Programming with Scala

Dick Murray
rmurray10127@gmail.com

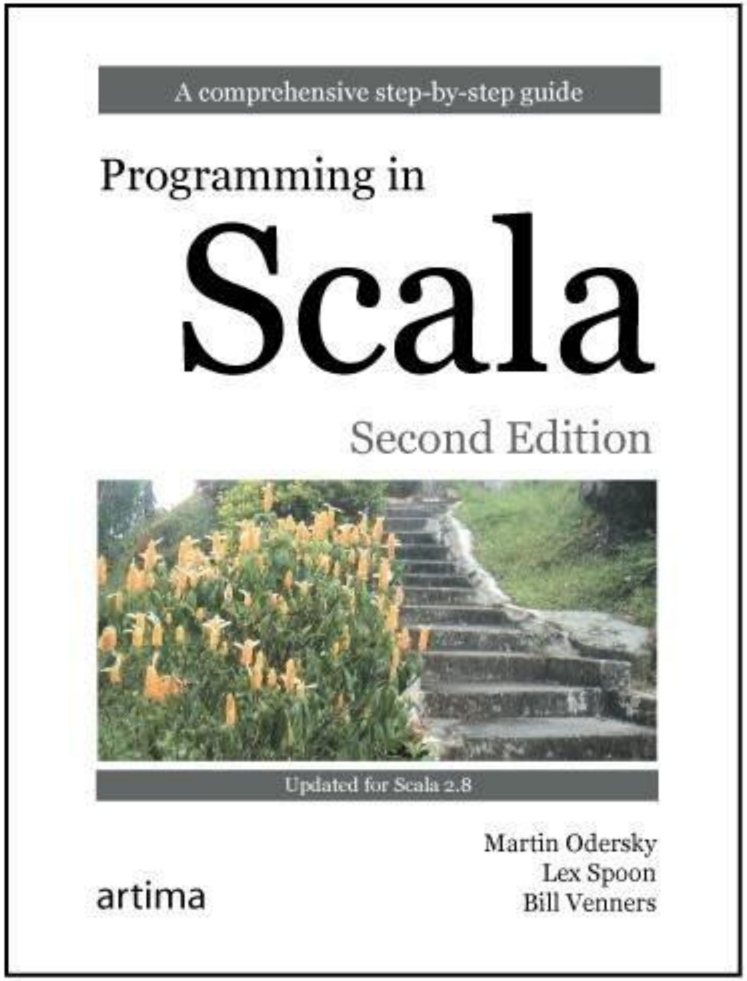
Agenda

- What is Scala?
- Functional Style of Programming
- Collections and Composition
- Patterns and Pattern Matching

What is Scala?

- SCA-lable LA-nguage
  - Invented by Martin Odersky at EPFL (École Polytechnique Fédérale de Lausanne)
  - Blend of OO and FP that can be applied to a wide range of programming tasks
  - Complementary strengths
    - OO – Build large systems which are adaptable
    - FP – Build interesting things quickly from simple parts
  - Runs on JVM and interoperates with all Java libraries
  - Advanced Static type system with highly effective type inference

Scala also means Stairs in Italian



A comprehensive step-by-step guide

Programming in

Scala

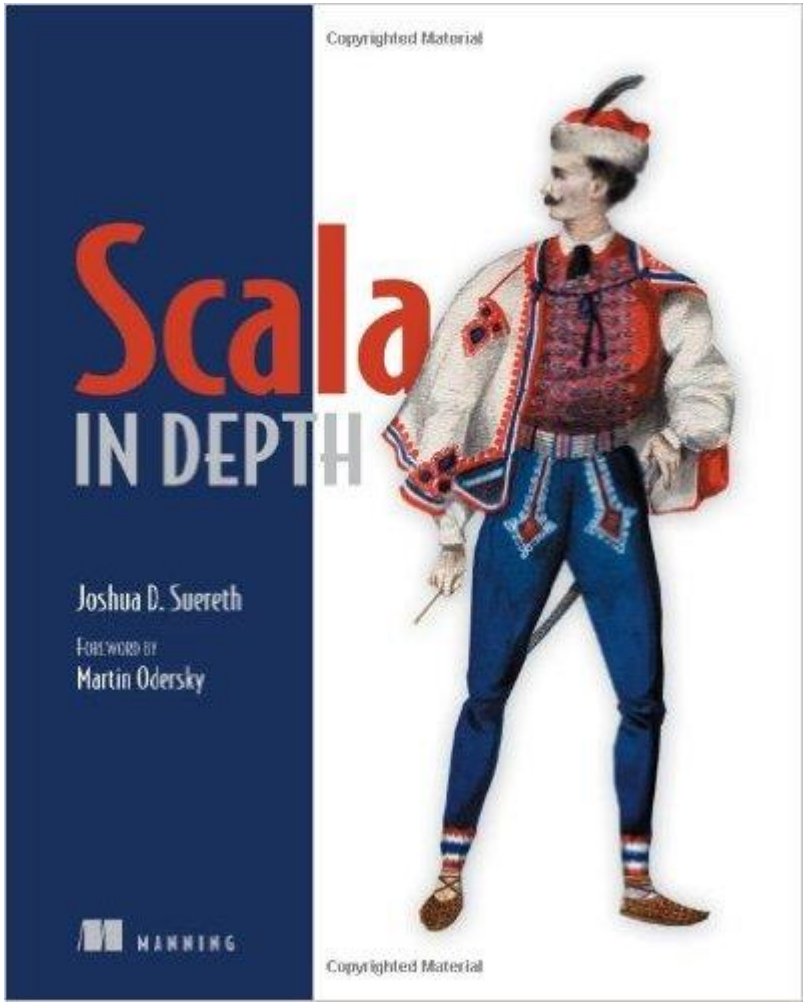
Second Edition

Updated for Scala 2.8

Martin Odersky
Lex Spoon
Bill Venners

artima

Best Books as ranked by community



Functional Style of Programming

- More tools in your toolbox
  - Functions
  - Immutability
  - Stateless (no side effects)
- Why*
  - Better ways of handling complexity (glue)
  - Better testing
  - OO makes code understandable by encapsulating moving parts. FP makes code understandable by minimizing moving parts – Michael Feathers on Twitter

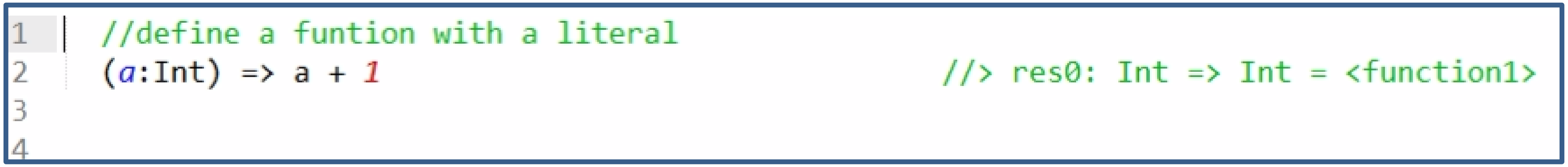
*John Hughes Paper: http://www.cs.kent.ac.uk/people/staff/dat/miranda/whyfp90.pdf

Functions as First Class Entities

- Functions are First Order
  - Functions may appear in any context that other first class entities (e.g. Integer) may appear
- Functions are Higher Order
  - Functions may be used as parameters to other functions
  - Functions may be returned as a value by another function

Functions as First Class Entities

First Order



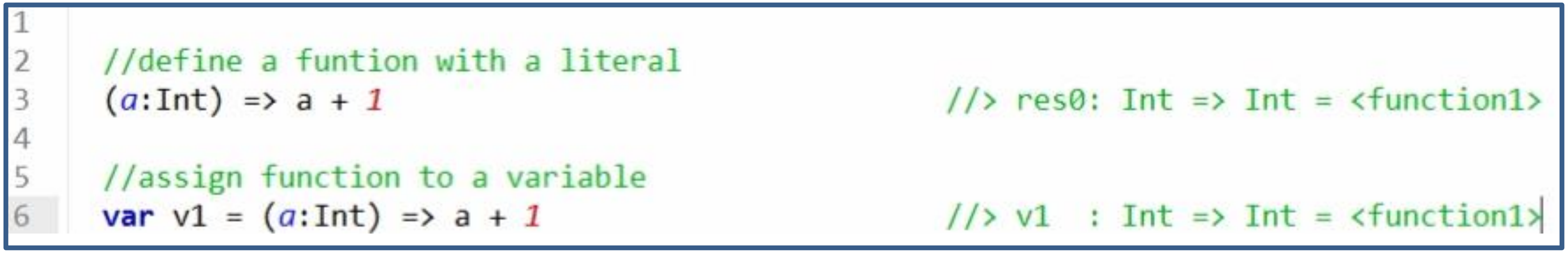
```
//define a funtion with a literal
(a:Int) => a + 1                                  //> res0: Int => Int = <function1>
```

Define a function with a literal

- Display shows REPL (read, evaluate,print,loop) tool in Scala IDE called worksheet.
  - Comments are preceded with // and are in green
  - Scala statements are to the left and are not green
  - Results from evaluating Scala statements are to the right, preceded by //> (first line) or //| (line continuation), and are in green

Functions as First Class Entities

First Order

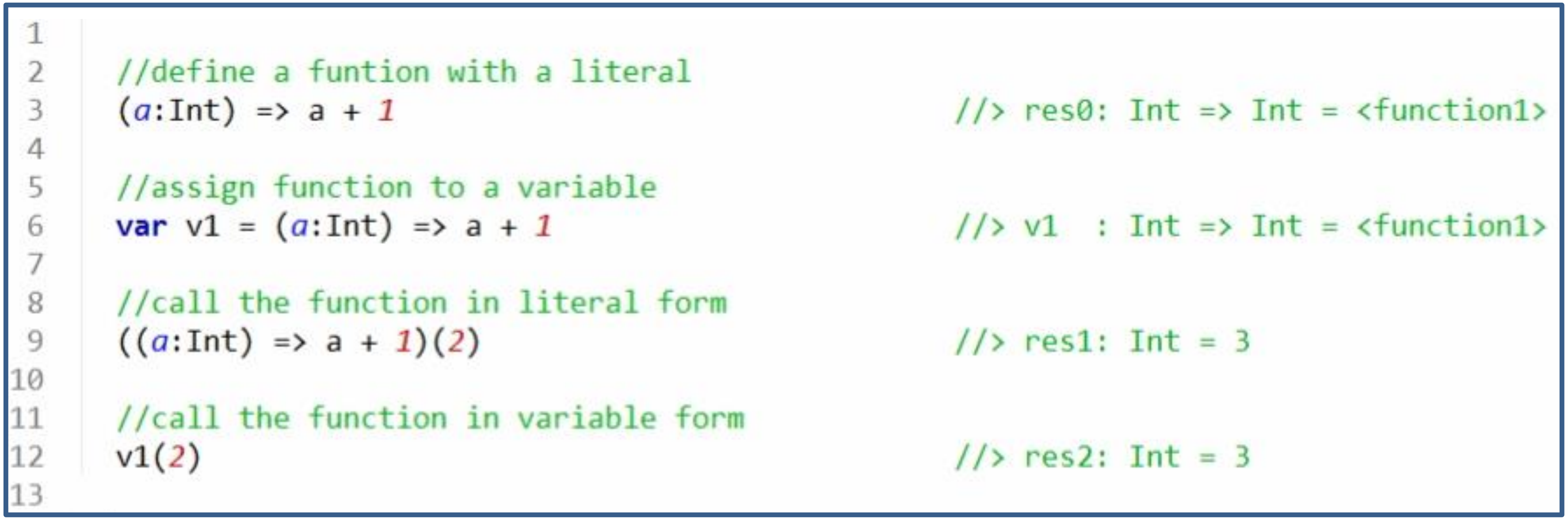


```

//define a funtion with a literal
(a:Int) => a + 1                                   //> res0: Int => Int = <function1>

//assign function to a variable
var v1 = (a:Int) => a + 1                          //> v1  : Int => Int = <function1>
```

Assign function definition to a variable

Functions a First Class Entities

First Order

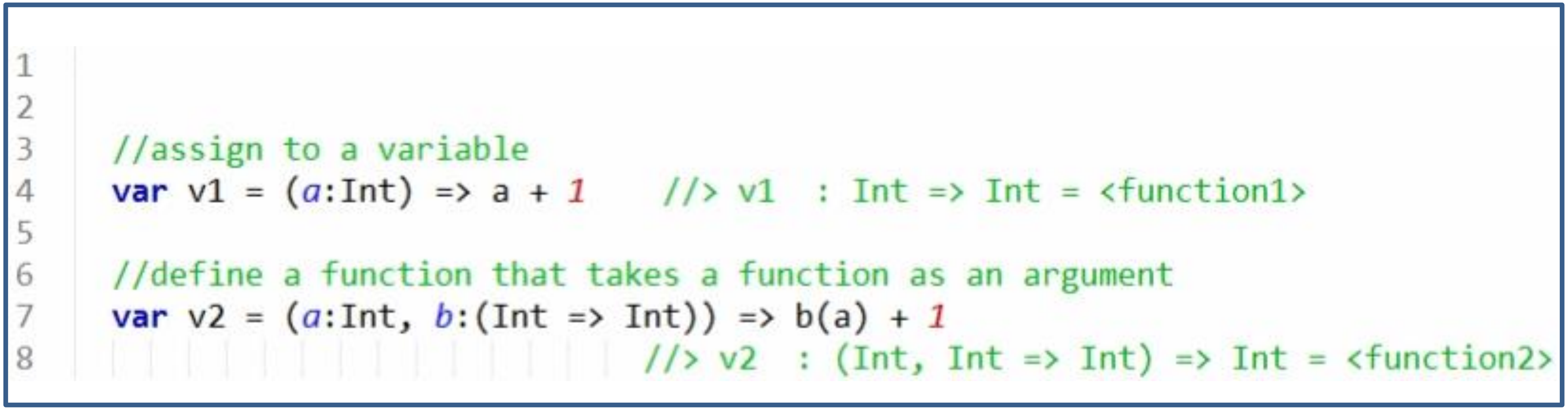


```

//define a funtion with a literal
(a:Int) => a + 1                                //> res0: Int => Int = <function1>

//assign function to a variable
var v1 = (a:Int) => a + 1                       //> v1  : Int => Int = <function1>

//call the function in literal form
((a:Int) => a + 1)(2)                           //> res1: Int = 3

//call the function in variable form
v1(2)                                           //> res2: Int = 3

