Université Paris Cité **Programmation Orientée Objet**
Année 2022-2023

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 double posX;    // Position sur l'axe horizontal
  private double posY;    // Position sur l'axe vertical

  public Figure(double posX, double posY) {
    this.posX = posX;
    this.posY = posY;
  }
  [...]
}
```

1. Complétez avec les méthodes concrètes :
   - `public double getPosX()` qui retourne la position horizontale du centre de la figure (abscisse) ;
   - `public double getPosY()` qui retourne la position verticale du centre de la figure (ordonnée) ;

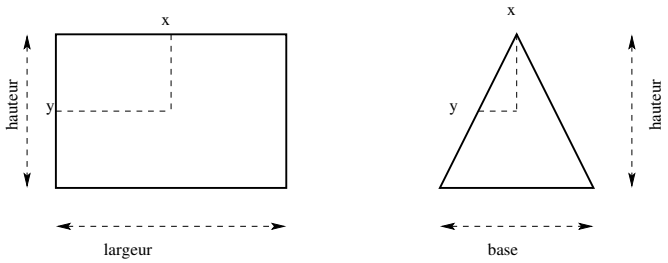
   Ainsi que la méthode abstraite :
   - `public abstract void affiche();` qui affichera un résumé des propriétés de la figure.
2. On souhaite que les deux accesseurs précédents ne puissent pas être redéfinis. Quelle modification faut-il faire pour le garantir ?

On veut maintenant définir les classes concrètes suivantes : `Rectangle`, `Carre`, `Ellipse`, `Cercle` 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

1. 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 [1]. 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.
2. 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

1. A l'aide des classes précédement définines, écrivez le code de la classe `Carre`. En particulier, n'oubliez pas d'adapter son constructeur.
2. De la même manière, écrivez le code de la classe `Cercle`.
3. Pour finir, écrivez le code de la classe `TriangleIsocele`. Nous ne considérons que les triangles positionnés comme sur le dessin ci-dessus. Le constructeur prend en arguments 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.
4. **(Bonus)** Définir une classe `Triangle` qui généralise la classe `TriangleIsocele` et qui permet de représenter des triangles quelconques.

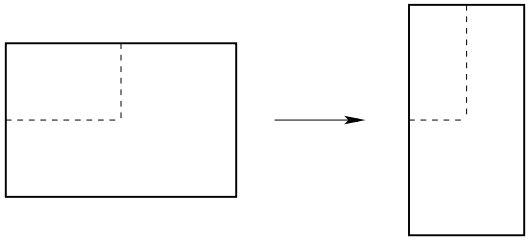
Exercice 4 Tous les carrés sont en particulier des rectangles. C'est pour cela que la classe `Carre` est héritée de la classe `Rectangle`. Par contre, tous les rectangles ne sont pas des carrés. Seul ceux ayant des cotés de même longueur peuvent être vu comme carrés.

1. Écrire dans la classe `Rectangle` une méthode `boolean estUnCarre()` revoyant `true` si le rectangle appellant peut-être vu comme un carré, et `false` sinon.
2. Écrire dans la classe `Rectangle` une méthode `Carre toCarre()` revoyant un carré correspondant au rectangle appellant si celui-ci peut-être vu comme un carré, et `null` sinon (en avertissant l'utilisateur).

De même, tous les cercles sont en particuliers des ellipses mais l'inverse n'est pas vrai.

3. Écrire dans la classe `Ellipse` une méthode `boolean estUnCercle()` revoyant `true` si l'ellipse appellante peut-être vu comme un cercle, et `false` sinon.
4. Écrire dans la classe `Ellipse` une méthode `Cercle toCercle()` revoyant un cercle correspondant à l'ellipse appellante si celle-ci peut-être vue comme un cercle, et `null` sinon (en avertissant l'utilisateur).
5. **(Bonus)** Faire de même pour la classe `Triangle` vis à vis de la classe `TriangleIsocele`.

Exercice 5 On définit l'interface `Deformable` qui contiendra la méthode `Figure deformation(double coeffH, double coeffV)`. Cette méthode correspond à l'application d'une déformation de coefficient `coeffH` sur l'axe horizontal, et `coeffV` sur l'axe vertical. Par exemple, dans le dessin ci-dessous, le rectangle de droite est la déformation du rectangle de gauche retournée par l'appel `deformation(0.5,1.5)`.



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

2. 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 quelle(s) classe(s) il convient de la définir ou la redéfinir. Vous pouvez faire ces questions dans l'ordre qui vous convient.

1. 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 figure `fig` passé en argument ;
2. Écrivez la méthode `double surface()` qui calculera la surface de la figure.

Rappel : Dans un triangle $surface = base \times hauteur / 2$ tandis que dans une ellipse $surface = \pi \times grand\_rayon \times petit\_rayon$.

3. Écrivez une méthode `deplacement(double x, double y)` qui déplace une figure (et la modifie donc) de `x` sur l'axe horizontal et de `y` sur l'axe vertical.

2 Tris

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

```
public static void triBulles(int tab[]) {
  boolean change;
  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. On veut maintenant pouvoir l'utiliser sur tout autre type de données muni d'une relation d'ordre. Pour cela, on introduit l'interface `Triable` suivante :

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

  // Retourne vrai ssi 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 ainsi représenter des **tableaux** d'éléments comparables et échangeables entre eux.

Exercice 7 On souhaite maintenant pouvoir trier les objets implémentant cette interface.

1. Écrivez dans l'interface `Triable` la méthode `static void triBulles(Triable t)` qui met en œuvre le tri à bulles pour les objets `Triable`s. Pour cela, inspirez-vous de l'implémentation présentée en début de section.

Mettons maintenant cela en application à l'aide des classes suivantes. Créez chacune d'entre elles sans oublier d'y ajouter un constructeur. Ajoutez aussi une méthode `public String toString()` et écrivez quelques tests.

2. 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).
3. 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 $0 = 00 = 000\ldots < 1 = 01 = 001\ldots < 10 < 11 < 100$ etc.).
4. Écrivez une classe `Dictionnaire` qui implémente l'interface `Triable` et permet à `triBulles(Triable t)` de trier des chaînes de caractères (en ordre alphabétique).
5. Pour finir, en reprenant le travail des exercices précédents, écrivez une classe `TableauBlanc` qui implémente l'interface `Triable` et permet à `triBulles(Triable t)` de trier des `Figure`s selon la distance entre leur centre approximatif et l'origine du tableau (le point de coordonées `posX = 0` et `posY = 0`).