Why You Should Learn Scala

if you're a Java developer

Bruce Cota

July 2, 2013

Scala Is

- A "SCAlable LAnguage" (in terms of large codebases, not just performance)
- Combines Haskell/ML style Functional Programming with Object Oriented Programming on a JVM.
  - "Haskell/ML style" == statically typed (as opposed to Lisp/Clojure)
- A complex language (my observation) – not easy to learn.

Rod Johnson: Scala in 2018 (Scaladays 2013 Keynote)

- Scala is the biggest thing since Java.
- Scala is the leading language for enterprise development.
- Java is “deeply uncool” but won't go away.
- Language fragmentation persists. Startups favor dynamic languages.
- Scala won by
  - Embrace the Java Ecosystem
  - Slow innovation
  - Fight only “winnable battles”.

Rod Johnson: Java vs Scala

	Java	Scala
Language	"Sucks"	Best thing since C
	verbose	lean
	lack of code reuse	Best reuse ever
	immutability is unnatural	Both FP and OO styles
	Restrictive	Flexible (blessing and a curse)
Community	pragmatic	purists and zealots
Culture of	readable code	clever code
	backwards compatible	not backwards compatible

Rod Johnson Scaladays 2013 Keynote

The Scala Community Needs To

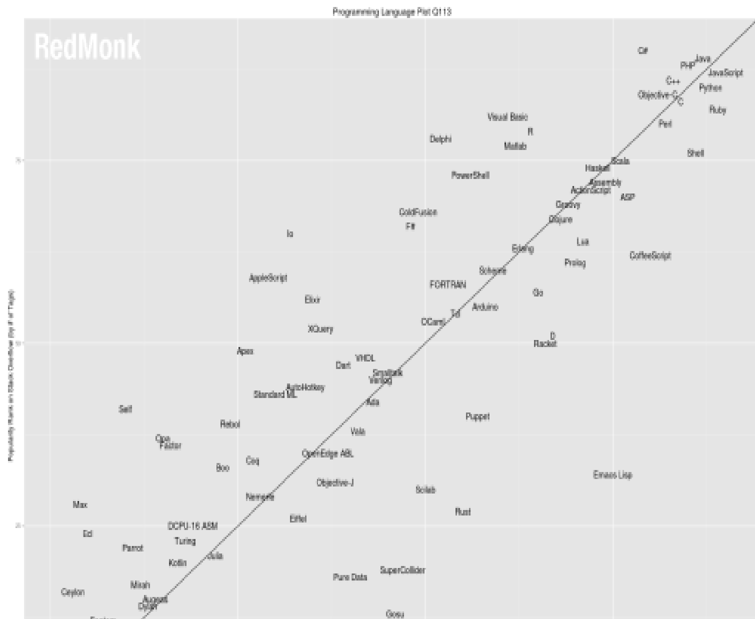
- Develop coding standards
- Stop denigrating OOP (go write Haskell)
- Be more welcoming.
- Stop reinventing wheels.
- Innovate less. ("Scala should be a little more boring")

But the community is moving in the right direction.

Martin Odersky Scaladays 2013 Keynote

- Scala is better than Java, but that's not enough – there has to be a "catalyst for initial adoption". p
- The catalyst for OOP was GUI widgets.
- The catalyst for FP may be concurrent programming.

Where is Scala Now? Redmonk Metrics Jan 2013



Where is Scala Now?

Redmonk Metrics Q1 2013

- #14 in number of Github projects (behind Javascript, Ruby, Python, Java, Shell(!), PHP, C, C++, Perl, Objective-C, and C#, Coffescript, and ASP)
- #17 in Stackflow tags (behind C#, Java, PHP, Javascript, C++, Python, Objective-C, C, VB, Ruby, Perl, R, Delphi, Matlab, and Shell)
- #12 combined – just ahead of Haskell, well ahead of Groovy and Clojure. Well behind #11 (Shell).
- Ranks worse in some other metrics (web searches, number of web sites, number of books published, etc)

Big name Adopters: Twitter, LinkedIn, FourSquare

Big Concepts in OOP

- Imperative Programming (Loops and State)
- Object Identity
- Classes
- Inheritance
- Interfaces as Types.
- Concepts Java doesn't have – Mixins, multimethods, multiple inheritance, etc.

