

## TP n° 6

## Introduction aux interfaces et aux classes abstraites : figures et tris

## 1 Figures

**Exercice 1** On considère une classe abstraite `Figure` dont voici le début :

```
public abstract class Figure {  
    // coordonnées du centre approximatif de la figure  
    private int posX;  
    private int posY;  
  
    public Figure(int x, int y) {  
        posX = x;  
        posY = y;  
    }  
    .....  
}
```

Complétez avec les méthodes concrètes :

- `public int getPosX()` qui retourne la position horizontale du centre de la figure (abscisse) ;
- `public int getPosY()` qui retourne la position verticale du centre de la figure (ordonnée) ;

Ainsi que la méthode abstraite

- `public abstract void affiche()` ; qui affichera un résumé des propriétés de la figure.

On souhaite que les deux accesseurs précédents ne puissent pas être redéfinis. Quelle modification faut-il faire pour le garantir ?

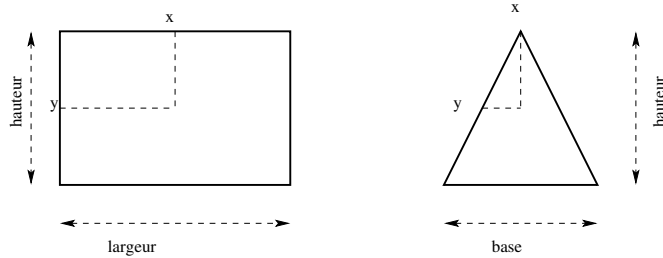
On veut maintenant définir les classes concrètes suivantes : `Rectangle`, `Carre`, `Ellipse`, `Circle` et `Triangle`. Remarquez que la hiérarchie entre ces objets n'est pas si simple à définir : soit vous réfléchissez en tant qu'inclusion des définitions, soit vous raisonnez sur le nombre d'attributs. Cependant les manipulations type/sous-types devraient vous orienter dans le choix de l'une d'elle. Pour l'instant, n'écrivez pas de code pour ces classes, mais représentez les dans un diagramme seulement.

**Exercice 2** — Écrivez le code de la classe `Rectangle`. En plus des attributs qui sont déjà définis dans `Figure`, la classe `Rectangle` doit contenir les attributs `largeur` et `hauteur` qui sont de type `double` et qui ne sont pas modifiables<sup>1</sup>. On doit passer en paramètres du constructeur la position du centre, la largeur et la hauteur. Dans la figure ci-dessous, `x` et `y` représentent la position du centre.

- Écrivez le code de la classe `Ellipse`. En plus des attributs qui sont déjà définis dans `Figure`, la classe `Ellipse` doit contenir les attributs `grand_rayon` et `petit_rayon` qui sont de type `double` et qui ne sont pas modifiables. On doit passer en paramètres du constructeur la position du centre, le `grand_rayon` et le `petit_rayon`.

---

1. C'est important. Un carré étant un rectangle on ne peut pas vraiment laisser ces dimensions être modifiables

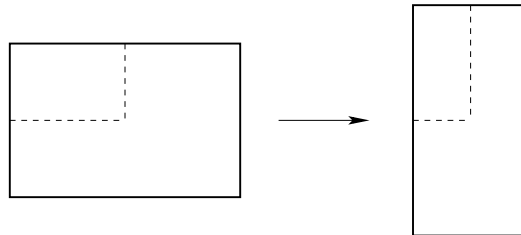


**Exercice 3** — Écrivez le code de la classe `Carre`, avec un constructeur adapté.

— De la même façon écrivez le code de la classe `Circle`.

**Exercice 4** Écrivez le code de la classe `Triangle`. On considère qu'un triangle est toujours isocèle et positionné comme sur le dessin ci-dessus. On doit passer en paramètres du constructeur la position du centre, ainsi que la base et l'hauteur, de type `double` (cf. dessin). Dans la figure ci-dessus,  $x$  et  $y$  représentent la position du centre,  $y$  est à la moitié de l'hauteur.

