CS 61A -- Lab 8 solutions

LAB ACTIVITIES:

1. Use a LET to keep both initial and current balance

```
(define (make-account init-amount)
 (let ((BALANCE INIT-AMOUNT))               ;;;  This is the change.
    (define (withdraw amount)
     (set! balance (- balance amount)) balance)
    (define (deposit amount)
     (set! balance (+ balance amount)) balance)
    (define (dispatch msg)
     (cond
       ((eq? msg 'withdraw) withdraw)
       ((eq? msg 'deposit) deposit)))
    dispatch))
```

2. Add messages to read those variables.

```
(define (make-account init-amount)
 (let ((balance init-amount))
    (define (withdraw amount)
     (set! balance (- balance amount)) balance)
    (define (deposit amount)
     (set! balance (+ balance amount)) balance)
    (define (dispatch msg)
     (cond
       ((eq? msg 'withdraw) withdraw)
       ((eq? msg 'deposit) deposit)
       ((EQ? MSG 'BALANCE) BALANCE)                ;; two lines added here
       ((EQ? MSG 'INIT-BALANCE) INIT-AMOUNT)))
    dispatch))
```

3. Add transaction history.

```
(define (make-account init-amount)
 (let ((balance init-amount)
       (TRANSACTIONS '()))                    ;; add local state var
    (define (withdraw amount)
     (SET! TRANSACTIONS (APPEND TRANSACTIONS
                             (LIST (LIST 'WITHDRAW AMOUNT))))    ;; update
     (set! balance (- balance amount)) balance)
    (define (deposit amount)
     (SET! TRANSACTIONS (APPEND TRANSACTIONS
                              (LIST (LIST 'DEPOSIT AMOUNT))))     ;; update
     (set! balance (+ balance amount)) balance)
    (define (dispatch msg)
     (cond
       ((eq? msg 'withdraw) withdraw)
       ((eq? msg 'deposit) deposit)
       ((eq? msg 'balance) balance)
       ((eq? msg 'init-balance) init-amount)
       ((EQ? MSG 'TRANSACTIONS) TRANSACTIONS)))      ;; message to examine it
    dispatch))
```

4. Why substitution doesn't work.

```
(plus1 5)  becomes

(set! 5 (+ 5 1))
5
```

The first line (the SET!) is syntactically wrong; "5" isn't a variable
and it doesn't make sense to substitute into an unevaluated part of a
special form. (Remember, SET! has the exact same syntax as DEFINE.)

The second line (returning the value 5) is syntactically okay but
gives the wrong answer; it ignores the fact that the SET! was supposed
to change the result.

The rest of the lab was about trying various uses of LAMBDA to build up
to an object-oriented system. There was one key thing to notice here:
the implementation of ASK.

```
(define (ask obj msg . args)
 (apply (obj msg) args) )
```

Will this work for the accounts from questions 1 and 2?