Scala Cheat Sheet

1 Scala Class Hierarchy

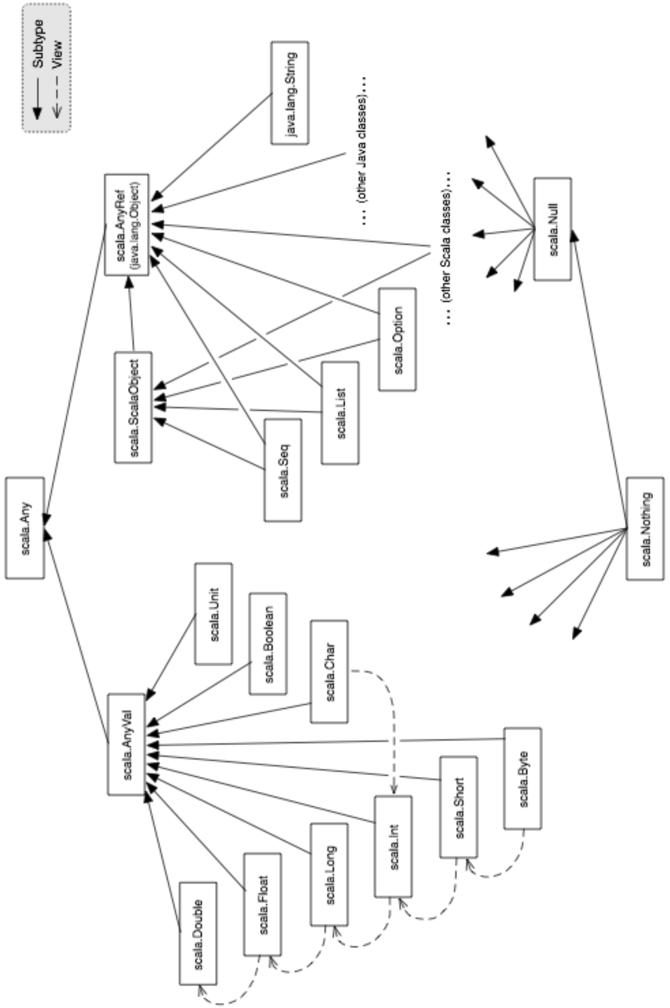
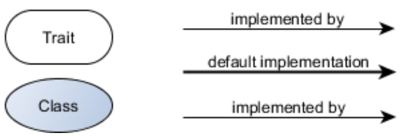


Figure 1: Scala class hierarchy, source: `http://www.scala-lang.org/old/node/128`

