CSC 735 – Data Analytics

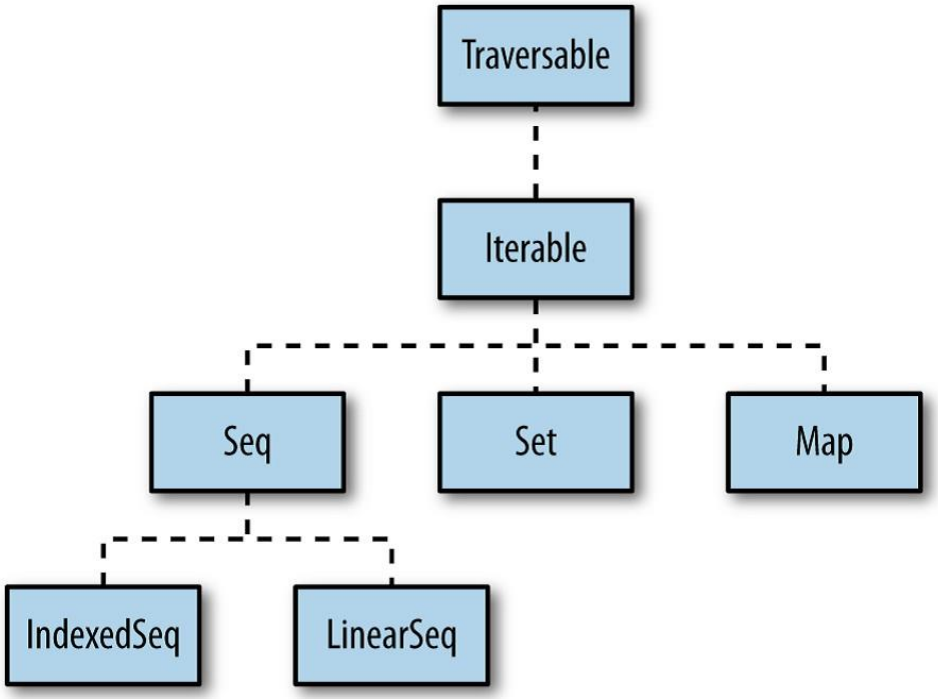
Introduction to Scala

Collections Overview

- A collection is a container data structure
- Scala has a rich collections library that includes collections of many different types
- Collections provide an easy-to-use interface that reduces the need to manually loop through all the elements
- They enable functional programming
- Collections provide a similar interface

Collections Overview (cont.)

- Scala collections are grouped into three main categories: sequences, sets, and maps



Mutable and Immutable Collections

- Scala supports mutable and immutable collections
- Scala prefers working with immutable collections
- When you do not specify a package for a collection, you are using an immutable version
- All collection classes are found in the package scala.collection
- Mutable objects are in scala.collection.mutable
  - scala.collection.mutable.Map
- Immutable objects
  - scala.collection.**im**mutable.**Map** or scala.collection.Map

