Université Paris Diderot **Programmation Orientée Objet**
L2-L3 **Année 2014-2015**

TP n° 5

Interfaces et classes astraites

Important : Dans ce TP, toutes les variables (d'objet ou de classe) seront **privées.**

1 Figures

Exercice 1 On va définir 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;
    }
    ............
}
```

On définira aussi dans `Figure` les méthodes concrètes :

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

Ainsi que la méthode abstraite

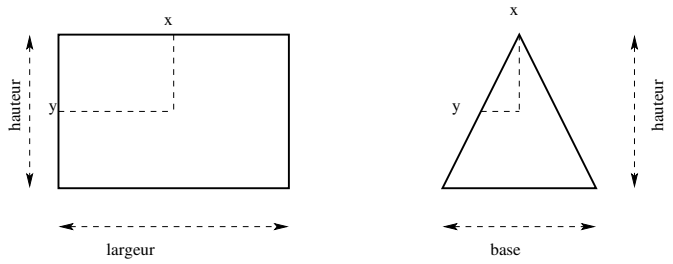
- `public abstract void affiche();` qui affichera un résumé des propriétés de la figure.

Écrivez le code de `Figure`.

On définira aussi les classes concrètes suivantes : `Rectangle`, `Carre` et `Triangle`.

Donnez la hiérarchie de classes que vous choisiriez. Pour l'instant, n'écrivez pas de code pour ces classes.

Exercice 2 Écrivez le code de la classe `Rectangle`. On doit passer en paramètres du constructeur la position du centre, la largeur et la hauteur. Dans la figure ci-dessous, `x` et `y` représente la position du centre.

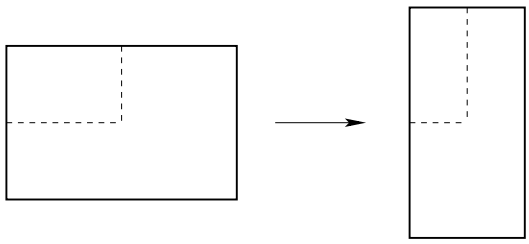


Exercice 3 Écrivez le code de la classe `Carre`. On doit passer en paramètres du constructeur la position du centre et la longueur d'un côté.

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, la base et la hauteur (cf. dessin). Dans la figure ci-dessus, x et y représente la position du centre, y est à la moitié de la hauteur.

Exercice 5 On définit l'interface `Deformable` qui correspond aux figures qu'on peut déformer horizontalement et verticalement de manière à obtenir une nouvelle figure.

Cette interface 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-dessous, 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 $base \times hauteur/2$.
- Écrivez une méthode `déplacement(int x, int y)` qui déplace une figure (et la modifie donc) dans la direction indiquée par `x` et `y`.

Exercice 7 S'il vous reste du temps...

1. Ajoutez d'autres figures dans cette hiérarchie de classes en réécrivant les méthodes qui doivent l'être.
2. Ajoutez d'autres méthodes calculant d'autres propritétés géométriques. Demandez-vous à chaque fois à quelle(s) classe(s) elles appartiennent, voire s'il serait pertinent de créer une nouvelle interface.

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 8 Écrivez la méthode `static void triBulles(Triable t)` qui met en œuvre le tri à bulles pour les objets `Triable`s. Elle est similaire à `static void triBulles(int tab[])`, à quelques différences près.

Exercice 9 Écrivez une classe `EntierTriable` qui implémente l'interface `Triable` et permet à `triBulles(Triable t)` de trier des entiers (selon leur ordre naturel). 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.

Exercice 11 Sur le même modèle, inventez et écrivez une autre classe qui implémente `Triable`.