Intermediate Scala

Practical Exercises

Lab 5b Functional Design Patterns Part 2b

1. The ***Id Monad*** is a special Monad, ***Id*** stands for Identity. It is designed to provide a context that does not add anything to the values that it encloses. For example,

   ```
   val id1 = Id(3)
   val id2 = Id(2)
   id1.flatMap( i => id2.map ( j => i + j ) )
   ```

   evaluates to **`Id(5)`**

   Provide a definition of this type, Id[T], and define the map and flatMap operations. Provide a definition of the Monad[_] typeclass for the Id Monad as well, and explore the effect of the Monad Transformers discussed in the chapter.

2. Look at the following type definition

   ```
   trait Generator[+T] {
     self => // alias for "this".

     def generate: T

     def map[S](f: T => S): Generator[S] = new Generator[S] {
       def generate = f(self.generate)
     }

     def flatMap[S](f: T => Generator[S]): Generator[S] =
       new Generator[S] {
         def generate = f(self.generate).generate
       }
   }
   ```

   This (monadic) type class is designed to support the generation of random values of the type T. As an example, an instance of this type class to create random Int values can be defined as:

   ```
   val integers = new Generator[Int] {
                    val rand = new java.util.Random
                    def generate = rand.nextInt()
                  }
   ```

Examine how this works… Create an instance of the type that uses this instance to generate random Boolean values.

Now use these two generators to build an instance of the class that generates random (Int, Boolean) tuples. This instance should be capable of generating tuples of any type, subject to suitable other generators being available.

Can you define an instance of this class that generates Lists of random lengths, containing randomly generated Int elements? In other words: Generator[List[Int]]?

3. Consider the method

   `def sum(List[Int])`

   defined to work using one of the fold… methods. Modify the method so that it can add together instances of the **`Option[Int]`**.

   Now change the code so that you can add together instances of the **`Trade`** class, shown in the slides, from a **`List`** containing all the buy or sell orders for a specific **`Stock`** symbol.