Immutable Sequences

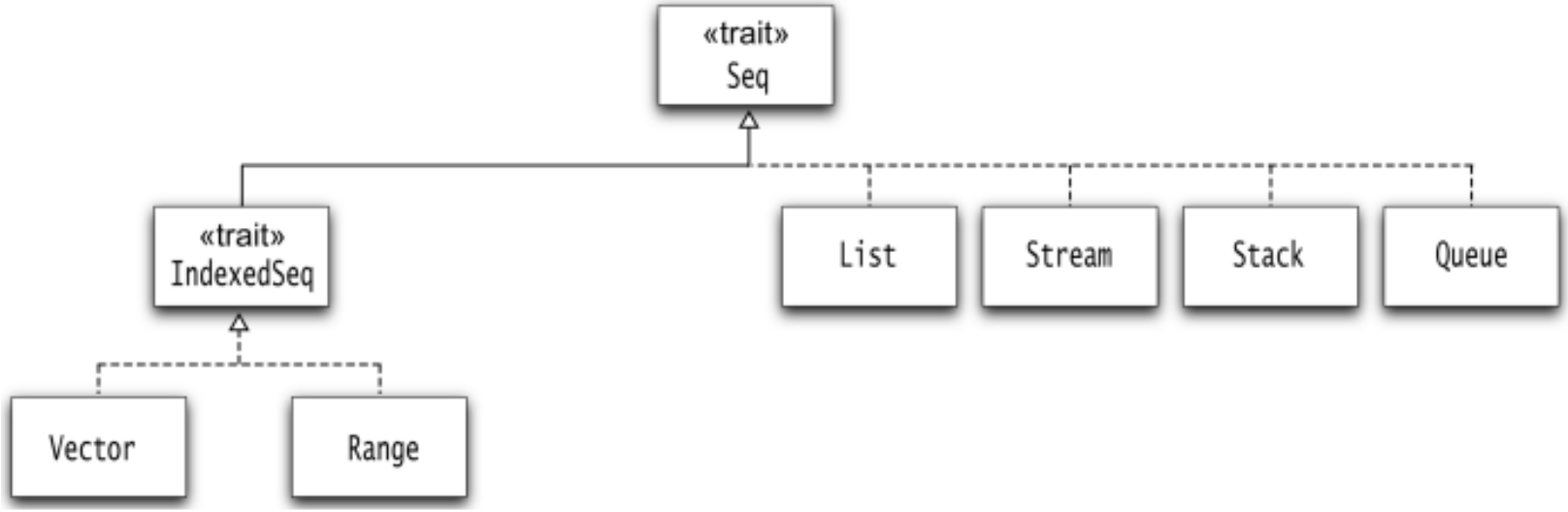


Figure 13–2 Immutable sequences

Mutable Sequences

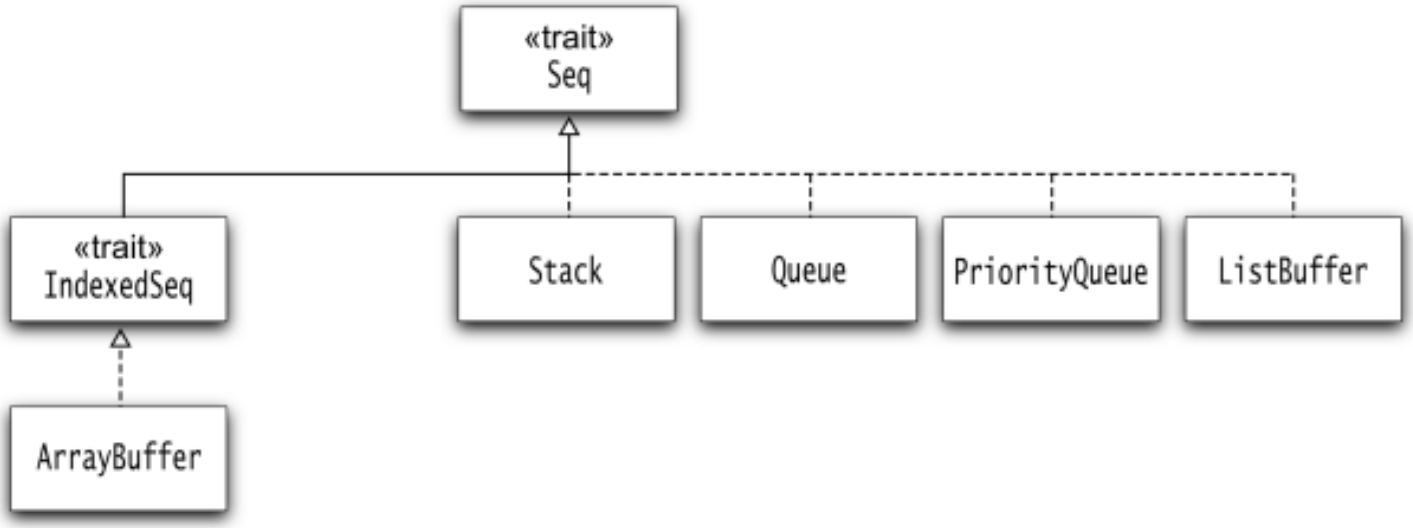
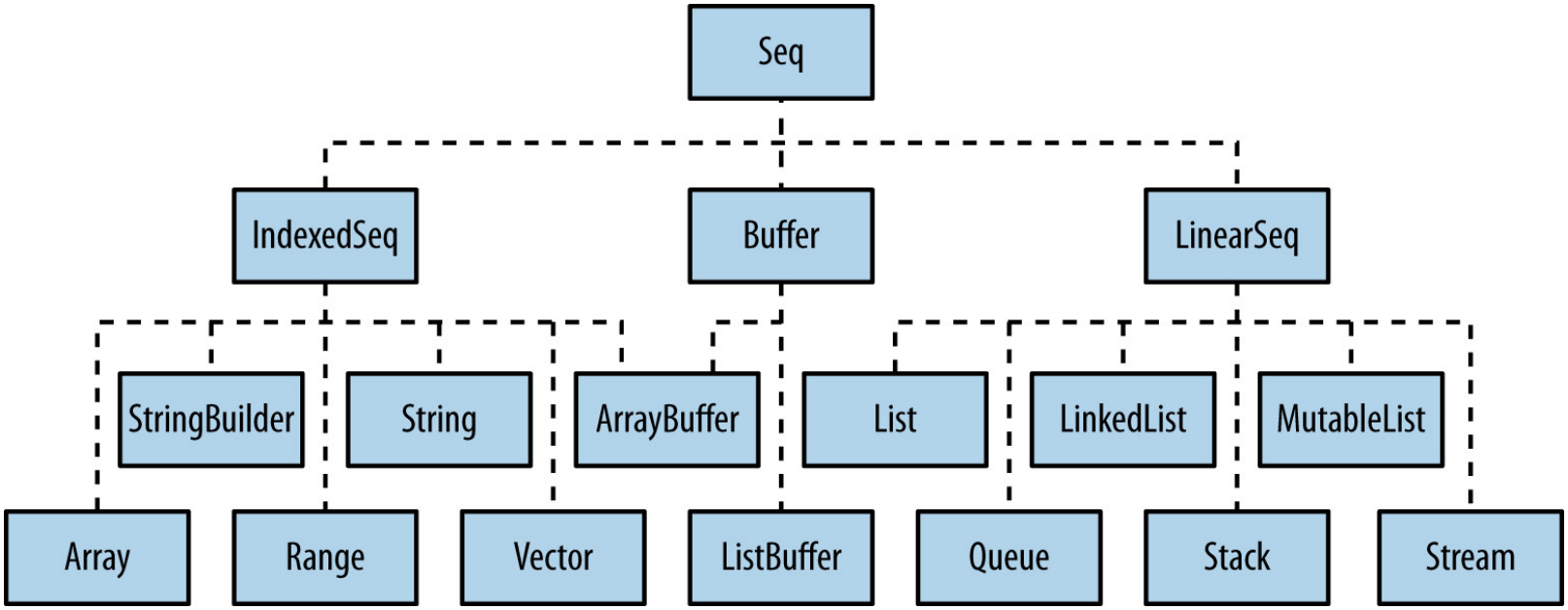


