

Collections

Introduction

- ❖ The Scala programming language has a rich library of collection classes, that allow us to create various collection types, such as maps, sets, lists, arrays and others.
- ❖ Most collection classes exist in three packages, `scala.collection`, `scala.collection.immutable` and `scala.collection.mutable`.

Introduction

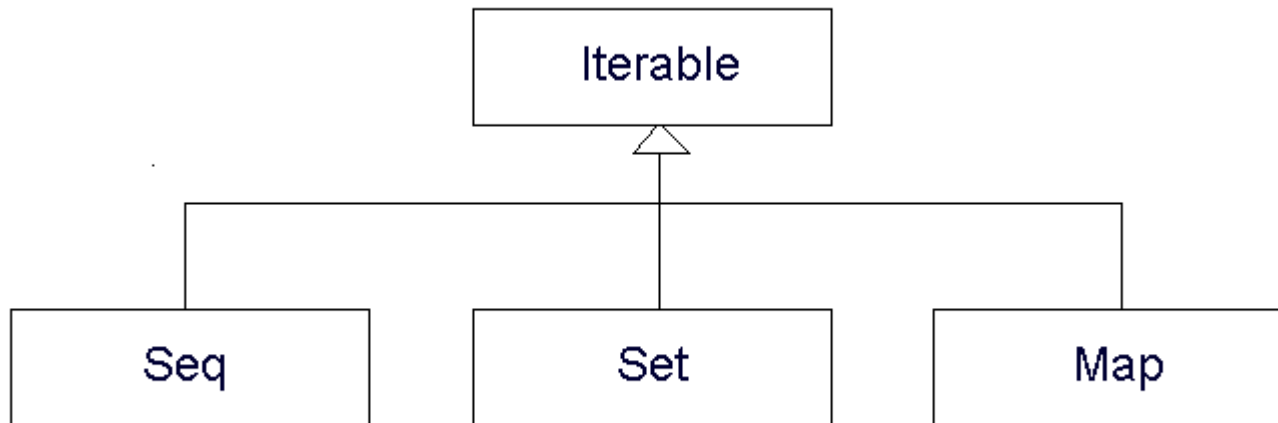
- ❖ The `scala.collection` package includes all high level abstract classes or traits. Most of them have both mutable and immutable implementations. These implementations reside in the `scala.collection.immutable` and `scala.collection.mutable` packages.

The Iterable Trait

- ❖ This is the main trait the collections library includes. Types mixed in with this trait can be iterated.
- ❖ This trait represents a collection that can get us an iterator we can use to iterate the elements.

The Iterable Trait

- ❖ The `Iterable` trait is the base trait for `Seq`, `Set` and `Map`.



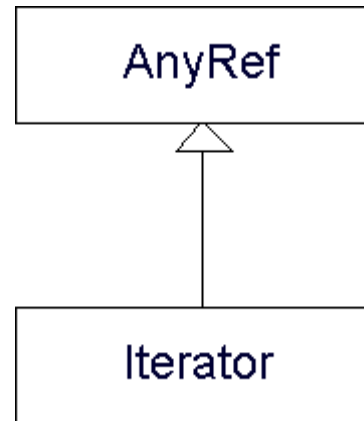
The Iterable Trait

- ❖ We get an iterator by calling the `iterator` method:

```
def iterator: Iterator[A]
```