```

Evaluate each form with parameter = 2

Functions a First Class Entities

Higher Order

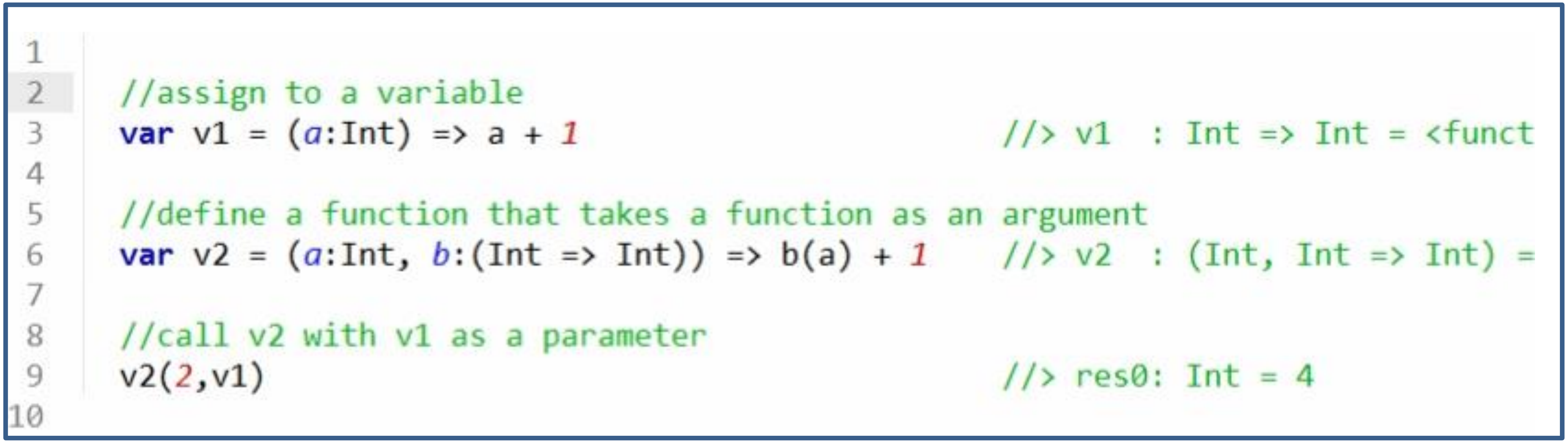


```


//assign to a variable
var v1 = (a:Int) => a + 1    //> v1  : Int => Int = <function1>

//define a function that takes a function as an argument
var v2 = (a:Int, b:(Int => Int)) => b(a) + 1
                                  //> v2  : (Int, Int => Int) => Int = <function2>
```

Define a function that takes a function as a parameter

Functions a First Class Entities

Higher Order

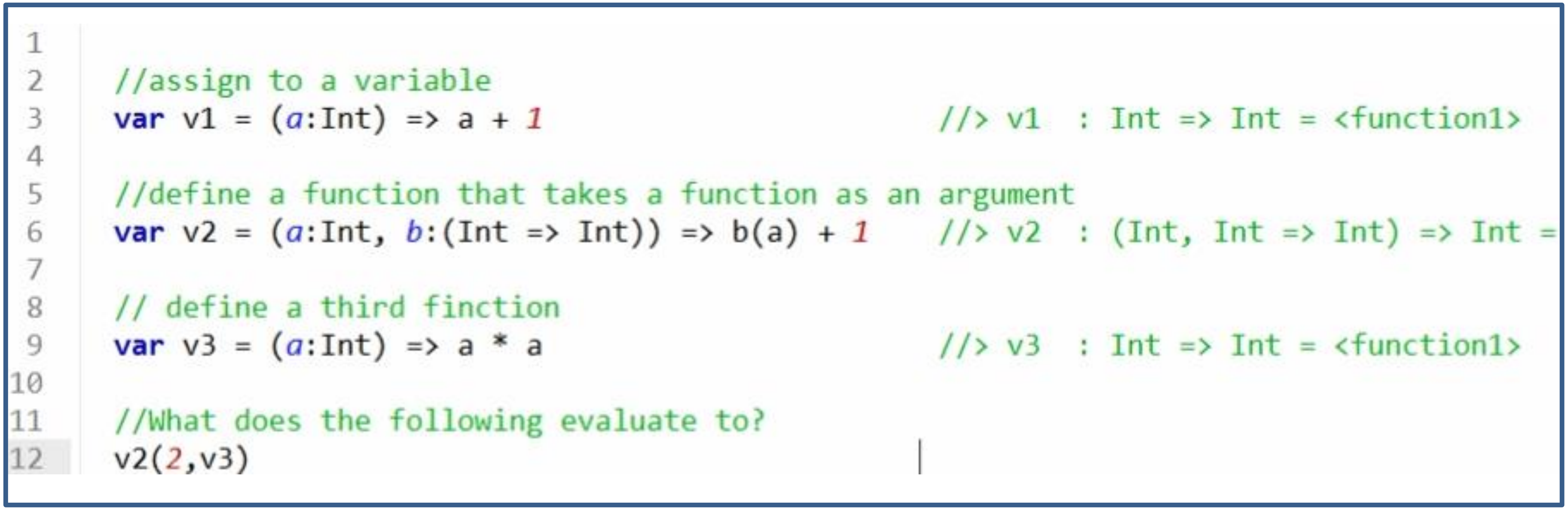


```

//assign to a variable
var v1 = (a:Int) => a + 1                       //> v1  : Int => Int = <funct

//define a function that takes a function as an argument
var v2 = (a:Int, b:(Int => Int)) => b(a) + 1    //> v2  : (Int, Int => Int) =

//call v2 with v1 as a parameter
v2(2,v1)                                        //> res0: Int = 4

```

Evaluate v2 with the v1 as parameter

Functions a First Class Entities

Higher Order



```

//assign to a variable
var v1 = (a:Int) => a + 1                       //> v1  : Int => Int = <function1>

//define a function that takes a function as an argument
var v2 = (a:Int, b:(Int => Int)) => b(a) + 1    //> v2  : (Int, Int => Int) => Int =

// define a third finction
var v3 = (a:Int) => a * a                       //> v3  : Int => Int = <function1>

//What does the following evaluate to?
v2(2,v3)
```

What does v2(2,v3) evaluate to?

Why is First Class Important?

- Remember the power of Unix pipe operator?
- ps –a | grep –v “\.py” | wc –l
  - ps Select all processes (generates 1 line of text per process
  - grep Searches text (passes thru all lines not containing .py
  - wc count words (-l counts lines)
- Pipe takes the output of one process and uses it as the input to the next process. (Build interesting things from simple parts)
- Functional style programming does something similar: “chained” methods on collections.

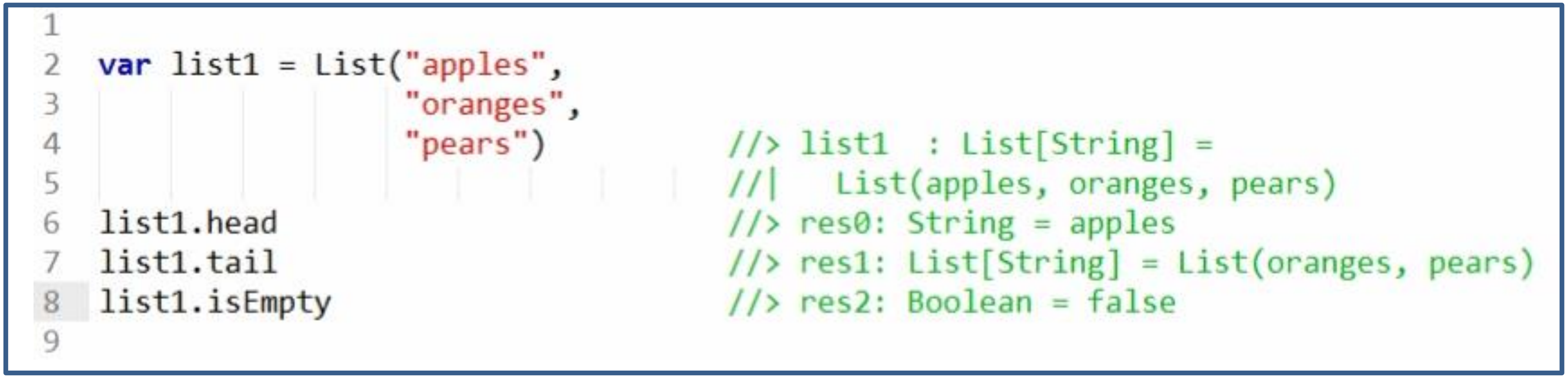
Collections and Composition

- Collections are a big deal in Scala
- They include collection operations, higher order methods that compose, iterate over, and extract elements.
- Three main collection types: List, Set, Map
  - Immutable, mutable
- We will concentrate on the List to explore collection operations.

Scala Lists

- Most commonly used data structure in Scala programming.
- Lists can be declared, constructed, and decomposed
- Lists are like arrays except
  - Lists are immutable
  - Lists have a recursive structure (i.e. linked list)
- Lists are homogeneous (maybe…)
- Lists will be important as we proceed to pattern matching

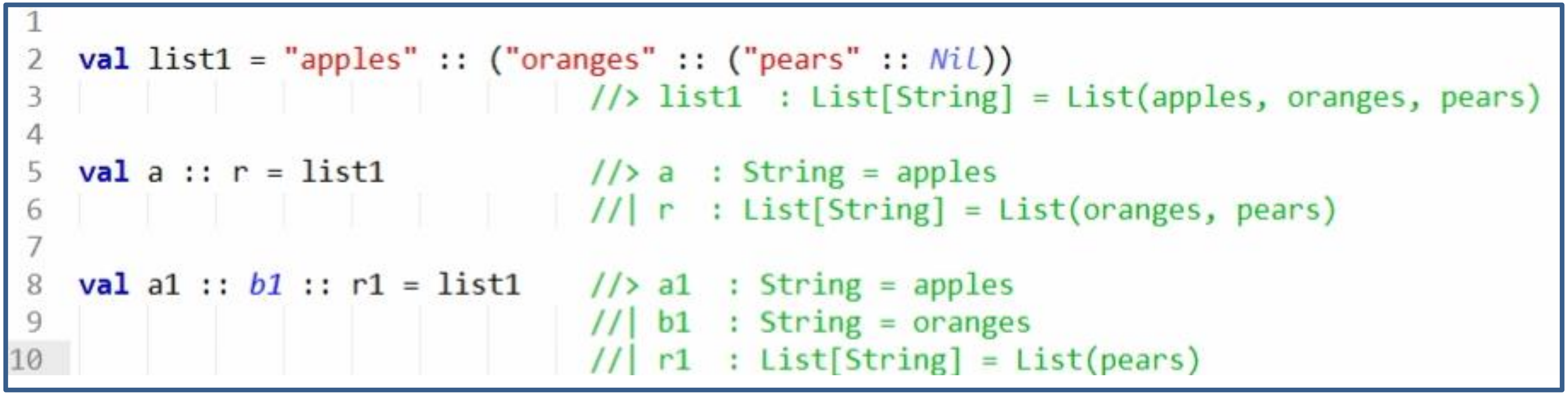
Scala Lists



```
var list1 = List("apples",
                 "oranges",
                 "pears")                //> list1  : List[String] =
                                         //|   List(apples, oranges, pears)
list1.head                               //> res0: String = apples
list1.tail                               //> res1: List[String] = List(oranges, pears)
list1.isEmpty                            //> res2: Boolean = false
```

- Classic (specifying) look at List of Strings (Note String type inferred)
- List methods used to identify components of the list

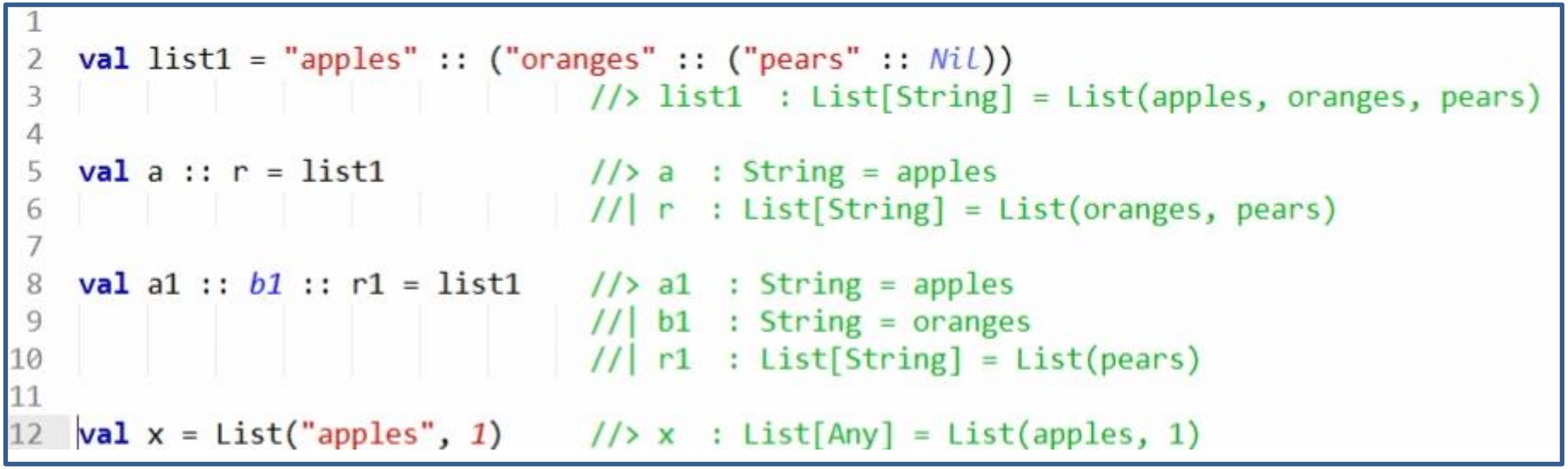
Scala Lists



```

val list1 = "apples" :: ("oranges" :: ("pears" :: Nil))
                              //> list1  : List[String] = List(apples, oranges, pears)

val a :: r = list1            //> a  : String = apples
                              //| r  : List[String] = List(oranges, pears)

val a1 :: b1 :: r1 = list1    //> a1  : String = apples
                              //| b1  : String = oranges
                              //| r1  : List[String] = List(pears)
```

