

# Scala: Collections



Muhammad Tahir

Lecture 9

# Contents

- ① Scala Collections
- ② Immutable Scala Collections
- ③ Chisel: Controller Implementation

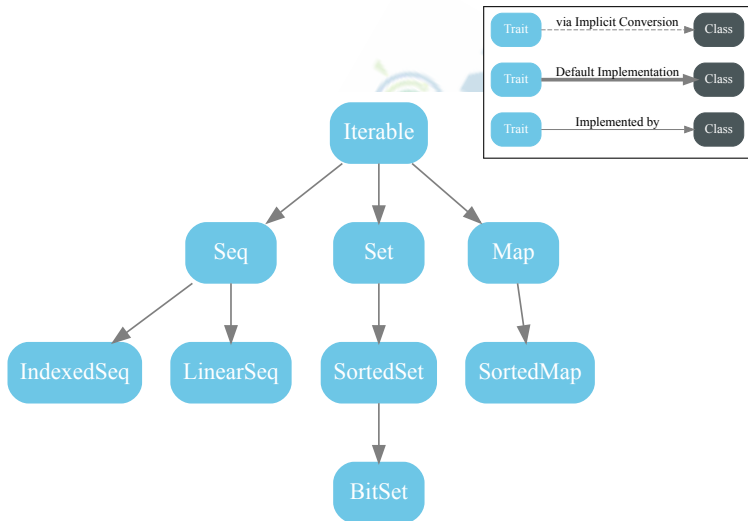
# Scala Collections

- Available in multiple packages or sub-packages
  - `scala.collection`: Includes both mutable and immutable collections
  - `scala.collection.mutable`: Includes mutable collections
  - `scala.collection.immutable`: Includes immutable collections

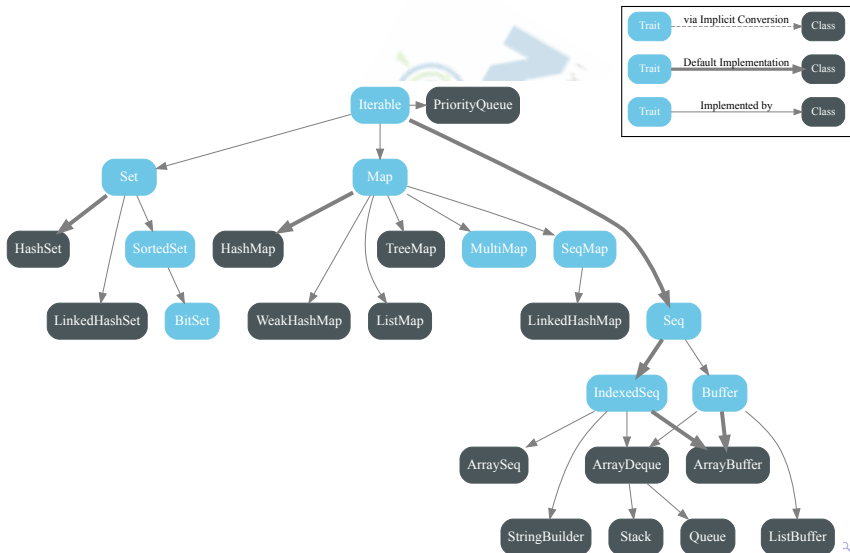
# Scala Collections

- Available in multiple packages or sub-packages
  - `scala.collection`: Includes both mutable and immutable collections
  - `scala.collection.mutable`: Includes mutable collections
  - `scala.collection.immutable`: Includes immutable collections
- Using `Seq` implies immutable collection
- Using `mutable.Seq` will refer to mutable counterpart

