Scala Basics cont'

Types, Loops, Strings, Reading Files, Data Structures

Lecture Question

- In a package named "lecture" create an object named "LectureQuestion" with a method named "fileSum" that takes a filename as a String and returns an Int which is the sum of the values in the file

- The input file will contain multiple lines each with multiple integer values separated by the '#' character

- Return the sum of all of the integer values in the file

- You may assume that the file exists and is properly formatted

Sample file contents:

```
3#1#8
12#9#25#10
-2#12
1#2
```

Submit a zip file of your project to AutoLab: File > Export to zip file

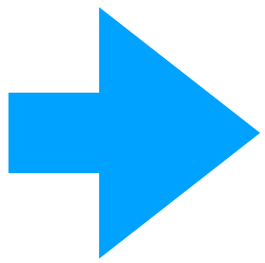
Scala Types

- All values in Scala are objects
  - Objects contain variables and methods
  - No primitive values in Scala
- We'll start with the following types:
  - Int
  - Long
  - Double
  - Boolean
  - Unit
  - String

Int

- A whole number
- 32 bit representation
- -2147483648 to 2147483647
  - Values outside this range will **overflow**
  - Overflow values will wrap around

```
val a: Int = 2147483647
println(a + 1)
```

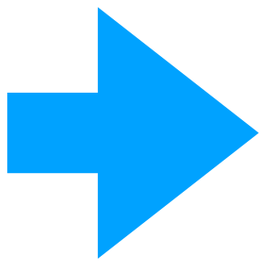


-2147483648

Integer Division

- When dividing two Ints the result is always an Int
- Decimal portion is removed
- Effectively returns the floor of the result

```
val ageInMonths: Int = 245
val monthsPerYear: Int = 12
val ageInYears = ageInMonths/monthsPerYear
println(ageInYears)
```

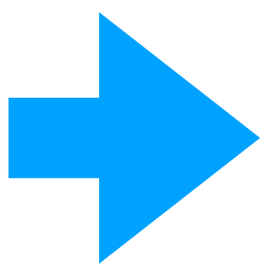


20

Long

- A whole number (Like Int)
- 64 bit representation
- -9223372036854775808 to 9223372036854775807
- Useful when you expect values that would overflow an Int

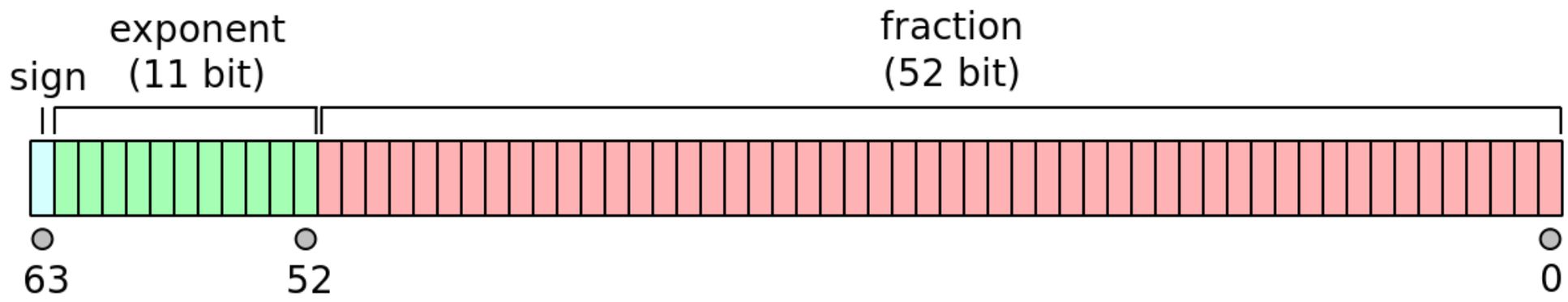
```
val a: Long = 2147483647
println(a + 1)
```



