Jim Duey
Intensive Systems Consulting, Inc.

@jimduey
www.intensivesystems.net

Prelude

- Functional Programming
  - Function composition (point free)
- Clojure
  - http://clojure.org

Why Monads

- Alan Kay: Programming and Scaling
  - The importance of DSL's

- Monads let you write DSL's
  - clean
  - elegant
  - usable

Function Composition

New functions from existing ones:

(def new-fn (comp fn1 fn2))

is equivalent to:

```
(def new-fn
     (fn [x]
          (fn1 (fn2 x))))
```

Function Composition

```
(defn double [x]
      (* 2 x))

(def inc-dbl (comp double inc))

> (inc-dbl 2)
6
```

Function Composition

```
(def new-fn (comp fn1 fn2))
```

only works if fn2 produces a result that fn1 expects.

Function Composition

(def bad-fn (comp double list))

what does that even mean?

Function Signatures

```
(defn double [x]
      (* 2 x))
```

It's signature is that it expects a single number and returns a single number.

Suppose ...

You had a number of functions that expected a value of any kind and returned a list of values.

```
(defn half-double [x]
    (list (/ x 2) (* x 2)))

(defn increase [x]
    (list (inc x) (+ 2 x)))
```

```
(def value (half-double 4)

> value
(2 8)
```

But we can't pass 'value' to the 'increase' function.

```
> (increase value)
java.lang.ClassCastException
```

Applicator needed

We need a function that will take ‘value’ and do the right thing with it to call ‘increase’:

```
> (mapcat increase value)
(3 4 9 10)
```

```
(defn m-bind [v f]
    (mapcat f v))
```

Another problem

But what if we want to call ‘m-bind’ with a single value?

> (m-bind 4 increase)
java.lang.IllegalArgument

> (m-bind (list 4) increase)
(5 6)

The Sequence Monad

```
(def m-result list)

(defn m-bind [v f]
      (mapcat f v))
```

```
> (m-bind (m-result 4) increase)
(5 6)
```

The Sequence Monad

```
(def m-result list)

(defn m-bind [v f]
  (mapcat f v))
```

The 3 Monad Laws

The Sequence Monad

```
(def m-result list)

(defn m-bind [v f]
      (mapcat f v))
```

monadic values: () (4) (8 2 14)

monadic functions: increase, half-double

So what?

Function composition of monadic functions:

Instead of:

```
(def bad-fn
    (comp increase half-double))
```

you can do:

```
(def new-monadic-fn
    (m-chain [half-double increase]))
```

Suppose ...

You had a number of functions that expected a value of any kind and returned a set of values.

```
(defn half-double [x]
    (hash-set (/ x 2) (* x 2)))

(defn increase [x]
    (hash-set (+ 1 x) (+ 2 x)))
```

The Set Monad

```
(def m-result hash-set)

(defn m-bind [v f]
    (apply union (map f v)))
```

monadic values: #{} #{4} #{8 2 14}

monadic functions: increase, half-double

So what?

Function composition of monadic functions

Comprehensions

```
> (for
      [a (list 1 2 3)
       b (list 10 20 30)]
    (+ a b))


(11 12 13 21 22 23 31 32 33)
```

So what?

Function composition of monadic functions

Comprehensions

```
> (domonad sequence-m
      [a (list 1 2 3)
       b (list 10 20 30)]
    (+ a b))
```

(11 12 13 21 22 23 31 32 33)

```
> (for   ;; list monad
      [a (half-double 16)
       b (half-double a)]
    (+ b 1))
```

```
(5 17 17 65)
```

```
> (let
      [a (double 16)
       b (double a)]
    (+ b 1))
```

```
65
```

```
> (domonad sequence-m
      [a (half-double 16)
       b (half-double a)]
    (+ b 1))
```

```
(5 17 17 65)
```

```
> (domonad set-m
      [a (half-double 16)
       b (half-double a)]
    (+ b 1))
```

```
#{5 17 65}
```

```
> (for   ;; list monad
      [a (half-double 16)
       b (half-double a)]
    (+ b 1))
```

```
(5 17 17 65)
```

```
> (let  ;; identity monad
      [a (double 16)
       b (double a)]
    (+ b 1))
```

```
65
```

```
> (domonad sequence-m
      [a (half-double 16)
       b (half-double a)]
    (+ b 1))
```

```
(5 17 17 65)
```

```
> (domonad set-m
      [a (half-double 16)
       b (half-double a)]
    (+ b 1))
```

```
#{5 17 65}
```

The Identity Monad

```
(def m-result identity)

(defn m-bind [v f]
    (f v))
```

Something Random

Suppose you had a function to generate a pseudo-random number given a seed.

```
(defn rand [seed]
    ...
    [rnd new-seed])
```

and you wanted to generate a triple of random numbers from a given seed.