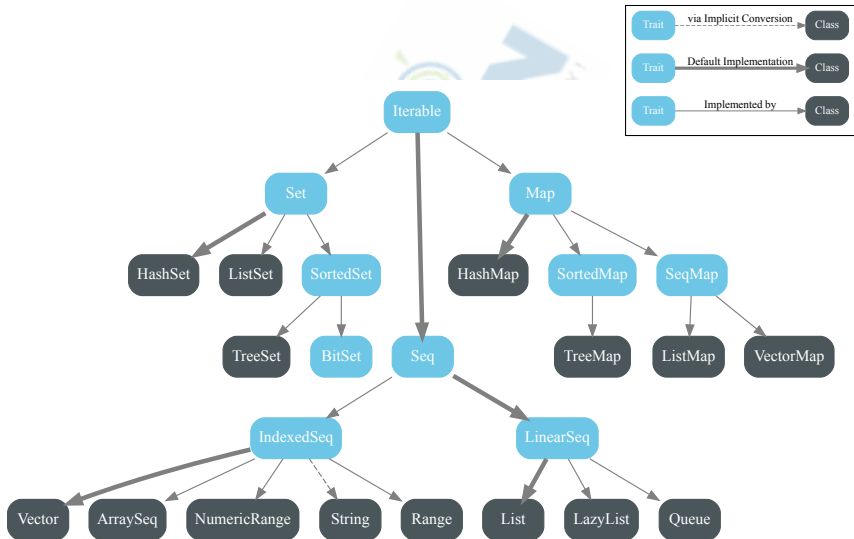
# Scala Collections



# Mutable Scala Collections



# Immutable Scala Collections



# Scala Mutable Collections: Array

Arrays are Mutable Collections of same data type

```
val arr1: Array [Int] = new Array [Int](4)
// Data type (Array of Int) and length (equal to 4) are
// mentioned explicitly
// Both of these attributes can be inferred implicitly

// Inferring data type
val arr2 = new Array [Int](5)

// Inferring data type and width
val arr3 = Array (1, 2, 3, 4, 5, 6)
```



## Selected Immutable Scala Collections

- Seq, Map, Set
- Seq
  - List, Queue, Stack,
  - Sting, Vector
- Set
  - Bitset, ListSet
- Map
  - ListMap, TreeMap

## Scala Immutable: Seq

- `Seq` is a base trait for sequences
- Sequences are special cases of iterable collections
- Sequences always have a defined order of elements
- Provide a method `apply` for indexing

## Seq: Illustration

- Example illustration of Seq and some common methods

```
val xseq = Seq(2 -> 'a', 5 -> 'b', 3 -> 'c')

println(xseq(1))
println(xseq.apply(0))
println(xseq.length)
println(xseq.toList)

// following is printed to the terminal window
(5,b)
(2,a)
3
List((2,a), (5,b), (3,c))
```

## Seq: Example Load Operation

- Uses UInt for indexing and yields the Seq element

```
// Seq and MuxLookup for indexing load operation
val raw_data = io.dmem.rdata
val ld_data = MuxLookup(ld_type, 0.U,
                        Seq(LD_LW -> raw_data.zext,
                           LD_LH -> raw_data(15, 0).asSInt,
                           LD_LB -> raw_data(7, 0).asSInt,
                           LD_LHU -> raw_data(15, 0).zext,
                           LD_LBU -> raw_data(7, 0).zext))
```

## Seq: CSRs

- Uses **BitPat** for indexing and yields the Seq element

```
// CSR register file
val csrFile = Seq(
  BitPat(CSR.CYCLE)      -> cycle,
  BitPat(CSR.TIME)       -> time,
  BitPat(CSR.INSTRET)    -> instret,
  BitPat(CSR.CYCLEH)     -> cycleh,
  BitPat(CSR.TIMEH)      -> timeh,
  BitPat(CSR.INSTRETH)   -> instreth,
  BitPat(CSR.MTVEC)      -> mtvec,
  BitPat(CSR.MIE)        -> mie.asUInt,
  BitPat(CSR.MSCRATCH)   -> mscratch,
  BitPat(CSR.MEPC)       -> mepc,
  BitPat(CSR.MCAUSE)     -> mcause,
  BitPat(CSR.MTVAL)      -> mtval,
  BitPat(CSR.MIP)        -> mip.asUInt,
  BitPat(CSR.MSTATUS)    -> mstatus.asUInt,
  BitPat(CSR.MISA)       -> misa
)

// reading CSR,
io.out := Lookup(csr_addr, 0.U, csrFile).asUInt
```

## Lists

- List are immutable collection
- Lists are implemented as linked lists
- Basic operations performed on lists are, `head`, `tail`, `isEmpty`

```
// List of Strings
val modules: List[String] = List("ALU", "Branch", "Control")

// List of Integers
val nums: List[Int] = List(1, 2, 3, 3, 4)

// Two dimensional list
val matrix: List[List[Int]] =
  List( List(1, 0, 0),
        List(0, 1, 0),
        List(0, 0, 1) )

// Display the lists
println(modules) // List(ALU, Branch, Control)
println(nums)    // List(1, 2, 3, 3, 4)
println(matrix)  // List(List(1, 0, 0), List(0, 1, 0), List(0, 0, 1))
```

