SCALA - A Tastier Java

REPL

```
scala welcome to scala version 2.8.0
scala> val l = List(1,2,3,4,5,6,7,8,9,10) l:
List[Int] = List(1,2,3,4,5,6,7,8,10)

scala> l.foldLeft(20)(_ + _) res1: Int = 75
```

Scala

Functional

```
scala> List("Foo", "Quux").map(_.length)
res1: List(3,4)
```

For comprehensions

```
case class User(name: String, bio: String)

val userProfiles = List(
  Map("name" -> "Odersky", "bio" -> "Scala
creator"),
  Map("name" -> "Pollak", "bio" -> "Lift
creator"),
  Map("name" -> "A Nobody"))
var validUsers = for {
  user <- userProfiles
  name <- user.get("name")
  bio <- user.get("bio")
} yield User(name, bio)

validUsers: List[User] = List(User(Odersky,
Scala creator), User(Pollak, Lift creator))
```

A Language that grows on you

Compatible

Scala compiles to Java bytecode, and runs on the JVM. This means you can still benefit from all the advantages that the Java platform has to offer.

High level

Scala helps to manage complexity by letting you raise the level of abstraction in your design. Scala code is shorter and more readable than Java code.

Statically typed

Scala has an advanced type system. It is more advanced than Java's, and also less verbose. Verbosity is avoided via type inference, and flexibility is added via patter matching and several ways to write and compose types.

A Chat Server

```
case class Messages(msgs: List[String])
object ChatServer extends Actor {
private var msgs: List [String] = Nil
private var listeners: List[Actor] = Nil

  def act = loop {
    react {
      case s: String =>
        msgs :: s
        listeners foreach (l => l ! Messages(msgs))
      case Add(who) =>
        listeners = who :: listeners
      case Remove(who) =>
        listeners -= who
      }
    }
  }
  this.start
  }

  class Chat extends CometActor with CometListenee {
    private var msgs: List[String] = Nil
      def render =
        <div>
          <ul>{msgs.revers.map(m => <li>{m}</li>)}</ul>
            {ajaxText("", s => {ChatServer ! s; Noop})}
        </div>
    protected def registerWith = ChatServer
      override def lowPriority = {
        case Messages(m) => msgs = m ; reRender(false)
      }
  }
}
```

Dense

```
scala> val l = 1 to 10
l: Range = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
l.filter(_ % 2 == 0).flatMap(i => l.filter(_
% 2 == 1).map(j => i * j))
res: scala.collection.VectorView[Int,Vector
[_]] =
VectorViewFN(2, 6, 10, 14, 18, 4, ... , 56,
72, 10, 30, 50, 70, 90)
```

Daunting?

```
trait Comonad[W[_]] extends Copointed[W] with Cojoin[W] { def cobind[A, B]
(a: W[A], f: W[A] => B) : W[B] = fmap(cojoin(a), f)
}
object Comonad { def comonad[W[_]](implicit j: Cojoin[W], p: Copointed[W])
=
new Comonad[W] { def cojoin[A](a: W[A]) = j.cojoin(a) def fmap[A, B](a: W
[A], f: A => B) = p.fmap(a, f) def copure[A](a: W[A]) = p.copure(a)
```

scalalabs