SCALA- collezioni e metodi

Info 3 AA 18/19

Angelo Gargantini

Immutability

- Why?
  - Immutable objects are automatically thread-safe
    (you don't have to worry about object being changed by another thread)
  - Compiler can reason better about immutable values -> optimization

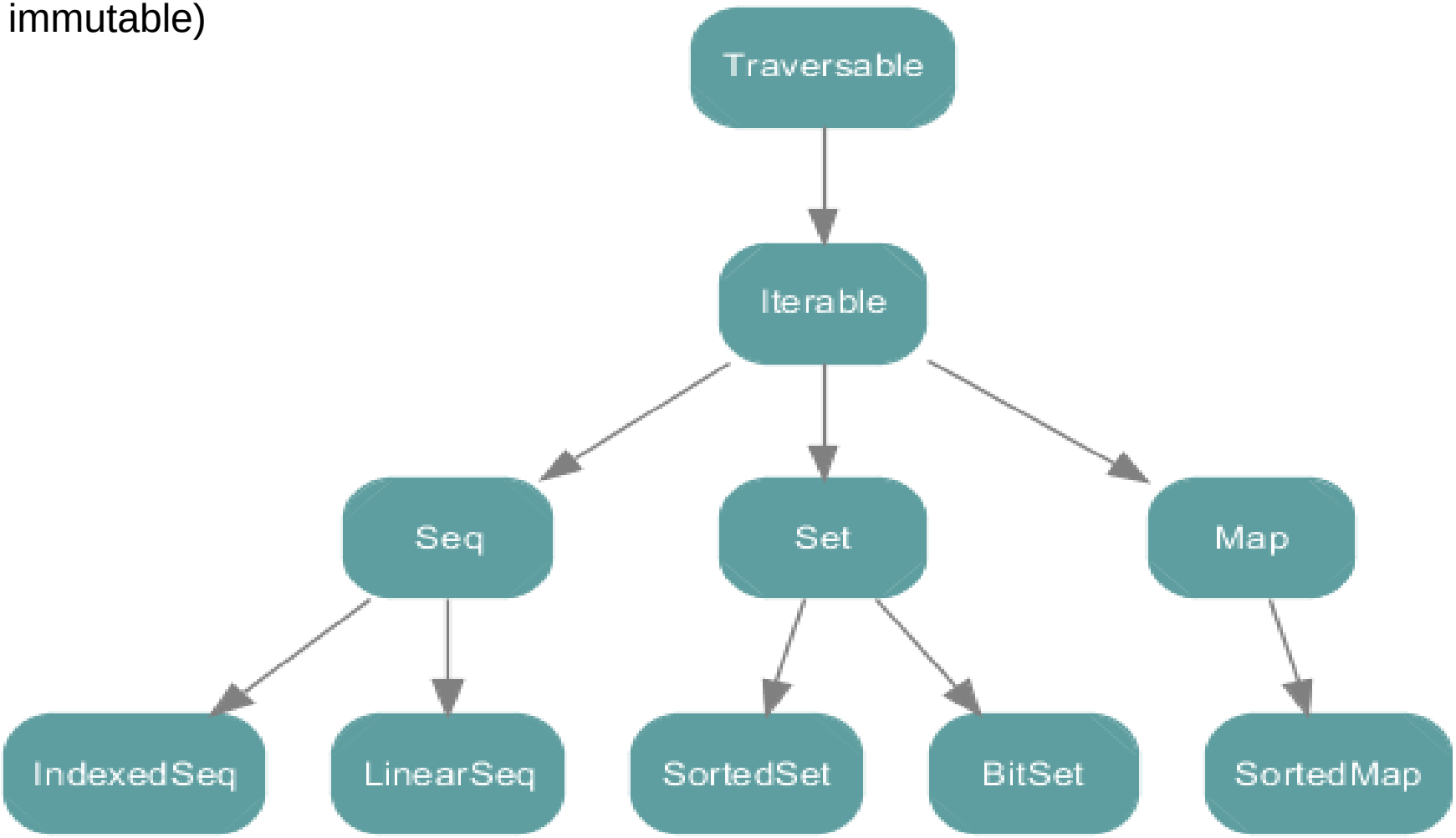
  - Steve Jenson from Twitter: *"Start with immutability, then use mutability where you find appropriate."*

