OBJECT ORIENTED PROGRAMMING

```
(define b
  (define b
    (lambda (mss val)
      (let ([x 0])
        (cond
          [(eq? msg show) x]
          [(eq? msg che)
            (let ([v (first val)]])
            (set! x (+ x v)))]
          [else (error !a "message not
            understood ~a"" msg)] ))))
```

```
> (b 1 show)
0
> (b 1 incs)
> (b 1 show)
0
```

how to do recersion with set!

```
> (a 'inc 10)
> (a 'show')
10'
```

show-field stout.

Object-save like environments
identifier -> value

inheritance - composition of environments.
~~in the~~

in the example, show, deposit, withdraw - balance
- lexical binding

three closures!

Objects are implanted as procedures.

```
(define make-ground-obj
   (λ ()
      (let ([table (make-hash)])
        (λ msg
          (match msg
            [(list (? symbol? rej ))]
              (hash-ref table key
    (λ ()
       (error 'obj: key not fond ~a" key))))]
    [ (list (? symbol ky) val)
       (hash-set (table ky val)]
    [else (error (make-ground-obj' " ... " )))))
```

```
> (define o (make-ground-obj))

> (o 'x 25);     o = {x ↦ 25}

> (o 'y 30);     o = {x ↦ 25
                      y ↦ 30}
```

```
> (o 'x 10) ;        O = {x ↦ 10, y ↦ 30}

    = (extend-env   x   10   o)

      (object-just made).

> (o 'z)   → error

> (o 'z 7) ;   {x ↦ 10,  y ↦ 30,  z ↦ z}
```

Objects are enironments.

METHODS

```
① (define showBalance
     (λ (self)
        (self 'balance)))
```

in Javascript.

```
var showbalance = function () {
    return this.balance;
}.
```

this -> key reform to an identifier -> objet

```
(define a (make-ground-obj))
   (define show-balance
      (λ (this)
          (this 'balance)))
   (define deposit
    (λ (this)
          ...))
   (define withdraw
      (λ (this)
          ....))

> (a 'balance 0)

> (a 'balance)

> (show-balance a)

> (a 'show-balance show-balance),

      ; a = { balance ↦ 0, show-balance ↦ c }.

> (a 'deposit deposit)

> (a 'withdraw withdraw)
```

methods - no common state.

$a = \{$ balance ↦ 0,
show-balance ↦ c_1,
deposit ↦ c_2,
withdraw ↦ c_3 $\}$

no free identifiers in the methods

```
> (define b (make-object))
> (b 'balance 500)
```

objects with methods should have a common state.

```
> ((a 'deposit) a)

> ((a 'deposit ) a 400) ; method call.

> ((a 'deposit ) b 500)

> (methodcall a 'deposit 400); a.deposit (400).
```