L8: Enumerations, Collections and Exceptions

Rachata Ausavarungnirun

(rachata.a@tggs.kmutnb.ac.th)

January 30th, 2020

Architecture Research Group

Software System Engineering

Thai-German Graduate School, KMUTNB

Can We Use Enumeration???

- In Scala, we can use trait for enum
- Example:
- trait Direction (Declare new object name Direction)
  case object North extends Direction
  case object South extends Direction
  case object East extends Direction
  case object West extends Direction

- Is the same as
  public enum Direction { NORTH, SOUTH, EAST, WEST }
  in java

Complex Pattern Matching

- Extract the first two element of a list?
- val (fst, snd) = xs match { case a::b::_ => (a, b) }
  - Note, this will give a warning and will fail if the list has <2 items

- Use if inside the case
- def quantify(num: Int) = num match {
  case n **if n > 100** => n.toString + " is huge"
  case n **if n > 10** => n.toString + " is large"
  case _ => "small" }

Let's Rewrite the min function

- Extract the minimum number in a list
- We can do the following:
-
```
def min(xs: List[Int]): Option[Int] = xs match {
case Nil => None
case h::Nil => Some(h)
case h::t => Some(Math.min(h, min(t).get))
}
```

(L has nothing)

(Tail is empty list)

give an int

One item

List

The remainder of the list

Polymorphic Types

- Given a list of integers and a position k, can you write a function nth that ***returns the k-th element in the lis***t (k starts from 0)?
- def nth(xs: List[Int], k: Int): Int =
  if (k==0) xs.head else nth(xs.tail, k-1)

- What if I keep asking for a list of String, Double, etc.
  - This gets annoying

- You can use a polymorphic type for this

Example

- We first declare
- def nth[A](xs: List[A], k: Int) = ??? [List of any type A]

- Then write the body of nth
- def nth[T](xs: List[T], k: Int): A =
  if (k==0) xs.head else nth(xs.tail, k-1)

- This code expend a list of element, each of type T, and return the k-th element of this same type T

Example 2: zip

- Remember our last in-class exercise? Let's use the concept of polymorphism for the zip
- def zip[A, B](xs: List[A], ys: List[B]): List[(A, B)] =
  (xs, ys) match {
  case (Nil, Nil) => Nil
  case (x::xs, y::ys) => (x, y)::zip(xs, ys)
  case _ => ??? // should not happen }

Tell the data type

(Empty List)

Assuming that two lists has same length

- This work for case class, case object too
  - We will get into this later

Collections Can Be Versatile (Build-in Library)

- Scala's collections come with library method
- Example: let's assume val L = List(1,2,3)
- You can do L.length to get the length of this list
- You can do L.exist(predicate) to check if the matching predicate exist [L.exist(3) = True]
- You can map a function to all elements using map
  - For example, you can multiply all elements by 2 using L.map(x=>2x)
- You can filter out elements
  - L.filter(x=>x<2) Filter value which more than 2
- Add all of them using L.sum
- Drop elements
- More info on scaladoc

multiply x by 2

L = (2,4,6)

Folding

Val L = List(1, 2, 3, 4)
x => 2x
List

L.fold(x => 2x) ⟹ 1x2
2+(2x2)
6+(2x3)
12+(2x4)
20
starting state

- Assume: def foo(lst): accum_state = (...initial state...)
  for elt in lst:
  accum_state = do_magic(accum_state, elt) return accum_state
  return value or L.foldleft accum_state do_func
- You can use fold for this by
  (0) (function)
- xs.foldLeft initialState doMagic

- This start from the list xs, and then accumulately perform the doMagic function to each element of xs from left to right

Exceptions

- What if your program run into a rare state such as
  - Accessing an empty list
  - Evaluate 2/0
- Basically the program wants to convey something is wrong
- Exception is a built-in feature of a language to handle these
- In Scala, we can use the throw keyword to raise an exception
- throw new IllegalArgumentException

(exception name)

Result Type of an Exception

- Remember Scala is a strongly-typed functional language
- Exception has a result type
- Example: [get head of the list]
- val hdOfList = xs match {
  case h::_ => h
  case Nil => throw new RuntimeException("xs can't be empty") }
- In this case, the exception will by correctly type Int if xs is a list of Int

What to Do With an Exception?

- Ignore → Your program terminate
- Catch and handle it
- def divMod(x: Int, y: Int) = :Int

```
try {
(x/y, x%y)
} catch {
case e: ArithmeticException => (0, 0)   (return 0)
}
```

Other Uses

- Let's consider this function
- def findLast(xs: List[Int], key: Int): Option[Int] = {
  def iterFind(xs: List[Int], location: Int): Option[Int] =
  xs match {
    case Nil => None
    case h::t => {
      val tailFound = iterFind(t, location+1)
      if (h==key && tailFound.isEmpty) Some(location)
      else tailFound
    }
    }
  iterFind(xs, 0) }

- Terminate the loop early → once you found the number throw exception and return the index

catch
  case e: Name => Index
  case e: Not found => -1

Other Uses

- What if I call findList(List(1,2,3,2,4,2,5), 2)
- This is going to be a long chain calls

- Solution: We can use exception to jump right out!

Other Uses

- def findLast(xs: List[Int], key: Int): Option[Int] = {
  case class FoundIndex(loc: Int) extends Exception
  def iterFind(xs: List[Int], location: Int): Option[Int] =
  xs match {
  case Nil => None
  case h::t => { val tailFound = iterFind(t, location+1)
  if (h==key) { throw FoundIndex(location) }
  else tailFound } }
  try { iterFind(xs, 0) } catch {
  case FoundIndex(loc) => Some(loc)
  }
  }

Before We Leave Today

In-class Exercise 7

- def unzip(xs: List[(Int,Int)]): (List[Int], List[Int]) reverses what zip does. Make it so that it's polymorphic. The input can be any List[(A, B)].
- def countWhile[T](xs: List[T], key: T): Int that counts the number of times key repeats itself in the prefix of xs.
- def topK(xs: List[Int], k: Int): List[Int] that tallies the elements of xs and return elements with the top k frequencies (if there are ties, break ties in any way you like). Look at Scaladoc for inspiration.
- Make a sum type called Dessert, which can be one of the following: – Pie(kind: String), – Smoothie(fruits: List[String]), – Cake(toppings: String)
- Then, write a function def isLiquid(what: Dessert): Boolean