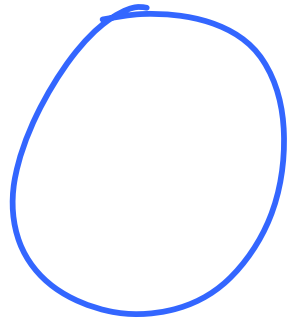


ADT

fields / methods

private

dynamic dispatch

ds assoc/ w/ some fns

bundles of data

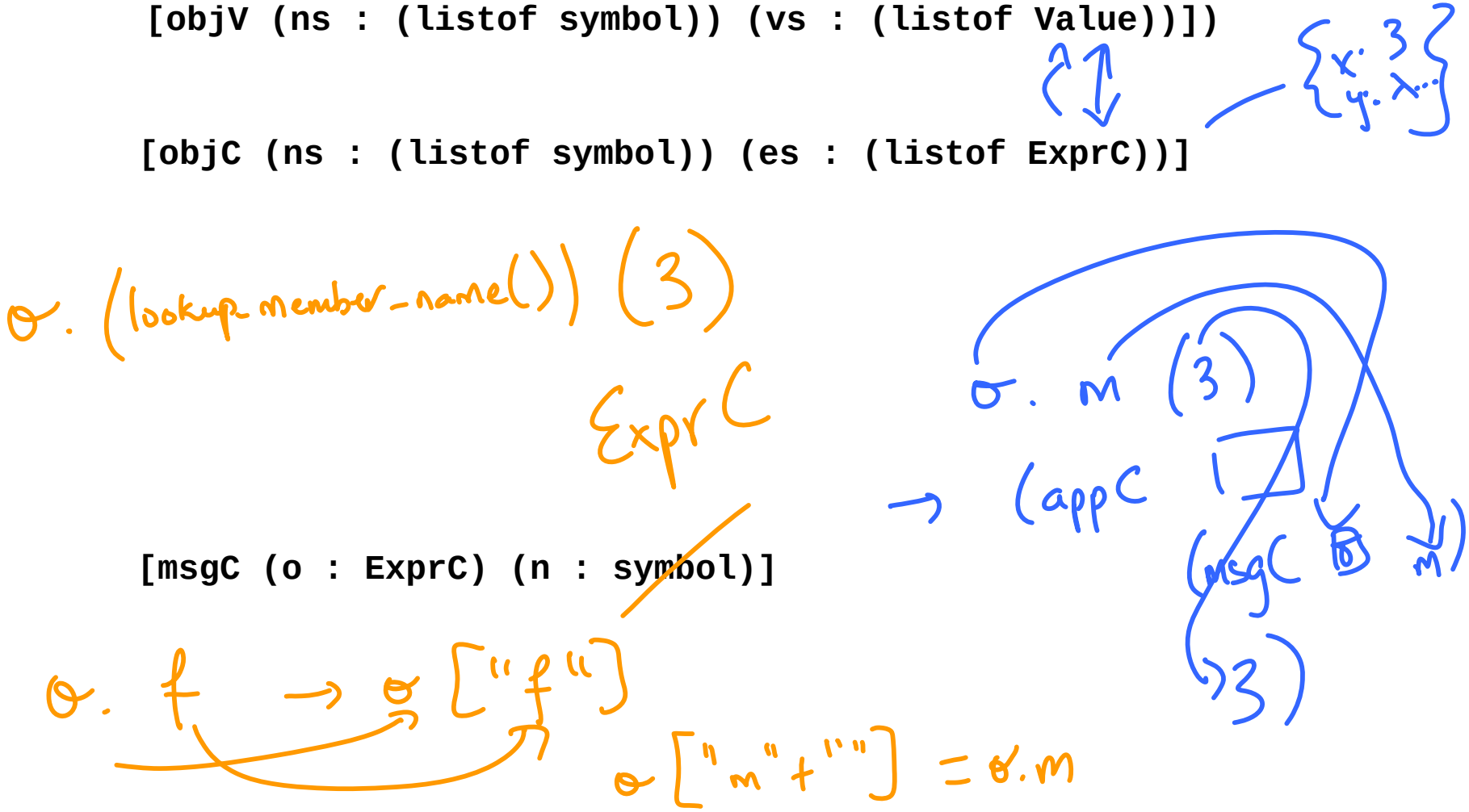
collection of state &
a way to change it

