define (class? obj) (tagged-list? obj 'class))

define (class-type class) (second class))

define (class-state class) (third class))

define (class-parent class) (fourth class))

define (class-methods class) (fifth class))

define (make-class type state parent methods) (list 'class type state parent methods))

define (make-methods . args) (define (mhelper lst result) (cond ((null? lst) result) (error "unmatched method (name,proc) pair") ((null? (cdr lst)) (error "invalid method name" (car lst))) (error "invalid method procedure" (cadr lst))) (mhelper args '()))

define (find-class-method methodname class) (let ((result (assq methodname (class-methods class)))) (if result (second result) #f))

define (instance? obj) (tagged-list? obj 'instance))

define (instance-state inst) (second inst))

define (instance-class inst) (third inst))

define (make-instance class . args) (let (inst (list 'instance (map (lambda [x] (list x #f)) (collect-state inst)))) (if (has-method? inst 'CONSTRUCTOR) ;; if it has a constructor, invoke it (apply invoke inst 'CONSTRUCTOR args) (void)) inst))

define (is-a type) (lambda (obj) (if memq type (collect-type (instance-class obj))) #t #f))