2147483648

Double

- Number with a whole number and a decimal portion
- 64 bit representation
- Values are truncated to fit in 64 bits
- Loss of precision!



https://en.wikipedia.org/wiki/Double-precision_floating-point_format

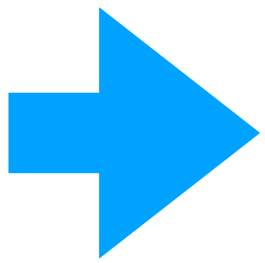
Double

- Values are represented in binary
  - Ex. 0.11 == 1/2 + 1/4 == 3/4
- In decimal we have values that cannot be stored without truncation
  - Ex. 1/3 != 0.33333333333333333333
- Values such as 0.1 cannot be represented as a sum of powers of 2
  - 0.1 (base 10) != 0.0001100110011001100110011001100110011001100110011001 (base 2)
  - But this the best we can do with Double representations

Double

- We need to be aware of this truncation in our programs
- In the code below, c == 0.3 is false!

```
val b: Double = 0.1
val c: Double = b * 3
println(c)
```

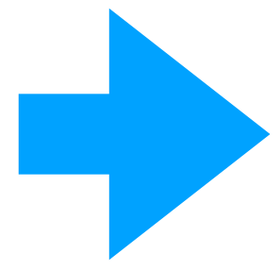


0.30000000000000004

Double

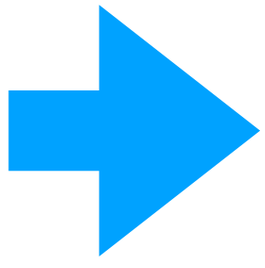
- Checking for equality with Doubles
- Allow a small amount of tolerance when comparing two doubles
- Math.abs(x - y) < small_value
    - As long as x and y are within a small value of each other this will be true

```
val b: Double = 0.1
val c: Double = b * 3
val expected: Double = 0.3
println(c == expected)
```



false

```
val epsilon: Double = 0.00000001
val b: Double = 0.1
val c: Double = b * 3
val expected: Double = 0.3
println(Math.abs(c - 0.3) < epsilon)
```



true

Boolean and Unit

- Boolean
  - true or false
- Unit
  - Nothing
  - Used to indicate a method/function that does not return a value
  - Ex: main and println both return Unit

String

- A sequence of characters (type Char)
- Declared with double quotes
  - val s:String = "valid string literal"
- Many useful methods. Examples:
  - startsWith(String) - check if this String starts with the given String
  - length() - number of characters in this String
  - .split(String) - Separates this String by the given String

Scala Type Conversions

```scala
package example

object Types {

  def main(args: Array[String]): Unit = {

    // Declaring variable
    var anInt: Int = 10
    var aDouble: Double = 5.8
    var aBoolean: Boolean = true
    var aString: String = "6.3"

    // Converting variable types
    var anotherDouble: Double = aString.toDouble
    var anotherString: String = anInt.toString

    // Truncates the decimal. anotherInt == 5
    var anotherInt: Int = aDouble.toInt
  }

}
```

Use the to<Type> methods to convert between types

For Loop

For Loop

```
for(<variable_name> <- <data_structure>){
  <loop_body>
}
```

Reads:

"for variable_name in data_structure execute loop_body"

For Loop

```
package example

object Loop {

  def printOneTo(n: Int): Unit = {
    for(i <- 1 to n){
      println("i == " + i)
    }
  }

  def printOneToAlternate(n: Int): Unit = {
    val numbers: Range = 1 to n
    for (i <- numbers) {
      println("i == " + i)
    }
  }

  def main(args: Array[String]): Unit = {
    printOneTo(10)
  }

}
```

Output:

i == 1
i == 2
i == 3
i == 4
i == 5
i == 6
i == 7
i == 8
i == 9
i == 10

"1 to n" creates a Range of integers that can be iterated over with a for loop
-Similar to range(n) in Python

