

Linear Collections



Sequences

```
trait Seq[+A] {  
  def head: A  
  def tail: Seq[A]  
}
```

A (very) general interface for data structures that

- have a well-defined order
- can be indexed

Supports various operations:

- *apply, iterator, length, reverse* for indexing and iterating
- concatenation, appending, prepending
- a lot of others: grouping, sorting, zipping, searching, slicing

Lists

```
sealed abstract class List[+A]  
case object Nil extends List[Nothing]  
case class ::[A](val hd: A, val tl: List[A]) extends List[A]
```

A LinearSeq *immutable* linked list

- *head, tail, isEmpty* methods are fast: $O(1)$
- most operations are $O(n)$: *length, reverse*

Sealed - has two subtypes:

- object Nil (empty)
- class ::

Lists in Action

Vararg *apply* factory method for building instances

```
val numbers: List[Int] = List(1,2,3)
```

Some of the most used operations: accessors and transforms

```
numbers.head           // 1
numbers.tail           // List(2,3)
numbers.map(_ + 1)      // List(2,3,4)
```

Handy for comprehensions

```
for (x <- numbers if x > 2)
  yield (x, x * x)      // List((3,9))
```



Lists in Action (2)

Lists reuse references to tails

sugar for
:::apply(42, simple)

```
// structural re-use
val simple = List(54)
val prepended = 42 :: simple

prepending.tail == simple           // true
```



A lot of useful functions are supported out of the box

```
// other utilities
List.fill(3)("apples")           // List("apples", "apples", "apples")
numbers foreach println          // prints all elements to console
numbers mkString ";"             // 1; 2; 3
numbers.reverse                  // List(3,2,1)
```

Arrays

```
final class Array[T]  
  extends java.io.Serializable  
  with java.lang.Cloneable
```

The equivalent of simple Java arrays

- can be manually constructed with predefined lengths
- can be *mutated* (updated in place)
- are interoperable with Java's `T[]` arrays
- indexing is fast
- has most Seq methods

Where's the Seq in the definition?!*



Arrays in Action

Instantiating "manually", with an exact length

```
val allocatedArray = new Array[String](3) // filled with nulls
```

Instantiating via the *apply* vararg factory method

```
val numbers = Array(1,2,3,4)
```

Multi-dimensional arrays = arrays of arrays

```
val multiDimArray = Array.ofDim(2,3) // a 2 x 3 array
```

Accessing elements: use *apply*

```
numbers(3) // 4
```

Arrays in Action (2)

Modifying an element in-place: use *update*

```
// syntax sugar - rewritten as numbers.update(1, 9)
numbers(1) = 9
```

Standard functions, uniform with the other Seqs

```
numbers :+ 5           // Array(1,9,3,4,5)
numbers.reverse       // Array(4,3,9,1) via a LOT of magic
```

Automatic conversion to Seq

```
// implicit* conversion to Seq[Int]
val numbersSeq: Seq[Int] = numbers
// utilities
numbersSeq.sum           // 17
numbersSeq.toList ←      // List(1,9,3,4)
```

also available in all collections:

toArray, toMap, toSet, toSeq,
toStream*

Vectors

```
final class Vector[+A]
```

A fast implementation for *immutable sequences*

- *effectively constant* indexed read and write: $O(\log_{32}(n))$
- fast element addition: append/prepend
- implemented as a fixed-branched trie (branch factor 32)
- good performance for large sizes

```
val noElements = Vector.empty
val numbers = noElements :+ 1 :+ 2 :+ 3      // Vector(1,2,3)
val modified = numbers updated (0, 7)      // Vector(7,2,3)
```

Vectors in Action

A small List vs Vector performance comparison

- measure time for *updated* calls
- *updated* modifies an element and returns a new collection

Testing for:

- a large number of elements (1 million)
- a significant number of random writes (1000)

```
val maxCapacity = 1000000  
val maxRuns = 1000
```

Set

```
trait Set[+A] extends ... {  
  def contains(elem: A): Boolean  
  def +(elem: A): Set[A]  
  def -(elem: A): Set[A]  
}
```

A general interface for a data structure that

- contains no duplicates
- can test whether it contains an element

Supports various operations:

- adding/removing an element
- union, difference, intersection
- conversion to/from other linear collections

Scala rocks

