Beginning Scala

Copyright © 2009 by David Pollak

ISBN-13 (pbk): 978-1-4302-1989-7

ISBN-13 (electronic): 978-1-4302-1990-3

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editors: Matthew Moodie, Steve Anglin
Technical Reviewer: Paul Snively
Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Michelle Lowman, Matthew Moodie, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Kylie Johnston
Copy Editor: Hastings Hart
Associate Production Director: Kari Brooks-Copony
Production Editor: Elizabeth Berry
Compositor: Octal Publishing, Inc.
Proofreader: Liz Welch
Indexer: Julie Grady
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 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 `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

Apress and friends of ED books 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 Special Bulk Sales–eBook Licensing web page at `http://www.apress.com/info/bulksales`.

The source code for this book is available to readers at `http://www.apress.com`.

CHAPTER 1

About Scala and How to Install It

Scala is a programming language that provides a best-of-all-worlds experience for developers. Java™ programmers will find that their libraries are fully interoperable with Scala code.[1] Dynamic-language users will find Scala's concise syntax and type inferencing a way to reduce the boilerplate needed when writing programs in Java and C#. Functional programmers will find Scala's powerful type system a great way to reason about code. Scala bridges a lot of the divides in programming languages.

In this chapter, I'll discuss my journey learning Scala as well as where Scala comes from, how to install it, and the Scala community.

Scala: A Great Language for the Java Virtual Machine

I've been writing computer programs for a long time. I've written commercial code in 6502 assembler, BASIC, Pascal, C, C++, Objective-C, Java, C#, Visual Basic, Smalltalk, Ruby, JavaScript, and Scala. Over the years, I've had a couple of "Oh my!" reactions to computer languages.

In 1996, when I first found Java, it was a revelation. I no longer had to worry about freeing memory, and Java had a sane and normal exception-handling mechanism. Overnight, 70 percent of the defects in my programs went away. For many years, I used and loved Java.

1. Scala can call any Java code, subclass any Java class, and implement any Java interface. Java code can call into Scala code if the Scala code subclasses a Java class or implements a Java interface. There are features of Scala that cannot be accessed from Java, including traits with defined methods, classes and methods that have names illegal in Java, and Scala's advanced types.

In the Java 1.0 and 1.1 days, the Java Virtual Machine (JVM) was slow, and garbage collection was a costly operation. Over the years, the JVM matured. The JVM's overall performance improved with HotSpot, and by JDK 1.3, Java application code was as fast as C++ application code. Java programs could run for weeks, months, and in some cases, years without restarting.[2]

■ **NOTE** While there is a lot of discussion of this issue, my experience is that well-tuned Java code runs as fast or faster than well-tuned C and C++ code. Integer, a pure Java spreadsheet that I wrote, performed as well as Excel in Sun benchmarks on a single-processor machine. Integer outperformed Excel by significant margins on symmetric multiprocessing (SMP) machines. Sam Pullara benchmarked Java's String classes against Objective-C's NSString classes and found that Java outperformed native code by a wide margin. See `http://www.theserverside.com/news/thread.tss?thread_id=25743` (the original citation is gone). I agree that a hyper-tuned C or C++ program can outperform a Java program and that Java programs require 100 percent more RAM to perform as well as their C or C++ counterparts, but for any moderately complex project that is not at the operating system kernel level, a JVM program will outperform a C or C++ program.

Over the years, Java, the language, failed to mature. Java stagnated in syntax, and web frameworks built on Java became increasingly top-heavy. It took more and more lines of Java, XML, and other pieces of glue to express simple concepts such as fields and to generate HTML forms from those fields. I used Java on most projects but became increasingly disillusioned with it.

Java 5 brought enumerations and generics. I found these to be welcome additions at the conceptual level, but at the coding level I could no longer use a simple text editor to write Java; I had to use an IDE (integrated development environment). The nasty declaration syntax for `HashMap<String, ArrayList<Integer>> = new HashMap<String, ArrayList<Integer>>();` meant that I really needed a tool that did code completion in order to write the simplest of programs.

Beyond Java

I started searching for a way to express the code in my brain in a simpler, more direct way. I found Ruby and Rails. I felt liberated. Ruby allowed me to express concepts in far fewer lines of code. Rails was so much easier to use than Spring MVC, Hibernate, and the other "streamlined" Java web frameworks. With Ruby and Rails, I got to express a lot more of

2. The web site `http://dogscape.com` ran for 13 months between restarts. I had to restart the Java process when I needed to upgrade the OS.

what was in my head in a shorter period of time. It was similar to the liberation I felt when I moved from C++ to Java.