Collezioni

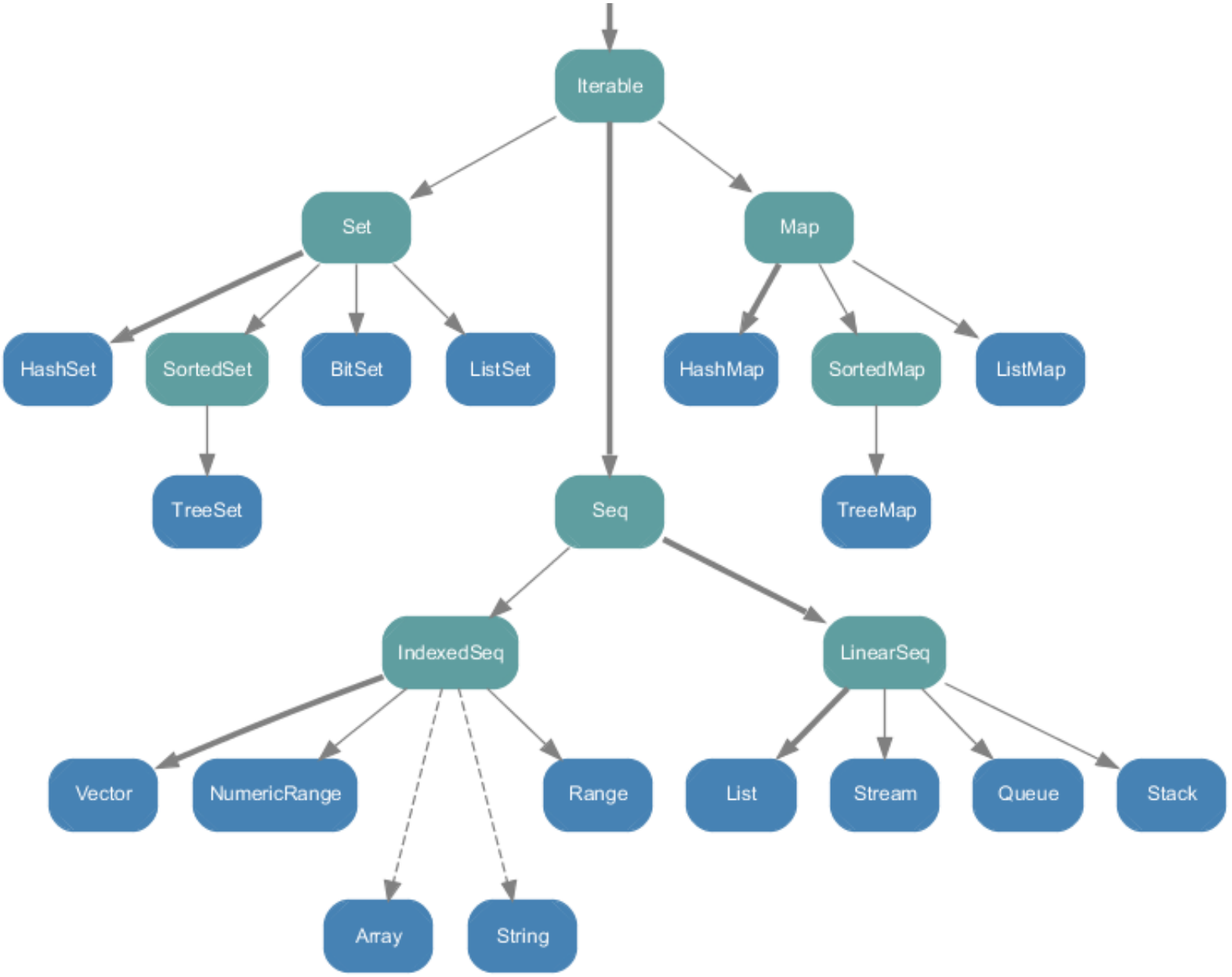
- Mutable and Immutable Collections
- Scala collections systematically distinguish between mutable and immutable collections.
  - A mutable collection can be updated or extended in place.
  - Immutable collections, by contrast, never change.
    - You have still operations that simulate additions, removals, or updates, but those operations will in each case return a new collection and leave the old collection unchanged.