# List Illustration

```
// List of Strings
val modules: List[String] = List("ALU", "Branch", "Control")
val peripherals = List("Uart", "Timer")

// List of Integers
val nums: List[Int] = List(1, 2, 3, 3, 4)

// split the list apart
val List(b, c, d) = modules
val a :: rest = modules

// concatenate two lists (using ::: or List.:::() or List.concat() )
val combined_list = modules ::: peripherals
println(rest)           // List(Branch, Control)
println(nums)           // List(1, 2, 3, 3, 4)
println(combined_list)  // List(ALU, Branch, Control, Uart, Timer)

// combine two lists of different types
println(modules:::nums) // List(ALU, Branch, Control, 1, 2, 3, 3, 4)
```

# Set

- **Set** is a collection of pairwise different elements
- The elements are of same type without any ordering
- Basic operations performed on sets are, **head**, **tail**, **isEmpty**

```
// An empty set of type integer
val set1 : Set[Int] = Set()

// A nonempty set of integer type
val set2 : Set[Int] = Set(1,3,3,5,5,7)
```



## Set Illustration

```
// An example set of type integer
val set1 : Set[Int] = Set(2,4,5)

// A second example set of integer type
val set2 : Set[Int] = Set(1,3,3,5,5,7)

// Union of sets with ++ as operator
var set_union1 = set1 ++ set2

// Union of sets with ++ as method
var set_union2 = set1.++(set2)

// Intersection of sets with & as method
var set_intersect1 = set1.&(set2)

println(set1)           // Set(2, 4, 5)
println(set2)           // Set(1, 3, 5, 7)
println(set2.head)      // 1
println(set2.tail)      // Set(3, 5, 7)
println(set1.isEmpty)   // false
println(set_union1)     // Set(5, 1, 2, 7, 3, 4)
println(set_union2)     // Set(5, 1, 2, 7, 3, 4)
println(set_intersect1) // Set(5)
```

# Map

- Map is a collection of key-value pairs
- Keys are unique, but values can be arbitrary

```
// An empty map with keys as strings and values as integers
var empty: Map[Char, Int] = Map()

// A map with keys and values.
val codes = Map("code1" -> 0xFF0000, "code2" -> 0xF0FFFF)
```

## Map Illustration

```
// Map of timer registers
val timer_regs_map = Map("cntReg" -> 10001014, "cmpReg" -> 10001018, "contReg"
    -> 10001010)

// Map of uart registers
val uart_regs_map = Map("txReg" -> 10001000, "rxReg" -> 10001004, "contReg" ->
    10001008)

println("Timer Regs: " + timer_regs_map)
println("Uart Regs: " + uart_regs_map)

// concatenating two maps using ++ operator
val combined_regs = (uart_regs_map ++ timer_regs_map)

combined_regs.keys.foreach{i => print( "Reg Name = " + i )
    println(" Address = " + combined_regs(i))}
```

```
// The output at the terminal
Timer Regs: Map(cntReg -> 10001000, cmpReg -> 10001004, contReg -> 10001010)

Uart Regs: Map(txReg -> 10001000, rxReg -> 10001004, contReg -> 10001008)

Reg Name = contReg Address = 10001010
Reg Name = rxReg Address = 10001004
Reg Name = txReg Address = 10001000
Reg Name = cmpReg Address = 10001018
Reg Name = cntReg Address = 10001014
```

# Tuple

- Allows to have heterogeneous data types in one collection
- Element access of the tuple is different from other collections

```
// Tuple illustration
val uTup = (2.5, true, "Chisel")
println(s" Data at location 1 is : ${uTup._1}")

// output at the terminal
Data at location 1 is : 2.5
```

# Chisel: Controller

## Control signals

```
class ControlSignals extends Bundle with Config {  
  val inst      = Input(UInt(XLEN.W))  
  val pc_sel    = Output(UInt(2.W))  
  val inst_kill = Output(Bool())  
  val A_sel     = Output(UInt(1.W))  
  val B_sel     = Output(UInt(1.W))  
  val imm_sel   = Output(UInt(3.W))  
  val alu_op    = Output(UInt(5.W))  
  val br_type   = Output(UInt(3.W))  
  val st_type   = Output(UInt(2.W))  
  val ld_type   = Output(UInt(3.W))  
  val wb_sel    = Output(UInt(2.W))  
  val wb_en     = Output(Bool())  
  val csr_cmd   = Output(UInt(3.W))  
  val illegal   = Output(Bool())  
  val en_rs1    = Output(Bool())  
  val en_rs2    = Output(Bool())  
}
```

