Série N°2

(A rendre avant 13/11/2021)

Classes et Objets

Réaliser une classe *Point* permettant de représenter un point sur un axe. Chaque point sera caractérisé par un nom (de type *char*) et une abscisse (de type *double*). On prévoira :

- un constructeur recevant en arguments le nom et l'abscisse d'un point,
- une méthode *affiche* imprimant (en fenêtre console) le nom du point et son abscisse,
- une méthode *translate* effectuant une translation définie par la valeur de son argument.

Écrire un petit programme utilisant cette classe pour créer un point, en afficher les caractéristiques, le déplacer et en afficher à nouveau les caractéristiques.

Ecrire un programme permettant de calculer la distance entre deux points.

Packages, Contrôle d'accès et Encapsulation

Ecrire un programme testant les modificateurs de visibilités (private, package (défaut), protected et public)
Ecrire un programme testant les getters et les setters.

Soit le programme suivant comportant la définition d'une classe nommée Point et son utilisation :

```
class Point
{ public Point (int abs, int ord) { x = abs ; y = ord ; }
public void deplace (int dx, int dy) { x += dx ; y += dy ; }
public void affiche ()
{ System.out.println ("Je suis un point de coordonnees " + x + " " + y) ;
}
private double x ; // abscisse
private double y ; // ordonnee
}
public class TstPnt
{ public static void main (String args[])
{ Point a ;
a = new Point(3, 5) ; a.affiche() ;
a.deplace(2, 0) ; a.affiche() ;
Point b = new Point(6, 8) ; b.affiche() ;
}}
```

Modifier la définition de la classe Point en supprimant la méthode affiche et en introduisant deux méthodes d'accès nommées abscisse et ordonnee fournissant respectivement l'abscisse et l'ordonnée d'un point. Adapter la méthode main en conséquence.

Membres statiques

Réaliser une classe qui permet d'attribuer un numéro unique à chaque nouvel objet créé (1 au premier, 2 au suivant...). On ne cherchera pas à réutiliser les numéros d'objets éventuellement détruits. On dotera la classe uniquement d'un constructeur, d'une méthode getIdent fournissant le numéro attribué à l'objet et d'une méthode getIdentMax fournissant le numéro du dernier objet créé. Écrire un petit programme d'essai.

Écrire une méthode statique d'une classe statique Util calculant la valeur de la "fonction d'Ackermann" A définie pour m>=0 et n>=0 par :

- A (m, n) = A (m-1, A(m, n-1)) pour m>0 et n>0,
- A (0, n) = n+1 pour n>0,
- A (m, 0) = A(m-1, 1) pour m>0.

Soit la classe Point ainsi définie :

```
class Point{
        public Point (int abs, int ord) { x = abs ; y = ord ; }
        public void affiche (){
```

```
        System.out.println ("Coordonnees " + x + " " + y) ;
        }
        private double x ; // abscisse
        private double y ; // ordonnee
}
```

Lui ajouter une méthode maxNorme déterminant parmi deux points lequel est le plus éloigné de l'origine et le fournissant en valeur de retour. On donnera deux solutions :

- maxNorme est une méthode statique de Point,
- maxNorme est une méthode usuelle de Point.

Héritage et polymorphisme

On dispose de la classe suivante (disposant cette fois d'un constructeur) :

```
class Point {
public Point (int x, int y) { this.x = x ; this.y = y ; }
public void affCoord()
{ System.out.println ("Coordonnees : " + x + " " + y) ;
}
private int x, y ;
}
```

Réaliser une classe PointNom, dérivée de Point permettant de manipuler des points définis par leurs coordonnées (entières) et un nom (caractère). On y prévoira les méthodes suivantes :

- constructeur pour définir les coordonnées et le nom d'un objet de type PointNom,
- affCoordNom pour afficher les coordonnées et le nom d'un objet de type PointNom.

Écrire un petit programme utilisant la classe PointNom.

Ecrire un programme donnant exemple au polymorphisme avec classe abstraite (voir TP).

Ecrire un programme donnant exemple au polymorphisme en utilisant un tableau d'objets (Menagerie, voir cours)

Classes abstraites et interfaces

On souhaite disposer d'une hiérarchie de classes permettant de manipuler des figures géométriques. On veut qu'il soit toujours possible d'étendre la hiérarchie en dérivant de nouvelles classes mais on souhaite pouvoir imposer que ces dernières disposent toujours des méthodes suivantes :

- void affiche ()
- void homothetie (double coeff)
- void rotation (double angle)

Écrire la classe abstraite Figure qui pourra servir de classe de base à toutes ces classes.

Compléter la classe abstraite Figure de l'exercice précédent, de façon qu'elle implémente :

- une méthode homoRot (double coef, double angle) qui applique à la fois une homothétie et une rotation à la figure,
- de méthodes statiques afficheFigures, homothetieFigures et rotationFigures appliquant une même opération (affichage, homothétie ou rotation) à un tableau de figures (objets d'une classe dérivée de Figure).

On souhaite disposer de classes permettant de manipuler des figures géométriques. On souhaite pouvoir caractériser celles qui possèdent certaines fonctionnalités en leur demandant d'implémenter des interfaces, à savoir :

- Affichable pour celles qui disposeront d'une méthode void affiche (),
- Tranformable pour celles qui disposeront des deux méthodes suivantes :
  void homothetie (double coeff)
  void rotation (double angle)

Écrire les deux interfaces Affichable et Transformable.