Functional Programming with Scala: Class 4

John Nestor
47 Degrees, Persist Software

uw@persist.com

November 7, 2015

Scala 2.12 Released

- Requires Java 8
- More direct mapping from Scala to Java byte code
- New ScalaDoc
- Some minor breaking changes
- 2.11 compiled code not compatible with 2.12
  - must use 2.12 libraries
- http://www.scala-lang.org/news/2.12.0

Tuesday

- 47 Degrees Open Office Hours 1:30-5:30
- Scala Meetup 6:30

Review

- Simplify
- Present solution(s)
- Code review and discussion

Lecture

Scala Collections

- scala.collection
- scala.collection.mutable
- * scala.collection.immutable

Immutable Collections (Collections.sc)

- Seq
  - List
  - Vector
- Set
  - HashSet
  - TreeSet
  - BitSet
- Map
  - HashMap
  - TreeMap

Collection Performance

- http://docs.scala-lang.org/overviews/collections/performance-characteristics

Map, FlatMap and Filter

- Map.sc

For Comprehensions

- ForComp.sc

Set and Map

- SetMap.sc

Option, Try and Future

- OptionTryFuture.sc
- Also define map, flatMap, filter
- Can be used in for comprehensions

Folds (Fold.sc)

- fold
  - op must be associative
  - can be used to implement sum and max
- foldLeft
  - op need not be associative
  - can be used to implement reverse
  - can be used to implement map, flatMap, filter
- foldRight

More Operations (OtherOps.sc)

- zip zipWithIndex
- sortBy
- groupBy

Categories

- Categories abstract other mathematical concepts
- A set objects: S
- A set of morphisms (arrows) f: a => b
  - where a and b are in S
- A binary composition operator compose
  - given f: A => B and g: B => C
    then (f compose g) exists
- Each object has an identity morphism
- ((f compose g) compose h) === (f compose (g compose h))

Categories

- A category abstracts other mathematical concepts
  - Use in computer science to describe types
  - Used in physics to describe vector spaces
  - Lots of mathematics examples
- “[Category theory] does not itself solve hard problems in topology or algebra. It clears away tangled multitudes of individually trivial problems. It puts the hard problems in clear relief and makes their solution possible.” - Colin McLarty

Functors

- Functor - abstracts: map
  - a category whose set consist of categories
  - maps a category A to another category B
    - each object in A maps to an object in B
    - each arrow in A maps to an arrow in B

Monads

- Monad serves as a wrapper
  - return
    - wraps an value
  - bind
    - unwraps the value a
      - operates on a to produce b
      - wraps b
- In Scala operators are slightly different (but equivalent)
  - val s = Seq(a)
  - val s1 = s.flatMap(f)

Monads

- Allow a context to be wrapper around a value and propagate that context as the value is transformed
  - Option
  - Try
  - Future
  - Model side effects
  - IO Monad

Monoid and Endofunctors

- A monoid
  - set closed under associative binary op
  - has an identity element
    - unlike a group elements need not have inverses
- An endofunctor is a functor that maps a category to itself
- A monad is just a monoid in the category of endofunctors

Assignment 4

Assign 4: Water

- Goals
  - Learn more functional programming
  - Solve Puzzle
    - Use Scala immutable collections
- Look at code skeleton (Water)
- ??? throws exception, replace with your code
- running WaterTest.main will produce output

Water

- Step: if A is not empty and B is not full, Pour A to B until first of
  - A is empty
  - B is full
- Find the shortest sequence of steps
  - may be more than one (pick one)
  - may not exist (return None)
- Don’t go back to an earlier state
- Use breadth first search
  - Start with Set(initialState)
  - Iterate: Old:Set(State) => New:Set(State)
    - All new States reachable in 1 step from some old State
    - Exclude any States previously seen from New
  - Stop when target State is seen or New is empty

Assign4 Output Should Look Something Like This

```
*** Test1 [3,5,8] ***
1 [0,0,8]
2 [0,5,3]
3 [3,2,3]
4 [0,2,6]
5 [2,0,6]
6 [2,5,1]
7 [3,4,1]
8 [0,4,4]
*** Test2 [5,11,13,24] ***
1 [0,0,0,24]
2 [5,0,0,19]
3 [5,11,0,8]
4 [0,11,5,8]
5 [0,3,13,8]
6 [5,3,8,8]
7 [0,8,8,8]
*** Test3 [4,5,10,10] ***
1 [0,0,10,10]
2 [0,5,10,5]
3 [4,1,10,5]
4 [0,1,10,9]
5 [1,0,10,9]
6 [1,5,5,9]
7 [4,2,5,9]
8 [0,2,9,9]
9 [2,0,9,9]
10 [2,5,4,9]
11 [4,3,4,9]
12 [3,3,4,10]
*** Test4 [2,2,100,100] ***
Too many steps 51
*** Test5 [2,2,100,100] ***
No Solution
```