

extending the Java platform with Scala  
(and some reasons it might be useful to do so)

Jason Nerothin  
jasonnerothin@gmail.com

# Scala is:

- a programming language
- free and open source
- statically typed
- functional & imperative
- JVM compatible
- pretty well tooled

latest release	2.8.0
website	<a href="http://scala-lang.org">scala-lang.org</a>
book	Programming In Scala
inventor	Martin Odersky

part 1: from imperative  
to functional

```
6 package imp2funcrfx
7
8 object Main {
9
10     def printArgs (args:Array[String]): Unit = {
11         var i = 0
12         while (i < args.length) {
13             println (args (i))
14             i+=1
15         }
16     }
17
18     def main (args:Array[String]) = {
19         printArgs (args)
20     }
21 }
22
23 }
```

imperative style

```
package imp2funcrfx
```

```
object Main {
```

```
  def printArgs(args:Array[String]): Unit = {  
    for( arg <- args )  
      println(arg)  
  }
```

```
  def main(args:Array[String]) = {  
    printArgs(args)  
  }
```

```
}
```

less imperative

```
5  
6 package imp2functx
```

```
7  
8 object Main {
```

```
9  
10 def printArgs(args:Array[String]): Unit =  
11     args.foreach(println)  
12 }
```

```
13  
14 def main(args:Array[String]) = {  
15     printArgs(args)  
16 }
```

```
17  
18 }  
19  
20
```

pretty functional

```
package imp2funcrfx

object Main {

  def formatArgs (args:Array[String]) = args.mkString("\n")

  def main (args:Array[String]) = {
    println (formatArgs (args))
  }
}
```

functional

```
package imp2funcrfx

object Main extends Application{

  def formatArgs(args:List[String]) = args.mkString("\n")

  val args = List("one", "two", "three")

  println(formatArgs(args))

}
```

down-right functional

# part 2: language primer

types	classes
	traits
values	fields
	methods
	packages
	singletons

# Scala namespaces

# equality

operator	Java	Scala
==	reference	value
eq	undefined	reference
equals()	value	undefined



```
package basics

class ImmutableClass (time:java.util.Date) {
    val date = time
}

object Instantiations extends Application{

    val ic = new ImmutableClass(new java.util.Date())
    println(ic.date.toString)

}

}
```

classes and objects

```

1 package fomoco
2 // need an axle
3 trait ThreeQuarterTonAxle {
4     def turn : String = "Whhhhirrrr!"
5     def weightRating : Int = 1500
6 }
7 // and some brakes
8 trait Brakes {
9     def stop : String = "Stopping now..."
10    def stoppingCapacity : Int = 2500
11 }
12 trait Drum extends Brakes{
13     override def stop : String = "Ka-THUNK!"
14     override def stoppingCapacity = 1500
15 }
16 trait Disc extends Brakes{
17     override def stop : String = "VVVRrrrrr...rr.r...tsh"
18 }
19 // and a steering mechanism
20 abstract class SteeringKnuckle ( open : Boolean ){
21     def closed : Boolean = !open
22 }
23 // ...as an assembly
24 class DanaSpicer44 extends SteeringKnuckle( false ) with ThreeQuarterTonAxle with Brakes {
25     def cargoWeight : Int = { stoppingCapacity + super.weightRating }
26 }

```

# mixins

# cost of code



IMHO

task	Java	Scala
testable code	medium	easy
code 4 side fx	easy	hard
immutability	hard	trivial
behavior stacking	hard	easy

part 3: concurrency

models	Java	Scala
state isolation	lock/monitor synchronization	message passing map reduction
language support	low-level + java.util.concurrent	scala.actors
Threading	Thread per task	Thread:Actor 1:1 OR 1:*
Complexity	high	low

```

5
6 class ShinyThing(desc:String){
7   override def toString() : String = {
8     desc
9   }
10 }
11
12 class Business extends Actor {
13   val products = Array[String]("Xbox","Wii","PS3")
14   val rand = new java.util.Random
15   def act(){
16     loop{
17       react{
18         case starryEyed : Customer =>
19           println("A NEWWWW customer {" + starryEyed.number + "}")
20           val thing = new ShinyThing(products(rand.nextInt(3)))
21           starryEyed ! thing
22           starryEyed ! "Pay me"
23       }
24     }
25   }
26 }
27
28 class Customer( num : Int) extends Actor {
29   def act(){
30     loop{
31       react{
32         case thing : ShinyThing => {
33           println("Oooh! My {" + num + "} new " + thing + "!")
34           Thread.sleep(75)
35         }
36         case msg =>
37           println("I {" + num + "} received mail: " + msg )
38       }
39     }
40   }
41 }
42 }
43 }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }

```

actors

```

46 object CustomerService extends Application{
47
48     val customers = Array(new Customer(1), new Customer(2), new Customer(3))
49     customers(0).start; customers(1).start; customers(2).start
50     def randomCustomer() : Customer = {
51         customers(new java.util.Random().nextInt(customers.length))
52     }
53
54     val biz = new Business
55     biz.start
56
57     var i=0
58     while( i<10){
59         i = i+1
60         biz ! randomCustomer
61     }
62     Thread.sleep(1500)
63     exit()
64

```

Output - Imp2FuncRfx (run)

```

compile:
run:
A NEWWWWW customer {3}!
A NEWWWWW customer {2}!
A NEWWWWW customer {1}!
Ooch! My {3} new PS3!
A NEWWWWW customer {1}!
Ooch! My {1} new PS3!
Ooch! My {2} new Xbox!
A NEWWWWW customer {1}!
A NEWWWWW customer {2}!
A NEWWWWW customer {3}!
A NEWWWWW customer {3}!
A NEWWWWW customer {1}!
A NEWWWWW customer {2}!
I {3} received mail: Pay me
I {1} received mail: Pay me
Ooch! My {3} new Wii!
I {2} received mail: Pay me
Ooch! My {1} new Xbox!
Ooch! My {2} new Xbox!
I {1} received mail: Pay me
I {2} received mail: Pay me
I {3} received mail: Pay me
Ooch! My {2} new Xbox!
Ooch! My {1} new Xbox!
Ooch! My {3} new Wii!
I {2} received mail: Pay me
I {1} received mail: Pay me
I {3} received mail: Pay me
Ooch! My {1} new Wii!
I {1} received mail: Pay me

```

# concurrency



IMHO

task	Java	Scala
code correctness	very hard	medium
buggy code	easy	medium
scalability	hard	easy

usw usf...

# Scala/Java Interop

invoking Scala from Java

define java iface

implement it in Scala

define factory iface in Java

implement it in Scala

define a Scala main that invokes your Java main and inits the Scala factory

invoking Java from Scala

call directly

Maven

# Scala Tribe?

- twitter: jasonnerothin
- interest sign-up sheet: <http://bit.ly/8XQD57>
- email: jasonnerothin@gmail.com