```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;OBJECT-ORIENTED PROGRAMMING
;;;
;;;  code from George Lugar and William Stubblefield
;;;    Artificial Intelligence and the Design of Expert Systems
;;;    Benjamin/Cummings, 1989

;;;define a new object, given its parent and an optional list
;;; of instance variables and bindings
(defun def-object (obj parent &optional vars)
  (setf (get obj 'ISA) parent)
  (setf (get obj 'VARIABLES) (evaluate-bindings vars)))

(defun evaluate-bindings (vars)
  (mapcar #'bind-eval-fn vars))

(defun bind-eval-fn (x)
  (list (first x) (eval (second x))))

;;;attach methods to an object
(defun def-method (obj name def)
  (setf (get obj 'METHODS)
        (replace-method name def (get obj 'METHODS))))

(defun replace-method (name def method-lst)
  (cond ((null method-lst)
         (acons name def nil))
        ((equal name (first (first method-lst)))
         (acons name def (rest method-lst)))
        (T (cons (first method-lst)
                 (replace-method name def (rest method-lst))))))

;;;search inheritance hierarchy
(defun inherit-method (obj method)
  (cond ((null obj) nil)
        ((atom obj)
         (or (get-method obj method)
             (inherit-method (get obj 'ISA) method)))
        (T (or (inherit-method (first obj) method)
               (inherit-method (rest obj) method)))))

(defun get-method (obj name)
  (cdr (assoc name (get obj 'METHODS))))

;;;evaluate a message to an object
(defun send (obj method &rest method-parameters)
  (let ((meth (inherit-method obj method))
        env)
    (cond (meth
           (setq env (build-env obj))
           (progv (cons 'SELF (mapcar #'first env))
                  (cons obj (mapcar #'second env))
             (apply meth method-parameters)))
```

```
          (T (print (list 'Unknown 'method method))))))

;;;walks the inheritance hierarchy, depth-first, constructing
;;; a list of inherited variable-binding pairs.
(defun build-env (obj)
  (cond ((null obj) nil)
        ((listp obj)
         (append (build-env (rest obj)) (build-env (first obj))))
        (T (append (build-env (get obj 'ISA))
                   (get obj 'VARIABLES)))))

(defun build-root ()
  (def-object 'ROOT nil)
  (def-method 'ROOT 'SHOW
    #'(lambda ()
        (terpri)
        (print (list self 'has 'parents))
        (pprint (get self 'ISA))
        (terpri)
        (print (list self 'has 'attached 'variables))
        (pprint (get self 'VARIABLES))
        (terpri)
        (print (list self 'has 'attached 'methods))
        (pprint (get self 'METHODS))
        (terpri)))
  (def-method 'ROOT 'SHOW-PARENTS
    #'(lambda ()
        (get self 'ISA)))
  (def-method 'ROOT 'SHOW-VALUE
    #'(lambda (name)
        (eval name)))
  (def-method 'ROOT 'SHOW-ENV
    #'(lambda ()
        (build-env self)))
  (def-method 'ROOT 'SET-VALUE
    #'(lambda (var value)
        (let ((pair (assoc var (get self 'VARIABLES))))
          (cond (pair (rplacd pair (list value)))
                (T (setf (get self 'VARIABLES)
                         (cons (list var value)
                               (get self 'VARIABLES))))))))
  '*ROOT-WORLD-READY*)


;;;;;;;;;;;;;;;;;;;;;;;;;
;;;object worlds
;;;rectangles and squares

(defun make-rectangle-world ()
  (def-object 'rectangle 'root
    '((numsides 4)
      (description "Four-sided planar figure, all angles = 90
degrees")))
  (def-object 'rectangle-1 'rectangle
```

```
    '((length 8)
      (width 4)))
  (def-object 'square 'rectangle
    '((description "Rectangle with equal sides")))
  (def-object 'square-1 'square
    '((side 10)))
  (def-method 'rectangle 'area
    #'(lambda ()
       (* length width)))
  (def-method 'square 'area
    #'(lambda ()
       (* side side)))
  '*RECTANGLE-WORLD-READY*)

;;;;;;;;;;;;;;;;;;;;;;;
;;;object-world
;;;rooms and thermostats

(defun make-thermostat-world ()
  (def-object 'thermostat 'root
    '((setting 65)))
  (def-object 'room 'root
    '((temperature 65)))
  (def-object 'heater 'root
    '((state 'off)))
  (def-object 'room-311 'room
    '((thermostat 'thermostat-311)))
  (def-object 'thermostat-311 'thermostat
    '((heater 'heater-311)
      (location 'room-311)))
  (def-object 'heater-311 'heater
    '((location 'room-311)))
  (def-method 'room 'change-temp
    #'(lambda (amount-of-change)
       (let ((new-temp (+ amount-of-change temperature)))
         (send self 'set-value 'temperature new-temp)
         (print
          (list 'Temperature 'in self 'changes 'to new-temp 'degrees.))
         (send thermostat 'check-temp))))
  (def-method 'thermostat 'check-temp
    #'(lambda ()
       (cond ((< (send location 'show-value 'temperature) setting)
              (send heater 'turn-on))
             (T (send heater 'turn-off)))))
  (def-method 'thermostat 'change-setting
    #'(lambda (temp)
       (send self 'set-value 'setting temp)
       (print (list 'New 'setting 'of self 'is temp 'degrees.))
       (send self 'check-temp)))
  (def-method 'heater 'turn-on
    #'(lambda ()
       (cond ()
             ((equal state 'off)
              (print (list 'Heater 'turns 'on 'in location))
```

```
              (send self 'set-value 'state 'on)))
        (send location 'change-temp 1)))
  (def-method 'heater 'turn-off
    #'(lambda ()
        (cond ()
              ((equal state 'on)
               (print (list 'Heater 'turns 'off 'in location))
               (send self 'set-value 'state 'off)))))
  '*THERMOSTAT-WORLD-READY*)

#|
;;;try this after loading the object file
;;; ignore the compiler warnings, this implementation is not smart
;;; about the dynamic binding of object values.  If it were smart,
;;; the code would look a lot more like the stream code.

(build-root)
(pprint (symbol-plist 'root))

;;;note that the methods functions are not visible, another weakness
;;; of the naive implementation
;;; here's the method-call for showing the symbol property list:

(send 'root 'show)

(make-rectangle-world)
(send 'rectangle 'show-value 'description)
(send 'rectangle-1 'show-value 'width)
(send 'rectangle-1 'show-value 'length)
(send 'rectangle-1 'area)
(send 'rectangle-1 'set-value 'width 6)
(send 'rectangle-1 'area)
(send 'square-1 'show-parents)
(send 'square 'show-parents)
(send 'rectangle 'show-parents)
(send 'square-1 'show-env)
(send 'square-1 'area)
(send 'square-1 'set-value 'side 14)
(send 'square-1 'area)

(make-thermostat-world)
(send 'room-311 'show-value 'temperature)
(send 'thermostat-311 'show-value 'setting)
(send 'heater-311 'show-value 'state)
(send 'room-311 'change-temp -5)
(send 'thermostat-311 'change-setting 70)
|#
```