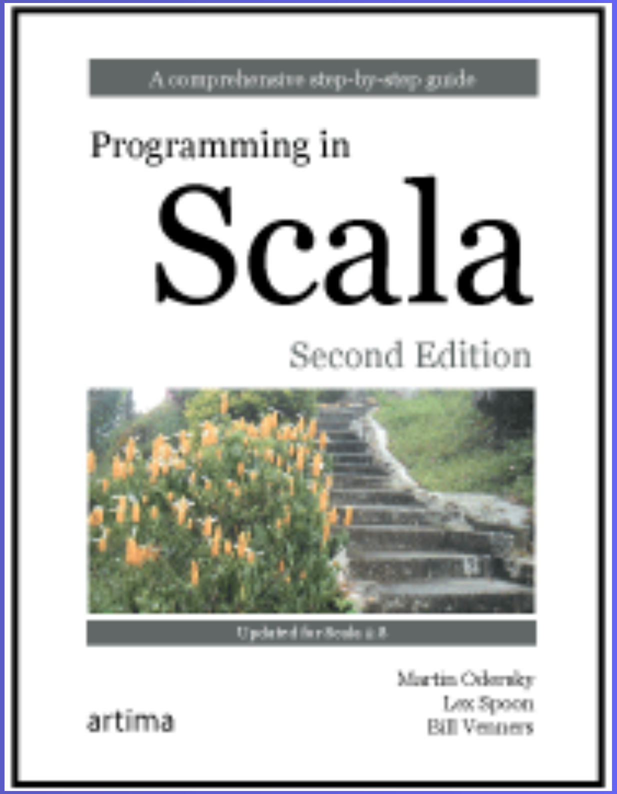
Stairway to Scala - Flight 2

Next steps in Scala

Bill Venners
Dick Wall

www.artima.com



Flight 2 goal

Get familiar with collections, look at what it means to be "functional," and learn how to process files with Scala scripts.

Parameterize arrays with types:

```
val greetStrings = new Array[String](3)

greetStrings(0) = "Hello"
greetStrings(1) = ", "
greetStrings(2) = "world!\n"

for (i <- 0 to 2)
  print(greetStrings(i))
```

All operations are method calls:

```
1 + 2

(1).+(2)
```

apply and update

```
greetStrings(i)
greetString.apply(i)

greetStrings(0) = "Hello"
greetStrings.update(0, "Hello")
```

Creating and initializing an array

```
val numNames = Array("zero", "one", "two")
```

```
val numNames2 = Array.apply("zero", "one", "two")
```

Creating and initializing a list

```
val oneTwoThree = List(1, 2, 3)
```

Lists are immutable

```
val oneTwo = List(1, 2)
val threeFour = List(3, 4)
val oneTwoThreeFour = oneTwo ::: threeFour
println(""+ oneTwo +" and "+ threeFour +" were not mutated.")
println("Thus, "+ oneTwoThreeFour +" is a new list.")
```

```
List(1, 2) and List(3, 4) were not mutated.
Thus, List(1, 2, 3, 4) is a new list.
```

Consing lists

```
val twoThree = List(2, 3)
val oneTwoThree = 1 :: twoThree
println(oneTwoThree)
```

```
List(1, 2, 3)
```

```
1 :: twoThree
twoThree.::(1)
```

Initializing lists with cons and Nil

```
val oneTwoThree = 1 :: 2 :: 3 :: Nil
println(oneTwoThree)
```

```
List(1, 2, 3)
```

Converting between Lists and Arrays

```
scala> Array(1,2,3).toList
res0: List[Int] = List(1, 2, 3)

scala> List(1,2,3).toArray
res1: Array[Int] = Array(1, 2, 3)
```