Something Random

```
(defn rand-triple [seed]
    (let
        [[r1 s1] (rand seed)
         [r2 s2] (rand s1)
         [r3 s3] (rand s2)]
      (list r1 r2 r3)))
```

Something Random

```
(defn rand-triple [seed]
    (let
        [[r1 s1] (rand seed)
         [r2 s2] (rand s1)
         [r3 s3] (rand s2)]
        [(list r1 r2 r3) s3]))
```

Something Random

```
(def rand-triple
     (domonad state-m
       [r1 rand
        r2 rand
        r3 rand]
       (list r1 r2 r3)))
```

```
> (def sum-list
    (domonad sequence-m
          [a [2 9 1]
           b [1 4]]
        (+ a b)))
```

```
> sum-list
(3 6 10 13 2 5)
```

The lists [2 9 1], [1 4] and ‘sum-list’ are monadic values for the sequence monad.

```
> (def rand-triple
      (domonad state-m
            [r1 rand
             r2 rand]
            (list r1 r2)))
```

```
> (rand-triple seed)
(32 7)
```

In the same way, the ‘rand’ and ‘rand-triple’ functions are monadic values for the state monad.

The State Monad

```
(defn m-result [x]
      (fn [state]
            [x state]))
```

The State Monad

```
(defn m-result [x]
      (fn [state]
            [x state]))

(defn m-bind [mv f]
            (fn [s]
                  (let [[v ss] (mv s)]
                        ((f v) ss))))
```

The State Monad

```
(defn m-result [x]
    (fn [state]
        [x state]))

(defn m-bind [mv f]
        (fn [s]
            (let [[v ss] (mv s)]
              ((f v) ss))))
```

monadic values: functions that take a state and return a value and a new state.

Testing Ring Handlers

- Make a sequence of requests to various routes
- Perform tests on the responses
- Return cookies to the handler on subsequent requests

You need a DSL to script a session with the handler.

That will maintain state between requests.

Simple Ring Handler

```
(def counter (atom 0))

(defn home []
  (swap! counter + 1)
  {:status  200
   :headers {"Content-Type"
             "text/html"}
   :body (str "counter: "
              @counter)})
```

```
(defroutes main-routes
          (GET "/" []
               (home))
          (route/not-found
                  “Not found"))

(def app
  (handler/site main-routes))
```

Scripter

```
(deftest test-not-found-route
        (script app
                [_ (click "/wrong")
                 _ (response-code? 404)]))
```

```
(defmacro script
  [handler & steps]
  `((domonad script-m
      ~@steps
      true)
    {:handler ~handler}))
```

```
(deftest test-not-found-route
   ((domonad script-m
        [_ (click "/wrong")
         _ (response-code? 404)]
        true)
    {:handler app}))
```

Scripter

```
(deftest test-counter
        (script app
                [_ (click "/")
                 _ (response-code? 200)
                 _ (body-contains? "counter: 1")

                 _ (click "/")
                 _ (response-code? 200)
                 _ (body-contains? "counter: 2")]))
```

Scripter

```
(def hit-counter
      (as-script
            [_ (click "/")
             _ (response-code? 200)]
           true))
```

```
(def hit-counter
      (domonad script-m
            [_ (click "/")
             _ (response-code? 200)]
           true))
```

Scripter

```
(def hit-counter
        (as-script
                [_ (click "/")
                 _ (response-code? 200)]
                true))

(deftest test-counter
        (script app
                [_ hit-counter
                 _ (body-contains? "counter: 1")

                 _ hit-counter
                 _ (body-contains? "counter: 2")]))
```

Questions

http://intensivesystems.net/writings.html

https://github.com/jduey/appraiser