Big Concepts in Functional Programming

- A program is an expression composed of smaller expressions.
- *Composability* (reusability)
- *Referentially Transparency* (immutability) – no side effects
  - Makes programs easier to reason about
  - No need for synchronization
  - *Monads* to do (often simple) things in pure FP
  - *Tail Recursion* (instead of loops)
- Pattern Matching (super duper switch statements)
- Type Constructors (generics)
- Function types are first class types (e.g. closures)
- Lazy Evaluation
- Union Types instead of Exception Handling

Scala: OOP and FP

- Scala fully supports both imperative-style OOP and FP
- Odersky Keynote
  - There are many ways to do things in Scala because of fusion of OOP and FP.
  - Booch's Definition: Objects have state, identity, and behavior
  - State and identity are usually bad (immutability and structural equality are usually better).
  - OOP provides a way of managing namespaces
  - "Use FP style by default, but don't be dogmatic."

OOP in Scala

- Classes and Inheritance
- Traits – interfaces with behavior
- Scala traits aren't "real" traits, because of limitations of JVM.

Immutability in Scala

- Most things are immutable by default, but mutability is there if you ask.
- Benefits
  - Easier to reason about programs if they are referentially transparent
  - No need to synchronize access to immutable objects
- `val` and `var`.
- Local state is considered ok

Functional Programming in Scala

- Recursion and Tail Recursion – annotations help!
- Pattern Matching

```
def title(p: Person) : String = p match {
  case t: Teacher => "Mr. " + t.last
  case s: Student => s.first
  case _ => "John Doe"
}
```

Pattern matching and tail recursion are often used together.

```
@tailrec def product(l: List[Int], acc: Int): Int =
l match {
  case head :: tail => product(tail, acc*head)
  case Nil => acc
}
```

FP or OOP?

But why not use method override instead of pattern matching?

```
class Person(first: String, last: String) {
  def sortable: String = last+","+first
  def title: String = last
}

class Teacher(val first: String, val last: String)
extends Person(first, last) {
  override def title = "Mr. " + last
}

class Student(val first: String, val last: String)
extends Person(first, last) {
  override def title = first
}
```

FP or OOP?

- To Loop or to (tail) Recurse? (beats me)
- Pattern Matching or Method Override? – Martin's advice:
  - If you expect to add classes, use method override
  - If expect to add methods, use pattern matching. (Example: Classes representing expressions in a parser.)
  - Sealed classes protect against unexpected new classes.

Sealed Classes and Pattern Matching

```
sealed class Expression

class Variable(name: String) extends Expression

class Constant(constant: String) extends Expression

class Method(target: Expression,
             method: String
             params: List[Expression]) extends Expressio
```

```
...
expression match {
  case Variable: ...
  case Constant: ...
  case Method: ...
}
```

Scala Type System

- Java 1.4

```
map.put(key, new BigLongClassName())
...
BigLongClassName bigLongClassName =
(BigLongClassName) map.get(key)
```

- Scala Version

```
map += (key -> new BigLongClassName())
val bigLongClassName = map.get(key)
```

- Of Course, Java 1.5+ has generics .. thanks to Martin Odersky

Scala Type System

- Type Inference
- Type Constructors (generics) – notations for variance, etc
- Functions are first class types
- Scala "culture" is fanatical about compile-time error checking (e.g. `@tailrec`, sealed classes).
- There's a lot more .. type variables, type lambdas, structural types ...

None instead of Null

- Use `Option[T]` type to avoid run-time NPE.
- A value is either a `Some(t)` (where “`t` is a `T`) or the singleton `None`
- `Option.map` *lifts* functions to `Option` type.
- `for` comprehension is helpful for more complex lifting.
- Scala still has `Null` (it has no choice).