But I found the ugly underbelly of Ruby and Rails. Ruby's runtime was so slow and flakey that I was embarrassed to deliver Ruby-based projects to my clients. I had to do my prototypes in Ruby and then port the code over to Java.

As my Ruby and Rails projects grew beyond a few thousand lines of code and as I added team members to my projects, the challenges of dynamic languages became apparent. We were spending more than half our coding time writing tests, and much of the productivity gains we saw were lost in test writing. Most of the tests would have been unnecessary in Java because most of them were geared toward making sure that we'd updated the callers when we refactored code by changing method names or parameter counts. Also, I found that working on teams where there were mind melds between two to four team members, things went well in Ruby, but as we tried to bring new members onto the team, the mental connections were hard to transmit to new team members.

I went looking for a new language and development environment. I was looking for a language that was as expressive as Ruby but as safe and high-performance as Java. In November 2006, I found this combination and a whole lot more in Scala.

Finding Scala

The same person, Martin Odersky, who wrote the Java compiler and Java Generics, designed Scala. Wow. Martin has a team dedicated to maintaining Scala as well as researching ways to mature the language. Beyond being an academic project, Scala is fast and concise, and it has even more type-safety features than does Java. Scala compiles down to Java bytecode, runs fast on the JVM, and is interoperable with Java code.

When I first found Scala, I couldn't believe its claims of performance and Java compatibility. But as I pushed on Scala, I found that it was stable. As I pushed on Scala, I found that it was fast. As I pushed on Scala, I found that it worked perfectly with all the Java libraries that I threw at it.

But most importantly, Scala taught me to program and reason about programming differently. I stopped thinking in terms of allocating buffers, structs, and objects, and of changing those pieces of memory. Instead, I learned to think about most of my programs as transforming input to output. This change in thinking has lead to lower defect rates, more modular code, and more testable code. Scala has also given me the tools to write smaller, more modular units of code and assemble them together into a whole that is maintainable, yet far more complex than anything that I could write in Java or Ruby for that matter.

After more than two years of writing and loving Scala, the only regrets that I have are that I didn't learn Lisp in college or take any programming language theory courses in grad school.[3] Each morning that I sit down at the keyboard and start coding Scala, I get a feeling of calm, peace, and power. I know that I'm going to have another day of taking the ideas in my head and reducing them to a running computer program that will be fast and low in defects.

So, please join me in exploring the Scala programming language. Most of the book will be oriented to writing simple code. I find that most of my work is done in this mode. We'll do a little exploration of Scala's type system and component architecture, but I find that I use those Scala features rarely, when I'm in "library writer" mode.

■ **NOTE** Scala is the first language that has let me think and code differently depending on whether I'm a library consumer or a library producer. Scala provides tools for me as an architect that allow me to specify complex relationships between types and classes and ultimately lets me reason about my code. Working with these tools requires a lot of thinking on my part, and often when I'm working in this mode, I will produce seven lines of code in a day and feel very proud of it. In this mode, I worry about view bounds, covariance and contravariance, implicit conversions, and so on. However, most of my coding time is spent in library consumer mode. In this mode, my code looks a whole lot like Ruby or Python code. I spit out hundreds of lines of code a day, and I know the type system is there to back me up so my tests focus on the logic, not the mechanics, of my code. This book is mostly geared toward simple code—library consumption. Chapter 7 will touch on some of the library-producer coding features of Scala.

A quick diversion: Warren Henning (`http://metacircular.wordpress.com/`) showed me this Scala code sample:

```
def fact(n: Int) = 1 to n reduceLeft (_*_)
```

It calculates the factorial of the input. It's simple, clean, functional, and very readable. I'll be writing a lot of code like this (although perhaps a bit more practical) throughout this book. Let's go and have some fun.

3. See `http://www.joelonsoftware.com/articles/ThePerilsofJavaSchools.html`. I never learned the functional way of thinking. While I've been programming professionally for 30 years or more, it's only in the last two years that I've come to appreciate recursion and the other things that were part of Lisp from the beginning.

Scala's History

Martin Odersky (`http://lampwww.epfl.ch/~odersky/`) evolved Scala over many years. He is an ACM Fellow and runs the Programming Methods Laboratory (LAMP) group at Swiss Federal Institute of Technology in Lausanne (EPFL). Odersky and his team balance between working with Scala as research into how to make programming languages better for all programmers and working with Scala as a tool for commercial software development.[4] They deliver timely updates to Scala along with rapid bug fixes and excellent support. Scala is open source software available under a BSD-like license.

In 1995, Martin Odersky and Philip Wadler (`http://homepages.inf.ed.ac.uk/wadler/`) discovered the JVM. They began doing computer science research and programming language development targeted to the JVM. In 1997, Sun approached Odersky to write the Java 1.1 compiler; he was the lead developer of javac from Java 1.1 through Java 1.4.