- Construction (building) look at List of Strings (Note String type inferred)
- The operator :: is called cons, from LISP, and is used to construct lists
- Pattern matching is used to take lists apart

Scala Lists: Detour



```
val list1 = "apples" :: ("oranges" :: ("pears" :: Nil))
                                  //> list1  : List[String] = List(apples, oranges, pears)

val a :: r = list1                //> a   : String = apples
                                  //| r   : List[String] = List(oranges, pears)

val a1 :: b1 :: r1 = list1        //> a1  : String = apples
                                  //| b1  : String = oranges
                                  //| r1  : List[String] = List(pears)

val x = List("apples", 1)         //> x   : List[Any] = List(apples, 1)
```

- Construction (building) look at List of Strings (Note String type inferred)
- List x contains a String and an Int. “Any” type is inferred

Detour: Type Inference

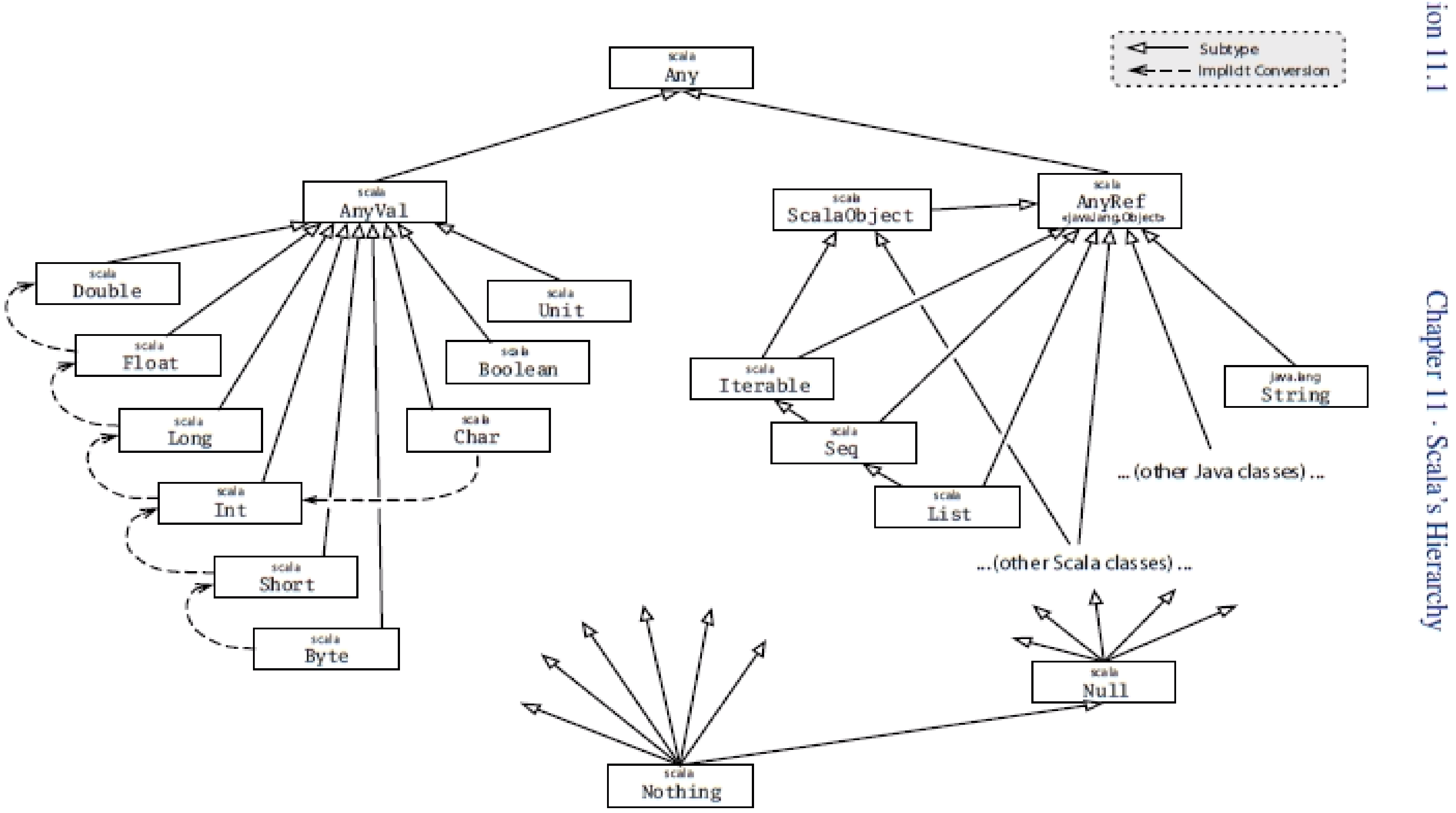
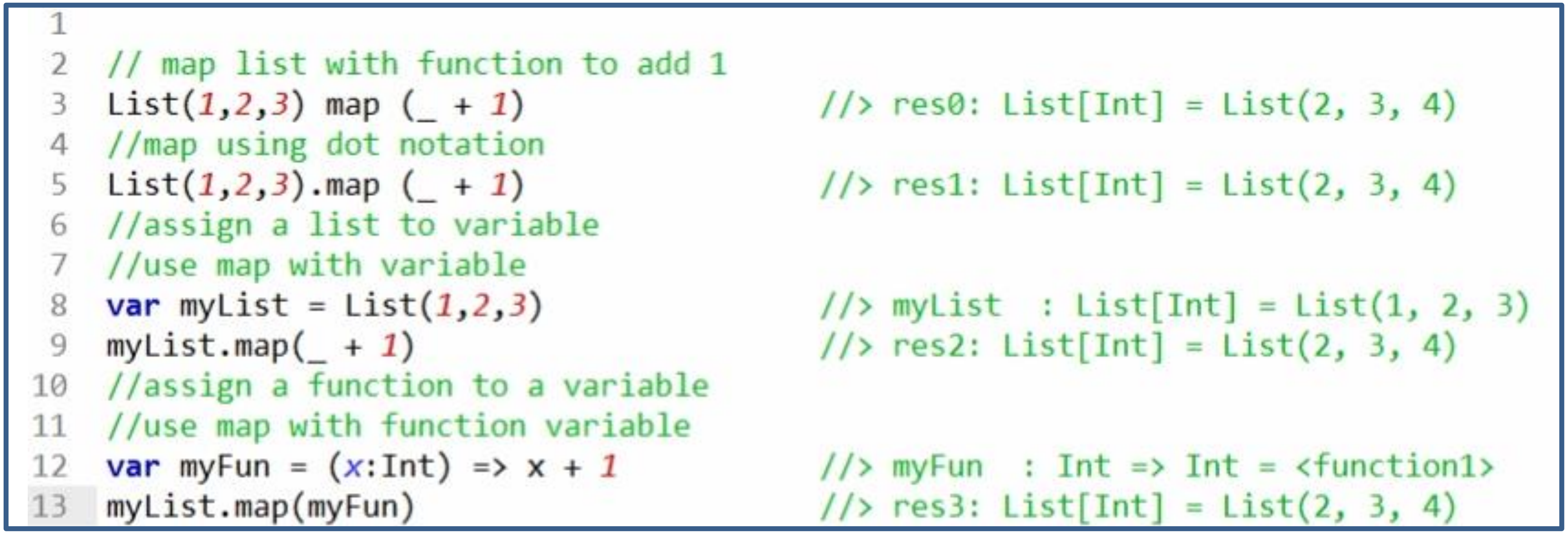


Figure 11.1 · Class hierarchy of Scala.

```
Val z = List(1,true)          //> z:  : List[AnyVal] = List(1, true)
```

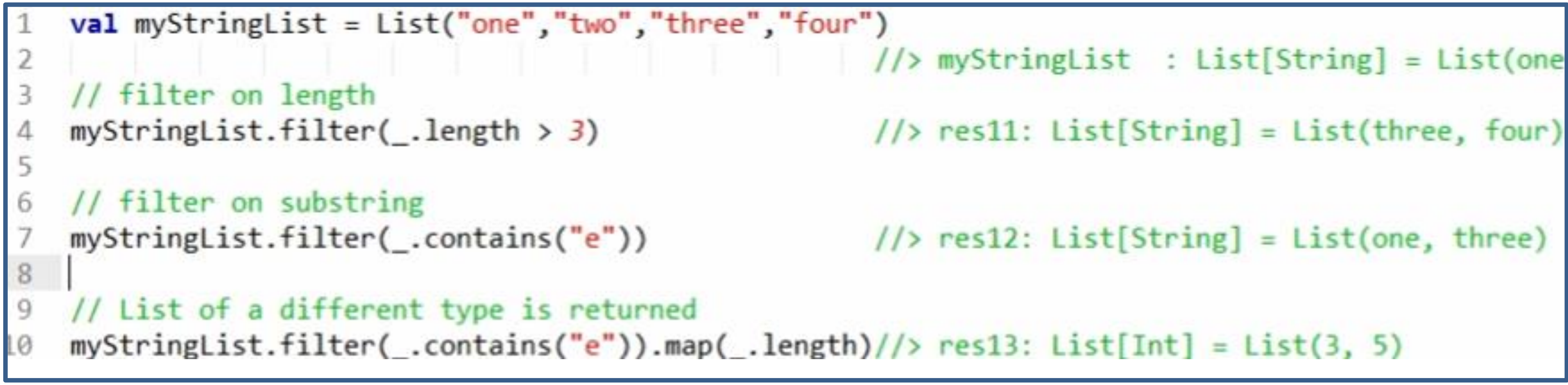
List Operation - map



```

// map list with function to add 1
List(1,2,3) map (_ + 1)                //> res0: List[Int] = List(2, 3, 4)
//map using dot notation
List(1,2,3).map (_ + 1)                //> res1: List[Int] = List(2, 3, 4)
//assign a list to variable
//use map with variable
var myList = List(1,2,3)               //> myList  : List[Int] = List(1, 2, 3)
myList.map(_ + 1)                      //> res2: List[Int] = List(2, 3, 4)
//assign a function to a variable
//use map with function variable
var myFun = (x:Int) => x + 1           //> myFun  : Int => Int = <function1>
myList.map(myFun)                      //> res3: List[Int] = List(2, 3, 4)
```

- map – xs map f
  - xs is a list of some type List[T]
  - f is a function of type T => U
  - map applies f to each element of xs returning the new list of type U.

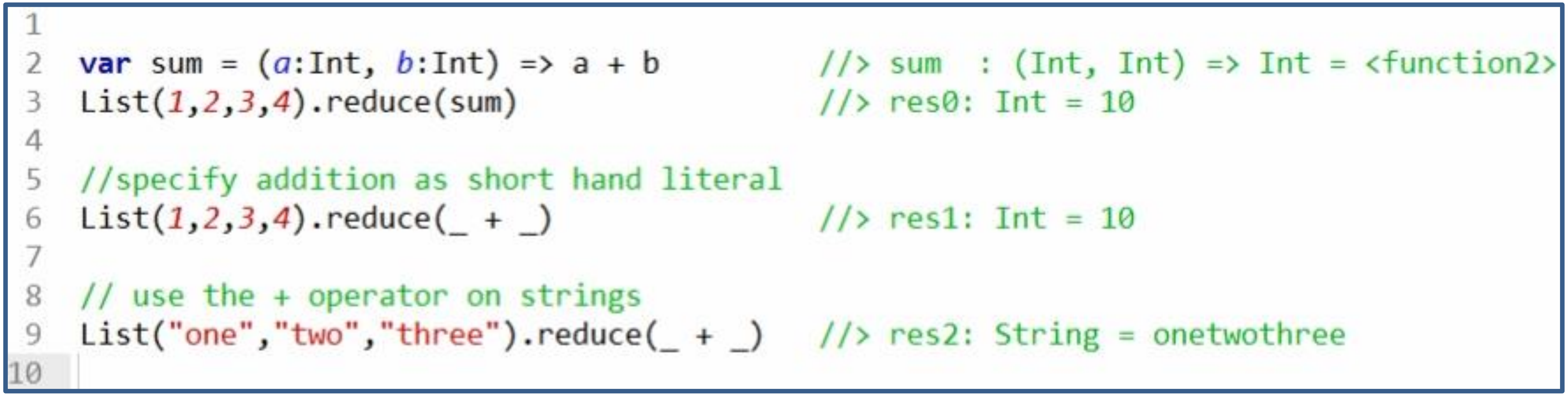
List Operation - filter



```
val myStringList = List("one","two","three","four")
                                                  //> myStringList  : List[String] = List(one
// filter on length
myStringList.filter(_.length > 3)                 //> res11: List[String] = List(three, four)

// filter on substring
myStringList.filter(_.contains("e"))              //> res12: List[String] = List(one, three)

// List of a different type is returned
myStringList.filter(_.contains("e")).map(_.length)//> res13: List[Int] = List(3, 5)
```

- xs filter p
  - xs is a list of type T
  - p is a function of type T => Boolean
  - filter applies p to xs returning a list of type T where for each x in xs, p(x) is true