For Loop + String Example

```
package example

object StringSplitter {

  def computePercentTrue(line: String): Double = {
    val splits: Array[String] = line.split(";")
    var totalCount: Double = 0
    var trueCount: Double = 0
    for (value <- splits) {
      val valueAsBoolean: Boolean = value.toBoolean
      if (valueAsBoolean) {
        trueCount += 1
      }
      totalCount += 1
    }
    trueCount / totalCount
  }

  def main(args: Array[String]): Unit = {
    val testInput = "true;false;true;true;true"
    val percentTrue = computePercentTrue(testInput) // expecting 0.8
    println("Percentage true == " + percentTrue)
  }

}
```

Given a String containing boolean values separated by semicolons, return the percentage of values that are true

For Loop + String Example

```
package example

object StringSplitter {

  def computePercentTrue(line: String): Double = {
    val splits: Array[String] = line.split(";")
    var totalCount: Double = 0
    var trueCount: Double = 0
    for (value <- splits) {
      val valueAsBoolean: Boolean = value.toBoolean
      if (valueAsBoolean) {
        trueCount += 1
      }
      totalCount += 1
    }
    trueCount / totalCount
  }

  def main(args: Array[String]): Unit = {
    val testInput = "true;false;true;true;true"
    val percentTrue = computePercentTrue(testInput) // expecting 0.8
    println("Percentage true == " + percentTrue)
  }

}
```

Split the String on semicolons
-Returns a data structure of Strings

For Loop + String Example

```
package example

object StringSplitter {

  def computePercentTrue(line: String): Double = {
    val splits: Array[String] = line.split(";")
    var totalCount: Double = 0
    var trueCount: Double = 0
    for (value <- splits) {
      val valueAsBoolean: Boolean = value.toBoolean
      if (valueAsBoolean) {
        trueCount += 1
      }
      totalCount += 1
    }
    trueCount / totalCount
  }

  def main(args: Array[String]): Unit = {
    val testInput = "true;false;true;true;true"
    val percentTrue = computePercentTrue(testInput) // expecting 0.8
    println("Percentage true == " + percentTrue)
  }

}
```

Iterate over each value

For Loop + String Example

```
package example

object StringSplitter {

  def computePercentTrue(line: String): Double = {
    val splits: Array[String] = line.split(";")
    var totalCount: Double = 0
    var trueCount: Double = 0
    for (value <- splits) {
      val valueAsBoolean: Boolean = value.toBoolean
      if (valueAsBoolean) {
        trueCount += 1
      }
      totalCount += 1
    }
    trueCount / totalCount
  }

  def main(args: Array[String]): Unit = {
    val testInput = "true;false;true;true;true"
    val percentTrue = computePercentTrue(testInput) // expecting 0.8
    println("Percentage true == " + percentTrue)
  }

}
```

Convert the Strings to Booleans

For Loop + String Example

```
package example

object StringSplitter {

  def computePercentTrue(line: String): Double = {
    val splits: Array[String] = line.split(";")
    var totalCount: Double = 0
    var trueCount: Double = 0
    for (value <- splits) {
      val valueAsBoolean: Boolean = value.toBoolean
      if (valueAsBoolean) {
        trueCount += 1
      }
      totalCount += 1
    }
    trueCount / totalCount
  }

  def main(args: Array[String]): Unit = {
    val testInput = "true;false;true;true;true"
    val percentTrue = computePercentTrue(testInput) // expecting 0.8
    println("Percentage true == " + percentTrue)
  }

}
```

Count the total number of values and the number that are true

For Loop + String Example

```
package example

object StringSplitter {

  def computePercentTrue(line: String): Double = {
    val splits: Array[String] = line.split(";")
    var totalCount: Double = 0
    var trueCount: Double = 0
    for (value <- splits) {
      val valueAsBoolean: Boolean = value.toBoolean
      if (valueAsBoolean) {
        trueCount += 1
      }
      totalCount += 1
    }
    trueCount / totalCount
  }

  def main(args: Array[String]): Unit = {
    val testInput = "true;false;true;true;true"
    val percentTrue = computePercentTrue(testInput) // expecting 0.8
    println("Percentage true == " + percentTrue)
  }

}
```

