POLE STAR
www.polestarglobal.com

Purplefinder Enterprise Platform Introduction to Scala

Peter Potts

September 2010

Resources

- Online Scala shell at www.simplyscala.com
- Download 2.8.0 final at www.scala-lang.org
- Run Scala shell on Windows or Linux
- Excellent Scala plugin for IntelliJ

Hello World

- Create file HelloWorld.scala

```
object HelloWorld extends Application {
    println("Hello world")
}
```

- Compile

```
scalac HelloWorld.scala
```

- Run

```
scala HelloWorld
```

Small but perfectly formed

- Scala has less syntax than Java
- Yet, Scala is more powerful than Java
- Drop semicolons and empty brackets

For example:

Scala only has

```
if (x) y else z
```

Java also has

```
x ? y : z
```

Methods

- def pretty(x: String): String = “<“ + x + “>”
- def pretty(x: String) = “<“ + x + “>”
- def pretty(x: String): Unit = println(“<“ + x + “>”)
- def pretty(x: String) {println(“<“ + x + “>”)}
- def pretty[T](x: T) = {

```
   val y = “<“ + x + “>”
   println(x)
   y
  }
```

Classes and Objects

- class User(val name: String) {
  def hello() {println(“Hello “ + name)}
  }
- new User(“Peter”).hello
- No static fields or methods
- object Queen extends User(“Liz”) {
  override def hello() {println(“Hi Ma’am”)}
  }
- Queen.hello

Sensible default scope

- “public” is the default scope in Scala
- “package” is the default scope in Java

POLE STAR
www.polestarglobal.com

Immutability

- Cannot be modified after creation
- A cornerstone of functional programming
- Only Java strings are immutable
- Scala defaults to immutability
- Blanket use of “final” pollutes the code
- Use, “val” for immutable values
- And “var” for mutable variables
- Scala “==” is like Java “equals”

Eliminate boilerplate

- Case classes
- Automatic factory class, getters & setters
- Automatic equals, hashCode & toString
- case class Person(name: String, var age: Int)
- val x = Person.apply(“a”, 1)
- x.name()
- x.age_=(2)
- x.==(Person(“a”, 2))

Infix, Postfix, Prefix

- Just syntactic sugar for methods
- 1+2 ≡ (1).+(2)
- “abc”.indexOf(“b”) ≡“abc” indexOf “b”
- “xyz”.toUpperCase() ≡ “xyz” toUpperCase
- -2.0 ≡ (2.0).unary_-

Enrich library classes

- class RichString(x: String) {def unary_- = x.reverse}
- new RichString(“abc”).unary_-
- -new RichString(“abc”)
- implicit def enrich(x: String) = new RichString(x)
- -“abc”

Functions as first class citizens

- val triple: (Int => Int) = { x => 3 *x }
- triple(4)
- val triple : Int => Int = x => 3 * x
- val triple = (x: Int) => 3 * x
- def triplicate(x: Int): Int = 3 * x
- def triplicate(x: Int) = 3 * x
- val triple = x => triplicate(x)
- val triple = triplicate(_)

Curried methods

- def add(x: Int, y: Int) = x + y
- add(3,4)
- def add(x: Int)(y: Int) = x + y
- add(3)(4)
- add(3) {

  4

  }
- Rename ‘add’ to ‘while’ we have makings of a DSL

Curried functions

- val add: Int => Int => Int = x => y => x + y
- add(3)(4)
- val inc = add(1)
- inc(6)

Java is a static language

- public interface Closeable {void close();}
- public class SafeClose {

  public void safeClose(Closeable resource) {

  try {resource.close();} catch (Exception e) {}

  }}
- public class UseResource extends SafeClose {

  …safeClose(resource);…

  }
- What about resources not subclass of Closeable?

Groovy is a dynamic language

- class SafeClose {

  void safeClose(def resource) {

  try {resource.close()} catch (Exception e) {}

  }}
- class UseResource extends SafeClose {

  …safeClose(resource)…

  }
- What happens if resource not implement close?

Scala is static in fact but dynamic in nature

- trait SafeClose {

  def safeClose[R <: {def close()}](resource: R) {

   try {resource.close} catch {case _ =>}

  }}
- class UseResource extends Other with SafeClose {

   ...safeClose(resource)...

  }
- Compilation checks close method is present.

Pattern matching

- abstract class Expr
- case class Num(num: Int) extends Expr
- case class Add(left: Expr, right: Expr) extends Expr
- def eval(expr: Expr): Int = expr match {

  ```
   case Num(num) => num
   case Add(left, Num(0)) => eval(left)
   case Add(left, right) => eval(left) + eval(right)
  }
  ```
- eval(Add(Num(3),Num(4)))

Arrays

- Arrays are not special in Scala
- Immutable by default
- Array(5, 4, 3, 2, 1)(2)
- Mutable alternative
- import scala.collection.mutable.ArrayBuffer
- val buffer = new ArrayBuffer[Int]
- buffer += 1
- buffer.toArray

List

- Immutable by default
- (“red” :: List(“green”, “blue”)).tail.head
- Mutable alternative
- import scala.collection.mutable.ListBuffer
- val buffer = new ListBuffer[Int]
- buffer += 1
- buffer.toList

Set

- Immutable by default
- Set(“red”, “green”, “blue”)
- “run spot run”.split(“ “).toSet
- Set(1, 2, 3, 4) -- List(2, 4)

Map

- Immutable by default
- val m = Map(1 -> "one", 2 -> "two") + (3 -> "three")
- 1->”one” ≡ Pair(1, “one”) ≡ Tuple2(1, “one”)
- m(2) ≡ m.apply(2)

Range

- 0 until 10 ≡ Range(0, 9)
- 1 to 10 ≡ Range(1, 10)
- for (i <- 1 to 10) println(i)
- (1 to 3).mkString(“<“, “:”, “>”)
- Array(“+”, “-”) .mkString(“&”)

Can syntax for map to list be simplified?

```
Map(
    “a” -> List(“apple”, “art”),
    “b” -> List(“bee”, “bye”, “bin”),
    “c” -> List(“cat”))
```

have but want

```
Map(
    “a” --> (“apple”, “art”),
    “b” --> (“bee”, “bye”, “bin”),
    “c” --> “cat”)
```

Map to list DSL

```
class Key[K](val k: K) {
    def -->[V](vs: V*): (K, List[V]) =  (k, vs.toList)
}
implicit def toKey[K](k: K): Key[K] = new Key(k)
```

Call by name

```
def thread(block: => Unit) {
   new Thread {
    override def run {block}
   }.start
}
thread {
   Thread.sleep(5000L)
   println("Just waited 5 seconds")
}
```

Higher order functions on collections

```
Iterator.
    continually(readLine).
    takeWhile(_  != “quit”).
    foreach(line => println("echo " + line))
```