Odersky and Wadler conspired on a language called Pizza that compiled to Java bytecode and had generics, first-class functions, and pattern matching. Along with some folks at Sun, they evolved Pizza into Generic Java (GJ), a generics layer on top of Java. GJ became JSR-014 and ultimately, in 2004, Java Generics.

In 2001, Odersky started work on Scala and released the first version in 2003. He and his team started using Scala as a teaching language at EPFL. In 2006, they released Scala 2.0.

Martin Odersky is the core of the Scala group, but he's attracted some extremely talented individuals[5] who have contributed significantly to Scala over the years. These folks include

- Lex Spoon (`http://www.lexspoon.org/`), who has written much of the Scala internals. Lex has been seen in the Squeak Smalltalk community.

- Burak Emir (`http://burak.emir.googlepages.com/`), who built Scala's XML support and has done a lot of work on Scala's pattern matching. Further, Burak has served as my guide and mentor in learning Scala.

- Adriaan Moors (`http://www.cs.kuleuven.be/~adriaan/`), who visited Martin's group for a number of months and provided enhancements to Scala's type system and Scala's parser combinator.

4. `http://dsc.sun.com/learning/javaoneonline/2007/pdf/TS-2844.pdf`
5. Personally, I think they're extremely, awesomely, wickedly cool and talented folks, but I tend to be over the top, so after you spend significant time in the Scala community, please find accolades of your own.

- Philipp Haller (`http://lamp.epfl.ch/~phaller/`), who wrote Scala's Actor support and has always been eager and excellent at adding features to Actors necessary for my work.

- Many other folks, including Don Syme, designer of F# (`http://blogs.msdn.com/dsyme/archive/2007/03/23/f-and-scala-in-lovely-lausanne.aspx`), who have influenced and shaped Scala over the years.

Martin provides the gravity to draw in some excellent brains into the building of Scala, but Martin is committed to Scala as a commercial language while satisfying the research needs of his team.

As of this writing, Scala is at version 2.7.3 and has matured significantly. More than half a dozen people have earned PhDs based on their Scala-based research. Scala is in production at some of the largest and best-known companies in the world, including SAP and Twitter.

Installing Scala

In order to run Scala, you must have Java 1.5 or greater installed on your computer. If you are running Mac OS X, you already have the Java Development Kit (JDK) on your machine. If you're running Windows, please download Java from `http://java.sun.com`. If you're running Linux or Solaris, please consult with your distribution's Java installation instructions.

You can download Scala from `http://www.scala-lang.org/downloads`.

Installing on Windows

First, you must set the `JAVA_HOME` environment variable and add the JDK's `bin` directory to your `PATH` variable. In Windows XP, right-click on My Computer and select the Properties menu item. In Vista, right-click Computer and select Change settings. In both, now select the Advanced tab and click the Environment Variables button. In the System variables area, click the New button. Set the Variable Name to `JAVA_HOME` and the Variable Value to the place that you installed the JDK. On my machine, that's `C:\Program Files\Java\jdk1.6.0_11`.

Next, select the `PATH` variable and click the Edit button. Move the cursor to the beginning of the Variable Value and type **%JAVA_HOME%\bin;** before the rest of the line. Click the OK buttons in the Edit System Variable dialog, in the Environment Variables dialog, and in the System Properties dialog.

Test that you can access the JDK. Click the Start button and select Run…; in the dialog, type **cmd** and click the OK button. A DOS window should appear with a prompt that looks like the following:

```
C:\Documents and Settings\dpp>
```

At this prompt, type **java -version** and press Enter. You should see something like the following:

```
java version "1.6.0_11"
Java(TM) SE Runtime Environment (build 1.6.0_11-b03)
Java HotSpot(TM) Client VM (build 11.0-b16, mixed mode, sharing)
```

Next, test to see that the Java compiler is installed. Type **javac -version**. You should see something like the following:

```
javac 1.6.0_11
```

Next, let's install Scala. Download the Scala lzPack installer from `http://scala-tools.org/downloads`. This download is typically the first on the page. Save this file to the Desktop or some other place that you can find it. Double-click on the `scala-2.7.3.final-installer.jar` file to start the install process. Once complete, you need to create the `SCALA_HOME` environment variable and add Scala to your `PATH` variable. Follow the same process that you used for creating `JAVA_HOME`, but name the variable `SCALA_HOME` and set the `PATH` to the place Scala was installed. Next, update the `PATH` variable to include **%SCALA_HOME%\bin;**.

