
FUNCTIONAL COMPOSITION WITH MONADS IN KOTLIN

Jaideep Ganguly, Sc.D.



CONTENTS

Contents i

1	Monad	1
1.1	Typeclass	1
1.2	Functions	1
1.3	Composition	2

Index 3

MONAD

1.1 Typeclass

Monad is a Typeclass

LISTING 1.1 – Typeclass.

```
1 sealed class Monad<out A> {  
2  
3     object None : Monad<Nothing>()  
4     data class Value<out A>(val value: A) : Monad<A>()  
5  
6     // Monad - Apply a function to a wrapped value and return a wrapped  
7     // value using flatMap (liftM or >=> in Haskell)  
8     inline infix fun <B> flatMap(f: (A) -> Monad<B>) : Monad<B> =  
9         when (this) {  
10             is None -> this  
11             is Value -> f(value)  
12         }  
13 }
```

1.2 Functions

LISTING 1.2 – Functions.

```
1 import Monad.None  
2 import Monad.Value  
3  
4  
5 fun mysqrt(a: Double) = when {  
6     a >= 0 -> Monad.Value(kotlin.math.sqrt(a))  
7     else -> Monad.None  
8 }  
9  
10 fun mylog(a: Double) = when {  
11     a > 0 -> Value(kotlin.math.ln(a))  
12     else -> None  
13 }  
14  
15 fun myinv(a: Double) = when {  
16     a >= 0 -> Value(1/a)  
17     a <= 0 -> Value(1/a)  
18     else -> None  
19 }
```

1.3 Composition

LISTING 1.3 – Class & Function.

```

1 import Monad.Value
2
3 fun testMonad() {
4     var vin : Monad<Double>
5     var vout: Monad<Double>
6
7     var listOfFun: List<(Double) -> Monad<Double>> =
8         mutableListOf<(Double)->Monad<Double>> (::mysqrt, ::mylog)
9     listOfFun += ::myinv
10
11
12     vin = Value(100.0)
13
14     // This is not composition
15     var iter = listOfFun.iterator()
16     while (iter.hasNext()) {
17         println(vin.flatMap(iter.next()))
18     }
19     println()
20
21
22     iter = listOfFun.iterator()
23     while (iter.hasNext()) {
24         vin = vin.flatMap(iter.next()) // Mutating
25         println(vin)
26     }
27     println()
28
29     // composition
30     vin = Value(100.0)
31     vout = vin.flatMap(::mysqrt).flatMap(::mylog).flatMap(::myinv)
32     println(vout)
33
34     // composition with infix
35     vout = vin flatMap ::mysqrt flatMap ::mylog flatMap ::myinv
36     println(vout)
37
38     vin = Monad.Value(-100.0)
39     vout = vin.flatMap(::mysqrt)
40     println(vout)
41
42
43     println(Value(100.00).flatMap(::mysqrt))
44     println(Value(-100.00).flatMap(::mysqrt))
45
46     println(Value(1000.0)
47         .flatMap(::mysqrt)
48         .flatMap(::mysqrt))

```

