;; Dan Grossman, CSE341, Programming Languages
;; Lecture 21: Dynamic Dispatch Precisely, and Manually in Racket

#lang racket

;; We can "use" dynamic dispatch in a language without it manually

;; Our "objects" will have:
;; * an immutable list of mutable "fields" (symbols and contents)
;; * an immutable list of immutable "methods" (symbols and functions taking sel f)
(struct obj {fields methods})

;; like assoc but for an immutable list of mutable pairs
(define (assoc-m v xs)
  (cond [(null? xs) #f]
        [(equal? v (mcar (car xs))) (car xs)]
        [(#t (assoc-m v (cdr xs)))]))

(define (get obj fld)
  (let ((pr (assoc-m fld (obj-fields obj)))
        (error "field not found")))

(define (set obj fld v)
  (let ((pr (assoc-m fld (obj-fields obj)))
        (error "field not found")))

(define (send obj msg . args) ; convenience: multi-argument functions (2+ arguments)
  (let ((pr (assoc msg (obj-methods obj)))
        (error "method not found" msg)))

(define (make-point _x _y)
  (list (cons 'x _x)
        (cons 'y _y)))

(define (make-color-point _x _y _c)
  (let ([pt (make-point _x _y)])
    (cons (mcons 'color _c)
          (cons (mcons 'theta _th)
                (object-fields pt)))))

(define (make-polar-point _r _th)
  (let ([pt (make-point _r _th)])
    (append (list (cons 'r _r)
                  (cons 'theta _th))
            (object-fields pt)) ; Java-style field extension
    ; overriding by being earlier in the list (see send function)