TP n° 6 :

Introduction aux classes abstraites et interfaces

1 Formes géométriques

On considère la classe abstraite `Forme` suivante, dont les extensions permettront de représenter des formes géométriques :

```
public abstract class Forme {
  /* coordonnées du centre la forme dans le plan */
  private double x;
  private double y;
  /* largeur et hauteur de la forme */
  private double width;
  private double height;

  public Forme(double x, double y, double width, double
      height) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
  }
}
```

Nous nous restreindrons ici au cas où ces formes sont des rectangles ou des ellipses dont les côtés ou les axes sont d'orientation verticale ou horizontale.

Exercice 1 Ajouter à la classe `Forme` :

1. Des accesseurs publics en lecture pour ses champs (`Eclipse` peut les générer pour vous, bouton droit sur le code + "Source").
2. Une méthode abstraite et publique `double surface()`.
3. Une redéfinition de `toString` affichant le nom de classe d'une forme et les valeurs des champs.

Exercice 2 Implémenter les deux extensions concrètes `Rectangle` et `Ellipse` de `Forme`. Chaque classe devra permettre, sans champs supplémentaires, de représenter les formes géométriques de même nom, sachant que :

1. Un rectangle est défini par son centre, sa largeur et sa hauteur. La surface d'un rectangle est le produit de sa largeur et de sa hauteur.

2. Une ellipse est définie par son centre, son rayon horizontal (la moitié de sa largeur) et son rayon vertical (la moitié de sa hauteur). Sa surface est égale à π fois le produit de ses deux rayons.

Exercice 3 Pour changer les dimensions d'une forme, il suffit de multiplier sa largeur par un certain facteur `sx` et sa hauteur par un certain facteur `sy`.

On souhaite écrire dans la classe `Forme` une méthode `Forme resized(double sx, double sy)` renvoyant une copie de la forme courante, de même centre mais de hauteur et largeur modifiés par les facteurs spécifiés.

Trouver le moyen d'implémenter cette méthode dans `Forme` sans modifier ses classes descendantes, et sans avoir à la modifier plus tard si l'on ajoutait de nouveaux descendants à cette classe.

Indication. Servez-vous de la méthode `clone` héritée d'`Object` dans `Forme`.

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

Cette implémentation du tri à bulles ne permet de trier qu'un tableau d'entiers. On souhaite pouvoir utiliser le même algorithme sur tout type de données muni d'une relation d'ordre et associé à une notion de position d'élément dans une collection finie de valeurs (par exemple un tableau d'entiers, mais aussi un tableau de chaînes, une liste de formes associé à un ordre induit par leur surface, etc). Pour cela, on introduit l'interface `Triable` suivante :

```java
public interface Triable {
  // échange les éléments de positions i et j :
  void echanger(int i, int j);

  // renvoie vrai ssi l'élément de position i
  // est plus grand que celui de position j :
  boolean plusGrand(int i, int j);

  // borne supérieure sur les positions :
  int taille();

  // implémentation par défaut d'un tri à bulles sur tout
  // objet dont la classe implémente cette interface :
  default void triBulles() {
    /* à compléter */
  }
}
```

Les objets des classes implémentant cette interface représenteront des collections d'éléments comparables et échangeables entre eux, chaque élément étant associé à une position comprise entre `0` et la valeur de retour de `taille` moins 1.

Exercice 4 Compléter l'implémentation par défaut de `triBulles`. Une invocation de la forme `o.triBulles()` devra trier les éléments de la collection `o` pour l'ordre implémenté par `plusGrand`, par l'algorithme du tri à bulles. Inspirez-vous bien sûr de l'implémentation en début de section.

Exercice 5 Définir les classes suivantes, chacune implémentant l'interface `Triable` pour l'ordre spécifié :

1. `TabEntiers` encpasulant un tableau d'entiers triable suivant l'ordre naturel sur les entiers.
2. `Dictionnaire` encapsulant un tableau de chaînes de caractères triable suivant l'ordre lexicographique.

N'oubliez pas d'écrire des `toString` et des tests.