Updated: 2021-01-21

3.8
Writing Good Scala code

Making the transition from Java to Scala (1)

- I would estimate that 99.9% of Scala programmers were (or still are) Java programmers;
- There are some things that need getting used to:
  - Perhaps the number one thing is the use of *var*. Many new Scala programmers don't know how to do things without mutable variables and collections. That's not surprising. It takes training and a knowledge of FP to be able to avoid *var*s.
  - Immutable collections: Scala collections are immutable by default (you have to import *mutable.xxx* in order to get the mutating version). This takes a little getting used to and causes confusion about the operators such as ++, +=, etc.
  - Equality: Java has primitives (int, double, etc.) while Scala doesn't. Therefore, Java provides two methods for testing equality: "==" for primitives and "equals" for objects. In Scala, the "==" method simply delegates to Java's *equals* method. If you want to test that two objects are the same object in Scala (rare), you use *eq*.

Making the transition from Java to Scala (2)

- More things that need getting used to:
  - Java loves to wrap everything in {} and separate statements with ";" —in Scala, you can eliminate most of this clutter.
  - In Java, you're allowed multiple returns from a method—in FP, this is very much frowned on. You don't need the *return* keyword in Scala—the final expression in a method or block is what is returned: if you ever find yourself using *return*, ask yourself why.
  - In Java8, functions are all defined as lambdas—but in Scala you can take advantage of the so-called *eta* expansion—that's to say that wherever Scala expects a function, you can provide a method (it's much clearer what's going on when you define your function as a method—for one thing a method has a name).
  - Defining classes in Scala: it's rare to define a straightforward class in Scala: a class is either an abstract class or a case class. You can do it, but there's no good reason to.

Making the transition from Java to Scala (3)

- Yet more things that need getting used to:
  - Class constructors: in Java it's common practice to provide several different constructors for a class. In Scala, you can do this too (syntax is different), but it's better practice to code these as additional "apply" methods in the companion object.
  - Case classes: Get used to using these for just about everything. If you just need a temporary way to combine values (say for the return from a method), you can use a tuple (you don't have to name the fields or given them types). But be aware that case classes are also tuples.
  - Modules: a single file in Scala can have any number of related classes (don't abuse this, though). Typically, you will have one (sealed) trait and a number of case classes which extend that trait. Each of these case classes may also have a companion object.

Making the transition from Java to Scala (4)

- Even more things that need getting used to:
  - In Java, it can be a pain to initialize a list. Not so in Scala. You can just write List(1,2,3)
  - How about adding an element to a list? In Java, you typically do that with *add* (a mutating method). In Scala, you do it immutably:

```
val xs1 = List(1)
val xs2 = xs1 :+ 2
```

    - Note that the : is always on the side of the collection.

```
val xs2 = 2 +: xs1
```

How to write good Scala code (1)

- Perhaps the first difference you notice from Java is the syntax:
  - No semi-colons (unless you need them to fit stuff on one line);
  - Fewer braces and parentheses (no-arg methods usually don't need parentheses);
- and built-in semantics:
  - Case classes, for instance, provide all the getters (possibly setters, too), *equals, hashCode, toString*, all that jazz—and the definition is really the constructor—and you don't need "new".
- In short, Scala code is a lot more compact than the equivalent Java code.
- So, try to use these improvements to the look-and-feel of the code to write really elegant, clear code.
- Have each class know about one domain and put all the methods that use that knowledge in the class—and have each method do only one simple thing.

The principle of Simple, Obvious, Elegant

- Most of the time when writing, say, a method in FP, you have a very limited set of variables which are in scope and, because each has some particular type, they can only be combined together in a small number of ways:
  - Don’t be afraid to follow where your instinct is leading you.
  - Let the IDE guide you by using ctrl-space (or whatever): it will show you the possible expansions, with the types that they result in.

Getting the help you need

- Your first place to go for anything is: https://www.scala-lang.org/
- The definitive book is *Programming in Scala* (3rd edition) by Odersky et al.
- But the book that will really teach you what's going on is *Functional Programming in Scala* by Rúnar Bjarnesson and Paul Chiusano (the "red book"). It's fun to read but not for the faint of heart.
- The Lightbend site is also good (more application-oriented).
- Of course, StackOverflow is great for all Scala questions.
- There are some great blogs out there too:
  - The Scala Times
  - The Neophyte's guide to Scala
  - (my) Scalaprof blog