Introduction to the Scala programming language (for Java programmers)

Olivier Curé

Université Paris-Est Marne la Vallée , LIGM UMR CNRS 8049, France

September 26, 2017

- Scala : "Scalable Language"
- Designed for scale: ability to integrate novel features easily
- Runs on the JVM
- An object-oriented and functional programming language
- Statically-typed
- Type inference

- Everything is an object, 2+3 is an invokation of '+' on objects of the Int class: 2.+(3)
- Functions behave like values and can be returned and provided as arguments.
- A lot of data structures are immutable.
- Type inference
- a REPL (Read Eval Print Loop) to accelerate learning curve (scala command)

- val to define an immutable variable, ex: val i=5 // i: Int=5
- var to define a mutable variable, ex: var j=3 // j: Int=3
- Types: Boolean, Byte (8 bits signed value), Short (16 bits), Int (32 bits), Long (64 bits), Float (32 bits single precision), Double (64 bits double precision), Char (16 bits unsigned Unicode char), String (seq of chars)

- if else, returns a value
- loops: do .. while(test), while(test) .., for(i<- 1 to 3), for(j<-3 until 8)
- val li = List(1,2,3,4,5)
- for(a<- li if a%2==0) print(a) // 24
- val arr = Array(1,2,3,4) // arr: Array[Int] = Array(1, 2, 3, 4) (Type inference)
- arr.foreach(println)
- arr(0)=10 // values of arr can change
- arr = Array(3,5,6) // error: reassignment to val (but it can not be associated to a new array

- Match expressions (like switch)

```
argument match {
case value1 => do something
case value2 => do something else
case _ => otherwise
}
```

- def to define a function
- def functionName(par: parType, ..) = { .. }
- def myMax(a:Int, b:Int) : Int =if (a>b) return a else return b
- def myMax(a:Int, b:Int) : Int =if (a>b) a else b
- def myMax(a:Int, b:Int) = if (a>b) a else b

- Unit, similar to void in Java
- def functionName() : Unit = // no return
- It is possible to put a function inside a function
- Function can have default values: def tr(age: Int = 21) : Boolean = if(age>=18) true else false
- tr() // true
- tr(14) // false
- Function literals : (x => x+1) // lambda

- map: evaluates a function over each element in a list and returns a list of the same cardinality.

```
val nums = List(1,2,3,4)
nums.map(x=>x*2)
res46: List[Int] = List(2, 4, 6, 8)
def square(n:Int) = n*n
nums.map(square)
res48: List[Int] = List(1, 4, 9, 16)
```

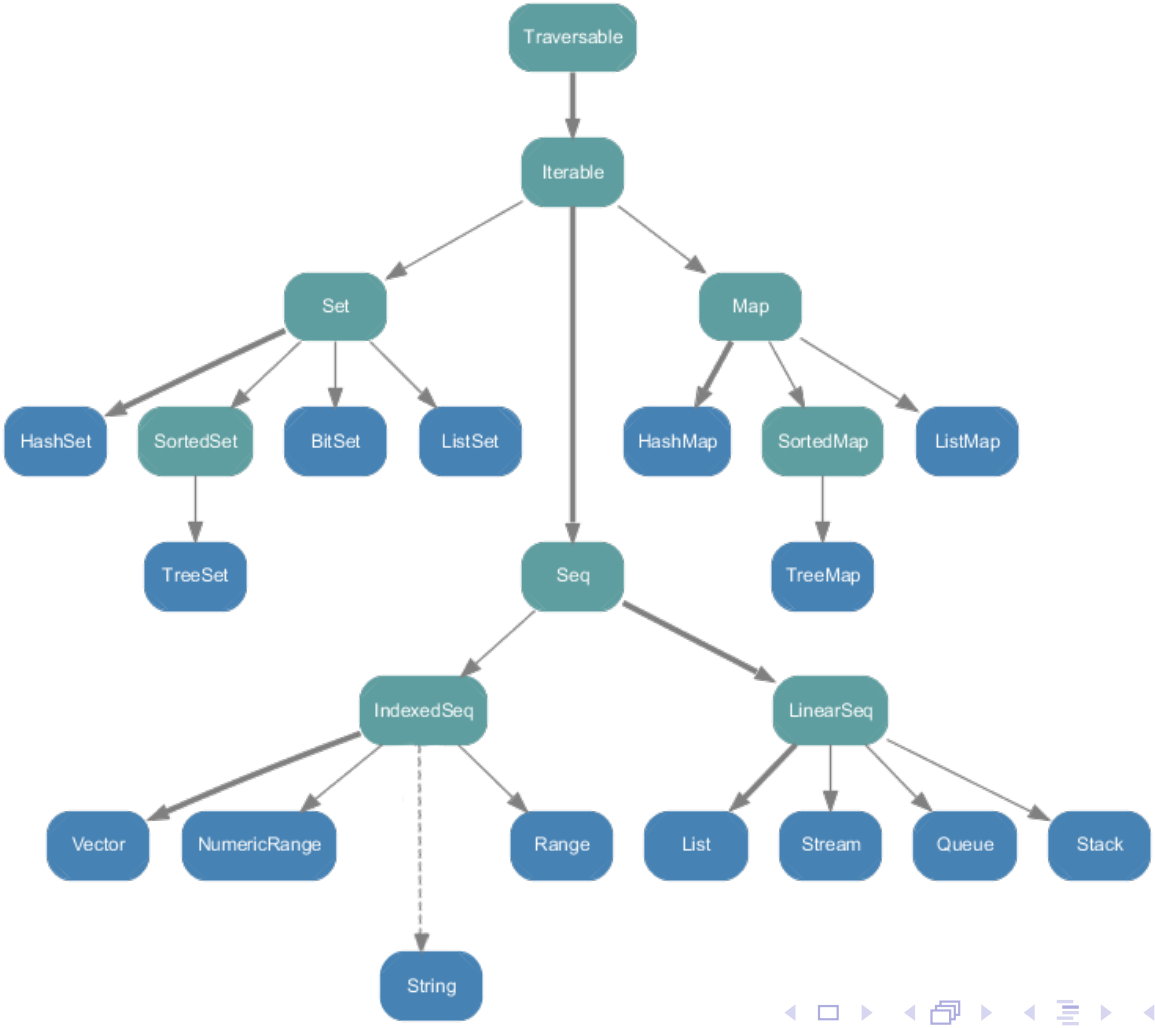
- filter: retains elements of the collection for which the passed function evaluates to true

```
val nums = List(1,2,3,4)
nums.filter(x=> x%2==0)
res49: List[Int] = List(2,4)
def isEven(n:Int) = n%2==0
nums.filter(isEven)
res50: List[Int] = List(2, 4)
```