2 Scala Collections

2.1 Scala Collections Hierarchy



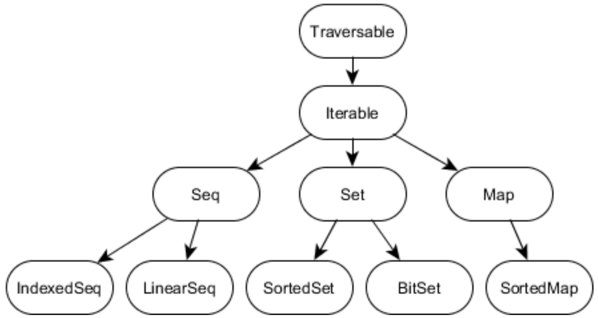


Figure 2: scala.collection

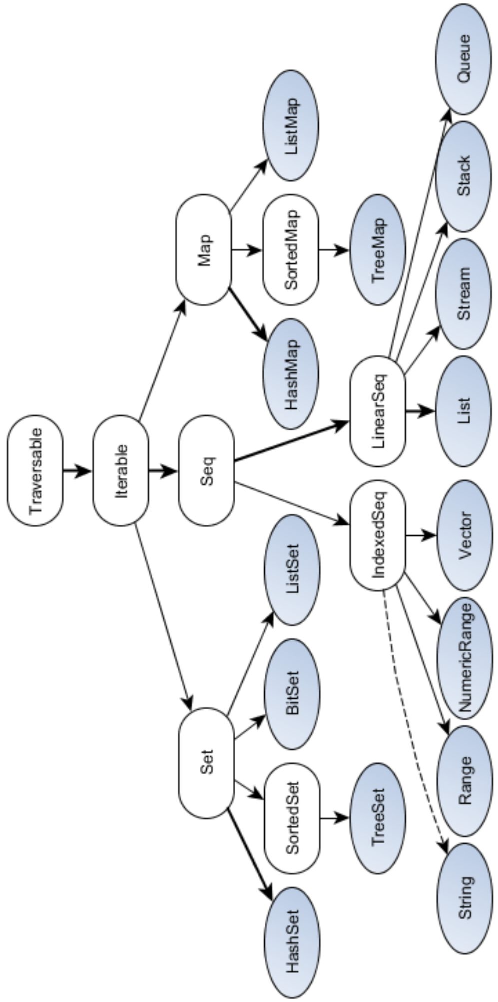


Figure 3: scala.collection.immutable

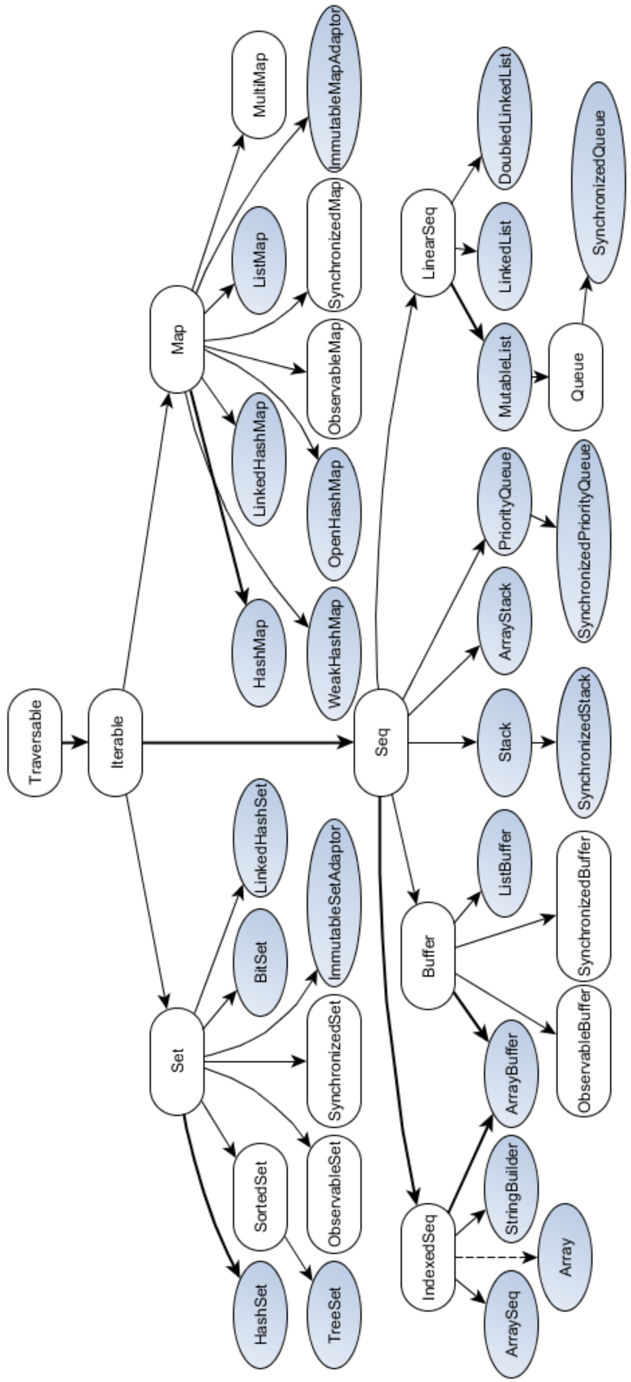


Figure 4: scala.collection.mutable

2.2 Trait Traversable

Table 1: Methods in Traversable

Category	Methods
Abstract	xs foreach f
Addition	xs ++ ys
Maps	xs map f, xs flatMap f, xs collect f
Conversions	toArray, toList, toIterable, toSeq, toIndexedSeq, toStream, toSet, toMap
Size info	isEmpty, nonEmpty, size, hasDefiniteSize
Element	head, headOption, last, lastOption,
Retrieval	xs find p
Sub-collection	xs.tail, xs.init, xs slice (from, to), xs take n, xs drop n, xs takeWhile p, xs dropWhile p, xs filter p, xs withFilter p, xs filterNot p
Subdivision	xs splitAt n, xs span p, xs partition p, xs groupBy f
Element Condition	xs forall p, xs exists p, xs count p
Fold	(z /: xs)(op), (xs : z)(op), xs.foldLeft(z)(op), xs.foldRight(z)(op), xs reduceLeft op, xs reduceRight op
Specific Fold	xs.sum, xs.product, xs.min, xs.max
String	xs addString (b, start, sep, end), xs mkString (start, sep, end), xs.stringPrefix
View	xs.view, xs view (from, to)

Reference: http://docs.scala-lang.org/overviews/collections/trait-traversable.html

2.3 Trait Iterable

All methods in this trait are defined in terms of an an abstract method, `iterator`, which yields the collections elements one by one.