# Chisel: Controller

## Controller definitions



```

19 object Control {
20
21
22   val default =
23     //
24     //           pc_sel  A_sel  B_sel  imm_sel  alu_op  br_type | st_type ld_type wb_sel | csr_cmd | en_rs1 |
25     //           |      |      |      |      |      |      |      |      |      |      |      |
26     List(PC_4,   A_XXX, B_XXX, IMM_X, ALU_XXX, , BR_XXX, N, ST_XXX, LD_XXX, WB_ALU, N, CSR.Z, Y, N, N)
27   val map = Array(
28     LUI  -> List(PC_4 , A_PC, B_IMM, IMM_U, ALU_COPY_B, BR_XXX, N, ST_XXX, LD_XXX, WB_ALU, Y, CSR.Z, N, N, N),
29     AUIPC-> List(PC_4 , A_PC, B_IMM, IMM_U, ALU_ADD , BR_XXX, N, ST_XXX, LD_XXX, WB_ALU, Y, CSR.Z, N, N, N),
30     JAL  -> List(PC_ALU, A_PC, B_IMM, IMM_J, ALU_ADD , BR_XXX, Y, ST_XXX, LD_XXX, WB_PC4, Y, CSR.Z, N, N, N),
31     JALR -> List(PC_ALU, A_RS1, B_IMM, IMM_I, ALU_ADD , BR_XXX, Y, ST_XXX, LD_XXX, WB_PC4, Y, CSR.Z, N, Y, N),
32
33     BEQ  -> List(PC_4 , A_PC, B_IMM, IMM_B, ALU_ADD , BR_EQ , N, ST_XXX, LD_XXX, WB_ALU, N, CSR.Z, N, Y, Y),
34     BNE  -> List(PC_4 , A_PC, B_IMM, IMM_B, ALU_ADD , BR_NE , N, ST_XXX, LD_XXX, WB_ALU, N, CSR.Z, N, Y, Y),
35     BLT  -> List(PC_4 , A_PC, B_IMM, IMM_B, ALU_ADD , BR_LT , N, ST_XXX, LD_XXX, WB_ALU, N, CSR.Z, N, Y, Y),
36     BGE  -> List(PC_4 , A_PC, B_IMM, IMM_B, ALU_ADD , BR_GE , N, ST_XXX, LD_XXX, WB_ALU, N, CSR.Z, N, Y, Y),
37     BLTU -> List(PC_4 , A_PC, B_IMM, IMM_B, ALU_ADD , BR_LTU, N, ST_XXX, LD_XXX, WB_ALU, N, CSR.Z, N, Y, Y),
38     BGEU -> List(PC_4 , A_PC, B_IMM, IMM_B, ALU_ADD , BR_GEU, N, ST_XXX, LD_XXX, WB_ALU, N, CSR.Z, N, Y, Y),

```

# Chisel: ListLookup

## Controller Implementation using ListLookup

```
class Control extends Module {  
  val io = IO(new ControlSignals)  
  val ctrlSignals = ListLookup(io.inst, Control.default,  
    Control.map)  
  
  // Control signals for Fetch  
  io.pc_sel      := ctrlSignals(0)  
  io.inst_kill   := ctrlSignals(6).toBool  
  
  // Control signals for Execute  
  io.A_sel      := ctrlSignals(1)  
  io.B_sel      := ctrlSignals(2)  
  io.imm_sel    := ctrlSignals(3)  
  io.alu_op     := ctrlSignals(4)  
  
  ...  
}
```

## Reading List I

- Read Chapters 16, 24 and 25 of [Odersky et al., 2016] for in-depth understanding of Scala Collections
- The tutorial available at [Tutorial, 2020] is good resource for quick reference



## References



Odersky, M., Spoon, L., and Venners, B. (2016).  
*Programming in Scala*.  
Artima Incorporation.



Tutorial (2020).  
Scala tutorial.  
<https://www.tutorialspoint.com/scala/index.htm>.