TD 10 : CLOS

Le but de ce TD est de manipuler la partie objet de LISP.

On veut manipuler des figures géométriques. Nous allons dans ce but définir des classes correspondant aux différents types de figures que nous serons amenés à manipuler :

- *Le point ;*
- *Le triangle ;*
- *Le cercle ;*
- *Le carré ;*
- *Le rectangle ;*
- *Le polygone ;*
- *La classe figure dont hérite les précédentes.*

La manipulation consistera à leur appliquer des transformations géométriques. A cette fin nous devrons créer des méthodes associées aux différentes classes.

Table des matières

Définition des classes

```
(defclass $dot ($figure)
(($x :accessor abscisse :initarg :abscisse :type real)
 ($y :accessor ordonnee :initarg :ordonnee :type real)
)
)

(defclass $cercle ($figure)
(($centre :accessor centre :initarg :centre :type $dot)
 ($rayon :accessor rayon :initarg :rayon :type real)
)
)

(defclass $triangle ($figure)
(($M1 :accessor M1 :initarg :M1 :type $dot)
 ($M2 :accessor M2 :initarg :M2 :type $dot)
 ($M3 :accessor M3 :initarg :M3 :type $dot)
)
)

(defclass $poly ($figure)
(($listeSommets :accessor listeSommets :initarg :listeSommets :type list)
)
)

(defclass $rectangle  ($figure)
(($sommetG :accessor sommetG :initarg :sommetG :type $dot)
 ($hauteur :accessor hauteur :initarg :hauteur :type real)
 ($largeur :accessor largeur :initarg :largeur :type real)
)
)

(defclass $carre  ($figure)
(($sommetG :accessor sommetG :initarg :sommetG :type $dot)
 ($cote :accessor cote :initarg :cote :type real)
)
)

(defclass $figure ()
(($color :accessor color :initarg :color :type string)))
```

Définition des méthodes

A noter un côté pratique des méthodes : elles agissent de la façon qu'on leur dit d'agir, comme une fonction, mais pour la classe qu'on lui associe. Ainsi, on peut associer à une même méthode plusieurs comportements selon la classe associée.

Pour le point :

(defmethod translate ((xx $dot) dx dy) (setf (abscisse xx) (+ dx (abscisse xx))) (setf (ordonnee xx) (+ dy (ordonnee xx))) xx)	Translation
(defmethod symx ((xx $dot)) (setf (ordonnee xx) (- 0 (ordonnee xx))) xx)	Symétrie d'axe Ox.
(defmethod symy ((xx $dot)) (setf (abscisse xx) (- 0 (abscisse xx))) xx)	Symétrie d'axe Oy.
(defmethod symO ((xx $dot)) (Symx P0) (Symy P0) xx)	Symétrie de centre O.
(defmethod zoom ((xx $dot) k) (setf (abscisse xx) (* k (abscisse xx))) (setf (ordonnee xx) (* k (ordonnee xx))) xx)	Homothétie de rapport k.

Pour le triangle :

Les méthodes du triangles sont simples : on applique celles du point à chacun de sommets qui définissent le triangle.

(defmethod translate ((tt $triangle) dx dy) (translate (M1 tt) dx dy) (translate (M2 tt) dx dy) (translate (M3 tt) dx dy) tt)	
(defmethod symx ((tt $triangle)) (symx (M1 tt)) (symx (M2 tt)) (symx (M3 tt)) tt)	
(defmethod symy ((tt $triangle)) (symy (M1 tt)) (symy (M2 tt)) (symy (M3 tt)) tt)	
(defmethod symO ((tt $triangle))	

```
(symO tt)
(symO tt)
tt
)

(defmethod zoom ((tt $triangle) k)
  (zoom (M1 tt))
  (zoom (M2 tt))
  (zoom (M3 tt))
  tt
  )
```

Pour le cercle :

Les méthodes du cercle demandent d'appliquer les méthodes du point au centre du cercle. Il faut ensuite prêter garde au comportement du rayon.

```
(defmethod translate ((cc $cercle) dx dy)
  (translate (centre cc) dx dy)
  cc
)

(defmethod symy ((cc $cercle))
  (symy (centre cc))
  cc
  )

(defmethod symx ((cc $cercle))
  (symx (centre cc))
  cc
  )

(defmethod symO ((cc $cercle))
  (symx (centre cc))
  (symy (centre cc))
  cc
  )

(defmethod zoom ((cc $cercle) k)
  (zoom (centre cc) k)
  (setf (rayon cc) (* (abs k) (rayon cc)))
  cc
  )
```

Homothétie : le rayon est modifié !!! On doit le multiplier par la valeur absolue du rapport.

Pour le carré :

```
(defmethod translate ((cc $carre) dx dy)
  (translate (sommetG cc) dx dy)
  cc
  )

(defmethod symy ((cc $carre))
  (if (>= (abscisse (sommetG cc)) 0)
      (setf (abscisse (sommetG cc)) (- (cote
cc) (abscisse cc)))
      (setf (abscisse (sommetG cc)) (+ (cote
cc) (abscisse cc)))
   )
   cc
  )

(defmethod symx ((cc $carre))
  (if (>= (ordonnee (sommetG cc)) 0)
      (setf (ordonnee (sommetG cc)) (- (cote
```

Translation : il suffit d'appliquer la translation du point au sommet gauche du carré.

Symétrie d'axe Oy : il faut tout redéfinir. Attention au signe de l'abscisse du sommet gauche !

