```
;;;;
;;;; Corrigé-Feuille8.stk       -- Correction Feuille 8 de Scheme
;;;;
;;;;           Author: Erick Gallesio [eg@essi.fr]
;;;;    Creation date: 22-May-2003 11:07 (eg)
;;;; Last file update: 22-May-2003 17:10 (eg)
;;;;

(load "tortue-lib")
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Version 1: On construit les objets à la main
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (FenetreGraphique)
  (let ((fg (initialiser-graphique)))
    (lambda (msg . args)
      (case msg
        ((ligne)     (when fg (apply afficher-ligne fg args)))
        ((effacer)   (when fg (effacer-graphique fg)))
        ((imprimer)  (when fg (imprimer-graphique fg (car args))))
        (else        (error "Message ~S difficile à interpréter" msg))))))



(define (Tortue)
  (let ((x       0)
        (y       0)
        (angle   0)
        (tracer? #t)
        (super   (FenetreGraphique)))

    ;; On y va
    (lambda (msg . args)
      (case msg
        ((position)      (list x y))
        ((avancer)       (let* ((pas  (car args))
                                (newx (+ x (* pas (cos angle))))
                                (newy (+ y (* pas (sin angle)))))
                           (if tracer?
                               (super 'ligne x y newx newy))
                           ;; Conserver les nouvelles coordonnées
                           (set! x newx)  (set! y  newy)))
        ((tourner)       (set! angle (+ angle (degre->radian (car args)))))
        ((baisser)       (set! tracer? #t))
        ((lever)         (set! tracer? #f))
        (else            (apply super msg args))))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Version 2: Utilisation de classes
;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;
;; Construction de defclass
;;

(define-macro (defclass name super slots .  methods)
  ;; Trouver d'abord la liste de tous les slots de la classe. C'est juste
  ;; slots s'il n'y a pas d'héritage. Sinon c'est slots + un symbole nommé
```



```
  ;; super qui contient un représentant de la super-classe
  (let ((les-slots (if (null? super)
                       slots
                       `((super ,super) ,@slots))))

    ;; On y va maintenant
    `(define (,name)
       (let ,les-slots
         ;; Générer les méthodes
         ,@(map (lambda (x) (cons 'define (cdr x))) methods)

         ;; Générer l'aiguillage
         (lambda (msg . args)
           (case msg
             ;; Générer les méthodes
             ,@(map (lambda(x) `(,(cadr x) (apply ,(cadr x) args)))
                    methods)
             ;; Et un else (ou l'autre) suivant que l'on a une super classe
             ;;ou pas
             (else ,(if (null? super)
                        `(apply super msg args)
                        `(error "Message ~S inconnu" msg)))))))))

#|
      Voyons l'expansion sur une classe sans héritage

       (pp (macro-expand '(defclass C1 ()
                            ((a 0) (b 10))
                            (method M (lambda () (cons a b))))))
       ==>
        (define (c1)
          (let ((a 0)
                (b 10))
            (define m (lambda ()(cons a b)))

            (lambda (msg . args)
              (case msg
                (m    (apply m args))
                (else (apply super msg args))))))

      Et maintenant avec héritage
       (pp (macro-expand '(defclass C1 (C0)
                            ((a 0) (b 10))
                            (method M (lambda () (cons a b))))))

       ==> (define (c1)
             (let ((super (c0))
                   (a 0)
                   (b 10))
               (define m (lambda ()(cons a b)))

               (lambda (msg . args)
                 (case msg
                   (m    (apply m args))
                   (else (error "Message ~S inconnu" msg))))))
|#


;;
;; Reconstruction de la Fenêtre graphique
;;
(defclass FenetreGraphique ()
```

```
  ;; Les slots (noter au passage qu'il n'est pas exporté)
  ((fg (initialiser-graphique)))

  ;; Les méthodes
  (method ligne (lambda (x0 y0 x1 y1)
                  (when fg (afficher-ligne fg x0 y0 x1 y1))))

  (method effacer (lambda ()
                    (when fg (effacer-graphique fg))))

  (method imprimer (lambda (fichier)
                     (when fg (imprimer-graphique fg fichier)))))

;;
;; Reconstruction de la tortue
;;
(defclass Tortue (FenetreGraphique)
  ((x 0) (y 0) (angle 0) (tracer? #t))

  (method position (lambda () (list x y)))
  (method avancer  (lambda (pas)
                     (let ((newx (+ x (* pas (cos angle))))
                           (newy (+ y (* pas (sin angle)))))
                       (if tracer?
                           (super 'ligne x y newx newy))
                       ;; Conserver les nouvelles coordonnées
                       (set! x newx) (set! y  newy))))
  (method tourner  (lambda (theta)
                     (set! angle (+ angle (degre->radian theta)))))
  (method baisser  (lambda ()
                     (set! tracer? #t)))
  (method lever    (lambda ()
                     (set! tracer? #f))))
```