Either instead of Exception

- Scala has `throw` and `try/catch`
- `Either[L,R]` type is preferred by functional programmers
- A value is either a `Left(l)` or a `Right(r)`
- By convention, a `Right` holds a value, and a `Left` is an error condition
- Lifting is messier. `for` comprehension doesn't work as well because `Either` is symmetrical.
- `Validation` from scalaz library purportedly works better ("right leaning")

Either instead of Exception

```
trait Grader {
  def grade(homework: Homework): Int

  def average(work: Seq[Homework]) = {
    val total = work.foldLeft(0)(_ + grade(_))
    total.asInstanceOf[Float]/work.size
  }
}


class Assistant extends Grader {
  def grade(homework: Homework): Int =
    if (hmwk.content.length < 1000)
      hmwk.content.hashCode % 100
    else
      throw new Exception("I don't know what to do!")
}
```

Either instead of Exception

```
trait Grader {
  def grade(homework: Homework): Either[Exception,Int]

  def average(work: Seq[Homework]) = { //broken
    val total = work.foldLeft(0)(_ + grade(_))
    total.asInstanceOf[Float]/work.size
  }
}

class Assistant extends Grader {
  def grade(homework: Homework): Either[Exception,Int] =
    if (hmwk.content.length < 1000)
      Right(hmwk.content.hashCode % 100)
    else
      Left(new Exception("I don't know what to do!"))
}
```

Fixed Grader

`Either` isn't a *monad* so we can't use for-comprehension.

```
trait Grader {
  type R = Either[Exception,Int]

  def grade(homework: Homework): R

  def average(work: Seq[Homework]) = {
    val total = work.foldLeft[R] (Right(0)) (
      (sum: R, hmwk: Homework)
        => sum match {
          case Left(exception) => Left(exception)
          case Right(s) =>  grade(hmwk) match {
            case Left(exception) => Left(exception)
            case Right(score) => Right(s+score)
          }
        })
    total.asInstanceOf[Float]/work.size
  }
}
```

Case Classes

- Convenience class to eliminate boilerplate
  1. Constructor args are automatically `val`
  2. Compiler generates `apply` and `unapply` (static) methods
  3. Compiler generates `equals`, `hashCode`, and `copy` methods.
  4. Cannot extend them

Implicits

- *Pimp my Library* – enhance third party libraries. Like compile time Monkey Patching.
- *Type Classes* – Making classes extend interfaces that they don't even know about!
- Could be an enormous mess

Lazy Computation/Pass By Name

- Pass By Name is a by-product of functions as first class types
- Lazy vals “cache” computations
- Streams

There's Much More!

- Named parameters with default values
- Implicit method parameters
- Type variables
- Structural types
- For comprehension

Conclusion

Java developers should learn Scala because

1. Functional Programming
   - Many useful and powerful concepts for writing better code
   - Something of a "Paradigm Shift" for OOP developers.
   - Having influence (leaking into Java, for example)
2. Scala is a good entry into FP for a Java developer.
   - It runs on a JVM
   - You can use Java libraries (Hibernate, Spring, etc)
   - You can use your favorite IDE
   - You can write pure OOP, pure FP, or anything in between.

Scala Ecosystem

- Typesafe – leading commercial entity.
- SBT – Simple Build Tool. (You could use gradle.)
- Akka – distributed/concurrent programming
- Slick and Squeryl — ORMs (Hibernate also works.)
- Spray – servlets based on Akka
- Lift — (older) framework for web development.
- Scalatra – web "micro-framework" for building web services
- Play framework – self contained "full stack" framework for web development

Resources

- Programming in Scala by Odersky et al from Artima is the standard Tome
- Manning is churning out books, notably
  - Scala in Action
  - Scala in Depth
  - Functional Programming in Scala
- quick start with Typesafe Activator at typesafe.com
- ww.playframework.com is another alternative
- Martin Odersky's Functional Programming course on Coursera