collections in package scala.collection

Higher level (bouth mutable and immutable)



Immutable


Iterable
Set
Map
HashSet
SortedSet
BitSet
ListSet
HashMap
SortedMap
ListMap
TreeSet
Seq
TreeMap
IndexedSeq
LinearSeq
Vector
NumericRange
Range
List
Stream
Queue
Stack
Array
String

List

- Lists are immutable (= contents cannot be changed)
- List**[String]** contains Strings

```
val lst = List("b", "c", "d")
lst.head                    // “b”
lst.tail                    // List(“c”, “d”)
val lst2 = "a" :: lst       // cons operator
```

Nil = synonym for empty list

```
val l = 1 :: 2 :: 3 :: Nil
```

- List concatenation

```
val l2 = List(1, 2, 3) ::: List(4, 5)
```

Foreach

```
val list3 = List("mff", "cuni", "cz")
```

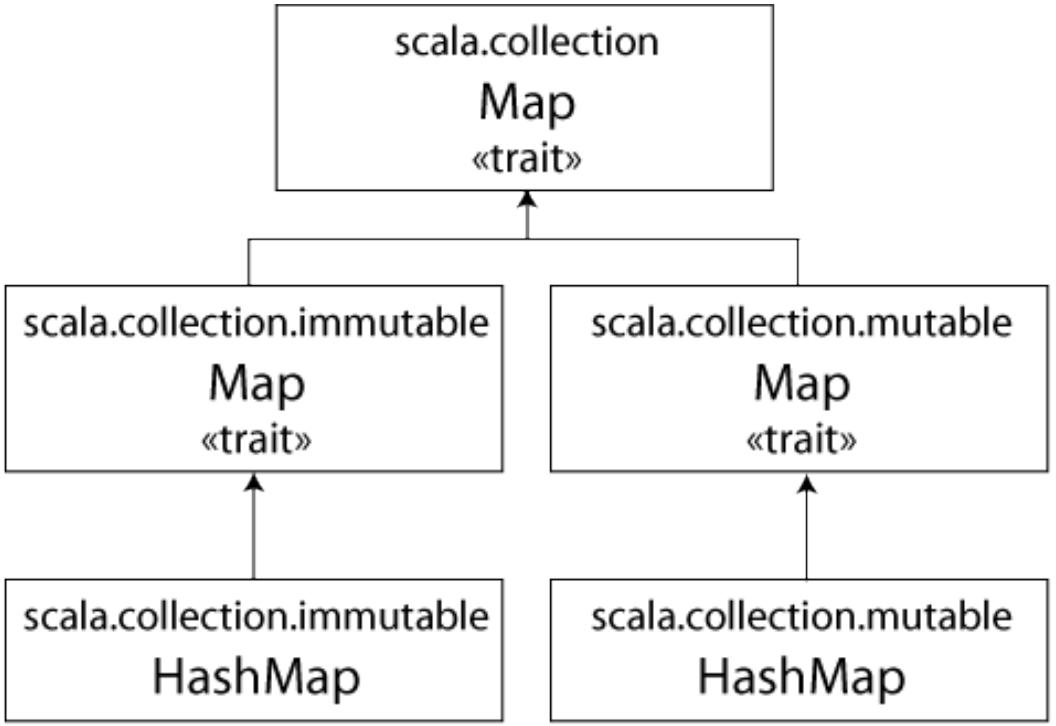
- Following 3 calls are equivalent

```
list.foreach((s : String) => println(s))
list.foreach(s => println(s))
list.foreach(println)
```

- For comprehensions

```
for (s <- list) println(s)
for (s <- list if s.length() == 4) println(s)
```

Maps and Sets



MAPS

```
import scala.collection._

val cache = new mutable.HashMap[String,String];
cache += "foo" -> "bar";

val c = cache("foo");
```

