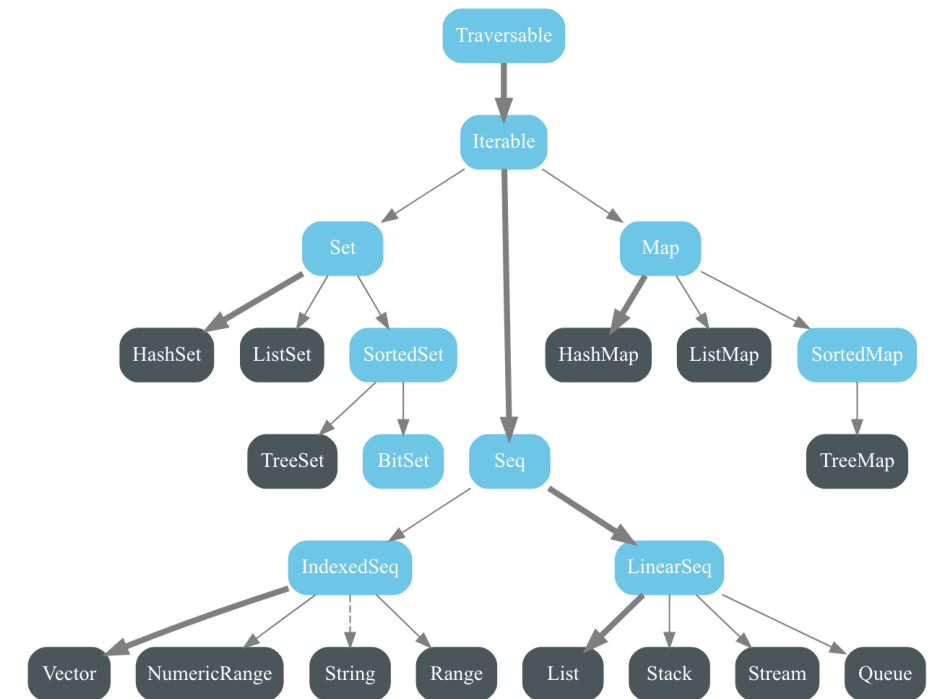


Data Structures

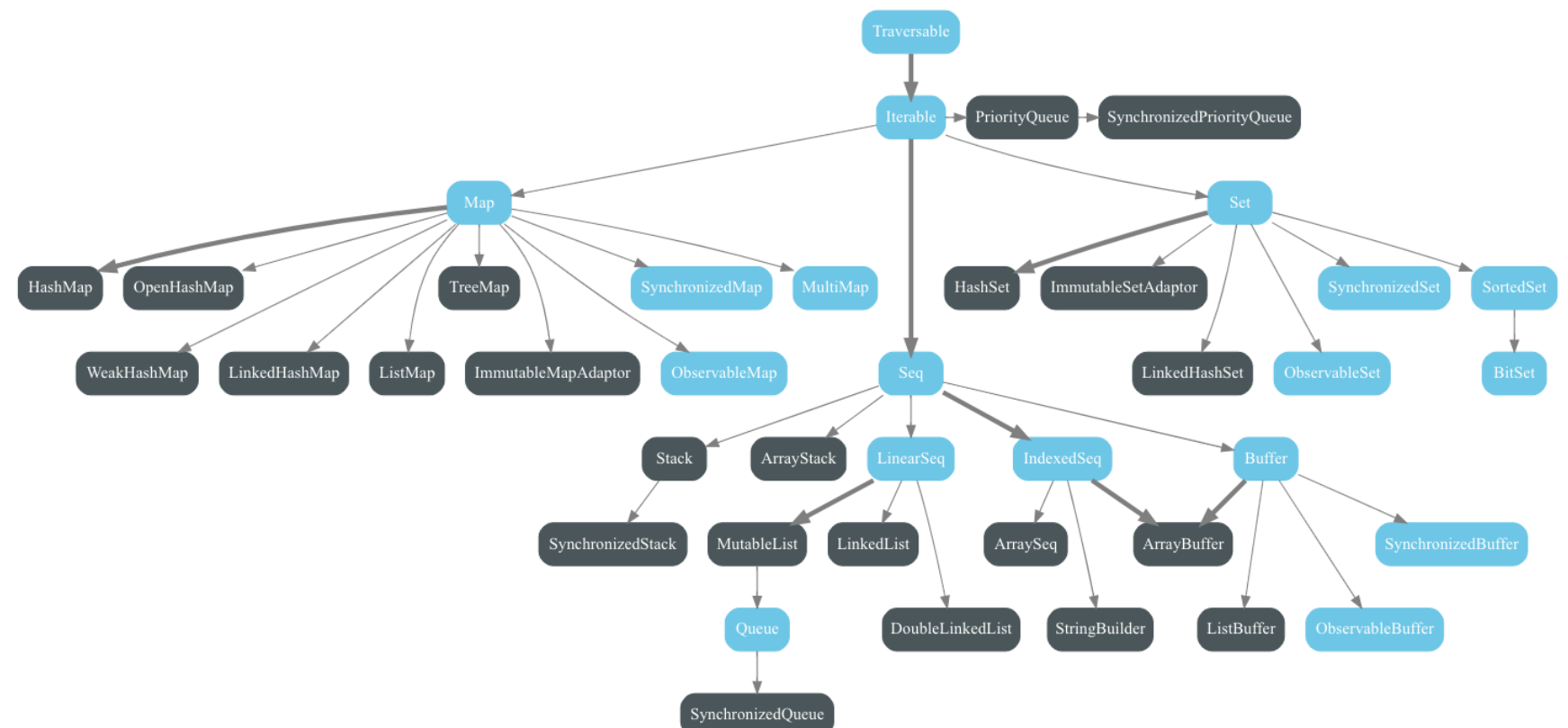
Data Structures

- Wide variety of Data Structures in Scala

Graph for immutable hierarchy:



Graph for mutable hierarchy:



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
 - $\text{address} = \text{first_address} + (\text{element_size} * \text{index})$
- Fixed Size
 - Since memory adjacent to the block may be used
 - Efficient when you know how many elements you'll need to store

Array

```
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)  
  }  
}
```

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 store the memory address in the last element
 - or new element stores address of first element
- Values cannot change

List

```
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)  
  }  
}
```

Scala Unit Testing

```
// return the maximum value that can be computed by multiplying two  
// different elements from the input list  
// -Two elements with the same value are considered different  
// If the list only contains 1 element, return that element squared  
// If the list is empty, return -1
```

```
def maxMultiply(input: List[Int]): Int = {  
  
  if (input.isEmpty) {  
    -1  
  } else if (input.length == 1) {  
    // round to avoid truncation errors  
    Math.round(Math.pow(input.head, 2)).toInt  
  } else {  
    var maxFound = input.head * input.apply(1)  
    var ix = 0  
    var iy = 0  
    for (x <- input) {  
      iy = 0  
      for (y <- input) {  
        if (iy != ix) {  
          if (x * y > maxFound) {  
            maxFound = x * y  
          }  
        }  
        iy += 1  
      }  
      ix += 1  
    }  
    maxFound  
  }  
}
```

Lecture Question

In a package named "lecture" there is an object named "DataStructures" with a method named "maxMultiply" attempting to solve the question from lecture. There will be a correct solution and 3 incorrect solutions in AutoLab:

```
// Return the maximum value that can be computed by multiplying two  
// different elements from the input list  
// -Two elements with the same value are considered different  
// If the list only contains 1 element, return that element squared  
// If the list is empty, return -1  
// All numbers are Ints
```

Unit Testing: In a package named "tests" create a class named "TestDataStructures" as a test suite that tests the maxMultiply method

Suggestion: It is helpful to store test cases in a Map with inputs as keys and expected outputs as values

* This question will be open until midnight