Creating and using a tuple

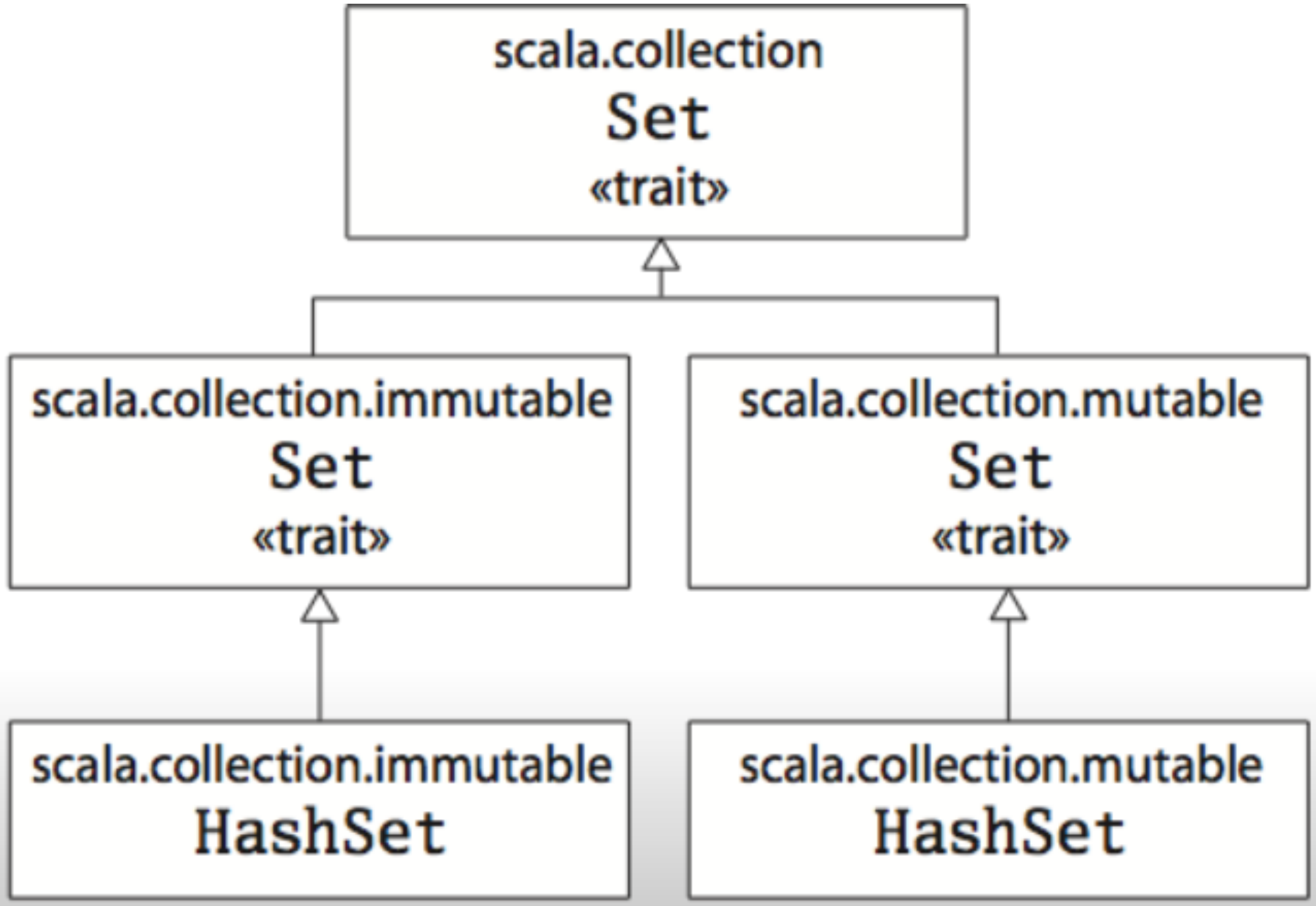
```
val pair = (99, "Luftballons")
println(pair._1)
println(pair._2)
```

```
99
Luftballons
```

Tuple types

```
(99, "Luftballons")
Tuple2[Int, String]

('u', 'r', "the", 1, 4, "me")
Tuple6[Char, Char, String, Int, Int, String]
```

Set hierarchy



Creating, initializing, and using an immutable set

```
var jetSet = Set("Boeing", "Airbus")
jetSet += "Lear"
println(jetSet.contains("Cessna"))
```

```
jetSet = jetSet + "Lear"
```

Creating, initializing, and using a mutable set

```
import scala.collection.mutable.Set

val movieSet = Set("Hitch", "Poltergeist")
movieSet += "Shrek"
println(movieSet)
```

