Object-Oriented System - Version 1

(define (make-speaker name)
  (lambda (message)
    (case message
      ((NAME) (lambda () name))
      ((CHANGE-NAME)
        (lambda (new-name) (set! name new-name)))
      ((SAY)
        (lambda (list-of-stuff)
          (if (not (null? list-of-stuff))
              (display-message list-of-stuff)
              'NUF-SAID))
        (else (no-method))))))

Abstract out retrieval of method from the object (given the message)...

(define (get-method message object)
  (object message))

(define (ask object message . args)
  (let ((method (get-method message object)))
    (if (method? method)
        (apply method args)
        (error "No method for message" message))))

(define (no-method) '(NO-METHOD))

(define (method? x)
  (cond ((procedure? x) #)
    ((eq? x (no-method)) #f)
    (else (error "Object returned non-message" x)))))
Object-Oriented System - Version 2

What if we want a speaker to call its own method?

Problem: no access to the "object" from inside itself! Solution: add explicit "self" argument to all methods

```scheme
(define (make-speaker name)
  (lambda (message)
    (case message
      ((NAME) (lambda (self) name))
      ((CHANGE-NAME)
        (lambda (self new-name)
          (set! name new-name)
          (ask self 'SAY (list 'call 'me name))))
      ((SAY)
        (lambda (self list-of-stuff)
          (if (not (null? list-of-stuff))
            (display-message list-of-stuff)
            'NUF-SAID))
        (else (no-method))))))

(define (ask object message . args)
  (let ((method (get-method message object)))
    (if (method? method)
      (apply method object args)
      (error "No method for message" message))))

(ask p 'CHANGE-NAME 'fred)
Call me fred

A Specialized Speaker (Subclass)

Want lecturers to be a kind of speaker - that inherit the behavior of speakers but add to that behavior:

```scheme
(define (make-lecturer name)
  (let ((speaker (make-speaker name)))
    (lambda (message)
      (case message
        ((LECTURE)
          (lambda (self stuff)
            (delegate speaker self
              'SAY '(Good Morning!))
            (delegate speaker self 'SAY stuff))
          (else (get-method message speaker))))))

(define d (make-lecturer 'Duane))
(ask d 'LECTURE '(Today we learn more))
Good Morning!
Today we learn more
Approach: Inheritance by Delegation

- Inherit behavior by adding an "internal" speaker
  - Get internal object to act on behalf of object by delegation
- If message is not recognized, pass the buck
- Can change or specialize behavior:
  - Add new methods
  - Change operation of methods

Another Subclass

Want a "Canadian Lecturer" that changes the basic way of talking: append "Eh?" to everything he says...

```scheme
(define (make-canadian-lecturer name)
  (let ((lecturer (make-lecturer name)))
    (lambda (message)
      (case message
        ((SAY)
          (lambda (self stuff)
            (delegate lecturer self
              'SAY (append stuff '(Eh?))))))
        (else (get-method message lecturer))))

(define (delegate to from message . args)
  (let ((method (get-method message to)))
    (if (method? method)
      (apply method from args)
      (error "No method" message))))

(define (ask object message . args)
  (apply delegate object object message args))

(define e (make-canadian-lecturer 'Eric))
(define (get-method message preferred . others)
  (define (loop objs)
    (let ((method (get-method-from-object message (car objs)))
      (rest (cdr objs)))
      (if (or (method? method) (null? rest))
        method
        (loop rest))))

(define (get-method-from-object message object)
  (object message))
```
Alternative Multiple Inheritance

We have lots of flexibility - suppose we want to pass the message on to multiple internal objects (not just some "preferred" one)?

\[
\begin{align*}
\text{(define eric} \\
\text{  (let ((comic (make-comic))} \\
\text{    (lecturer (make-canadian-lecturer 'Eric))))} \\
\text{  (lambda (message) } \\
\text{    (lambda (self . args) } \\
\text{      (apply delegate-to-all } \\
\text{        (list lecturer comic) } \\
\text{        self } \\
\text{        args))))))
\end{align*}
\]

(ask eric 'SAY '(The sky is blue))
The sky is blue Eh?
The sky is blue ha ha

\[
\begin{align*}
\text{(define (delegate-to-all to-list from message . args) } \\
\text{  (foreach} \\
\text{    (lambda (to-whom) } \\
\text{      (apply delegate to-whom from message args)) } \\
\text{  to-list)}
\end{align*}
\]