, UBS, and Morgan Stanley, and it's unlikely to fall out of favour anytime soon. If you're nervous about using it in production, don't be; it's backed by an international organization and regularly scores well in popularity indexes.

The tooling is still behind Java though. Powerful IDEs like IntelliJ's IDEA and Eclipse make refactoring Java code straightforward but aren't quite there yet for Scala. The same is true of compile times: Scala is a lot slower to compile than Java. These things will improve over time and, on balance, they're not the biggest hindrances I encounter when developing.

Scala's future is tied to the future of the JVM and the JVM is still going strong. Various other functional languages are emerging however; Kotlin and Clojure in particular are interesting and may compete. If you're not interested in JVM based languages but just the benefits of functional programming, Haskel and ELM are becoming more widely adopted in industry.

CHAPTER 2

Installing Scala

Getting Started

There are a couple of ways to get started with Scala.

1. Run Scala interactively with the interpreter.
2. Run shorter programs as shell scripts.
3. Compile programs with the `scalac` compiler.

The Scala Interpreter

Before working with an IDE, it's probably worth getting familiar with the Scala interpreter, or REPL.

Download the latest Scala binaries (from `http://scala-lang.org/downloads`) and extract the archive. Assuming you have Java installed, you can start using the interpreter from a command prompt or terminal window straight away. To start up the interpreter, navigate to the exploded folder and type[3]

```
bin/scala
```

You'll be faced with the Scala prompt.

```
scala> _
```

You can type commands followed by `enter`, and the interpreter will evaluate the expression and print the result. It reads, evaluates, and prints in a loop so it's known as a REPL.

[3]If you don't want to change into the install folder to run the REPL, set the `bin` folder on your path.

© Toby Weston 2018
T. Weston, *Scala for Java Developers*, https://doi.org/10.1007/978-1-4842-3108-1_2

If you type `42*4` and hit `enter`, the REPL evaluates the input and displays the result.

```
scala> 42*4
res0: Int = 168
```

In this case, the result is assigned to a variable called `res0`. You can go on to use this, for example, to get half of `res0`.

```
scala> res0 / 2
res1: Int = 84
```

The new result is assigned to `res1`.

Notice the REPL also displays the type of the result: `res0` and `res1` are both integers (`Int`). Scala has inferred the types based on the values.

If you add `res1` to the end of a string, no problem; the new result object is a string.

```
scala> "Hello Prisoner " + res1
res2: String = Hello Prisoner 84
```

To quit the REPL, type

```
:quit
```

The REPL is a really useful tool for experimenting with Scala without having to go to the effort of creating the usual project files. It's so useful that the community provided a Java REPL[4] as far back as 2013. Interestingly, Oracle followed suit and introduced the official Java REPL called *JShell* in Java 9 in 2017.

Scala Scripts

The creators of Scala originally tried to promote the use of Scala from Unix shell scripts. As competition to Perl, Groovy, or bash scripts on Unix environments it didn't really take off, but if you want to you can create a shell script to wrap Scala.

```
#!/bin/sh
exec scala "$0" "$@"
```

[4] http://www.javarepl.com/

```
3   !#
4   object HelloWorld {
5     def main(args: Array[String]) {
6       println("Hello, " + args.toList)
7     }
8   }
9   HelloWorld.main(args)
```

Don't worry about the syntax or what the script does (although I'm sure you've got a pretty good idea already). The important thing to note is that some Scala code has been embedded in a shell script and that the last line is the command to run.

You'd save it as a `.sh` file—for example, `hello.sh`—and execute it like this:

```
./hello.sh World!
```

The `exec` command on line 2 spawns a process to call `scala` with arguments; the first is the script filename itself (`hello.sh`) and the second is the arguments to pass to the script. The whole thing is equivalent to running Scala like this, passing in a shell script as an argument:

```
scala hello.sh World!
```

scalac

If you'd prefer, you can compile `.scala` files using the Scala compiler.

The `scalac` compiler works just like `javac`. It produces Java bytecode that can be executed directly on the JVM. You run the generated bytecode with the `scala` command. Just like Java though, it's unlikely you'll want to build your applications from the command line.

All the major IDEs support Scala projects, so you're more likely to continue using your favorite IDE. We're not going to go into the details of how to set up a Scala project in each of the major IDEs; if you're familiar with creating Java projects in your IDE, the process will be very similar.

For reference though, here are a few starting points.

- You can create bootstrap projects with Maven and the `maven-scala-plugin`.
- You can create a new Scala project directly within IntelliJ IDEA once you've installed the `scala` plugin (available in the JetBrains repository).
- Similarly, you can create a new Scala project directly within Eclipse once you have the Scala IDE plugin. Typesafe created this and it's available from the usual update sites. You can also download a bundle directly from the scala-lang or scala-ide.org sites.
- You can use SBT and create a build file to compile and run your project. SBT stands for Simple Build Tool and it's akin to Ant or Maven, but for the Scala world.
- SBT also has plugins for Eclipse and IDEA, so you can use it directly from within the IDE to create and manage the IDE project files.