Table 2: Methods in Iterable

Category	Methods
Abstract	xs.iterator
Iterator	xs grouped n, xs sliding n
Subcollection	xs takeRight n, xs dropRight n
Zipper	xs zip ys, xs zipAll (ys, x, y), xs.zipWithIndex
Comparison	xs sameElements ys

Reference: http://docs.scala-lang.org/overviews/collections/trait-iterable.html

In the inheritance hierarchy below `Iterable` you find three traits: `Seq`, `Set`, and `Map`. A common aspect of these three traits is that they all implement the `PartialFunction` trait with its `apply` and `isDefinedAt` methods. However, the way each trait implements `PartialFunction` differs.

2.4 Seq

All methods in this trait are defined in terms of an an abstract method, `iterator`, which yields the collections elements one by one.

Table 3: Methods in Seq

Category	Methods
Indexing and Length	xs(i), xs isDefinedAt i, xs.length, xs.lengthCompare ys, xs.indices
Index Search	xs indexOf x, xs lastIndexOf x, xs indexOfSlice ys, xs lastIndexOfSlice ys, xs indexWhere p, xs segmentLength (p, i), xs prefixLength p
Addition	x +: xs, xs :+ x, xs padTo (len, x)
Update	xs patch (i, ys, r), xs updated (i, x), xs(i) = x(only available for mutable.Seqs)
Sorting	xs.sorted, xs sortWith lt, xs sortBy f
Reversal	xs.reverse, xs.reverseIterator, xs reverseMap f
Comparison	xs startsWith ys, xs endsWith ys, xs contains x, xs containsSlice ys, (xs corresponds ys)(p)
Multiset	xs intersect ys, xs union ys, xs diff ys, xs.distinct

Reference: http://docs.scala-lang.org/overviews/collections/seqs.html

Table 4: Methods in Buffer

Category	Methods
Addition	buf += x, buf += (x, y, z), buf ++= xs, x +=: buf, xs ++=: buf, buf insert (i, x), buf insertAll (i, xs)
Removal	buf -= x, buf remove i, buf remove (i, n), buf trimStart n, buf trimEnd n, buf.clear()
Cloning	buf.clone

2.5 Set

Table 5: Methods in Set

Category	Methods
Test	xs contains x, xs(x), xs subsetOf ys
Addition	xs + x, xs + (x, y, z), xs ++ ys
Removal	xs - x, xs - (x, y, z), xs -- ys, xs.empty
Set operation	xs & ys, xs intersect ys, xs | ys, xs union ys, xs & ys, xs diff ys

Reference: http://docs.scala-lang.org/overviews/collections/sets.html

Mutable sets offer in addition methods to add, remove, or update elements, which are summarized in below.

Table 6: Methods in mutable.Set

Category	Methods
Addition	xs += x, xs += (x, y, z), xs ++= ys, xs add x
Removal	xs -= x, xs -= (x, y, z), xs --= ys, xs remove x, xs retain p, xs.clear()
Update	xs(x) = b
Cloning	xs.clone

2.6 Map

Table 7: Methods in Map

Category	Methods
Lookup	ms get k, ms(k), ms getOrElse (k, d), ms contains k, ms isDefinedAt k
Addition	ms + (k -> v), ms + (k -> v, l -> w), ms ++ kvs
Removal	ms - k, ms - (k, 1, m), ms -- ks
Update	ms updated (k, v)
Subcollection	ms.keys, ms.keySet, ms.keyIterator, ms.values, ms.valuesIterator
Transformation	ms filterKeys p, ms mapValues f

Reference: http://docs.scala-lang.org/overviews/collections/maps.html

Table 8: Methods in mutable.Map

Category	Methods
Addition	ms += (k -> v), ms += (k -> v, l -> w), ms ++= kvs,
Removal	ms -= k, ms -= (k, l, m), ms --= ks, ms remove k, ms retain p, ms.clear()
Update	ms(k) = v, ms put (k, v), ms getOrElseUpdate (k, d)
Transformation	ms transform f
Cloning	xs.clone

2.7 Performance Characteristics

Table 9: Performance characteristics of sequence types

	head	**tail**	**apply**	**update**	**prepend**	**append**	**insert**
immutable							
List	C	C	L	L	C	L	-
Stream	C	C	L	L	C	L	-
Vector	eC	eC	eC	eC	eC	eC	-
Stack	C	C	L	L	C	C	L
Queue	aC	aC	L	L	L	C	-
Range	C	C	C	-	-	-	-
String	C	L	C	L	L	L	-

Reference: http://docs.scala-lang.org/overviews/collections/performance-characteristics.html

	head	tail	apply	update	prepend	append	insert
mutable							
`ArrayBuffer`	C	L	C	C	L	aC	L
`ListBuffer`	C	L	L	L	C	C	L
`StringBuilder`	C	L	C	C	L	aC	L
`MutableList`	C	L	L	L	C	C	L
`Queue`	C	L	L	L	C	C	L
`ArraySeq`	C	L	C	C	-	-	-
`Stack`	C	L	L	L	C	L	L
`ArrayStack`	C	L	C	C	aC	L	L
`Array`	C	L	C	C	-	-	-