List Operation -reduce

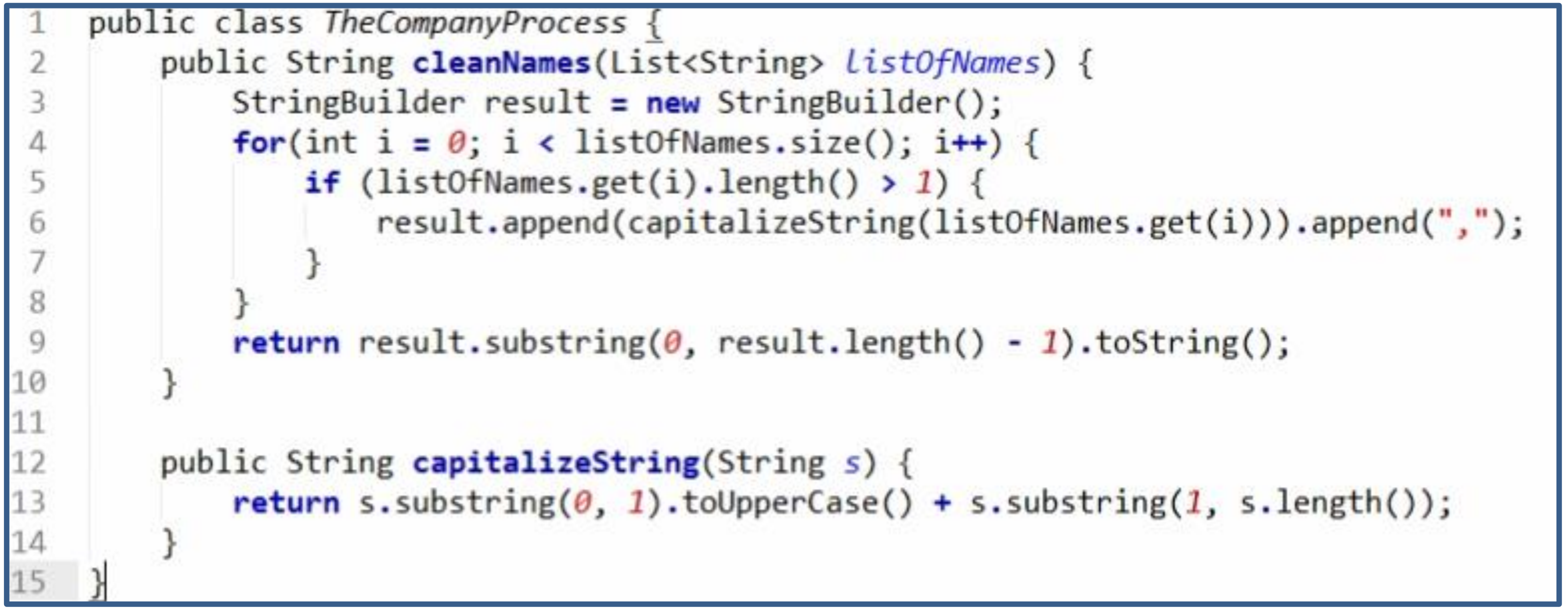


```
var sum = (a:Int, b:Int) => a + b         //> sum  : (Int, Int) => Int = <function2>
List(1,2,3,4).reduce(sum)                 //> res0: Int = 10

//specify addition as short hand literal
List(1,2,3,4).reduce(_ + _)               //> res1: Int = 10

// use the + operator on strings
List("one","two","three").reduce(_ + _)   //> res2: String = onetwothree
```

- xs reduce f
  - – xs is a list of type T where xs.length >1
  - – f is a binary operator of type (T , T) => T
  - – Reduce reduces a list of type T to a single value of type T through successive applications of f.

Clean Names* - Java

Problem: return the names in a comma-delimited string that contains no single-letter names, with each name capitalized.

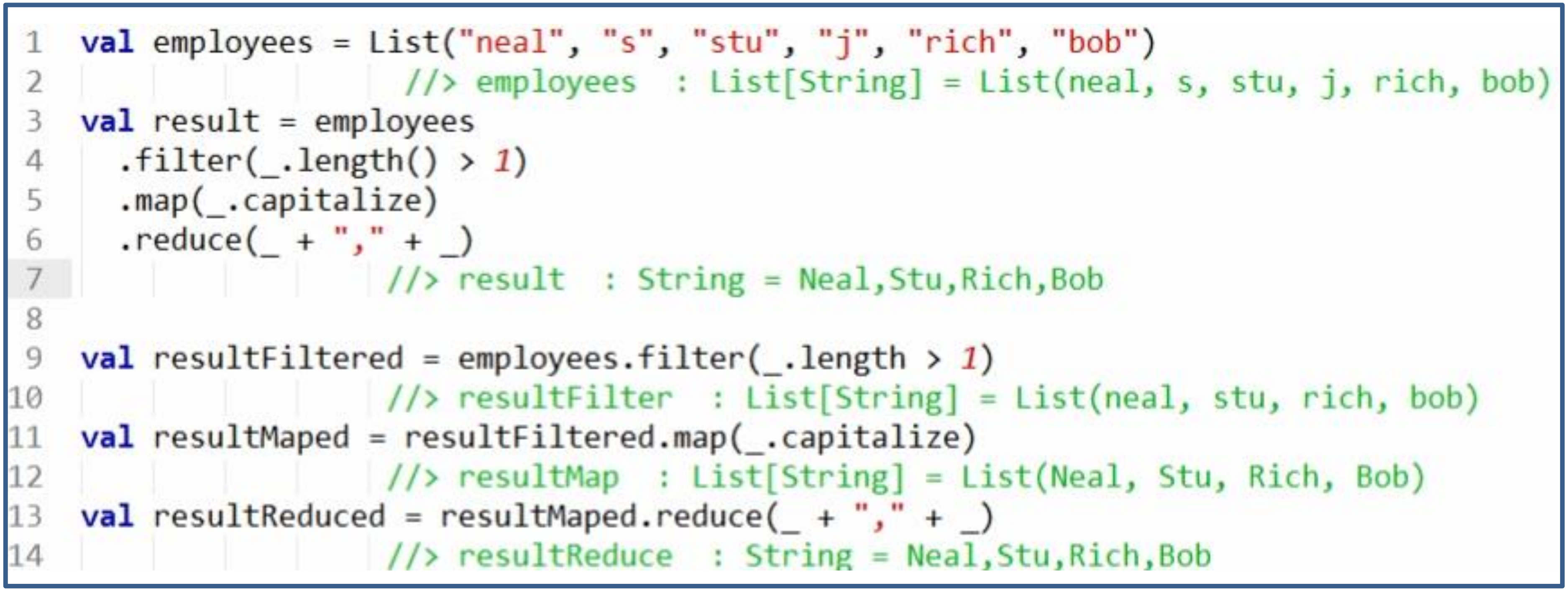


```java
public class TheCompanyProcess {
    public String cleanNames(List<String> listOfNames) {
        StringBuilder result = new StringBuilder();
        for(int i = 0; i < listOfNames.size(); i++) {
            if (listOfNames.get(i).length() > 1) {
                result.append(capitalizeString(listOfNames.get(i))).append(",");
            }
        }
        return result.substring(0, result.length() - 1).toString();
    }

    public String capitalizeString(String s) {
        return s.substring(0, 1).toUpperCase() + s.substring(1, s.length());
    }
}
```

Input: List(“neal” , “s” , “stu” , “j” , “rich” , “bob”)
Result: Neal,Stu,Rich,Bob

* http://www.ibm.com/developerworks/library/j-jn10/index.html

Clean Names Scala



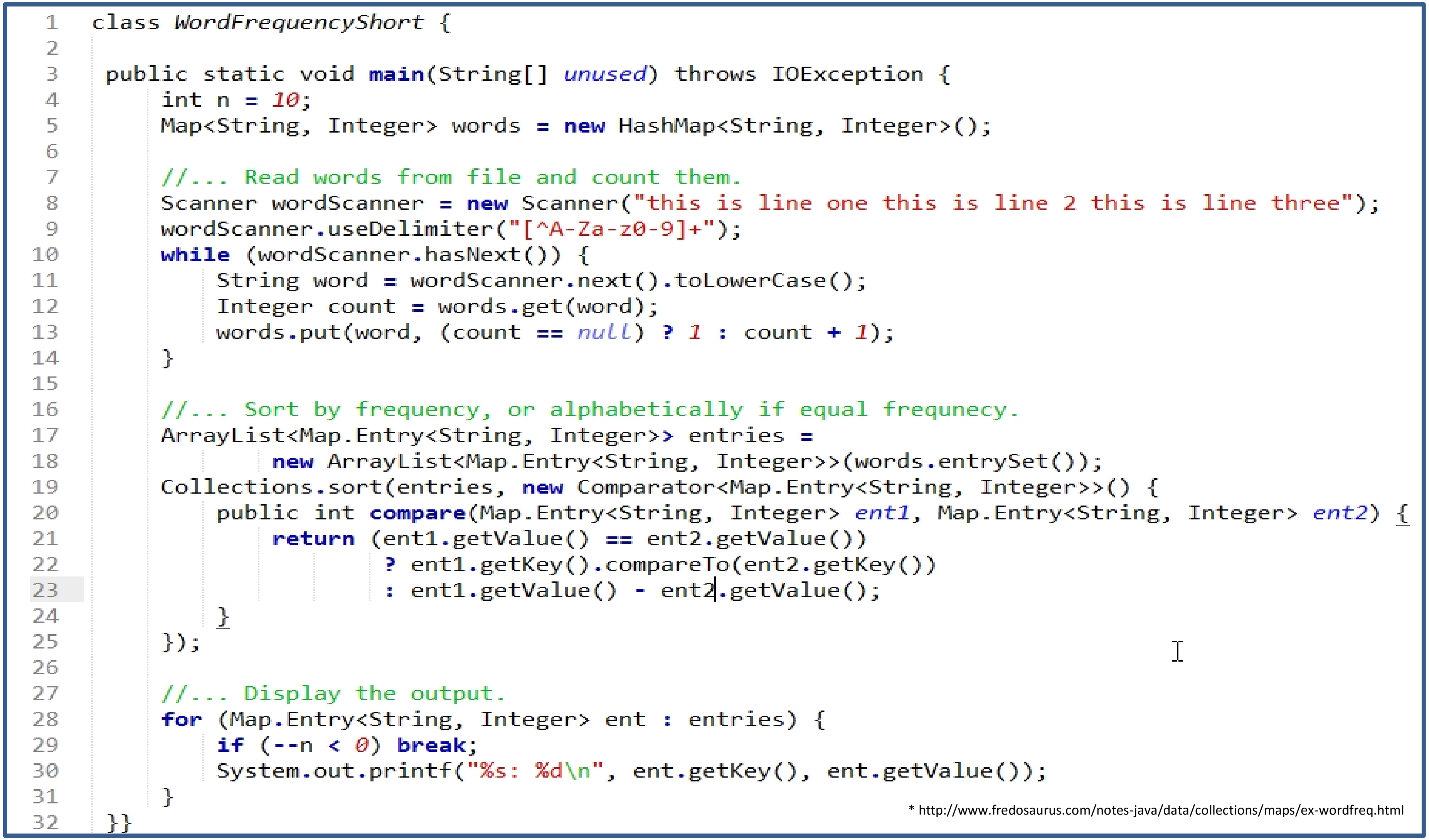
```
val employees = List("neal", "s", "stu", "j", "rich", "bob")
                  //> employees  : List[String] = List(neal, s, stu, j, rich, bob)
val result = employees
  .filter(_.length() > 1)
  .map(_.capitalize)
  .reduce(_ + "," + _)
                 //> result  : String = Neal,Stu,Rich,Bob

val resultFiltered = employees.filter(_.length > 1)
                 //> resultFilter  : List[String] = List(neal, stu, rich, bob)
val resultMaped = resultFiltered.map(_.capitalize)
                 //> resultMap  : List[String] = List(Neal, Stu, Rich, Bob)
val resultReduced = resultMaped.reduce(_ + "," + _)
                 //> resultReduce  : String = Neal,Stu,Rich,Bob
```

Input: List("neal" , "s" , "stu" , "j" , "rich" , "bob")
Result: Neal,Stu,Rich,Bob

Consider a problem

- Determine the word frequency in a list of sentences.
- Here are some random thoughts
  - Count the word frequencies in each line and sum them
  - Use a hashMap: word, count
  - Lines are unimportant, make a single sentence
  - Iterate through each word, increase its count in the hashMap
  - Iterate through the hashMap printing the results

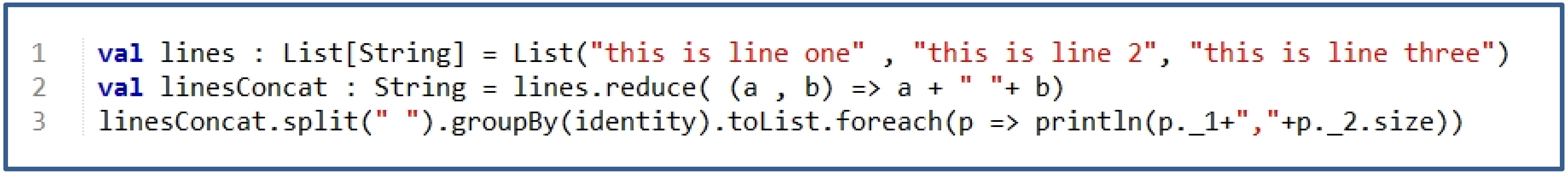
Word Frequency problem – Java*



```java
class WordFrequencyShort {

 public static void main(String[] unused) throws IOException {
     int n = 10;
     Map<String, Integer> words = new HashMap<String, Integer>();

     //... Read words from file and count them.
     Scanner wordScanner = new Scanner("this is line one this is line 2 this is line three");
     wordScanner.useDelimiter("[^A-Za-z0-9]+");
     while (wordScanner.hasNext()) {
         String word = wordScanner.next().toLowerCase();
         Integer count = words.get(word);
         words.put(word, (count == null) ? 1 : count + 1);
     }

     //... Sort by frequency, or alphabetically if equal frequnecy.
     ArrayList<Map.Entry<String, Integer>> entries =
             new ArrayList<Map.Entry<String, Integer>>(words.entrySet());
     Collections.sort(entries, new Comparator<Map.Entry<String, Integer>>() {
         public int compare(Map.Entry<String, Integer> ent1, Map.Entry<String, Integer> ent2) {
             return (ent1.getValue() == ent2.getValue())
                     ? ent1.getKey().compareTo(ent2.getKey())
                     : ent1.getValue() - ent2.getValue();
         }
     });

     //... Display the output.
     for (Map.Entry<String, Integer> ent : entries) {
         if (--n < 0) break;
         System.out.printf("%s: %d\n", ent.getKey(), ent.getValue());
     }
 }}
```