Compute the average

-Note: If these values were Ints this would be integer division

Reading Files

Reading Files

```scala
package example

import scala.io.Source

object FileReader {

  def convertFileToString(filename: String): String = {
    var contents: String = ""
    val file: BufferedSource = Source.fromFile(filename)
    for (line <- file.getLines()){
      contents += line + "\n"
    }
    contents
  }

  def main(args: Array[String]): Unit = {
    val filename = "data/testFile.txt"
    val contents = convertFileToString(filename)
    println(contents)
  }

}
```

Read the contents of a file into a String line-by-line

-Assumes "data/testFile.txt" exists in the project

Reading Files

```
package example

import scala.io.Source

object FileReader {

  def convertFileToString(filename: String): String = {
    var contents: String = ""
    val file: BufferedSource = Source.fromFile(filename)
    for (line <- file.getLines()){
      contents += line + "\n"
    }
    contents
  }

  def main(args: Array[String]): Unit = {
    val filename = "data/testFile.txt"
    val contents = convertFileToString(filename)
    println(contents)
  }

}
```

Import the Source object from the standard library

Reading Files

```
package example

import scala.io.Source

object FileReader {

  def convertFileToString(filename: String): String = {
    var contents: String = ""
    val file: BufferedSource = Source.fromFile(filename)
    for (line <- file.getLines()){
      contents += line + "\n"
    }
    contents
  }

  def main(args: Array[String]): Unit = {
    val filename = "data/testFile.txt"
    val contents = convertFileToString(filename)
    println(contents)
  }

}
```

Call scala.io.Source.fromFile(filename: String): BufferedSource

Reading Files

```
package example

import scala.io.Source

object FileReader {

  def convertFileToString(filename: String): String = {
    var contents: String = ""
    val file: BufferedSource = Source.fromFile(filename)
    for (line <- file.getLines()){
      contents += line + "\n"
    }
    contents
  }

  def main(args: Array[String]): Unit = {
    val filename = "data/testFile.txt"
    val contents = convertFileToString(filename)
    println(contents)
  }

}
```

Call BufferedSource.getLines() to get the lines in a data structure of Strings

Reading Files

```
package example

import scala.io.Source

object FileReader {

  def convertFileToString(filename: String): String = {
    var contents: String = ""
    val file: BufferedSource = Source.fromFile(filename)
    for (line <- file.getLines()){
      contents += line + "\n"
    }
    contents
  }

  def main(args: Array[String]): Unit = {
    val filename = "data/testFile.txt"
    val contents = convertFileToString(filename)
    println(contents)
  }

}
```

Do whatever you need to do with the content of the file (Just printing to the screen in this example)