interfaces / abst defs

```
(define-type Value
  [numV (n : number)]
  [closV (arg : symbol) (body : ExprC) (env : Env)]
  [objV (ns : (listof symbol)) (vs : (listof Value))])

  [objC (ns : (listof symbol)) (es : (listof ExprC))]

  [msgC (o : ExprC) (n : symbol)]
```



(δ o (x (n)

(δ (msg o n a)
((o n) a))

(case m
[(add1) (λ (n) (+ n 1))]
[(sub1) (λ (n) (- n 1))]))

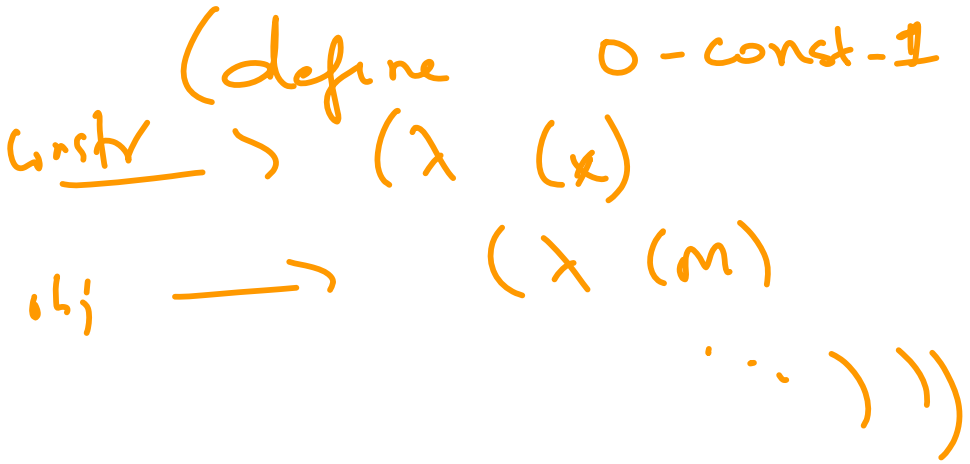
((o 'add1) 3) → 4

(msg o 'add1 3) → 4

```
(define o-1
  (lambda (m)
    (case m
      [(add1) (lambda (x) (+ x 1))]
      [(sub1) (lambda (x) (- x 1))])))
```

Constructors

```
(define (o-constr-1 x)
  (lambda (m)
    (case m
      [(addX) (lambda (y) (+ x y))])))
```



State

```
(define (o-state-1 count)
  (lambda (m)
    (case m
      [(inc) (lambda () (set! count (+ count 1)))]
      [(dec) (lambda () (set! count (- count 1)))]
      [(get) (lambda () count)])))
```

Static Members

```
(define o-static-1
  (let ([counter 0])
    (lambda (amount)
      (begin
        (set! counter (+ 1 counter))
        (lambda (m)
          (case m
            [(inc) (lambda (n) (set! amount (+ amount n)))]
            [(dec) (lambda (n) (set! amount (- amount n)))]
            [(get) (lambda () amount)]
            [(count) (lambda () counter)])))))))
```

Self-Reference

```
(define o-self!
  (let ([self 'dummy])
    (begin
      (set! self
            (lambda (m)
              (case m
                [(first) (lambda (x) (msg self 'second (+ x 1)))]
                [(second) (lambda (x) (+ x 1))])))
      self)))
```

(λ (self x)

(δ (msg o m a)

((o m) o a))