Même commentaire.

```
cc) (ordonnee cc)))
      (setf (ordonnee (sommetG cc)) (+ (cote
cc) (ordonnee cc)))
   )
   cc
  )

(defmethod symO ((cc $carre))
  (symx cc)
  (symy cc)
  cc
  )

(defmethod zoom ((cc $carre) k)
  (zoom (sommetG cc) k)
  (setf (cote cc) (* (abs k) (cote cc)))
  cc
  )
```

Homothétie : attention au côté qui doit être multiplié par la valeur absolue du rapport.

Pour le rectangle :

Ces méthodes sont les mêmes que celles du carré, sauf lorsqu'on manipule le côté : ici, le côté est la largeur ou la hauteur, il faut prendre garde à quelle caractéristique utiliser.

```
(defmethod translate ((rr $rectangle) dx dy)
  (translate (sommetG rr) dx dy)
  rr)

(defmethod symy ((rr $rectangle))
  (if (>= (abscisse (sommetG rr)) 0)
      (setf   (abscisse   (sommetG   rr))   (-
(largeur rr) (abscisse rr)))
      (setf   (abscisse   (sommetG   rr))   (+
(largeur rr) (abscisse rr)))
   )
   rr
  )

(defmethod symx ((rr $rectangle))
  (if (>= (ordonnee (sommetG rr)) 0)
      (setf   (ordonnee   (sommetG   rr))   (-
(hauteur rr) (ordonnee rr)))
      (setf   (ordonnee   (sommetG   rr))   (+
(hauteur rr) (ordonnee rr)))
   )
   cc
  )

(defmethod symO ((rr $rectangle))
  (symx cc)
  (symy cc)
  cc
  )

(defmethod zoom ((rr $rectangle) k)
  (zoom (sommetG rr) k)
  (setf  (hauteur  rr)  (*  (abs  k)  (hauteur
cc)))
  (setf  (largeur  rr)  (*  (abs  k)  (largeur
cc)))
  rr
  )
```

Homothétie : il faut multiplier la largeur ET la hauteur par la valeur absolue du rapport.

Pour le polygone :

Pour les méthodes du polygone il suffit d'appliquer les méthodes du point à chaque élément de la liste des sommets qui le définie. On utilise pour ça un dolist.

```
(defmethod translate ((pp $poly) dx dy)
  (dolist (i (listeSommets))
    (translate i dx dy)
    )
  pp
)

(defmethod symx ((pp $poly))
  (dolist (i (listeSommets))
    (symx i)
    )
  pp
)

(defmethod symy ((pp $poly))
  (dolist (i (listeSommets))
    (symy i)
    )
  pp
)

(defmethod symO ((pp $poly))
  (dolist (i (listeSommets))
    (symO i)
    )
  pp
)

(defmethod zoom ((pp $poly) k)
  (dolist (i (listeSommets))
    (zoom i k)
    )
  pp
)
```

Exemples d'exécution

Méthodes appliquées à un point :

```
>(setq P0 (make-instance '$dot :abscisse 3 :ordonnee 4 :color "noir"))
#<$DOT @ #x2126baa2>

> (abscisse P0)
3
>(ordonnee P0)
4

> (symx P0)
#<$DOT @ #x21332dfa>
> (symy P0)
#<$DOT @ #x21223952>
>(ordonnee P0)
-4
>(abscisse P0)
-3

>(symO P0)
#<$DOT @ #x21223952>
>(abscisse P0)
3
>(ordonnee P0)
4

(translate P0 1 1)
#<$DOT @ #x2132716a>
> (abscisse P0)
4
>(ordonnee P0)
5

>(zoom P0 -2)
#<$DOT @ #x2136a692>
>(abscisse P0)
-8
>(ordonnee P0)
-10
```

Méthodes appliquées à un cercle :

```
>(setq C1 (make-instance '$cercle
           :centre (make-instance '$dot :abscisse 0 :ordonnee 0 :color "pink" )
           :rayon 2
           )
    )
#<$CERCLE @ #x21220952>
>(setq cara (list (abscisse (centre C1))(ordonnee (centre c1))(rayon C1)))
(0 0 2)

>(translate C1 1 1)
#<$CERCLE @ #x213c5f3a>
>(setq cara (list (abscisse (centre C1))(ordonnee (centre c1))(rayon C1)))
(1 1 2)

> (symX C1)(symY C1)
#<$CERCLE @ #x213c5f3a>
#<$CERCLE @ #x213c5f3a>
>(setq cara (list (abscisse (centre C1))(ordonnee (centre c1))(rayon C1)))
(-1 -1 2)
```

```
> (symO C1)
#<$CERCLE @ #x21266702>
> (setq cara (list (abscisse (centre C1))(ordonnee (centre c1))(rayon C1)))
(1 1 2)

> (zoom C1 -5)
#<$CERCLE @ #x213acb62>
> (setq cara (list (abscisse (centre C1))(ordonnee (centre c1))(rayon C1)))
(-5 -5 10)
```

La méthode duplicate

Elle est très similaire (pas de piège) pour toutes les figures. Voici l'exemple pour le point :

```
(defmethod duplicate ((xx $dot) dx dy)
  (let ((copie 0))
    (setq copie (make-instance '$dot :abscisse (abscisse xx) :ordonnee (ordonnee
xx)))
    (translate copie dx dy)
    )
  )
```

Exécution :

```
>(abscisse P0)
3
>(ordonnée P0)
4

> (setq P01 (duplicate P0 1 1))
#<$DOT @ #x211e658a>

> P01
#<$DOT @ #x211e658a>

> (abscisse P01)
4
> (ordonnee P01)
5

>(abscisse P0)
3
>(ordonnée P0)
4
```

Ça marche ! Pour les autres figures, il faut remplacer dans la méthode $dot par la classe appropriée, et remplir le make-instance avec les bons paramètres.