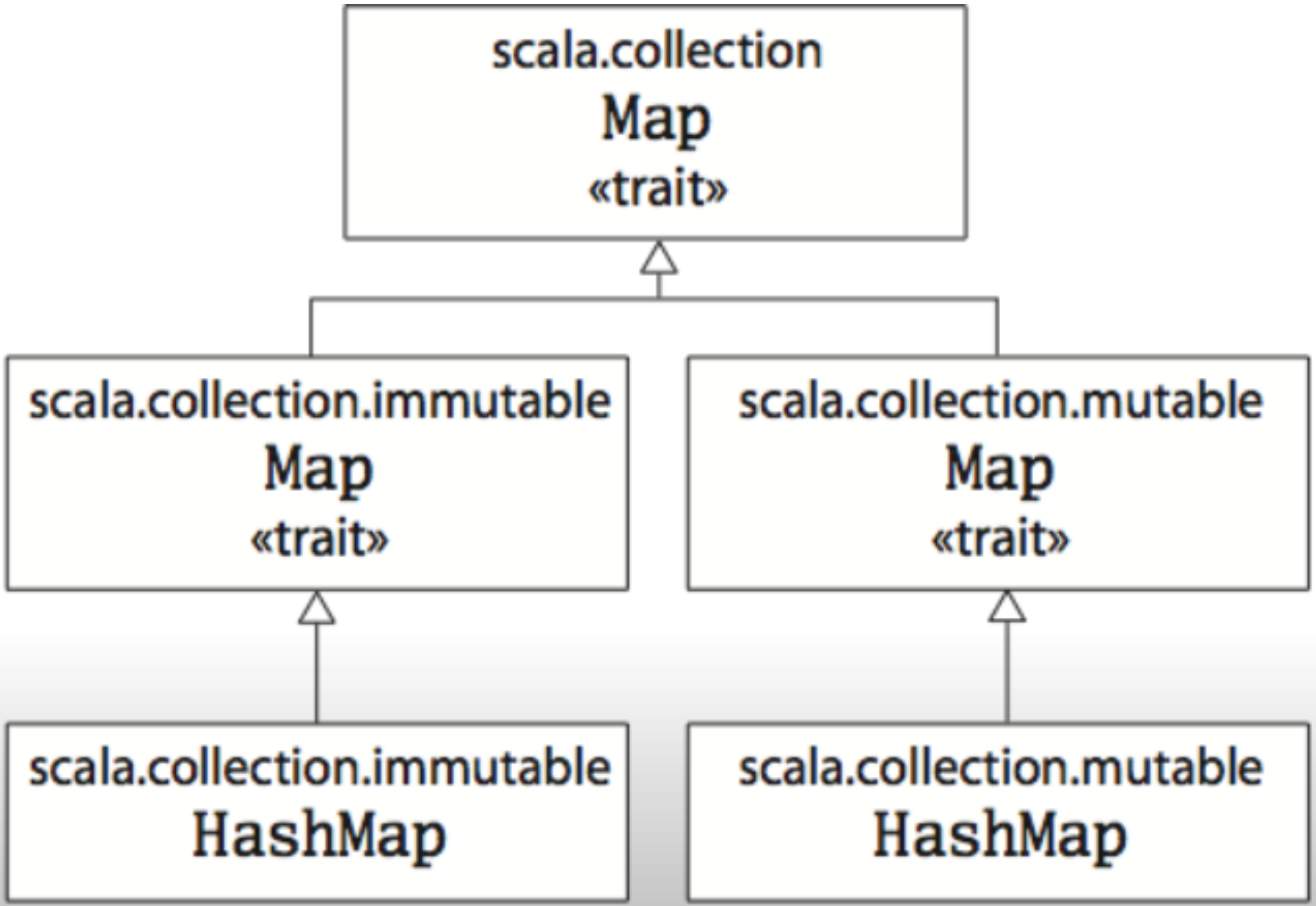
If you need a set class other than the default

```
import scala.collection.immutable.HashSet

val hashSet = HashSet("Tomatoes", "Chilies")
println(hashSet + "Coriander")
```

Importing a package

```
import scala.collection._

val mut = mutable.Set(1, 2, 3)
val imm = immutable.Set(4, 5, 6)
```

Map hierarchy



Creating, initializing, and using a mutable map

```
import scala.collection.mutable.Map

val treasureMap = Map[Int, String]()
treasureMap += (1 -> "Go to island.")
treasureMap += (2 -> "Find big X on ground.")
treasureMap += (3 -> "Dig.")
println(treasureMap(2))
```

```
Find big X on ground.
```

Implicit conversions

```
3 -> "Dig."

(3).->("Dig.")

any2ArrowAssoc(3).->("Dig.")
```

Creating, initializing, and using an immutable map

```
val romanNumeral = Map(
  1 -> "I", 2 -> "II", 3 -> "III", 4 -> "IV", 5 -> "V"
)
println(romanNumeral(4))
```

An imperative method

```scala
def printArgs(args: Array[String]): Unit = {
  var i = 0
  while (i < args.length) {
    println(args(i))
    i += 1
  }
}
```

More functional...

```
def printArgs(args: Array[String]): Unit = {
  for (arg <- args)
    println(arg)
  }

def printArgs(args: Array[String]): Unit = {
  args.foreach(println)
}
```

Fully functional

```
def formatArgs(args: Array[String]): String =
args.mkString("\n")

println(formatArgs(args))

val res = formatArgs(Array("zero", "one", "two"))
assert(res == "zero\none\ntwo")
```

A balanced attitude for Scala programmers

Prefer vals, immutable objects and methods without side effects. Reach for them first. Use vars, mutable objects, and methods with side effects when you have a specific need and justification for them.

Reading lines from a file

```
import scala.io.Source

if (args.length > 0) {
  for (line <- Source.fromFile(args(0)).getLines)
    print(line.length +" "+ line)
}
else
  Console.err.println("Please enter filename")
```

$ scala countchars1.scala countchars1.scala

```
23 import scala.io.Source
1
23 if (args.length > 0) {
1
50 for (line <- Source.fromFile(args(0)).getLines)
36 print(line.length +" "+ line)
2 }
5 else
47 Console.err.println("Please enter filename")
```