* http://www.fredosaurus.com/notes-java/data/collections/maps/ex-wordfreq.html

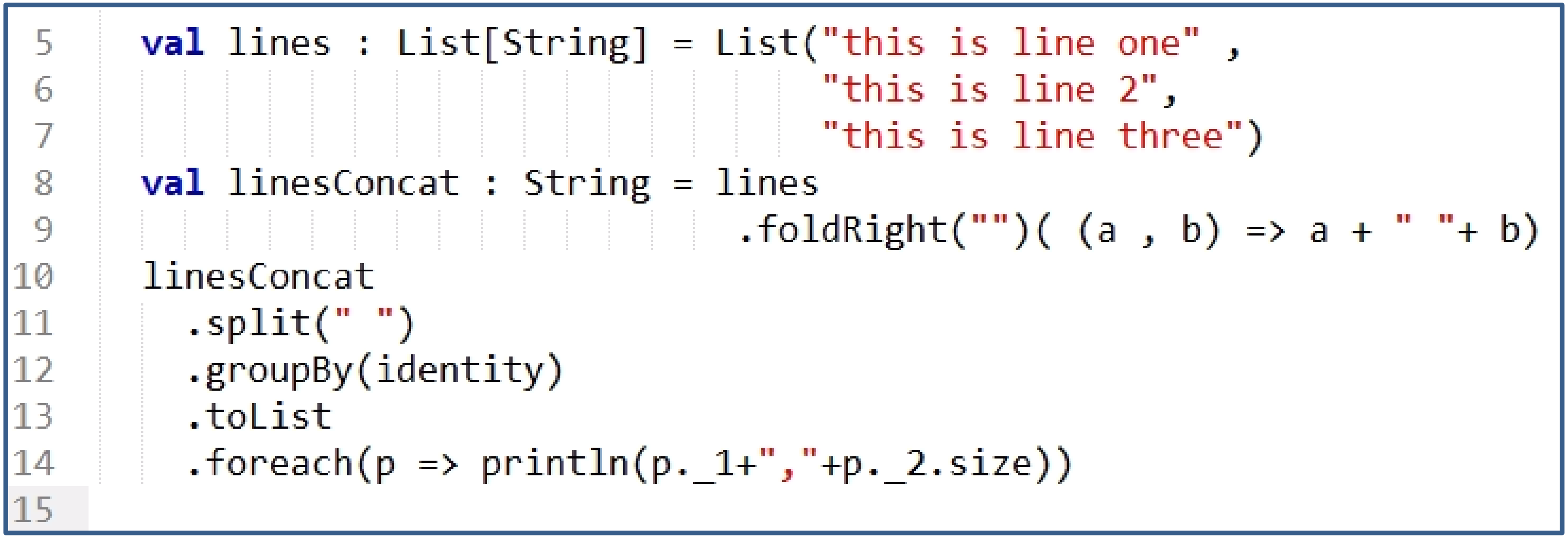
Word Frequency Problem - Scala



```
1  val lines : List[String] = List("this is line one" , "this is line 2", "this is line three")
2  val linesConcat : String = lines.reduce( (a , b) => a + " "+ b)
3  linesConcat.split(" ").groupBy(identity).toList.foreach(p => println(p._1+","+p._2.size))
```

- Line 1 defines a List of Strings where each string is a line of the file.
- Line 2 concatenates each element in the list
- Line 3… well… let’s look at it a bit closer after we run the code.

Word Frequency Problem - Scala

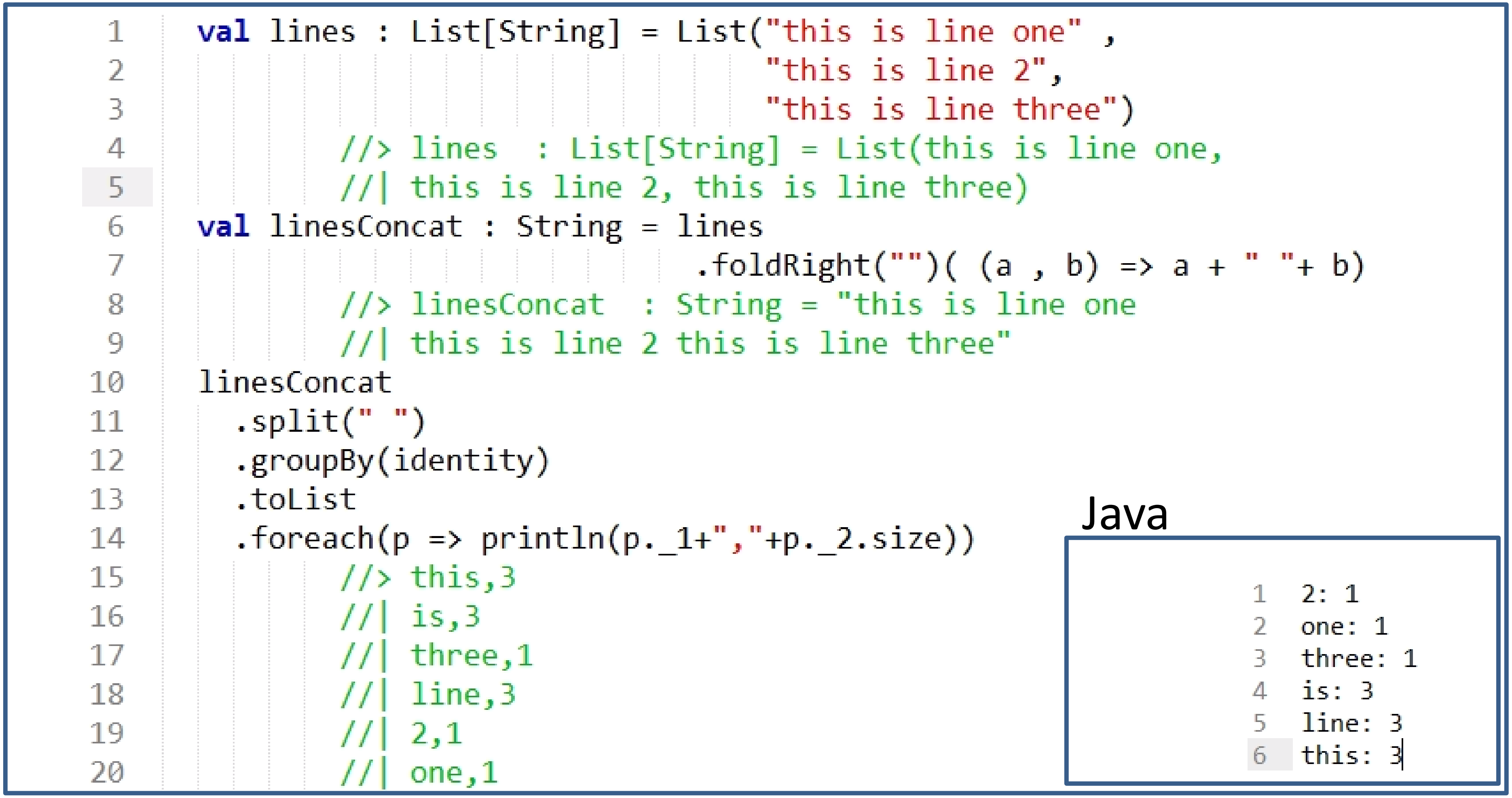


```
 5   val lines : List[String] = List("this is line one" ,
 6                                   "this is line 2",
 7                                   "this is line three")
 8   val linesConcat : String = lines
 9                               .foldRight("")( (a , b) => a + " "+ b)
10   linesConcat
11     .split(" ")
12     .groupBy(identity)
13     .toList
14     .foreach(p => println(p._1+","+p._2.size))
15
```

- Line 1 defines a List of Strings where each string is a line of the file.
- Line 2 concatenates each element in the list
- Line 3 applies several built in collection methods.

Run the Solution in REPL

Scala



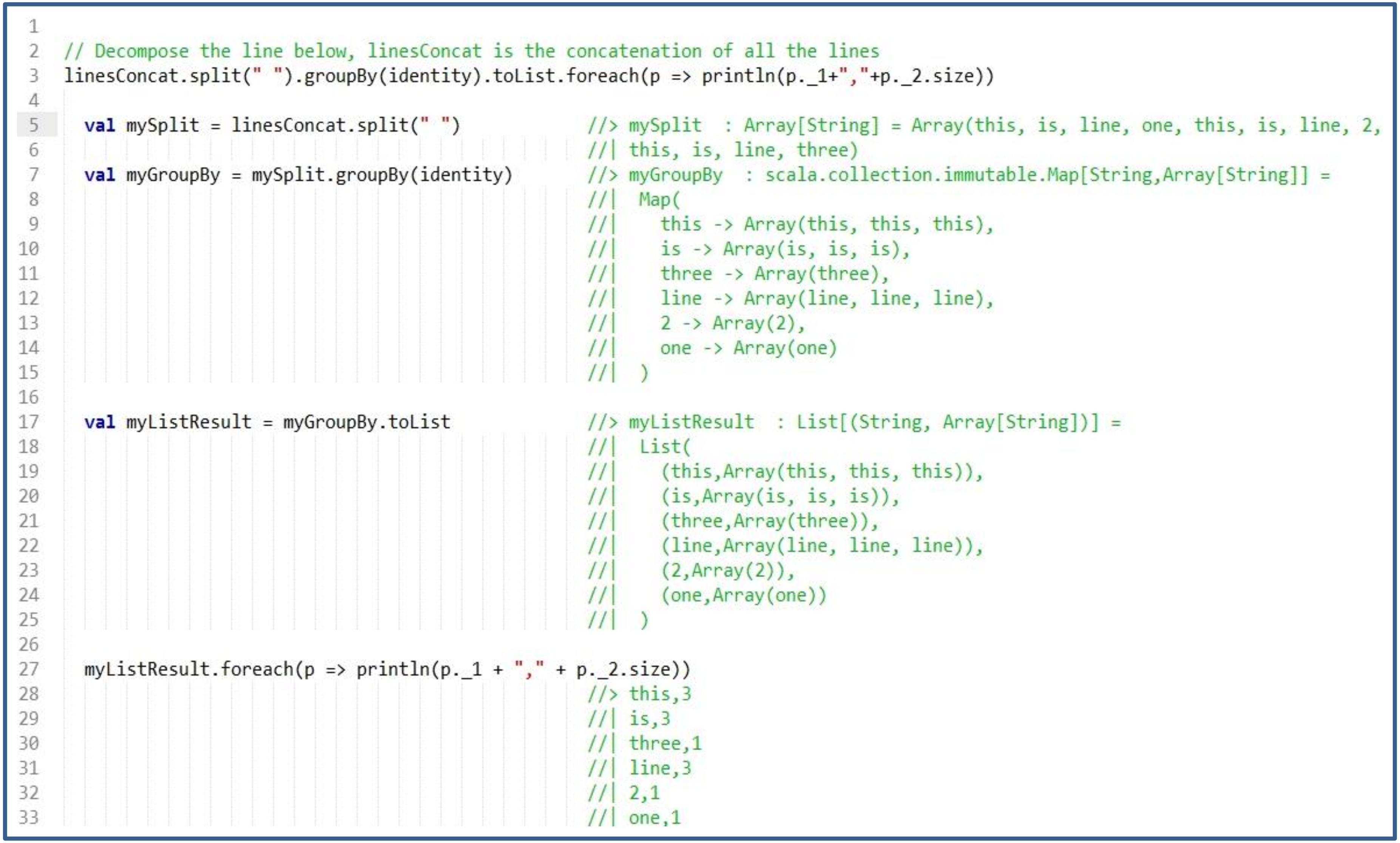
```scala
val lines : List[String] = List("this is line one" ,
                                "this is line 2",
                                "this is line three")
        //> lines  : List[String] = List(this is line one,
        //| this is line 2, this is line three)
val linesConcat : String = lines
                            .foldRight("")( (a , b) => a + " "+ b)
        //> linesConcat  : String = "this is line one
        //| this is line 2 this is line three"
linesConcat
  .split(" ")
  .groupBy(identity)
  .toList
  .foreach(p => println(p._1+","+p._2.size))
        //> this,3
        //| is,3
        //| three,1
        //| line,3
        //| 2,1
        //| one,1
```

Java

```
2: 1
one: 1
three: 1
is: 3
line: 3
this: 3
```

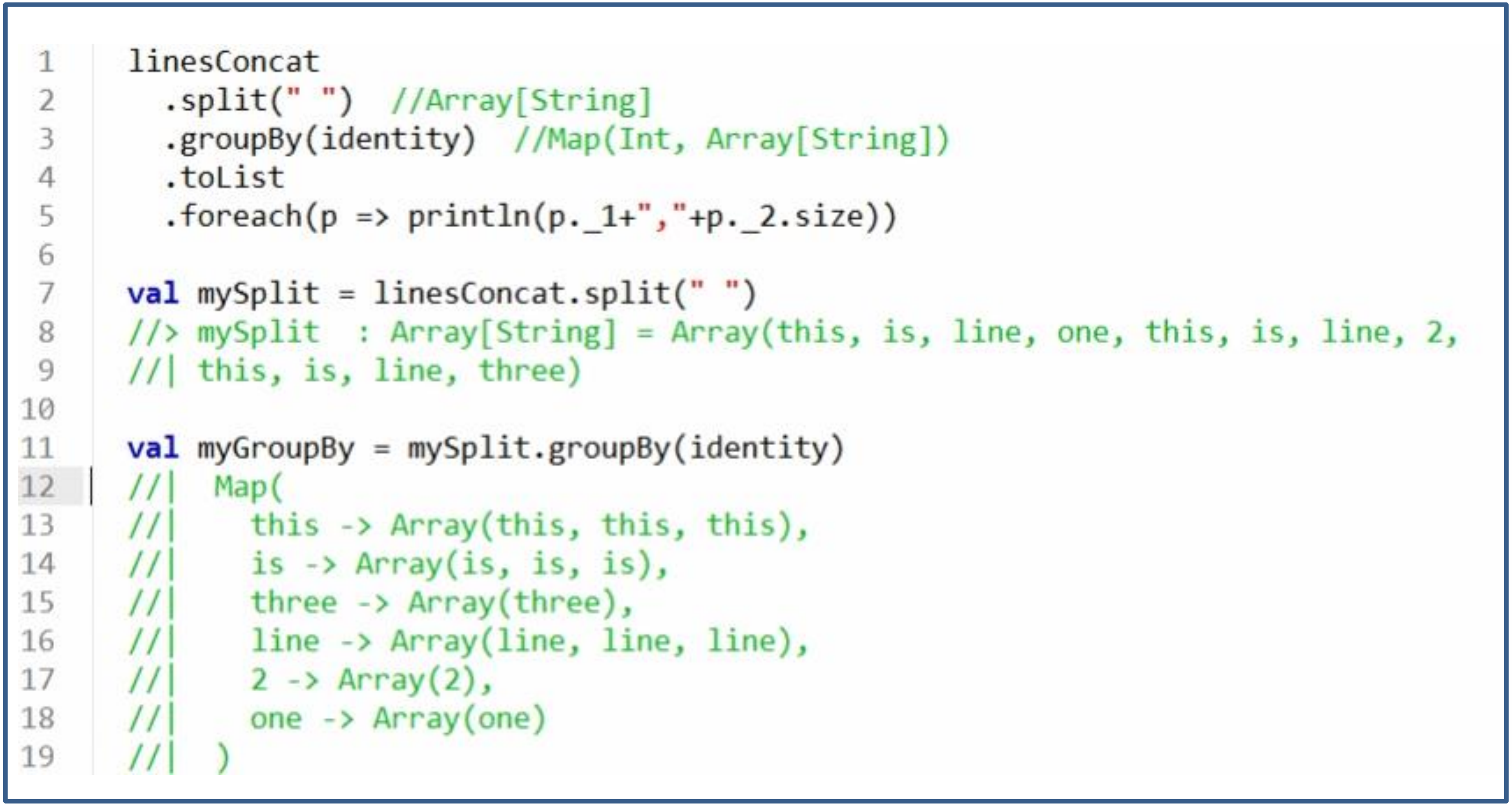
Let's examine the last line of the Scala program