- The rest of Map and Set interface looks as you would expect

Mutable List: ListBuffer

- ListBuffer[T] is a mutable List
- Like Java's ArrayList<T>

```
import scala.collection.mutable._

val list = new ListBuffer[String]
list += "Vicky"
list += "Christina"

val str = list(0)
```

+= add element

(*i*) to access the i-th element

scala.Seq

- scala.Seq is the supertype that defines methods like:
  - filter, fold, map, reduce, take, contains, …
- List, Array, Maps… descend from Seq

From Java to Scala

- Iterator <=> java.util.Iterator
- Iterator <=> java.util.Enumeration
- Iterable <=> java.lang.Iterable
- Iterable <=> java.util.Collection
- mutable.Buffer <=> java.util.List
- mutable.Set <=> java.util.Set
- mutable.Map <=> java.util.Map
- mutable.ConcurrentMap <=> java.util.concurrent.ConcurrentMap

algorithms

Iterate – foreach function

- Every collection in Scala's library defines (or inherits) a foreach method

```
val names = List("Daniel", "Chris", "Joseph")
    names.foreach { name =>
      println(name)
    }}
```

- foreach is a "higher-order" method, due to the fact that it accepts a parameter which is itself another method
  - `name => println(name)`

```
names.foreach(println)
```

Foreach - istruzione

- There are times that we just want to use a syntax which is similar to the for-loops available in other languages.

```
val nums = List(1, 2, 3, 4, 5)

var sum = 0
for (n <- nums) {
  sum += n
}
```

Oppure se volessi usare il metodo:

```
var ss = 0;
def sinc(x:Int) ={
      ss += x;
    }

nums.foreach(sinc)
```

Folding

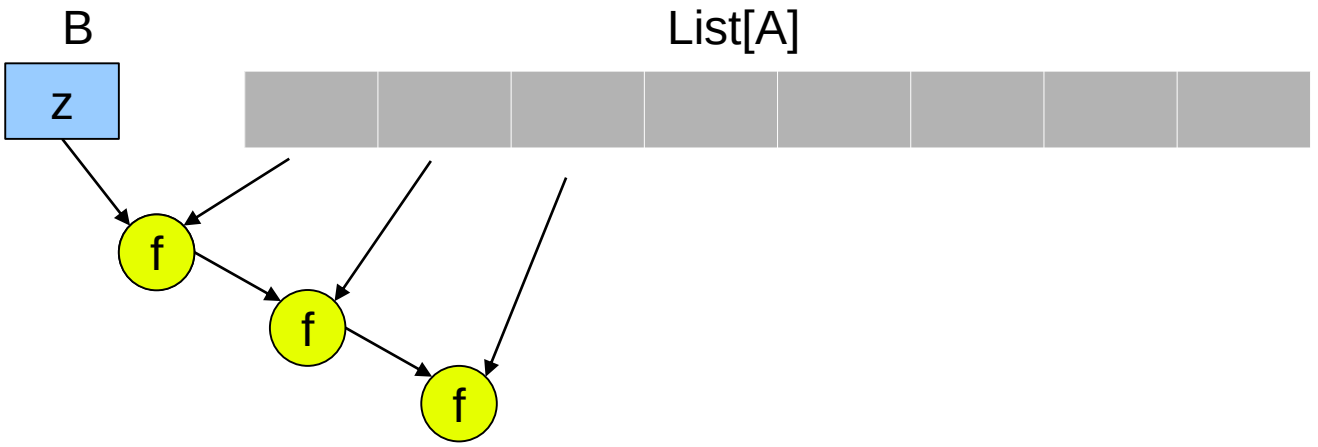
- Looping is nice, but sometimes there are situations where it is necessary to somehow combine or examine every element in a collection, producing a single value as a result.
- For List[A]:

```
def foldLeft[B](z: B)(f: (B, A) => B): B
```

- The foldLeft function goes through the whole List[A], from head to tail, and passes each value to f. For the first list item, that first parameter, z, is used as the first parameter to f. For the second list item, the result of the first call to f is used as the B type parameter.