The Iterator Trait

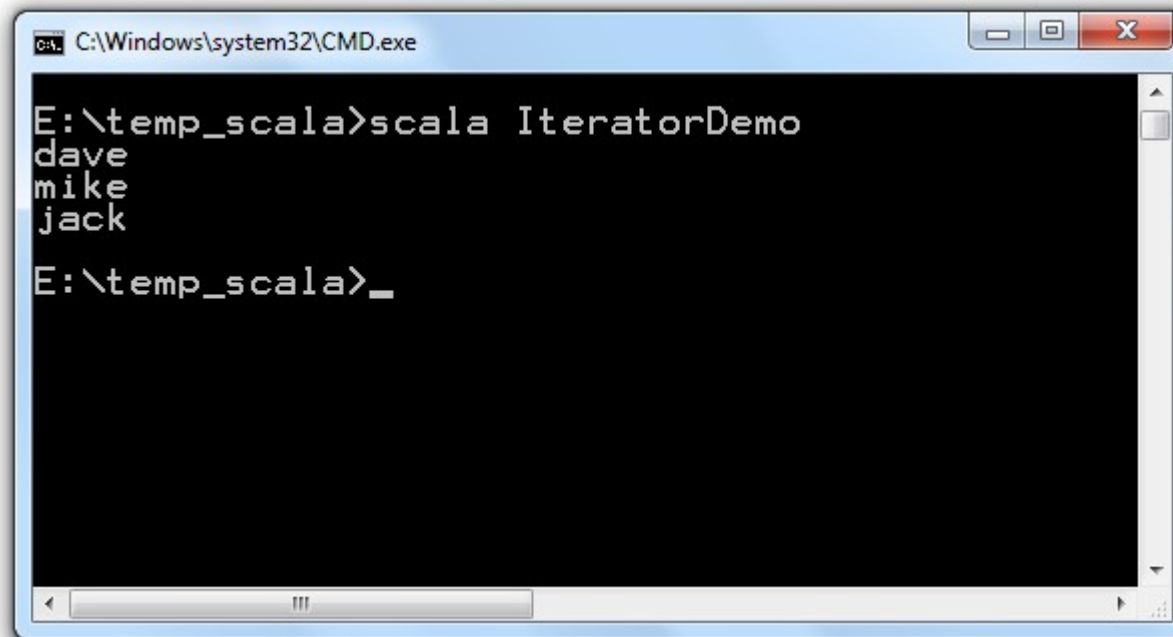
- ❖ The `Iterator` trait extends `AnyRef`. We can get an iterator through which we will iterate both finite and infinite collections of elements.



The Iterator Trait

```
object IteratorDemo
{
  def main(args: Array[String])
  {
    var ob:List[String] = List("dave","mike","jack")
    var iterator:Iterator[String] = ob.iterator
    while(iterator.hasNext)
    {
      println(iterator.next)
    }
  }
}
```


The Iterator Trait



```
C:\Windows\system32\CMD.exe
E:\temp_scala>scala IteratorDemo
dave
mike
jack
E:\temp_scala>_
```

Sequences

- ❖ The `Seq` trait defines a sequence of ordered elements.
- ❖ Classes (e.g. `Array`, `List`, `Queue`, `Stack`) that extend this trait describe collections of ordered elements.

Lists

- ❖ List is one of the most commonly used data structure in the Scala programming language. All elements the List object holds should be of the same type.

Lists

- ❖ We can easily create new List object by writing the word List followed by parentheses with the list values inside.

...

```
val colors = List('yellow', 'brown', 'blue', 'green', 'black')
```

```
val numbers = List(1, 20, 22, 12, 82, 8, 4)
```

...

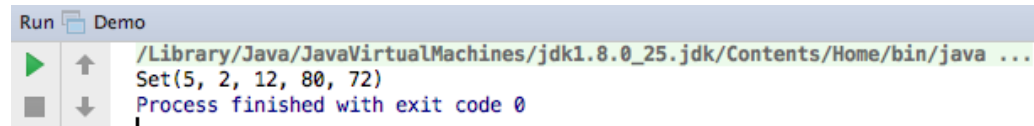
Sets

- ❖ The `Set` trait defines a sequence of unique elements.
- ❖ The Scala programming language offers both mutable and immutable versions of sets.
- ❖ By default, the Scala programming language uses the immutable `Set`. If we want to use the mutable `Set`, we should explicitly import the `scala.collection.mutable.Set` class.

