Environment Sketches

1. Sketch the following, assuming the (+ bar baz) was evaluated.

   (define foo
     (lambda (bar)
       ((lambda (baz) (+ bar baz))
        ((lambda (yucky) (+ 4 yucky)) 2)))))

2. Sketch make-root-object assuming the (lambda (msg) was evaluated.

3. Sketch make-named-object assuming the (lambda (msg) was evaluated.

4. Sketch make-instance assuming the (lambda (msg) was evaluated.

5. Link these sketches together as if create-named-object had been called.

6. Add to your sketch the frames that would be generated by a call to the NAME method. Only add frames which would hang under frames already in the diagram.

7. Add to your sketch the frames that would generated by a call to the IS-A method.
Parent Trap

Original code:

```scheme
(define (make-foo self name location)
  (let ((mobile-part (make-mobile-thing self name location)))
    (make-handler
      'foo
      (make-methods
       'INSTALL
       (lambda ()
         (ask mobile-part 'INSTALL)
         (ask self 'EMIT (list (ask self 'NAME)
                               "born in" (ask self 'LOCATION)))))
    mobile-part)))
```

New code, which calls `make-mobile-thing` everywhere `mobile-part` was before:

```scheme
(define (make-foo self name location)
  (make-handler
    'foo
    (make-methods
     'INSTALL
     (lambda ()
       (ask (make-mobile-thing self name location) 'INSTALL)
       (ask self 'EMIT (list (ask self 'NAME)
                               "born in" (ask self 'LOCATION)))))
    (lambda (msg)
      ((make-mobile-thing self name location) msg))))
```

8. What is the difference between the two versions for the following method calls:

```scheme
(define f (create-foo 'x gates))
(ask f 'LOCATION)
(ask f 'CHANGE-LOCATION dreyfoos)
(ask f 'LOCATION)
```

9. What would happen if all the `make-mobile-thing` expressions in the new version were replaced with `create-mobile-thing`?
Specialization

(define *all-stations* '(("I am the very model of a software engineer")
                           ("Breaking news: Carry Cdr assaulted by Lord Vandimort")))

(define (make-radio self)
  (let ((root-part (make-root-object self))
        (station 0))
    (make-handler 'radio
      (make-methods 'INSTALL
        (lambda ()
          (ask clock 'ADD-CALLBACK
            (create-clock-callback 'pass-time self 'TIMESTEP)))
        'TIMESTEP
        (lambda ()
          (ask self 'PLAY (list-ref *all-stations* station)))
        'PLAY
        (lambda (tune)
          (display-message (cons "Tunes emit from the radio: " tune)))
        'SET-STATION!
        (lambda (stat)
          (set! station stat)))
    root-part))

(define (make-boombox self)
  (let ((radio-part (make-radio self))
        (cd '())
        (current-track 0))
    (make-handler 'boombox
      (make-methods 'TIMESTEP
        (lambda ()
          (cond ((null? cd)
                      (ask radio-part 'TIMESTEP))
                ((< current-track 10)
                (ask self 'PLAY (list-ref cd current-track))
                (set! current-track (+ current-track 1)))
                (else
                 (set! cd '())
                 (set! current-track 0)
                 (ask self 'PLAY (list "Click! CD over!")'))))
      'PLAY
      (lambda (tune)
        (display-message (cons "Tunes blare from the boombox:" tune)))
      'PLAY-CD
      (lambda (the-cd)
        (set! cd the-cd))
    radio-part)))
(define r (create-radio))
(ask clock 'TICK) ; expr 1
(define r (create-boombox))
(ask clock 'TICK) ; expr 2

10. What is printed out for expression 1?

11. What is printed out for expression 2?