Reference: `http://docs.scala-lang.org/overviews/collections/performance-characteristics.html`

Table 10: Performance characteristics of set and map types

	lookup	add	remove	min
immutable				
`HashSet/HashMap`	eC	eC	eC	L
`TreeSet/TreeMap`	Log	Log	Log	Log
`BitSet`	C	L	L	eC^1
`ListMap`	L	L	L	L
mutable				
`HashSet/HashMap`	eC	eC	eC	L
`WeakHashMap`	eC	eC	eC	L
`BitSet`	C	aC	C	eC^1
`TreeSet`	Log	Log	Log	Log

Footnote 1: Assuming bits are densely packed.

The entries in these two tables are explained as follows:

C	The operation takes (fast) constant time.
eC	The operation takes effectively constant time, but this might depend on some assumptions such as maximum length of a vector or distribution of hash keys.
aC	The operation takes amortized constant time. Some invocations of the operation might take longer, but if many operations are performed on average only constant time per operation is taken.
Log	The operation takes time proportional to the logarithm of the collection size.
L	The operation is linear, that is it takes time proportional to the collection size.
-	The operation is not supported.

3 Scala Parallel Collections

3.1 Creating a Parallel Collection

Two ways to create a parallel collection: `new` and `par`.

3.2 Semantics

Conceptually, Scalas parallel collections framework parallelizes an operation on a parallel collection by recursively splitting a given collection, applying an operation on each partition of the collection in parallel, and re-combining all of the results that were completed in parallel.

These concurrent, and out-of-order semantics of parallel collections lead to the following two implications:

1. Side-effecting operations can lead to non-determinism
2. Non-associative operations lead to non-determinism

3.3 Concrete Parallel Collection Classes

`mutable.ParArray`, `immutable.ParVector`, `immutable.ParRange`, `mutable.ParHashSet`, `mutable.ParHashMap`, `mutable.ParTrieMap`

3.4 Performance characteristics

Table 11: Performance characteristics of sequence types

	head	tail	apply	update	prepend	append	insert
`ParArray`	C	L	C	C	L	L	L
`ParVector`	eC	eC	eC	eC	eC	eC	-
`ParRange`	C	C	C	-	-	-	-

`http://docs.scala-lang.org/overviews/parallel-collections/concrete-parallel-collections.html`

Table 12: Performance characteristics of set and map types

	lookup	add	remove
immutable			
`ParHashSet/ParHashMap`	eC	eC	eC
mutable			
`ParHashSet/ParHashMap`	C	C	C
`ParTrieMap`	eC	eC	eC

3.5 Parallel Collection Conversions

Every sequential collection can be converted to its parallel variant using the `par` method. Certain sequential collections have a direct parallel counterpart. For these collections the conversion is efficient it occurs in constant time, since both the sequential and the parallel collection have the same data-structural representation (one exception is mutable hash maps and hash sets which are slightly more expensive to convert the first time `par` is called, but subsequent invocations of par take constant time). It should be noted that for mutable collections, changes in the sequential collection are visible in its parallel counterpart if they share the underlying data-structure.

Table 13: Sequential collections and their direct parallel counterparts

Sequential	Parallel
mutable	
`Array`	`ParArray`
`HashMap`	`ParHashMap`
`HashSet`	`ParHashSet`
`TrieMap`	`ParTrieMap`
immutable	
`Vector`	`ParVector`
`Range`	`ParRange`
`HashMap`	`ParHashMap`
`HashSet`	`ParHashSet`

Source: `http://docs.scala-lang.org/overviews/parallel-collections/conversions.html`

Other collections, such as lists, queues or streams, are inherently sequential in the sense that the elements must be accessed one after the other. These collections are converted to their parallel variants by copying the elements into a similar parallel collection. For example, a functional list is converted into a standard immutable parallel sequence, which is a parallel vector.

Every parallel collection can be converted to its sequential variant using the seq method. Converting a parallel collection to a sequential collection is always efficient it takes constant time. Calling seq on a mutable parallel collection yields a sequential collection which is backed by the same store updates to one collection will be visible in the other one.

3.6 Architecture of the Parallel Collections Library

Two core abstractions: `Splitter`s and `Combiner`s.

Splitter

```
trait Splitter[T] extends Iterator[T] {
    def split: Seq[Splitter[T]]
}
```

Combiner

```
trait Combiner[Elem, To] extends Builder[Elem, To] {
    def combine(other: Combiner[Elem, To]): Combiner[Elem, To]
}
```


Github: `https://github.com/soulmachine/scala-cheat-sheet`
My blog: `http://www.soulmachine.me`
Last update: March 28, 2014