Figure 13–3 Mutable sequences

Sequences

- The Seq trait represents sequences
- Sequences branch into two main categories
  1. Indexed Sequences
  2. Linear Sequences



Indexed Sequences

- In an indexed sequence, any element can be accessed by an index which makes random element access efficient
- Examples: Array, Vector, Range

Indexed Sequences

- In an indexed sequence, any element can be accessed by an index which makes random element access efficient
- Examples: Array, Vector, Range
- By default, specifying that you want an IndexedSeq, Scala creates a Vector:

```
scala>  val x = IndexedSeq(1,2,3)
x: IndexedSeq[Int] = Vector(1, 2, 3)
```

Linear Sequences

- A LinearSeq is a collection that can be efficiently split into head and tail components
- Examples: List, Stack, Queue

Linear Sequences

- A LinearSeq is a collection that can be efficiently split into head and tail components
- Examples: List, Stack, Queue
- Accessing a LinearSeq through an index is inefficient
- We work with them using the head, tail, and isEmpty methods

Linear Sequences

- A LinearSeq is a collection that can be efficiently split into head and tail components
- Examples: List, Stack, Queue
- Accessing a LinearSeq through an index is inefficient
- We work with them using the head, tail, and isEmpty methods
- By default specifying that you want a LinearSeq, Scala creates a List:

Linear Sequences

- A LinearSeq is a collection that can be efficiently split into head and tail components
- Examples: List, Stack, Queue
- Accessing a LinearSeq through an index is inefficient
- We work with them using the head, tail, and isEmpty methods
- By default specifying that you want a LinearSeq, Scala creates a List:

```
scala>  val seq = scala.collection.immutable.LinearSeq(1,2,3)
seq: scala.collection.immutable.LinearSeq[Int] = List(1, 2, 3)
```

Ranges

- A Range is a range of values, such as 1,2,3,4,5 or 10,20,30
- A Range stores only the start, end, and increment
- You construct Range objects with the **to** and **until** methods

Ranges

- A Range is a range of values, such as 1,2,3,4,5 or 10,20,30
- A Range stores only the start, end, and increment
- You construct Range objects with the **to** and **until** methods

```
scala> val a1 = 0 to 10 by 2
a1: scala.collection.immutable.Range = inexact Range 0 to 10 by 2
scala> val iter = a1.iterator
iter: Iterator[Int] = non-empty iterator
scala> while (iter.hasNext) print (iter.next() + " ")
0 2 4 6 8 10
```

Ranges

- A Range is a range of values, such as 1,2,3,4,5 or 10,20,30
- A Range stores only the start, end, and increment
- You construct Range objects with the **to** and **until** methods

```
scala> val a1 = 0 to 10 by 2
a1: scala.collection.immutable.Range = inexact Range 0 to 10 by 2
scala> val iter = a1.iterator
iter: Iterator[Int] = non-empty iterator
scala> while (iter.hasNext) print (iter.next() + " ")
0 2 4 6 8 10
```

```
scala> val a2 = 0 until 10 by 2
a2: scala.collection.immutable.Range = Range 0 until 10 by 2
scala> val iter = a2.iterator
iter: Iterator[Int] = non-empty iterator
scala> while (iter.hasNext) print (iter.next() + " ")
0 2 4 6 8
```

