#lang racket

(define (no-method name)
  (list 'no-method name))

(define (no-method? x)
  (if (pair? x)
      (eq? (car x) 'no-method)
      false))

(define (method? x)
  (not (no-method? x)))

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

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

(define (make-speaker)
  (define (self message)
    (cond ((eq? message 'say)
           (lambda (stuff) (display stuff)))
          (else (no-method message))))
  self)

(define (make-lecturer)
  (let ((speaker (make-speaker)))
    (define (self message)
      (cond ((eq? message 'lecture)
             (lambda (stuff)
                (ask self 'say stuff)
                (ask self 'say '(you should be taking notes))))
             (else (get-method speaker message))))
    self))

(define (make-arrogant-lecturer)
  (let ((lecturer (make-lecturer))
        (define (self message)
          (cond ((eq? message 'say)
                 (lambda (stuff)
                   (ask lecturer 'say
                     (append '(it is obvious that) stuff))))
                 (else (get-method lecturer message))))
          self))

(define s (make-speaker))
;(ask s 'say 'hello)
(define l (make-lecturer))
(define al (make-arrogant-lecturer))
#lang racket

(define (no-method name)  
  (list 'no-method name))

(define (no-method? x)  
  (if (pair? x)  
      (eq? (car x) 'no-method)  
      false))

(define (method? x)  
  (not (no-method? x)))

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

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

(define (make-speaker)  
  (define (self message)  
    (cond ((eq? message 'say)  
            (lambda (self stuff) (display stuff)))  
          (else (no-method message))))
  self)

(define (make-lecturer)  
  (let ((speaker (make-speaker)))  
    (define (self message)  
      (cond ((eq? message 'lecture)  
              (lambda (self stuff)  
                (ask self 'say stuff)  
                (ask self 'say '(you should be taking notes))))  
             (else (get-method speaker message))))
    self))

(define (make-arrogant-lecturer)  
  (let ((lecturer (make-lecturer)))  
    (define (self message)  
      (cond ((eq? message 'say)  
              (lambda (self stuff)  
                (ask lecturer 'say  
                     (append '(it is obvious that) stuff))))  
             (else (get-method lecturer message))))
    self))

(define s (make-speaker))  
(define l (make-lecturer))  
(define al (make-arrogant-lecturer))