foldLeft

```
def foldLeft[B](z: B)(f: (B, A) => B): B
```



z: first element
f: function to be applied

Risultato finale

Folding Esempi

Somma di tutti i numeri in nums

```
val sum = nums.foldLeft(0)((total, n) => total + n)
```

oppure

```
def myf(x: Int, y: Int) = x+y
val sum = nums.foldLeft(0)(myf)
```

Fold --> Reduce

- Fold has a closely related operation in Scala called “reduce” which can be extremely helpful in merging the elements of a sequence where leading or trailing values might be a problem. Consider the ever-popular example of transforming a list of String(s) into a single, comma-delimited value:
-
```
var nn = List("a","b", "c")// voglio stampare "a,b,c"
println(nn.foldLeft("")((x, y) => x + "," + y))
```

Stampa però: `,a,b,c`

Reduce

- Solution: use a reduce, rather than a fold.
  - Reduce distinguishes itself from fold in that it does not require an initial value to “prime the sequence”. Rather, it starts with the very first element in the sequence and moves on to the end.

```
def reduceLeft(f: (A, A) => A): A
```

Esempi

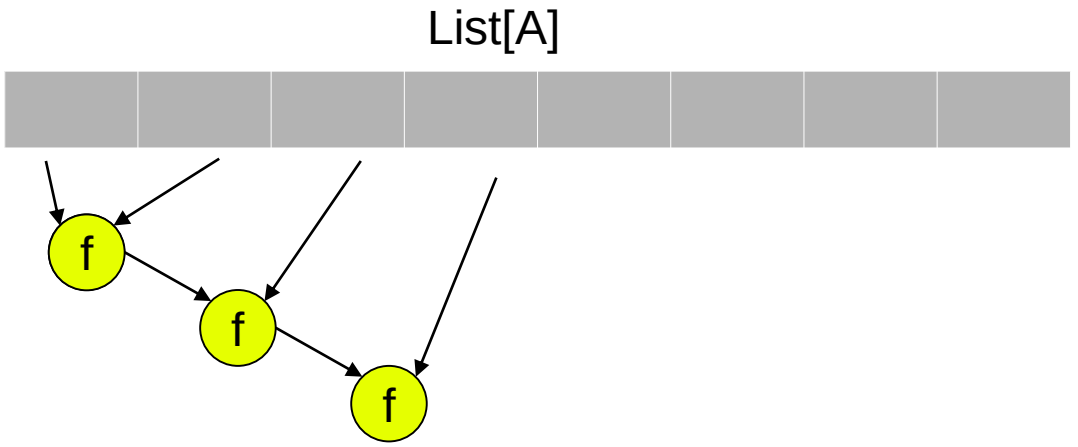
```
println(nn.reduceLeft((x, y) => x + "," + y))
--> a,b,c
```

- Altro esempio: calcolo del max in lista

```
l.reduceLeft((x, y) => if (x>y) x else y )
```

foldLeft

```
def reduceLeft(f: (A, A) => A): A
```



Non c’è il primo elemento

Risultato finale

Esempi di Folder/Reduce

- Fai la somma/prodotto dei numeri in una lista
- Restituisci la stringa piu' lunga
- Trova la dimensione della stringa piu' lunga
- ....

Filter/map

- fold can be an extremely useful tool for applying a computation to each element in a collection and arriving at a single result
- if we want to apply a method to every element in a collection in-place (as it were), creating a new collection of the same type with the modified elements?
- Esempi, data una lista, costruire la lista dei doppi

```
var ll = List(3, 4, 5)
// lista dei doppi
def doppio(x: Int) = 2 * x
println(ll.map(doppio))
```

Esempi + Filter

- La lista delle lunghezze di una stringa

```
nomi.map( x => x.length())
```

- Filter
- Alcune volte voglio estrarre delle liste filtrando il contenuto
- Ad esempio: data una lista estrarre la lista pari

```
def pari = (x: Int) => (x % 2 == 0)
println(ll.filter(pari))
```

-

Using Map+Reduce

- Spese si usa map insieme a reduce:
  - Con map trasformo i dati per renderli piu' trattabili
  - Con reduce ottengo un dato sintetico
- Sono algoritmi che si possono parallelizzare
  - Vedi google framework mapreduce
  - http://it.wikipedia.org/wiki/MapReduce
- Vedi
  - http://spark.apache.org/