Expand the last line of Scala



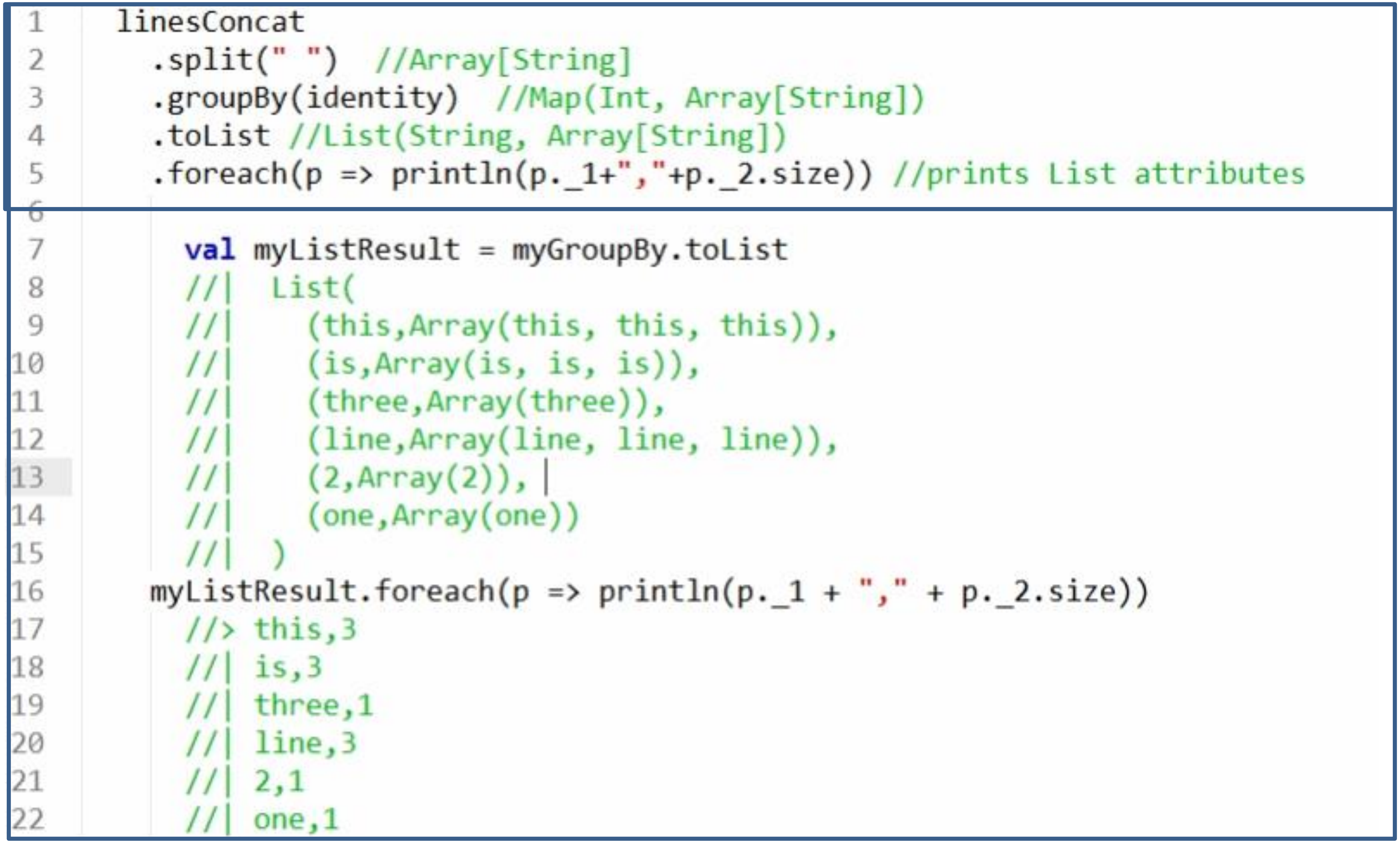
```
// Decompose the line below, linesConcat is the concatenation of all the lines
linesConcat.split(" ").groupBy(identity).toList.foreach(p => println(p._1+","+p._2.size))

  val mySplit = linesConcat.split(" ")            //> mySplit  : Array[String] = Array(this, is, line, one, this, is, line, 2,
                                                  //| this, is, line, three)
  val myGroupBy = mySplit.groupBy(identity)       //> myGroupBy  : scala.collection.immutable.Map[String,Array[String]] =
                                                  //|  Map(
                                                  //|    this -> Array(this, this, this),
                                                  //|    is -> Array(is, is, is),
                                                  //|    three -> Array(three),
                                                  //|    line -> Array(line, line, line),
                                                  //|    2 -> Array(2),
                                                  //|    one -> Array(one)
                                                  //|  )

  val myListResult = myGroupBy.toList             //> myListResult  : List[(String, Array[String])] =
                                                  //|  List(
                                                  //|    (this,Array(this, this, this)),
                                                  //|    (is,Array(is, is, is)),
                                                  //|    (three,Array(three)),
                                                  //|    (line,Array(line, line, line)),
                                                  //|    (2,Array(2)),
                                                  //|    (one,Array(one))
                                                  //|  )

  myListResult.foreach(p => println(p._1 + "," + p._2.size))
                                                  //> this,3
                                                  //| is,3
                                                  //| three,1
                                                  //| line,3
                                                  //| 2,1
                                                  //| one,1
```

Expand the last line of Scala



```
linesConcat
  .split(" ")  //Array[String]
  .groupBy(identity)  //Map(Int, Array[String])
  .toList
  .foreach(p => println(p._1+","+p._2.size))

val mySplit = linesConcat.split(" ")
//> mySplit  : Array[String] = Array(this, is, line, one, this, is, line, 2,
//| this, is, line, three)

val myGroupBy = mySplit.groupBy(identity)
//|  Map(
//|    this -> Array(this, this, this),
//|    is -> Array(is, is, is),
//|    three -> Array(three),
//|    line -> Array(line, line, line),
//|    2 -> Array(2),
//|    one -> Array(one)
//|  )
```

Expand the last line of Scala

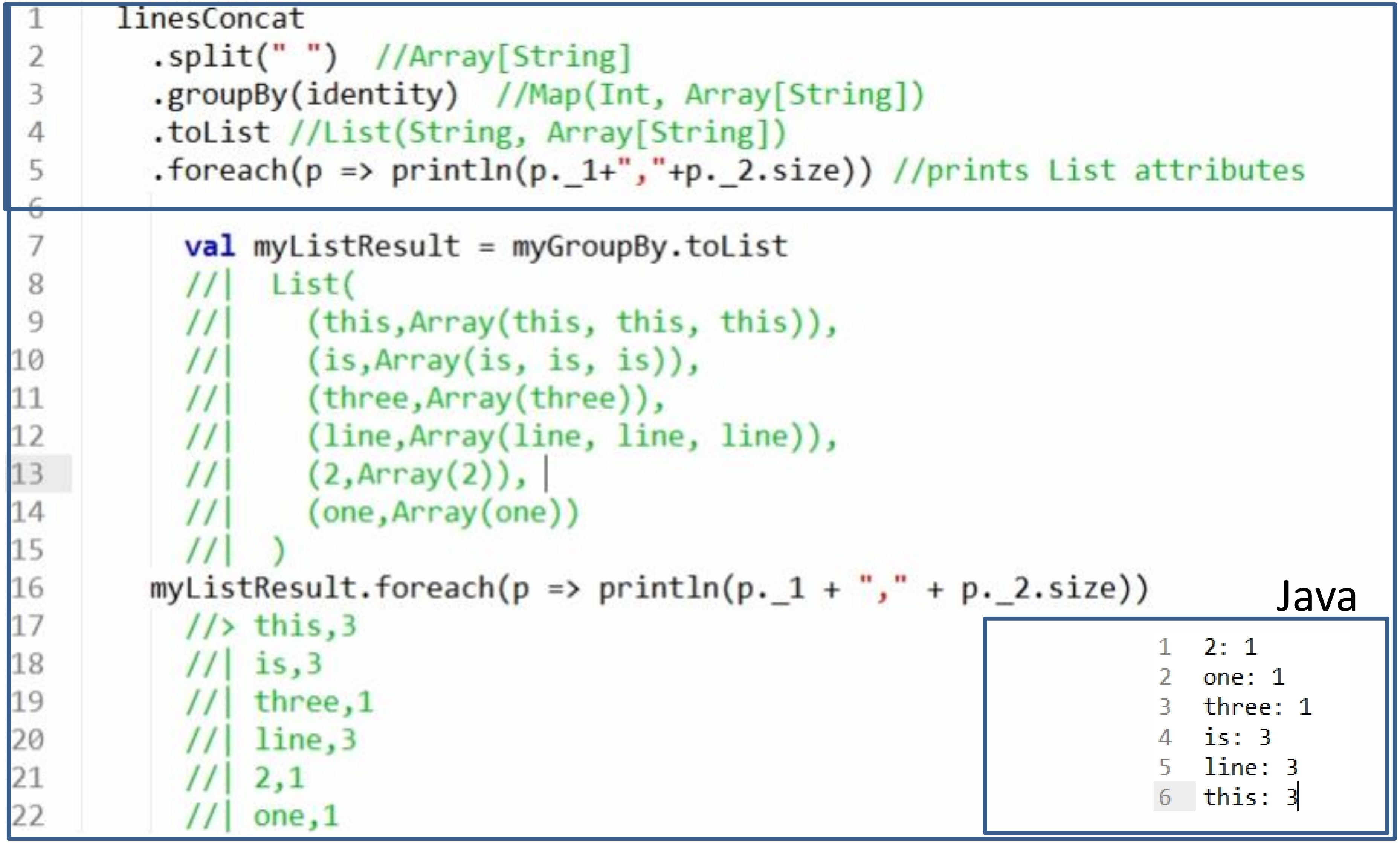


```
linesConcat
  .split(" ")  //Array[String]
  .groupBy(identity)  //Map(Int, Array[String])
  .toList //List(String, Array[String])
  .foreach(p => println(p._1+","+p._2.size)) //prints List attributes

    val myListResult = myGroupBy.toList
    //|  List(
    //|    (this,Array(this, this, this)),
    //|    (is,Array(is, is, is)),
    //|    (three,Array(three)),
    //|    (line,Array(line, line, line)),
    //|    (2,Array(2)),
    //|    (one,Array(one))
    //|  )
  myListResult.foreach(p => println(p._1 + "," + p._2.size))
    //> this,3
    //| is,3
    //| three,1
    //| line,3
    //| 2,1
    //| one,1
```

Functional Style: Transform to Solution

- Started with a list of sentences
- Transformed to a string of words with .foldright
- Transformed to an String Array of words with .split
- Transformed to a Map(String, Array[String]) with .groupBy(identity)
- Transformed to a list of 2-tuples (String, Array[String]) with .toList
- “cherry picked” our results from each 2-tuple with .foreach(p => println(p._1 +space+ p._2.size)