Reading Files

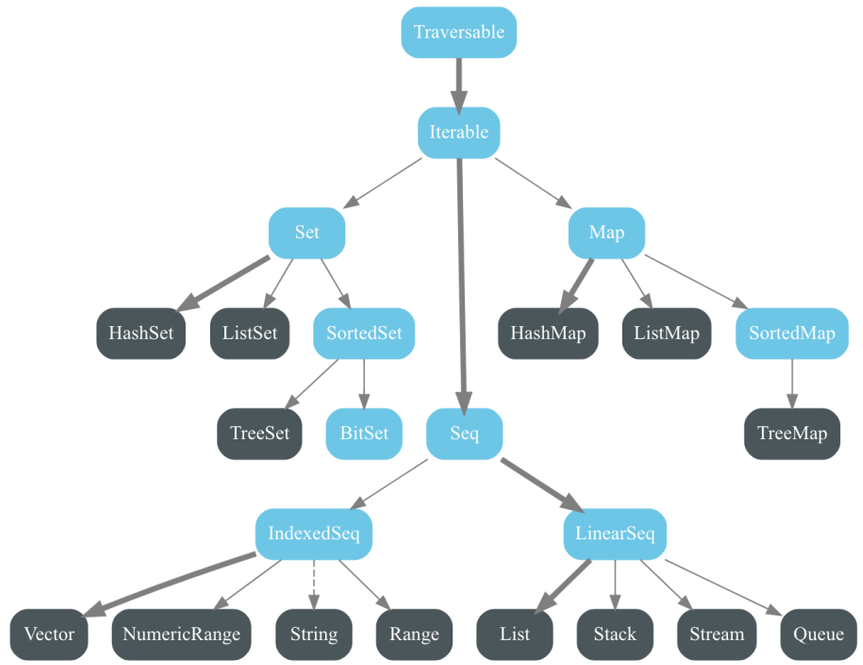
Example in IntelliJ

Data Structures

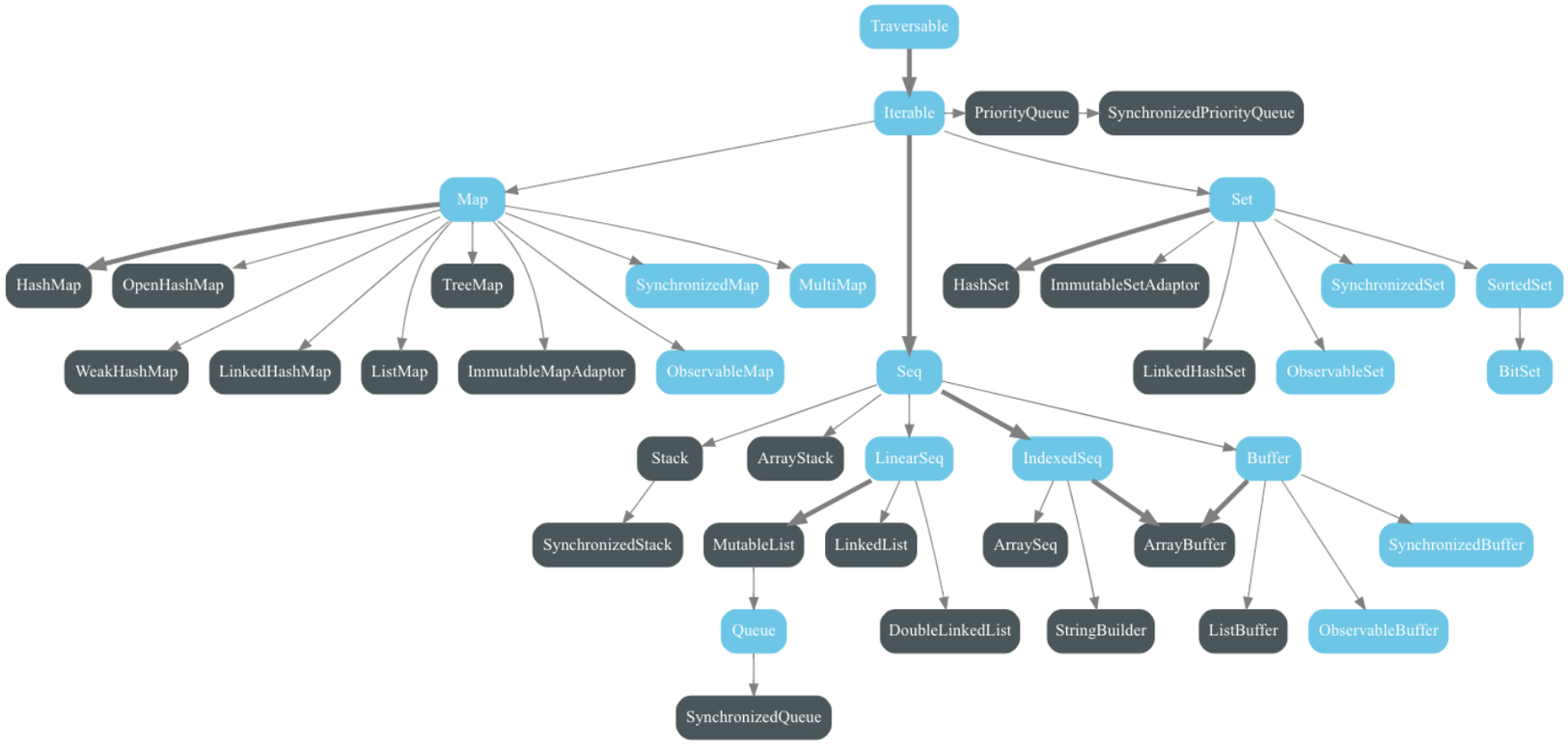
Data Structures

- Wide variety of Data Structures in Scala

Graph for immutable hierarchy:



Graph for mutable hierarchy:



https://docs.scala-lang.org/tutorials/FAQ/collections.html

Data Structures

- Let's keep it simple and focus on 3 for now
- Array
  - Sequential
  - Fixed Size
- List
  - Sequential
- Map
  - Key-Value Store

Array

- Sequential
  - One continuous block of memory
  - Random access based on memory address
    - address = first_address + (element_size * index)
- Fixed Size
  - Since memory adjacent to the block may be used
  - Efficient when you know how many elements you'll need to store

Array

```scala
def arrayExample(): Unit = {
  // Create new Array of Int
  val arr: Array[Int] = Array(2, 3, 4)

  // Change a value by index
  arr(1) = 20

  // Access a value by index
  val x: Int = arr(1)

  // Iterate over elements
  for (element <- arr) {
    println(element)
  }

  // Iterate over indices
  for (index <- 0 to arr.length) {
    println(index)
  }

  // Iterate over indices - alternate
  for (index <- arr.indices) {
    println(index)
  }

}
```

List

- Sequential
  - Spread across memory
  - Each element knows the memory address of the next element
    - Follow the addresses to find each element
- Variable Size
  - Store new element anywhere in memory
  - Add the new memory address to the last element
    - Or new element stores address of first element
- Values cannot change [In Scala]

List

```scala
def listExample(): Unit = {
  // Create new Array of Int