Sets

- ❖ When creating a new `Set` object we can either create an empty set or pass over the elements to the constructor.

```
object Demo
{
  def main(args:Array[String]):Unit =
  {
    var set1:Set[Int] = Set()
    var set2:Set[Int] = Set(12,5,2,72,80)
    print(set2)
  }
}
```



```
Run Demo
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...
Set(5, 2, 12, 80, 72)
Process finished with exit code 0
```

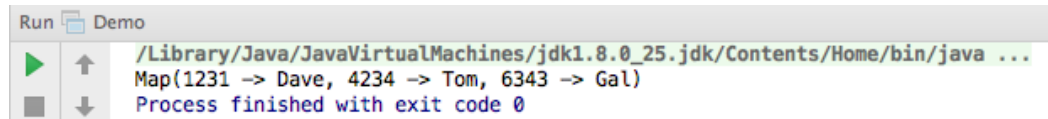
Maps

- ❖ The `Map` trait defines a sequence of non-ordered unique key-value elements.
- ❖ The keys are unique. The values don't need to be unique.
- ❖ The Scala programming language offers both mutable and immutable versions of maps. By default, Scala uses the immutable `Map`. In order to use the mutable `Map`, we will need to import the `scala.collection.mutable.Map` class explicitly.

Maps

- ❖ When creating a new empty map we should specify the types of the keys and the values. When creating a non empty map we can avoid it.

```
object Demo
{
  def main(args:Array[String]):Unit =
  {
    val map1:Map[Int,String] = Map()
    val map2 = Map(1231->"Dave", 4234->"Tom", 6343->"Gal")
    print(map2)
  }
}
```



```
Run Demo
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...
Map(1231 -> Dave, 4234 -> Tom, 6343 -> Gal)
Process finished with exit code 0
```

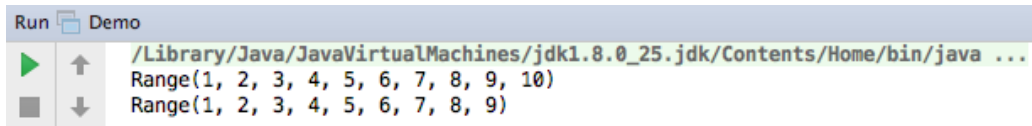

Range

- ❖ The `Range` collection represents a range of `Int` numbers. Creating a new `Range` object can be fairly simple. We just need to specify the range using two `Int` numbers and using the `to` or the `until` keywords.

Range

```
package com.lifemichael.samples

object Demo
{
  def main(args:Array[String]):Unit =
  {
    val a = 1 to 10
    println(a)
    val b = 1 until 10
    println(b)
  }
}
```



```
Run Demo
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...
Range(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
Range(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

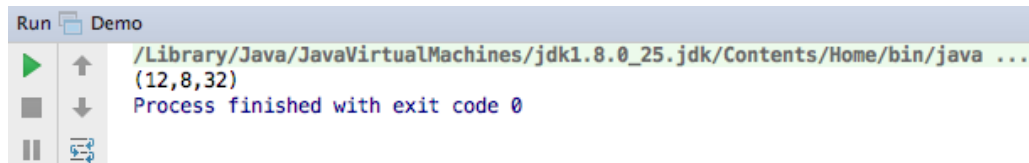
Tuples

- ❖ Tuple in Scala has a fixed number of items. When passing over a tuple we actually pass over all items together, as a whole.
- ❖ Unlike arrays and lists the tuple can hold objects with different types. The objects the tuple holds must be immutable.

Tuples

- ❖ The simplest way for creating a new tuple would be putting the values together in parentheses.

```
object Demo
{
  def main(args:Array[String]):Unit =
  {
    val numbers = (12,8,32)
    print(numbers)
  }
}
```



```
Run Demo
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...
(12,8,32)
Process finished with exit code 0
```

Options

- ❖ The Scala `Option[T]` is a container for zero or one element of a given type.
- ❖ The `Option[T]` can be either an object of the type `Some[T]` or of the type `None`. Object of the type `None` represents a missing value.

Options

- ❖ The following code shows that we can assign a variable of the type `Option` either with a reference for a `Some` object or with a reference for `None` object.

```
object Demo
{
  def main(args:Array[String]):Unit =
  {
    var temp:Option[Int] = Some(5)
    temp = None
  }
}
```