Scala Overview
where objects and functions meet.

Walter Cazzola

Dipartimento di Informatica
Università degli Studi di Milano
e-mail: cazzola@di.unimi.it

Scala: Scalable Language

My First Scala Program: A Special Form of HelloWorld

```scala
class Upper {
  def upper(strings: String*): Seq[String] = {
    strings.map((s: String) => s.toUpperCase())
  }
}

val up = new Upper
Console.println(up.upper("A", "First", "Scala", "Program"))
```

Interpreted as a script

```
[15:39]cazzola@surtur:~/lp/scala>scala upper.scala
ArrayBuffer(A, FIRST, SCALA, PROGRAM)
```

Or into an interactive section

```
[15:39]cazzola@surtur:~/lp/scala>scala
Welcome to Scala version 2.8.1.final (Java HotSpot(TM) 64-Bit Server VM, Java 1.6.0_21).
Type in expressions to have them evaluated.
Type :help for more information.
scala> :load upper.scala
Loading upper.scala...
defined class Upper
ArrayBuffer(A, FIRST, SCALA, PROGRAM)
```

Scala: Scalable Language
History & Motivations

The design of Scala starts in 2001 at École Polytechnique Fédérale (EPFL) of Lausanne by Martin Odersky:
- The first working release is out at the end of 2003;
- Last stable release is 2.11.4 (end of 2014).

It runs on the JVM and interoperates with the Java libraries.

Scalable language
- succinct, elegant and flexible syntax (50%-75% of code reduction);
- interactive interpreter and
- support for embedded domain specific languages

Scala merges object-oriented and functional programming.
Scala is statically typed, it supports
- abstract and path-dependent types;
- generic classes and polymorphic methods;
- (a limited form of) type inference.

Scala: Scalable Language
My First Scala Program: A Special Form of HelloWorld (Cont’d)

```scala
object Upper {
  def upper(strings: String*) = strings.map(_.toUpperCase())
}

println(Upper.upper("A", "First", "Scala", "Program"))
```

- the keyword `object` introduce a class with a single instance;
- don’t exist static methods but methods of singleton objects
- _ as a wildcard

```
[15:39]cazzola@surtur:~/lp/scala>scala
Welcome to Scala version 2.8.1.final (Java HotSpot(TM) 64-Bit Server VM, Java 1.6.0_21).
Type in expressions to have them evaluated.
Type :help for more information.
scala> :load upper2.scala
Loading upper2.scala...
defined module Upper
ArrayBuffer(A, FIRST, SCALA, PROGRAM)
```
Scala: Scalable Language
My First Scala Program: A Special Form of HelloWorld (Cont'd)

object Upper {
  def main(args: Array[String]) = {
    args.map(_.toUpperCase()).foreach(println("", _))
    println(""")
  }
}

- main as a method of a singleton object;
- two independent uses of the _ wildcard.

Compiled to Bytecode

```scala
object Upper {
  def main(args: Array[String]) = {
    args.map(_.toUpperCase()).foreach(println("", _))
    println(""")
  }
}
```

- main as a method of a singleton object;
- two independent uses of the _ wildcard.

Compiled to Bytecode

```scala
Compiled to bytecode
```

Scala: Scalable Language
Types

```scala
class Rational(n: Int, d: Int) extends AnyRef {
  val num = n
  val den = d
  def this(n: Int) = this(n, 1)

  def +(that: Rational): Rational = {
    new Rational(num*that.den + that.num*den, den*that.den)
  }

  def +(i: Int): Rational = {
    new Rational(num+i*den, den)
  }

  override def toString = {
    num + "/" + den
  }
}
```

```scala
scala> :load rational.scala
Loading rational.scala...
defined class Rational
```

```scala
scala> val r1 = new Rational(1)
r1: Rational = 1/1
```

```scala
scala> val r2 = new Rational(2,3)
r2: Rational = 2/3
```

```scala
scala> r1+r2
res2: Rational = 5/3
```

```scala
scala> r1.+(r2)
res3: Rational = 5/3
```

Scala: Scalable Language
Types

Type Hierarchy

```scala
Any is the root of the whole hierarchy
- AnyRef is the root for the reference classes (both Java and Scala classes) and coincides with Object;
- AnyVal is the root for all the basic types.

Two different 'empty' values
- Null for all the reference types and it is instantiate by null
- Nothing for all types and it can't be instantiate.

It can be used to define Empty as List[Nothing] for any List[T].
```
Scala: Scalable Language
Pure Object-Oriented Paradigm

