Programmation Orientée Objets

TD n°3 : Composition et Héritage

Exercice 1 : Agrégation forte / faible

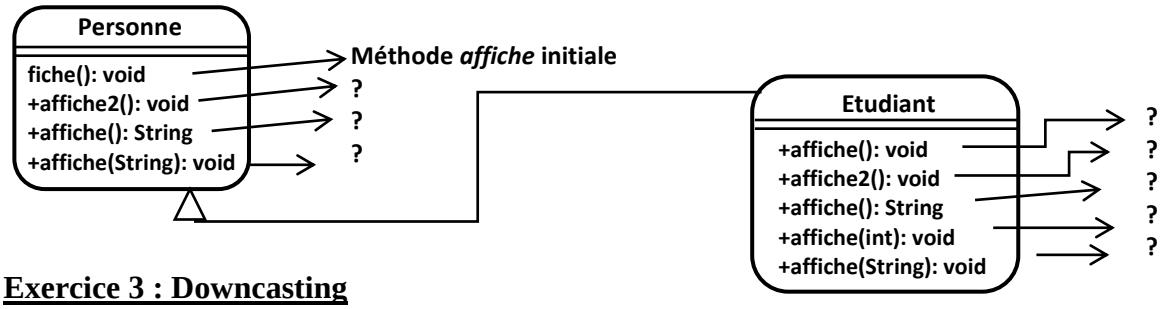
Modifier le code ci-dessous afin d'implanter une agrégation forte de `Point` dans `Cercle`.

```
public class Point {
   private int x; // Abscisse
   private int y; // Ordonnée
   public Point(int x, int y) {
      this.x = x, this.y = y;
   }
   public void translater(int dx, int dy) {
      this.x += dx; this.y += dy;
   }
   public double distance(Point p) {
      return Math.sqrt(pow(this.x-p.x,2)
                       +pow(this.y-p.y,2));
   }
   public int getX() { return x; }
   public int getY() { return y; }
   public void setX(int x) { this.x = x; }
   public void setY(int y) { this.y = y; }
}
```

```
public class Cercle {
   public Point centre;
   private int rayon;
   public Cercle(int x, int y, int r) {
      centre = new Point(x,y);
      rayon = r;
   }
   public Cercle(Point p, int r) {
      centre = p;
      rayon = r;
   }
   public void translater(int dx, int dy) {
      centre.translater(dx,dy);
   }
   public double distance(Cercle c) {
      return centre.distance(c.centre);
   }
}
```

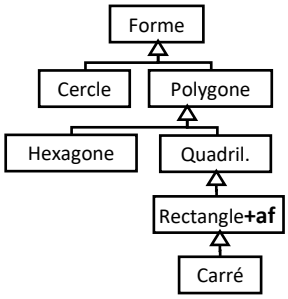
Exercice 2 : Surcharge et redéfinition

Préciser pour chaque méthode ci-dessous s'il s'agit d'une nouvelle méthode, d'une surcharge ou d'une redéfinition de la méthode `affiche()` initiale, ou si l'écriture d'une telle méthode est interdite.



Exercice 3 : Downcasting

Pour chacune des lignes suivantes, que se passe-t-il : à la compilation ? à l'exécution ?




```
Polygone p = new Carré();
Cercle c = p;
Rectangle r = p;
Rectangle r = (Rectangle) p;
Hexagone h = (Hexagone) p;
```

Exercice 4 : Polymorphisme, surcharge et redéfinition

Pour les appels de méthodes ci-dessous, quel choix est fait à la compilation ? Quel est l'affichage à l'exécution ?

```
class Poly {
       int aff(Poly p) {
              return 1;
       }
}
class Rect extends Poly {
       int aff(Poly p) {
              return 2;
       }

       int aff(Rect r) {
              return 3;
       }
}
```

```
Poly p = new Poly();
Poly pr = new Rect();
Rect r = new Rect();
```

```
// Partie p
System.out.println(p.aff(p));
System.out.println(p.aff(pr));
System.out.println(p.aff(r));

// Partie pr
System.out.println(pr.aff(p));
System.out.println(pr.aff(pr));
System.out.println(pr.aff(r));

// Partie r
System.out.println(r.aff(p));
System.out.println(r.aff(pr));
System.out.println(r.aff(r));
```

Exercice 5 : Diagramme de classes

Dessiner le diagramme de classes modélisant l'intégralité des données du sujet de TP.

