Functional Composition & Monads in Kotlin

Jaideep Ganguly, Sc.D.

Contents

CHAPTER 1

Monad

1.1 Class & Function

Listing 1.1 – Class & Function.

```
fun testMonad() :  TC<DC>  {
    var inp: TC<DC>

//    var listOfFun: List<(Value) -> TC<Value>> = mutableListOf<(Value)->TC<Value>>()
//    listOfFun += ::mysqrt
//    listOfFun += ::mylog
//    listOfFun += ::myinv

    var dc = DC(100.0,"NOTOK")
    inp = TC.Value(dc)
//    inp = execute(inp, listOfFun) as TC.Value<Value>

    println(inp.value.data)
    println(inp.value.rem)

    var out = inp.flatMap(::mysqrt).flatMap(::myinv) as TC.Value
//    var out = inp flatMap ::mysqrt flatMap ::myinv
    println(out.value.data)
    println(out.value.rem)
//
//    var dci: DCI = DCI("Hello")
//    inp.data.DCI = dci

    return(inp)

}

fun <T> execute(input: TC<T>, fns: List<(T) -> TC<T>>): TC<T> =
        fns.fold(input) { inp, fn -> inp.flatMap(fn) }

fun main(args: Array<String>) {
    testMonad()

/*
    var listOfFun: List<(Value) -> TC<Value>> = mutableListOf<(Value) -> TC<Value>>()
    listOfFun += ::mysqrt
    listOfFun += ::mylog
    listOfFun += ::myinv

    var inp1: TC<Value> = TC.Value(Value("Jaideep", 100.0))
    var inp2: TC<Value> = TC.Value(Value("Jaideep", 200.0))

    runBlocking {
        val startTime = System.currentTimeMillis()
        val deferred1 = async { execute(inp1, listOfFun) }
        val value1 = deferred1.await() as TC.Value<Value>

        val deferred2 = async { execute(inp2, listOfFun) }
```