Lists

- A List is a linear sequence of elements of the same type
  - linked list implementation
- It is an immutable data structure
- It is a recursive data structure
- It is not an efficient data structure for accessing elements by their indices
  - access time is proportional to the position of an element in a list

Creating A List

- A few ways to create a list:

```
val xs = List(10,20,30,40)
```

Creating A List

- A few ways to create a list:

```
val xs = List(10,20,30,40)

val ys = (1 to 100).toList
```

Creating A List

- A few ways to create a list:

```
val xs = List(10,20,30,40)

val ys = (1 to 100).toList

val zs = someArray.toList
```

Basic List Operations

- The method **head** returns the first element
- The method **tail** returns a list with all the elements except the first
- The method **isEmpty** returns true if a list is empty
- The **::** operator makes a new list from a given head and tail

```
scala> 5 :: List(2, 7, 10)
res19: List[Int] = List(5, 2, 7, 10)
scala> res19.head
res20: Int = 5
scala> res19.tail
res21: List[Int] = List(2, 7, 10)
```

Example

```
//file name: list_sum.scala
val lst = (1 to 10).toList
```

Example

```
//file name: list_sum.scala
val lst = (1 to 10).toList

def sumList(lst: List[Int]):Int = {
  if (lst == Nil) //lst.isEmpty
     0
  else
     lst.head + sumList(lst.tail)
}

print(sumList(lst)) //55
```

Reading from a Text File

- To read from console, we used

```
import scala.io.StdIn
```

Reading from a Text File

- To read from console, we used

```
import scala.io.StdIn
```

- To read from other sources, use

```
import scala.io.Source
```

Reading from a Text File

- To read from console, we used

```
import scala.io.StdIn
```

- To read from other sources, use

```
import scala.io.Source
```

- To read from a file, use

```
val objectName = Source.fromFile(filename)
```

Reading from a Text File

- To read from console, we used

```
import scala.io.StdIn
```

- To read from other sources, use

```
import scala.io.Source
```

- To read from a file, use

```
val objectName = Source.fromFile(filename)
```

```
scala> val fileSource = Source.fromFile("file1.txt") // path to the file
fileSource: scala.io.BufferedSource = non-empty iterator
```

Reading from a Text File

- The method mkString returns the whole content of the file as a string

Reading from a Text File

- The method mkString returns the whole content of the file as a string

```
//content of file1.txt
Line 1
Line 2
Line 3
```

Reading from a Text File

- The method mkString returns the whole content of the file as a string

```
scala> fileSource.mkString
res0: String =
Line 1
Line 2
Line 3
```

```
//content of file1.txt
Line 1
Line 2
Line 3
```

Reading from a Text File

- The method mkString returns the whole content of the file as a string

```
scala> fileSource.mkString
res0: String =
Line 1
Line 2
Line 3
```

```
//content of file1.txt
Line 1
Line 2
Line 3
```

```
scala> fileSource
res45: scala.io.BufferedSource = empty iterator
```

Reading from a Text File

- The method mkString returns the whole content of the file as a string

```
scala> fileSource.mkString
res0: String =
Line 1
Line 2
Line 3
```

```
//content of file1.txt
Line 1
Line 2
Line 3
```

```
scala> fileSource
res45: scala.io.BufferedSource = empty iterator
```

```
scala> fileSource.mkString
res2: String = ""
```

Reading from a Text File

- The method mkString returns the whole content of the file as a string

```
scala> val content =
fileSource.mkString
res0: String =
Line 1
Line 2
Line 3
```

```
//content of file1.txt
Line 1
Line 2
Line 3
```

Reading from a Text File

- Process file one char at a time
- fileSource is Iterator[Char]

```
val fileSource = Source.fromFile("file1.txt")
for (value <- fileSource) println(value)
```

Reading from a Text File

```
import scala.io.Source

val fileSource = Source.fromFile("file1.txt")
val lineIterator = fileSource.getLines
//lineIterator: Iterator[String] = non-empty iterator
for (line <- lineIterator) println(line)
Line 1
Line 2
Line 3
```

- The method `getLines` returns an Iterator of type Iterator[String]
- Each element is a line in the file without the EOL

Reading from a Text File

```
import scala.io.Source

val fileSource = Source.fromFile("file1.txt")
val lineIterator = fileSource.getLines
//lineIterator: Iterator[String] = non-empty iterator
for (line <- lineIterator) println(line)
Line 1
Line 2
Line 3

fileSource.close
```

- Remember to close file when done