Programmation Orientée Objets

TP n°3 : Composition et Héritage

Consigne : à partir du TP n°3, rendu de TP = un fichier .jar (fichiers source) + diagramme UML.

Exercice 1 : Classe « Point »

Créer un projet « `Dessin` » contenant un package « `dessin` ». Définir une classe `Point` qui modélise un point dans un espace à 2 dimensions (les coordonnées sont représentées sous forme de réels). En plus des constructeurs et accesseurs (dont un constructeur vide qui positionne aléatoirement les coordonnées dans l'intervalle [0;300[) cette classe doit fournir une méthode `double distance(Point p)` telle que, si `p1` et `p2` sont des instances de `Point` :

- `p1.distance(p2)` retourne la distance qui sépare les points `p1` et `p2`.
- `p1.distance()` retourne la distance de `p1` par rapport à l'origine.

NB : `Math.sqrt(double d)` retourne $\sqrt{d}$. `Math.pow(double a, double b)` retourne a^b.
NB : `Math.random()` retourne une valeur réelle aléatoire comprise dans l'intervalle [0;1[.
Ajouter une classe « `Test` » (qui contient le « `main` »), qui sera utilisée à la fin de chaque exercice pour tester soigneusement les nouveaux ajouts.

Exercice 2 : Classe « Polygone »

Définir une classe `Polygone` qui modélise un polygone dans un espace à 2 dimensions. Le nombre de côtés du polygone est donné en paramètre de construction et ne pourra pas être changé par la suite, par contre les points définissant les extrémités du polygone pourront être manipulés. La classe doit fournir les méthodes :

- `int nbrCotes()`: retourne le nombre de côtés du polygone.
- `Point getPoint(int n)` : retourne le point numéro `n` (numérotés de 0 a (N-1)).
- `void changePoint(int n, Point p)` et `void changePoint(int n, double x, double y)`.
- `double perimetre()` calcule et retourne la valeur du périmètre du polygone.

S'agit-il d'une agrégation faible ou d'une agrégation forte ?

Exercice 3 : Polygone aléatoire (facultatif)

Modifier le constructeur de `Polygone` afin d'instancier par défaut un polygone dont les coordonnées des points sont choisies au hasard dans l'intervalle [0;300[, puis modifiées au hasard point par point jusqu'à ce que le périmètre du polygone atteigne (ou dépasse) la valeur 600.

Exercice 4 : Triangle, Quadrilatère et Rectangle

Écrire une classe `Triangle`, qui hérite de `Polygone`, telle qu'il soit possible de créer un triangle à partir de 3 `Point`, et que l'on puisse à tout moment changer la position d'un des sommets du triangle.
Écrire une classe `Quadrilatère` puis une classe `Rectangle`. On supposera que les rectangles issus de cette classe sont horizontaux. Définir une méthode `construire(Point p1, Point p2)` dans `Rectangle` qui utilise `p1` et `p2` comme points 0 et 2 (supposés être les points opposés, haut gauche et bas droite) et qui en déduit les deux autres points (dans l'ordre, haut droite et bas gauche). En tirer un constructeur `Rectangle(Point p1, Point p2)`.

Exercice 5 : Hauteur, Largeur, Surface

Définir les méthodes `double hauteur()`, `double largeur()` et `double surface()` dans `Rectangle`.
Écrire les méthodes `surface()` de `Polygone` et de toutes ses sous-classes, quand c'est nécessaire.
Utiliser la formule de Héron et considérer pour simplifier que les polygones manipulés sont convexes.

Exercice 6 : Ensemble de polygones

Définir une classe `EnsemblePolygone` qui comprend un tableau de polygones dont la taille maximale est définie lors de la création de cet ensemble. Définir une méthode `ajouterPolygone(Polygone pol)` qui ajoute un polygone à cet ensemble (à condition que la capacité de cet ensemble ne soit pas dépassé).
Définir `sommePerimetres()` qui retourne la somme de tous les périmètres des polygone de l'ensemble.

Exercice 7 : toString()

Redéfinir la méthode `toString()` des polygones afin que chaque polygone affiche l'information suivante `<classe de polygone>: <p0>, ..., <pn>`, où chaque point s'écrit `(<x>,<y>)`. On redéfinira ainsi la méthode `toString()` de `Point`. On pourra utiliser la méthode `String getClass().getSimpleName()`.

Exercice 8 : Tous les polygones

Définir dans `EnsemblePolygone`, une méthode `affiche()` qui affiche tous les polygones qu'il contient.