Output not in correct order



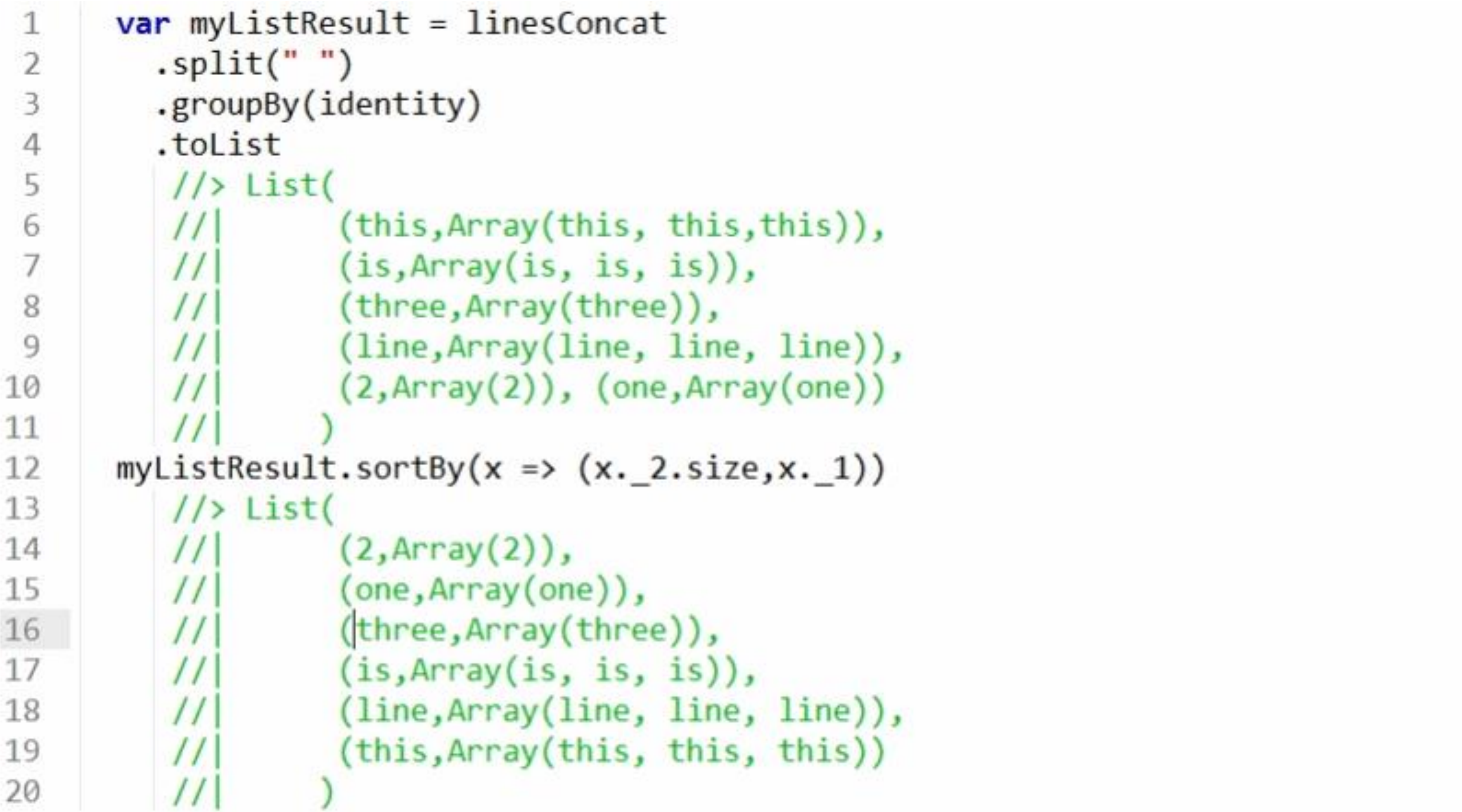
```
  linesConcat
    .split(" ")  //Array[String]
    .groupBy(identity)  //Map(Int, Array[String])
    .toList //List(String, Array[String])
    .foreach(p => println(p._1+","+p._2.size)) //prints List attributes

      val myListResult = myGroupBy.toList
      //|  List(
      //|    (this,Array(this, this, this)),
      //|    (is,Array(is, is, is)),
      //|    (three,Array(three)),
      //|    (line,Array(line, line, line)),
      //|    (2,Array(2)),
      //|    (one,Array(one))
      //|  )
    myListResult.foreach(p => println(p._1 + "," + p._2.size))
      //> this,3
      //| is,3
      //| three,1
      //| line,3
      //| 2,1
      //| one,1
```

Java

```
2: 1
one: 1
three: 1
is: 3
line: 3
this: 3
```

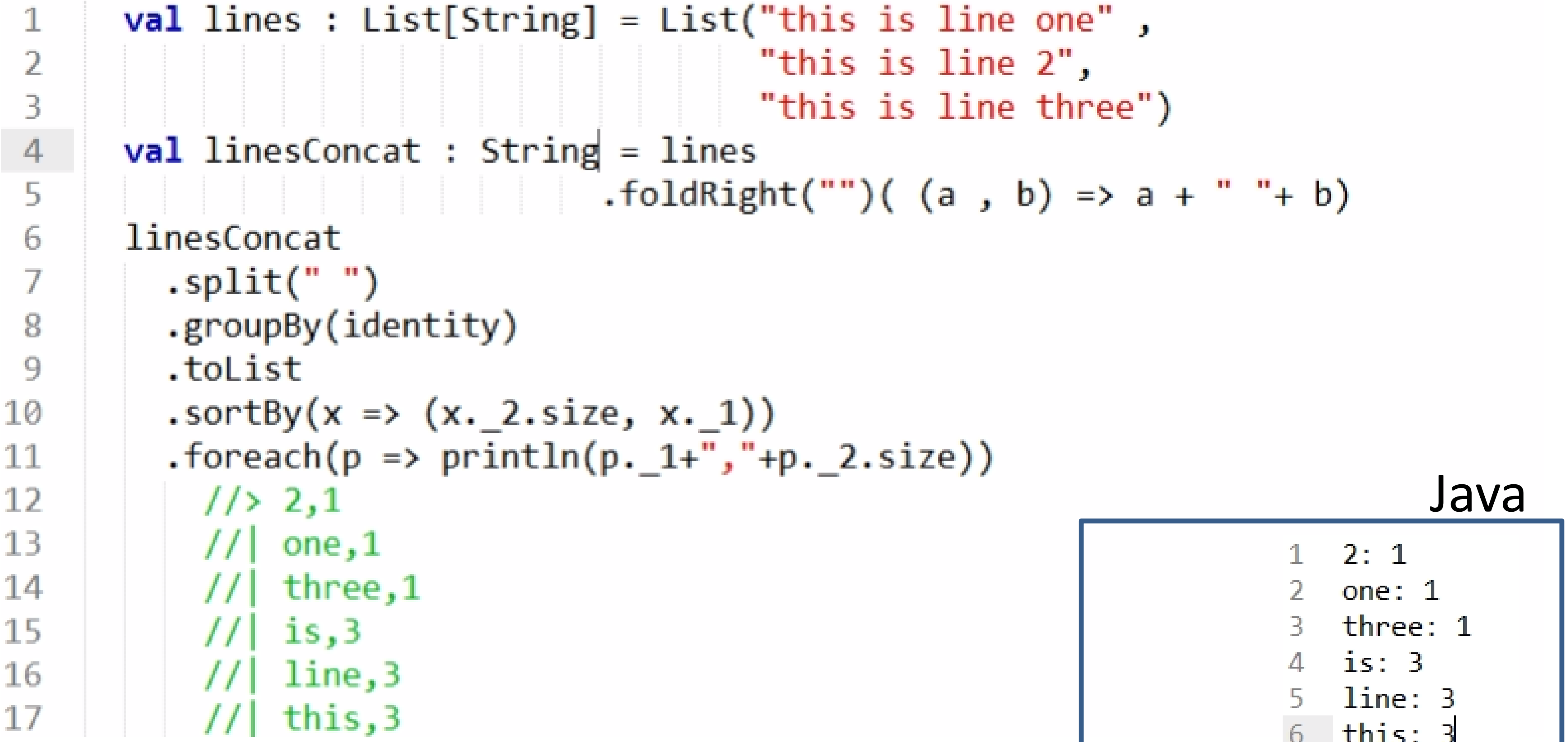
Insert a sort after the .toList

(isolated listing)



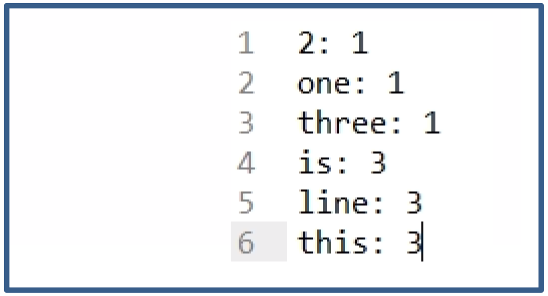
```
var myListResult = linesConcat
  .split(" ")
  .groupBy(identity)
  .toList
   //> List(
   //|      (this,Array(this, this,this)),
   //|      (is,Array(is, is, is)),
   //|      (three,Array(three)),
   //|      (line,Array(line, line, line)),
   //|      (2,Array(2)), (one,Array(one))
   //|     )
myListResult.sortBy(x => (x._2.size,x._1))
   //> List(
   //|      (2,Array(2)),
   //|      (one,Array(one)),
   //|      (three,Array(three)),
   //|      (is,Array(is, is, is)),
   //|      (line,Array(line, line, line)),
   //|      (this,Array(this, this, this))
   //|     )
```

Insert a sort after the .toList
Full Listing



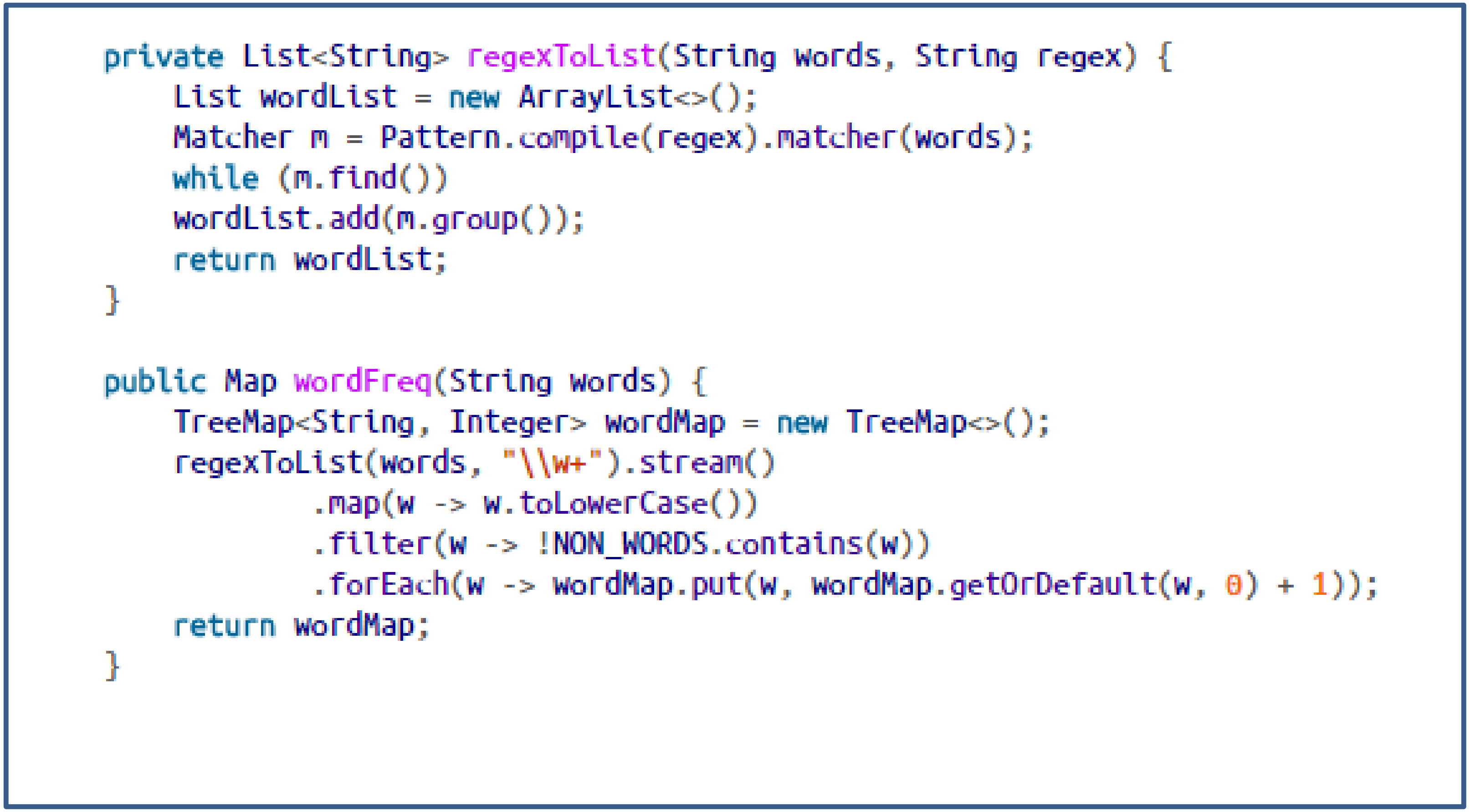
```scala
val lines : List[String] = List("this is line one" ,
                                "this is line 2",
                                "this is line three")
val linesConcat : String = lines
                        .foldRight("")( (a , b) => a + " "+ b)
linesConcat
  .split(" ")
  .groupBy(identity)
  .toList
  .sortBy(x => (x._2.size, x._1))
  .foreach(p => println(p._1+","+p._2.size))
    //> 2,1
    //| one,1
    //| three,1
    //| is,3
    //| line,3
    //| this,3
```

Java



```
2: 1
one: 1
three: 1
is: 3
line: 3
this: 3
```

Word Count in Java 8



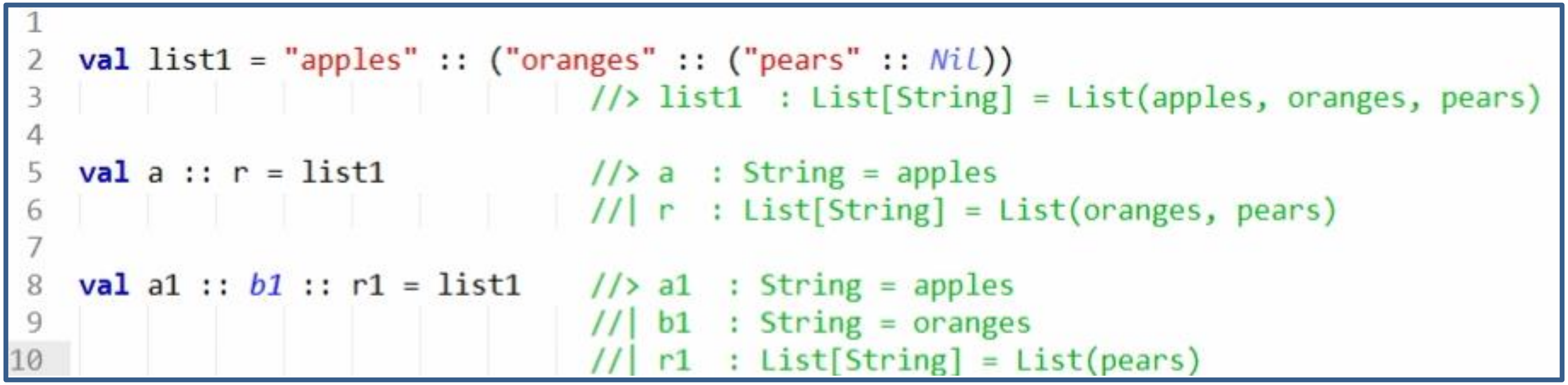
```
private List<String> regexToList(String words, String regex) {
    List wordList = new ArrayList<>();
    Matcher m = Pattern.compile(regex).matcher(words);
    while (m.find())
    wordList.add(m.group());
    return wordList;
}

public Map wordFreq(String words) {
    TreeMap<String, Integer> wordMap = new TreeMap<>();
    regexToList(words, "\\w+").stream()
            .map(w -> w.toLowerCase())
            .filter(w -> !NON_WORDS.contains(w))
            .forEach(w -> wordMap.put(w, wordMap.getOrDefault(w, 0) + 1));
    return wordMap;
}
```

Functional Style Programming

1. Favor functions that return immutable results over iterative methods that change state. (favor methods that do not have side effects)
2. Favor solutions that use the building blocks you have over creating new stuff. Complect – to join by weaving or twining together.
3. Favor thinking about what you must do over how you will do it.

