MOOC Intro POO Java
Tutoriels semaine 4

Les tutoriels sont des exercices qui reprennent des exemples du cours et dont le corrigé est donné progressivement au fur et à mesure de la donnée de l'exercice lui-même.

Ils sont conseillés comme un premier exercice sur un sujet que l'étudiant ne pense pas encore assez maîtriser pour aborder par lui-même un exercice «classique».

Figures géométriques revisitées

Le but de cet exercice est de reprendre un exemple de "figures géométriques" pour illustrer la notion de polymorphisme. Nous utiliserons une collection hétérogène de figures géométriques.

Il s'agit ici de reprendre le code obtenu au terme du tutoriel de la semaine 3 (`FiguresGeometriques.java`) et de le modifier de sorte à travailler avec un tableau de figures géométriques contenant à la fois des rectangles et des cercles.

[Essayez de le faire par vous même avant de regarder la solution qui suit]

Sans polymorphisme, nous serions obligés de créer deux tableaux différents, puisque l'on manipule deux types d'objets.

On souhaiterait maintenant plutôt pouvoir écrire notre programme principal comme suit :

```
class FiguresGeometriques2 {
    public static void main(String[] args) {
        Figure[] f = new Figure[3];
        f[0] = new RectangleColore(1.2, 3.4, 12.3, 43.2, 4);
        f[1] = new Cercle(2.3, 4.5, 12.2);
        f[2] = new Rectangle(1.3, 3.6, 2.3, 56.2);
        for (int i = 0; i < f.length; i++){
            f[i].affiche();
            System.out.println("La surface de cette forme est :" + f[i].surface());
        }
    }
}
```

et pouvoir indifféremment mettre dans notre tableau `f` des cercles et des rectangles. En termes informatiques, cela revient à utiliser le polymorphisme et la résolution dynamique des liens.

Comme la résolution dynamique des liens se fait automatiquement en Java, ceci ne pose en fait aucun problème particulier, et le code ci-dessus est presque correct.

Nous avons cependant un petit problème à résoudre.

[Essayez de le découvrir par vous même avant de regarder la solution qui suit]

En fait, nous souhaitons ici faire afficher aussi la surface des figures. Cependant, la méthode `surface` n'est pas définie dans `Figure`, donc si vous compilez le programme tel qu'il est, vous obtiendrez un message d'erreur du type

```
cannot resolve symbol
symbol  : variable surface ()
location: class Figure
    f[i].surface());
```

On aimerait donc définir le calcul de surface dans la classe `Figure`, mais cela pose un second problème: la surface d'un cercle et celle d'un rectangle ne se calculent pas de la même manière.

La solution consiste ici à déclarer la méthode `surface` commme abstraite dans la classe `Figure`

```
class Figure {
    private double x;  // abscisse du centre
    private double y;  // ordonnée du centre

    public Figure(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public abstract double surface();

    public void affiche() {
        System.out.println("centre = (" + x + ", " + y + ")");
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public void setCentre(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
```

Cela ne suffit pas encore tout à fait: une classe qui contient une méthode abstraite ne peut pas être instanciée. Il faut donc également déFigure comme abstraite. Le code correct est donc:

```
abstract class Figure { ...
```

Voilà, notre classe `Figure` est maintenant utilisable comme on le souhaite.

[Le code source final est disponible sur la page des exercices sous le nom `FiguresGeometriques2.java`]
