Simulating Objects in Racket

We can simulate object-oriented programming in Racket using a combination of lexical scoping and side effects.

First, let's see how functions can share state.

```
;; define the variables incr and get, and just give them null as
;; a value for now (it will be reset later)
(define incr  '())
(define get '())

(let ((n 0))
   (set! incr (lambda (i) (set! n (+ n i))))
   (set! get (lambda () n)))
```

The variables `incr` and `get` are global variables, now bound to functions. The variable `n` is shared by them, but is hidden -- it is *not* a global variable.

Now try evaluating the following expressions:

```
(get)

(incr 10)
(get)

(incr 100)
(get)
```

Now we can add dispatching to simulate a simple object. (This example is adapted from *Structure and Interpretation of Computer Programs*.) Here we define a bank account object, with a field `my-balance`, and methods `balance`, `deposit`, and `withdraw`.

This example is also available as a separate web page bankaccount.rkt.

```
(define (make-account)
   (let ((my-balance 0))

      ;; return the current balance
      (define (balance)
         my-balance)

      ;; make a withdrawal
     (define (withdraw amount)
        (if (>= my-balance amount)
           (begin (set! my-balance (- my-balance amount))
                  my-balance)
           "Insufficient funds"))

     ;; make a deposit
     (define (deposit amount)
        (set! my-balance (+ my-balance amount))
        my-balance)

     ;; the dispatching function -- decide what to do with the request
     (define (dispatch m)
        (cond ((eq? m 'balance) balance)
              ((eq? m 'withdraw) withdraw)
              ((eq? m 'deposit) deposit)
              (else (error "Unknown request -- MAKE-ACCOUNT"  m))))

      dispatch))
```

Note that the variable my-balance is local to the `make-account` function.

Using the account:

```
(define acct1 (make-account))
(define acct2 (make-account))
((acct1 'balance))               => 0
((acct1 'deposit) 100)           => 100
((acct1 'withdraw) 30)           => 70
((acct1 'withdraw) 200)          => "Insufficient funds"
```

```
;; acct2 is a different account from acct1!
((acct2 'balance))               => 0
```