- foreach to iterate over a collection
- var l = List(1,2,3)
- l.foreach(x=>print(x))
- l.foreach(print)

- Scala collections are different from Java collections
- Spark collections and Scala collections are different
- Scala collections can be mutable or immutable
- Mutable collections can be updated or extended in place
- Immutable collections never change: additions, deletions or updates operators return a new collection and leave the original one unchanged.
- Collections: Array, List, Set, Map, Tuple

Collections hiearchy



- Iterable proposes a foreach method.
- Sequence: an immutable iterable with a length, elements start at 0 and have fixed index positions.
  - LinearSeq is the list, IndexedSeq is the array
- Map: an iterable consisting of pairs (k/v)
  - HasMap (hash table implem.), ListMap ((inverse order of insertion), keys sorted in sort order, red-black tree
- Set: an iterable with no duplicate elements.
  - HashSet (set using a hast table implem.), TreeSet (AVL tree), BitSet (to save memory, only for integers).

- Array (Mutable, entries of the same type), val ar = Array(1,2,3)
- List (Immutable, same type), val li = List(4,5,6)
  - append to the head with '::' (named cons), concatene lists with ':::', add to the tail with ':+'
  - in general, a ':' at the end of an operator means an invocation on the right operand.
  - reverse a list with reverse, li.reverse
  - It is more efficient to add elements on the left of a list and reverse (than append on the right) or use a ListBuffer (mutable list) and perform a .toList.
  - Nil is the empty list (or List())

- Tuple: allows different types for each element. construct with '( )'.
  - val tup = (1,"abc",List(1,2,3))
  - tup: (Int, String, List[Int]) = (1,abc,List(1, 2, 3))
  - print(tup._2) // abc

- Associative array (Map):immutable and mutable

```
var caps = Map("UK"->"London","FR"->"Paris")
caps("UK")
res31: String = London
caps += ("JP"->"Tokyo")
```

- Set : immutable and mutable, no duplicates, elements of the same type.

```
val s1 = Set(1,2,3)
val s2 = s1 + 0
val s3 = s2 - 2
s.contains(1) val set = Set(2,3,4,5,5)
set: scala.collection.immutable.Set[Int] = Set(2, 3, 4, 5)
val setm = scala.collection.mutable.Set(3,4,5,6)
setm += 10
res6: setm.type = Set(5, 6, 3, 10, 4)
```

- Classes defined with 'class' like in Java
- No static members in a class, all present in an singleton bject instead. They are called companion object have the same name as the class and they must be in the same file.
- A standalone object is a singleton with no companion class. Used as the entry point of an app (where you have the def main 'args: Array[String] function)
- public visibility by default.

- Traits are like Java interfaces