As in Smalltalk:
- everything is an object and any operation is a method

Scala> 1.+2
res0: Double = 3.0
Scala> 3.14.+res0
res3: Double = 6.140000000000001

Identifiers
- alphanumerics (variables) or a string on a given set of characters (operators/methods)
- e.1 id e.2 is the short for e.1.id(e.2)

Immutable/mutable variables

Scala> val array: Array[String] = new Array(3)
array: Array[String] = Array(null, null, null)
Scala> array = new Array(2)
<console>:6: error: reassignment to val
array = new Array(2)
Scala> array(0) = "Hello"
Scala> array
res7: Array[String] = Array("Hello", null, null)
Scala> var price: Double = 100
price: Double = 100.0
Scala> price += price*.20
Scala> price
res9: Double = 120.0

Scala: Scalable Language
Case Classes
abstract class Bool {
def and(b: => Bool): Bool
def or(b: => Bool): Bool
}
case object True extends Bool {
def and(b: => Bool) = b
def or(b: => Bool) = this
}
case object False extends Bool {
def and(b: => Bool) = this
def or(b: => Bool) = b
}
def bottom: () => Nothing = () => bottom()
scala> :load short-circuit.scala
Loading short-circuit.scala...
defined class Bool
defined module True
defined module False
bottom: () => Nothing
Scala> True and bottom()
java.lang.StackOverflowError
Scala> True or bottom()
res4: object True = True

Scala: Scalable Language
Option: None and Some instead of Null
Options are used to smoothly integrate functions and objects

val RegionCapitals = Map(
  "Val d'Aosta" -> "Aosta", "Piemonte" -> "Torino", "Liguria" -> "Genova",
  "Lombardia" -> "Milano", "Emilia Romagna" -> "Bologna"
// ...
)
println( "Get the capital cities wrapped in Options:")
println( "Liguria: " + RegionCapitals.get("Liguria") )
println( "Lombardia: " + RegionCapitals.get("Lombardia") )
println( "Padania: " + RegionCapitals.get("Padania") + "," )

println( "Get the capital cities themselves out of the Options:")
println( "Liguria: " + RegionCapitals.get("Liguria").get )
println( "Lombardia: " + RegionCapitals.get("Lombardia").getOrElse("Oops!") )
println( "Padania: " + RegionCapitals.get("Padania").getOrElse("Oops2!") )

Scala: Scalable Language
Functions and Methods
Methods & functions
- functions are high-order;
- (parametric) polymorphism limited to methods

Methods are values of a particular class with method apply
- they are similarly called: succfun(2) and succmeth(2) but the first is the short for succfun.apply(2)

Parametric polymorphism for methods
Scala: Scalable Language

Comprehensions are a mechanism
- to traverse a set of something
- to "comprehend" what we find and
- computing something new from it

```scala
def sum_evens = (L:List[Int]) => (var sum = 0; for (X <- L if X%2 == 0) sum += X; sum)
```

Yielding
- to get a new collection from a comprehension

```scala
val is_prime = (X:Int) => (val divisors = (X:Int) => (for { Y <- List.range(2,math.sqrt(X).toInt) if (X % Y == 0) yield Y divisors(X).length == 0 })
```

Scala: Scalable Language

Some (Known) Functions

- to get a new collection from a comprehension
- to "comprehend" what we find and
- to traverse a set of something;

```scala
map(A,B)(f: A => B, list: List[A]): List[B] = List match {
  case Nil => Nil
  case hd::tl => f(hd)::map(A,B)(f,tl)
}
def reduce[T](f:T=>T,T):List[T] = {
  def reduce2(acc:T, list: List[T]):T = {
    case Nil => acc
    case hd::tl => reduce2(f(acc,hd), tl)
  }
  reduce2(list.head, list.tail)
}
```

References

- Martin Odersky and Matthias Zenger.
  Scalable Component Abstractions
- Nathanial Schüirli, Stéphane Duvasse, Oscar Nierstrasz, and Andrew P. Black.
  Traits: Composable Units of Behaviour.
- Venkat Subramaniam.
  Programming Scala.
  The Pragmatic Bookshelf, June 2009.
- Dean Wampler and Alex Payne.
  Programming Scala.
  O’Reilly, September 2009.