Pattern Matching

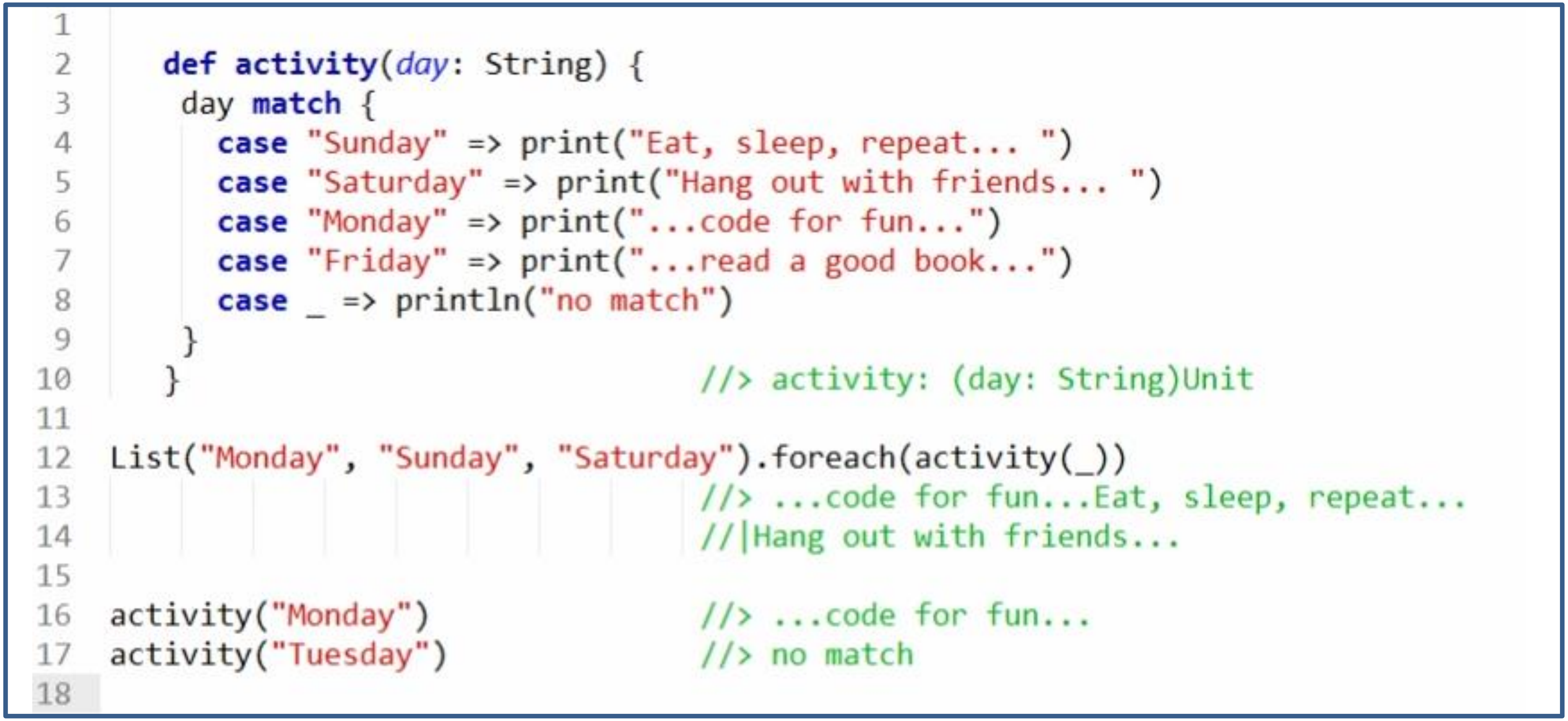
- Pattern matching is the second most important feature of Scala, after function values
- Previously used pattern matching to decompose Lists.



```
val list1 = "apples" :: ("oranges" :: ("pears" :: Nil))
                                     //> list1  : List[String] = List(apples, oranges, pears)

val a :: r = list1                   //> a   : String = apples
                                     //| r   : List[String] = List(oranges, pears)

val a1 :: b1 :: r1 = list1           //> a1  : String = apples
                                     //| b1  : String = oranges
                                     //| r1  : List[String] = List(pears)
```

- That’s Not All!!

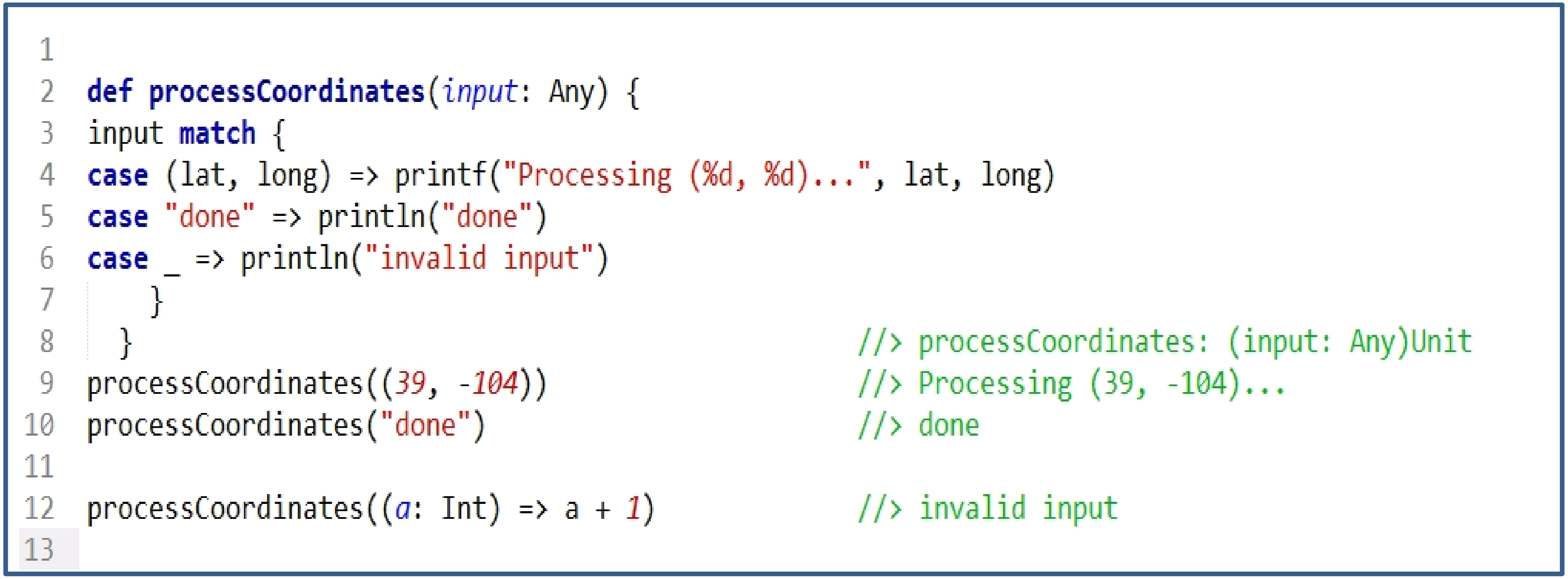
Pattern Matching



```
   def activity(day: String) {
    day match {
      case "Sunday" => print("Eat, sleep, repeat... ")
      case "Saturday" => print("Hang out with friends... ")
      case "Monday" => print("...code for fun...")
      case "Friday" => print("...read a good book...")
      case _ => println("no match")
    }
   }                              //> activity: (day: String)Unit

List("Monday", "Sunday", "Saturday").foreach(activity(_))
                                 //> ...code for fun...Eat, sleep, repeat...
                                 //|Hang out with friends...

activity("Monday")               //> ...code for fun...
activity("Tuesday")              //> no match
```

- Pattern matching is not a Java switch statement! It is much more powerful
- …not switch. No fall through, find match and leave block.
- Note the use of underscore (_) to indicate wild card matches anything

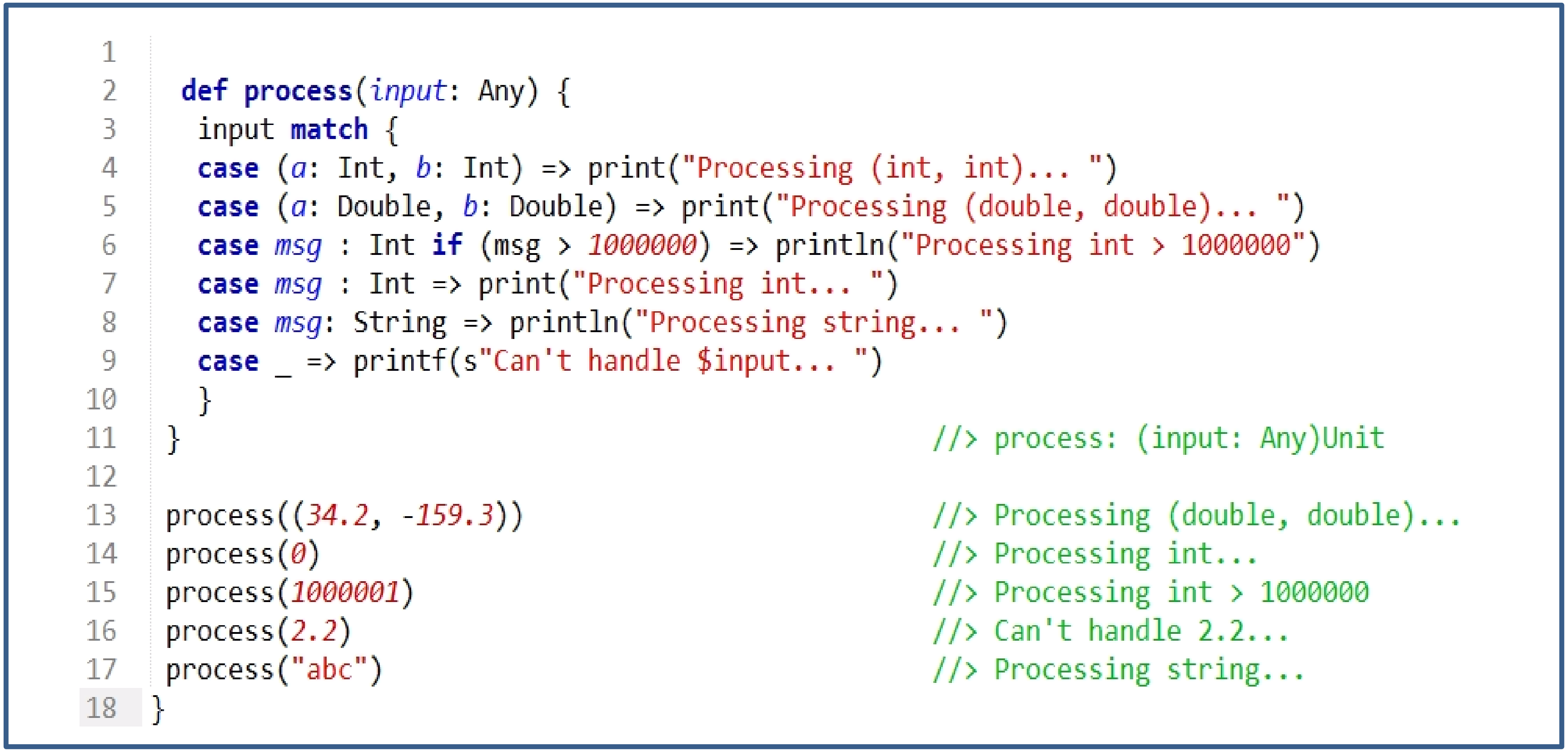
Pattern Matching



```
1
2  def processCoordinates(input: Any) {
3  input match {
4  case (lat, long) => printf("Processing (%d, %d)...", lat, long)
5  case "done" => println("done")
6  case _ => println("invalid input")
7      }
8    }                                       //> processCoordinates: (input: Any)Unit
9  processCoordinates((39, -104))            //> Processing (39, -104)...
10 processCoordinates("done")                //> done
11
12 processCoordinates((a: Int) => a + 1)     //> invalid input
13
```

- Input type is Any. Patterns can match instances of different types
- (lat, long) is of type tuple. Not only matches, but binds variables when matched
- Note: Type Any includes function type as seen in line 12. The function processCoordinates will accept a function literal as argument. Another example of functions as first class entities.

Pattern Matching



```scala

  def process(input: Any) {
   input match {
   case (a: Int, b: Int) => print("Processing (int, int)... ")
   case (a: Double, b: Double) => print("Processing (double, double)... ")
   case msg : Int if (msg > 1000000) => println("Processing int > 1000000")
   case msg : Int => print("Processing int... ")
   case msg: String => println("Processing string... ")
   case _ => printf(s"Can't handle $input... ")
   }
 }                                                //> process: (input: Any)Unit

 process((34.2, -159.3))                          //> Processing (double, double)...
 process(0)                                       //> Processing int...
 process(1000001)                                 //> Processing int > 1000000
 process(2.2)                                     //> Can't handle 2.2...
 process("abc")                                   //> Processing string...
}
```

- Matching by types, even types within types [(a: Int, b: Int) is a 2-tuple of integers]

Summary

- Scala is a blend of programming styles. I encourage you to use this blend in your design thinking.
- Complect… What does it mean?
- Scala feels like a dynamically typed language while exhibiting the advantages of a statically typed language. Why?
- Favor immutability. Why?

Thank You