**Exercice 5** On définit l'interface `Deformable` qui contiendra la méthode `Figure deformation(double coeffH, double coeffV)` où `coeffH` est le coefficient de déformation horizontale, et où `coeffV` est celui de déformation verticale. Par exemple, dans le dessin ci-dessus, le rectangle de droite est la déformation du rectangle de gauche `rec` retournée par l'appel `rec.deformation(0.5, 1.5)`.



Quelles figures implémenteront `Deformable`? Quel devra être dans chaque cas le type réel de l'objet référencé par la valeur de retour?

Implémentez cette interface dans toutes les classes où c'est possible.

**Exercice 6** Les méthodes des questions suivantes sont à ajouter (sauf indication contraire) dans `Figure`. Pour chacune d'entre elles, demandez-vous si elle doit être abstraite ou non, et dans quelles classes il convient de la définir ou la redéfinir. Vous pouvez faire ces questions dans l'ordre qui vous convient.

- Écrivez la méthode `double estDistantDe(Figure fig)` qui calculera la distance entre le centre de la figure sur laquelle est appelée la méthode et le centre de `fig`;
- Écrivez la méthode `double surface()` qui calculera la surface de la figure.  
**Rappel** surface d'un triangle  $base \times hauteur / 2$ .  
surface d'une ellipse  $\pi \times grand\_rayon \times petit\_rayon$ .
- Écrivez une méthode `déplacement(int x, int y)` qui déplace une figure (et la modifie donc) dans la direction indiquée par `x` et `y`.

## 2 Tris

Le tri à bulles est un algorithme classique permettant de trier un tableau d'entiers. Il peut s'écrire de la façon suivante en Java :

```
static void triBulles(int tab[])
{
    boolean change = false;
    do {
        change = false;
        for (int i=0; i<tab.length - 1; i++) {
            if (tab[i] > tab [i+1]) {
                int tmp = tab[i+1];
                tab[i+1] = tab[i];
                tab[i] = tmp;
                change = true;
            }
        }
    } while (change);
}
```

Cette implémentation du tri à bulles permet de trier un tableau d'entiers. Maintenant on veut pouvoir utiliser tout autre type de données (muni d'une relation d'ordre) sans avoir à réécrire l'algorithme à chaque fois. Pour cela on va supposer définie comme suit, l'interface **Triable** :

```
public interface Triable {
    // échange les éléments en positions i et j
    void echange(int i, int j);

    // retourne vrai si l'élément de position i est plus grand que
    // l'élément de position j
    boolean plusGrand(int i, int j);

    // nombre d'éléments à trier
    int taille();
}
```

Les objets des classes implémentant cette interface devront représenter ainsi des **tableaux** d'éléments, comparables entre eux, que l'on souhaite trier.

**Exercice 7** Écrivez dans l'interface **Triable** la méthode `static void triBulles(Triable t)` qui met en œuvre le tri à bulles pour les objets **Triables**. Elle est similaire à `static void triBulles(int tab[])`, à quelques différences près.

**Exercice 8** Écrivez une classe **TabEntiersTriable** qui implémente l'interface **Triable** et permet à `triBulles(Triable t)` de trier un tableau d'entiers (selon leur ordre naturel). N'oubliez pas les constructeurs et la méthode `toString()` afin de pouvoir tester le tri.

**Exercice 9** Écrivez une classe **TriBinaire** qui implémente l'interface **Triable** et permet à `triBulles(Triable t)` de trier un tableau de chaînes de bits (selon leur ordre naturel, c'est-à-dire  $0 = 00 = 000 \dots < 1 = 01 = 001 \dots < 10 < 11 < 100$  etc.). N'oubliez pas les constructeurs et la méthode `toString()` afin de pouvoir tester le tri.

**Exercice 10** Écrivez une classe **Dictionnaire** qui implémente l'interface **Triable** et permet à `triBulles(Triable t)` de trier des chaînes de caractères (en ordre alphabétique). N'oubliez pas les constructeurs et la méthode `toString()` afin de pouvoir tester le tri.