#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)
   ((eqv? v (mcar (car xs))) (car xs))
   (else (assoc-m v (cdr xs))))

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

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

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

(define (make-point _x _y)
  (list (mcons _x _x)
         (mcons _y _y))

(define (make-color-point _x _y _c)
  (let ([pt (make-point _x _y)])
    (ob)
    (append (list (cons (mcons 'color _c)
                        (obj-fields pt))
                 (append (list (cons 'get-color (lambda (self args) (get self 'color))
                                (cons 'set-color (lambda (self args) (set self 'color (car args))))
                                (cons 'distToOrigin (lambda (self args)
                                                      (let ([a (send self 'get-x)]
                                                            [b (send self 'get-y)]
                                                            ([sqrt (+ (* a a) (* b b))]))) (send self 'set-r-theta r theta))))
                 (cons 'get-y (lambda (self args)
                                [theta (atan b a)]
                                [r (sqrt (+ (* a a) (* b b)))]
                                (send self 'set-r-theta r theta))))
                 (cons 'get-x (lambda (self args)
                                [theta (atan b a)]
                                [r (sqrt (+ (* a a) (* b b)))]
                                (send self 'set-r-theta r theta))))
                 (cons (mcons 'theta _th)
                     (obj-fields pt)) ; Java-style field extension
                 (append ; overriding by being earlier in the list (see send function)