Functional Composition with Monads in Kotlin

Jaideep Ganguly, Sc.D.

Contents

CHAPTER 1

Monad

1.1 Typeclass

Monad is a Typeclass

Listing 1.1 – Typeclass.

```
sealed class Monad<out A> {

    object None : Monad<Nothing>()
    data class Value<out A>(val value: A) : Monad<A>()

    // Monad - Apply a function to a wrapped value and return a wrapped
    // value using flatMap (liftM or >>= in Haskell)
    inline infix fun <B> flatMap(f: (A) -> Monad<B>) : Monad<B> =
            when (this) {
                is None  -> this
                is Value -> f(value)
            }
 }
```

1.2 Functions

Listing 1.2 – Functions.

```
import Monad.None
import Monad.Value


fun mysqrt(a: Double) = when {
    a >= 0 -> Monad.Value(kotlin.math.sqrt(a))
    else   -> Monad.None
}

fun mylog(a: Double) = when {
    a > 0  -> Value(kotlin.math.ln(a))
    else   -> None
}

fun myinv(a: Double) = when {
    a >= 0 -> Value(1/a)
    a <= 0 -> Value(1/a)
    else   -> None
}
```

1.3 Composition

Listing 1.3 – Class & Function.

```
import Monad.Value

fun testMonad() {
    var vin : Monad<Double>
    var vout: Monad<Double>

    var listOfFun: List<(Double) -> Monad<Double>> =
            mutableListOf<(Double)->Monad<Double>>(::mysqrt,::mylog)
    listOfFun += ::myinv


    vin = Value(100.0)

    // This is not composition
    var iter = listOfFun.iterator()
    while (iter.hasNext()) {
        println(vin.flatMap(iter.next()))
    }
    println()


    iter = listOfFun.iterator()
    while (iter.hasNext()) {
        vin = vin.flatMap(iter.next())      // Mutating
        println(vin)
    }
    println()

    // composition
    vin = Value(100.0)
    vout = vin.flatMap(::mysqrt).flatMap(::mylog).flatMap(::myinv)
    println(vout)

    // composition with infix
    vout = vin flatMap ::mysqrt flatMap ::mylog flatMap ::myinv
    println(vout)

    vin = Monad.Value(-100.0)
    vout = vin.flatMap(::mysqrt)
    println(vout)


    println(Value(100.00).flatMap(::mysqrt))
    println(Value(-100.00).flatMap(::mysqrt))

    println(Value(1000.0)
            .flatMap(::mysqrt)
            .flatMap(::mysqrt)
```