Programmation 3 : feuille 9

Premiers pas en programmation objet

On considère la classe sans créneau :

```
(defclass polygon () ())

(defgeneric number-of-sides (p)
  (:documentation "returns the number of sides of a polygon"))

(defgeneric sides-lengths (p)
  (:documentation "returns the list of sides of a polygon"))

(defgeneric perimeter (p))

(defgeneric area (p))

(defmethod perimeter ((p polygon)) (reduce #'+ (sides-lengths p)))
```

Exercice 9 .1

Parmi les polygones que l'on souhaite traiter, on considère le cas des triangles.

1. Définir une classe `triangle` sous-classe de la classe `polygon` avec un seul créneau `number-of-sides` initialisé à `3`.
2. Créer la méthode `number-of-sides` pour la classe `triangle` grâce au mot-clé `:reader`. Tester la méthode `number-of-sides` sur une instance de la classe `triangle`.
3. Modifier la classe `triangle` de manière à ce que le créneau soit partagé par toutes les instances de la classe.
4. Implémenter la méthode `area` pour les triangles.

Exercice 9 .2

1. Pour les triangles quelconques, définir une classe `plain-triangle` sous-classe de `triangle` ayant trois créneaux `side-a`, `side-b` `side-c` correspondant aux longueurs des trois côtés.
2. Écrire une fonction `correct-side-length (a b c)` qui vérifie que les valeurs a, b, c représentent bien un triangle, c'est-à-dire qu'elles sont positives et vérifient les inégalités $a < b + c$ et $a > |b - c|$.
3. Définir la fonction `make-plain-triangle (a b c)` qui créée une instance de `plain-triangle`. On utilisera une assertion pour vérifier que le triangle est correct avant de créér une instance.
4. Implémenter la méthode `sides-lengths` pour cette classe. Créér une instance de `plain-triangle` et tester les méthodes `sides-lengths`, `perimeter` et `area`.
5. Écrire une méthode `resize-a ((triangle plain-triangle) new-a)` qui permet de modifier le créneau `side-a` d'un triangle quelconque à condition qu'on obtienne encore un triangle.

Pour certaines formes régulières comme les triangles isocèles, rectangles et équilatéraux, le calcul de l'aire et celui du périmètre d'un triangle peut être simplifié.

Exercice 9 .3

1. Définir une classe `isocele`. Définir la fonction `make-isocele (cote base)`. Implémenter la méthode `sides-lengths` pour les triangles isocèles.
2. Faire de même pour les triangles équilatéraux.
3. Redéfinir les méthodes `perimeter` et `area` pour les triangles équilatéraux.

Exercice 9 .4

On veut rajouter un créneau `area` qui mémorise l'aire du triangle (calculée à partir des autres créneaux). Si on autorise la modification des côtés du triangle, il faut penser au recalcul de l'aire.

Première solution : On peut calculer l'aire à la création d'une instance et la recalculer à chaque modification d'un côté.

1. Rajouter un créneau `area` à la classe triangle avec son "reader" `area`.
2. Écrire des méthodes "after" pour
   `initialize-instance ((triangle triangle) &rest initargs &key &allow-other-keys)`
   et `resize-a ((triangle plain-triangle))`.

Deuxième solution : On peut également procéder par mémoïsation, c'est-à-dire ne pas calculer l'aire à l'avance (laisser le créneau à `nil`) et la calculer la première fois qu'elle est demandée. À la modification d'un côté, il suffit de remettre le créneau à `nil`.

Reprendre le code comme il était avant l'implémentation de la première solution, puis écrire deux versions :

1. en mettant le créneau `area` dans la classe `triangle`
2. en mettant le créneau `area` dans une classe `mixin`.

Exercice 9 .5

Voici une classe pour des cercles colorés :

```
(defclass color ()
  ((r :initarg :r :reader red-of)
   (g :initarg :g :reader green-of)
   (b :initarg :b :reader blue-of)))

(defclass circle ()
  ((radius :initarg :radius)
   (color :initarg :color)))
```

1. Écrire une fonction `make-circle`, constructeur de cercle (rouge par défaut).
2. Définir les fonctions et méthodes nécessaires pour pouvoir évaluer des expressions du genre :

```
(let ((l (list (make-circle 10 :color (make-color))
               (make-plain-triangle 3 5 6)
               (make-isocele 3 4)
               (make-triangle-rectangle 3 4))))
  (mapcar #'perimeter l))
```