Open a new command prompt and type **scala**. You should see the following:

```
Welcome to Scala version 2.7.3.final (Java HotSpot(TM) Client VM, Java 1.6.0_11).
Type in expressions to have them evaluated.
Type :help for more information.
```

Congratulations, you've installed Scala.

Installing on Mac OS X and Linux

Make sure you've got the Java JDK 1.5 or 1.6 installed. By default, the JDK is installed on Mac OS X 10.4 and greater. Download the Scala lzPack installer from `http://scala-tools.org/downloads`. Open a terminal window and change directories to the location where you downloaded the Scala installer. Type the following:

```
sudo java -jar scala-2.7.3.final-installer.jar
```

Now select `/usr/local/share` as the place to install Scala. Once Scala is installed, run the following commands:

```
cd /usr/local/share
sudo ln -s scala-2.7.3-final scala
cd /usr/local/bin
sudo ln -s ../share/scala/bin/scala scala
sudo ln -s ../share/scala/bin/fsc fsc
sudo ln -s ../share/scala/bin/scalac scalac
sudo ln -s ../share/scala/bin/scaladoc scaladoc
```

Change to your home directory and type **scala**. You should see the following:

```
Welcome to Scala version 2.7.3.final (Java HotSpot(TM) 64-Bit Server VM, Java
1.6.0_07).
Type in expressions to have them evaluated.

Type :help for more information.
scala>
```

Congratulations, you've installed Scala on your machine.

The Scala Community

The Scala community is a wide-ranging, rich, and vibrant group of people. It's a warm and welcoming place for newbies, but it also offers academically rigorous debate and lots of ideas from the cutting edge of computer science made practical.

There are lots of cool people from lots of different backgrounds in the Scala community.

- You'll find academics like Martin and his team. They actively participate in the community, answering questions from newbies and from seasoned folks about Scala basics, design choices, and more.
- There's a cohort of extremely skilled and experienced functional programming gurus, including David MacIver. They help drive Scala forward.
- James Iry is another highly skilled functional programming guru who has a magic touch when it comes to explaining complex Scala concepts to just about anyone.
- Jon Pretty and Jamie Webb of Sygneca provide consulting services and a ton of backbone for the Scala list and Scala community.
- You may hear from Jorge Ortiz who, like James, is great at explaining complex Scala topics to folks at all skill levels.

The list of Scala mailing lists managed by EPFL (Martin's group) can be found at `http://www.scala-lang.org/node/199`.

The main Scala list is available at `scala@listes.epfl.ch`, or you can access it on the Web via Nabble at `http://www.nabble.com/Scala-f14147.html`. This list is for discussing Scala, asking questions, and making observations. It's a great place for intermediate and advanced Scala developers to exchange ideas and information.

The Scala User list is oriented toward newbies, and it's a great place for folks to learn about Scala by seeing the kind of questions that other newbies have about the language. The e-mail for the list is `scala-user@listes.epfl.ch`, and it's available on Nabble via a web interface at `http://www.nabble.com/Scala---User-f30217.html`.

The Scala Debate list is a place for seasoned Scala folks to discuss the language, make suggestions, and conspire to move Scala forward. You can post to the list via e-mail at `scala-debate@listes.epfl.ch` or via Nabble at `http://www.nabble.com/Scala---Debate-f30218.html`. Note that this list is a hard-core, no-holds-barred discussion of Scala that sees some minor flame wars occasionally break out among the mathematically inclined. If you venture into this forum, keep in mind that it does, from time to time, devolve into discussion that is not kind and gentle and that often the discussion is indecipherable for those of us who do not have a PhD in mathematics. If this is the kind of discussion that is interesting to you, please check out Lambda the Ultimate at `http://lambda-the-ultimate.org`. LtU provides a forum for a broad spectrum of programming language theory (PLT) discussions.[6]

Lift is the leading web framework for Scala. The Lift community is hosted at Google. You can learn more at `http://groups.google.com/group/liftweb/`. While I'm the lead developer for Lift, we're not going to focus much on web development in this book, but if you're a web developer, I'd love it if you'd join the Lift community.

Those are the main mailing lists in the Scala community. As you explore with Scala, please join the online discussion and share your questions and thoughts.

Summary

You've learned a little about what attracted me to Scala, a little about where Scala comes from, and you've installed Scala. You've learned about, and hopefully joined, the Scala community. Now it's time to explore Scala and see whether it resonates with you the same way it resonates with me. Turn the page and write your first Scala program.

6. Paul Snively, the technical editor of this book, is an editor on LtU. It's a pleasure and an honor to have him participate in this project. If you dive deep into PLT, I hope you find Paul to be a compass and a beacon as I do.