  var list: List[Int] = List(2, 3, 4)

  // Access the first element
  val x: Int = list.head

  // Access a value by position
  val y: Int = list.apply(1)

  // Add an element to the end of the list (append)
  list = list :+ 50

  // Add an element to the beginning of the list (prepend)
  list = 70 :: list

  // Iteration
  for(element <- list){
    println(element)
  }
}
```

Map

- Key-Value Store
  - Values stored at keys instead of indices
  - Multiple different implementations
    - Default is HashMap (CSE250 topic)
- Variable Size
- Variable Values

Map

```
def mapExample(): Unit = {
  // Create new Map of Int to Int
  var myMap: Map[Int, Int] = Map(2 -> 4, 3 -> 9, 4 -> 16)

  // Add an key-value pair
  myMap = myMap + (5 -> 25)

  // Access a value by key (Crashes if key not in map)
  val x: Int = myMap(3)

  // Access a value by key with default value if key not in map
  val y: Int = myMap.getOrElse(100, -1)

  // Iteration
  for((key, value) <- myMap){
    println("value " + value + " stored at key " + key)
  }
}
```

Lecture Question

- In a package named "lecture" create an object named "LectureQuestion" with a method named "fileSum" that takes a filename as a String and returns an Int which is the sum of the values in the file

- The input file will contain multiple lines each with multiple integer values separated by the '#' character

- Return the sum of all of the integer values in the file

- You may assume that the file exists and is properly formatted

Sample file contents:

```
3#1#8
12#9#25#10
-2#12
1#2
```

Submit a zip file of your project to AutoLab: File > Export to zip file