Variations autour de la relation Segment-Point

Exercice 1 : Caractériser la relation entre Segment et Point

L'objectif de cet exercice est de comprendre les notions de composition et d'agrégation en s'appuyant sur les classes Point (listing 2) et Segment (listing 3).

1.1 Que donne l'exécution du programme du listing 1 ? En déduire la nature de la relation entre les classes Point et Segment (agrégation ou composition).

Listing 1 – La classe TestRelationSegmentPoint

```
1  /** Programme illustrant la relation entre les classes Segment et Point
2    * (agrégation ou composition ?).
3    * @author     Xavier Crégut (cregut@enseeiht.fr)
4    * @version    1.1
5    */
6  public class TestRelationSegmentPoint {
7
8      public static void main(String[] args) {
9          // Créer trois points
10         Point p1 = new Point(3, 2);
11         Point p2 = new Point(6, 9);
12         Point p3 = new Point(11, 4);
13
14         // Créer deux segments à partir de ces trois points
15         // (le point p2 est utilisé pour constuire s12 et s23)
16         Segment s12 = new Segment(p1, p2);
17         Segment s23 = new Segment(p2, p3);
18
19         // Afficher les deux segments
20         System.out.print("s12 = "); s12.afficher(); System.out.println();
21         System.out.print("s23 = "); s23.afficher(); System.out.println();
22
23         // Translater le point p2 qui a servi à initialiser les segments
24         System.out.println("> p2.translater(5, -10);");
25         p2.translater(5, -10);
26
27         // Afficher les deux segments
28         System.out.print("s12 = "); s12.afficher(); System.out.println();
29         System.out.print("s23 = "); s23.afficher(); System.out.println();
30     }
31 }
```

1.2 Comment faire pour obtenir l'autre relation ?

Exercice 2 : Composition et polymorphisme

Listing 2 – La classe Point

```
1  /** Définition d'un point avec ses coordonnées cartésiennes.
2   *  @author     Xavier Crégut (cregut@enseeiht.fr)
3   *  @version    1.2
4   */
5  public class Point
6  {
7      private double x;           // abscisse
8      private double y;           // ordonnée
9
10     /** Construire un point à partir de son abscisse et de son ordonnée.
11       * @param  x       abscisse
12       * @param  y       ordonnée */
13     public Point(double x_, double y_) {
14         x = x_;
15         y = y_;
16     }
17
18     /** Abscisse du point */
19     public double getX()                { return x; }
20
21     /** Ordonnée du point */
22     public double getY()                { return y; }
23
24     /** Changer l'abscisse du point
25       * @param x_       la nouvelle abscisse
26       */
27     public void setX(double x_)         { x = x_; }
28
29     /** Changer l'ordonnée du point
30       * @param y_       la nouvelle ordonnée
31       */
32     public void setY(double y_)         { y = y_; }
33
34     /** Afficher le point */
35     public void afficher() {
36         System.out.print("(" + getX() + "," + getY() + ")");
37     }
38
39     /** Distance par rapport à un autre point */
40     public double distance(Point autre) {
41         return Math.sqrt(Math.pow(autre.getX() - getX(), 2)
42                          + Math.pow(autre.getY() - getY(), 2));
43     }
44
45    /** Translater le point.
46      * @param dx_ déplacement suivant l'axe des X
47      * @param dy_ déplacement suivant l'axe des Y
48      */
49     public void translater(double dx_, double dy_) {
50         x += dx_;
51         y += dy_;
52     }
53 }
```

Listing 3 – La classe Segment

```
1  /** La classe Segment décrit un segment en fonction de ses deux points
2   *  extrémités.
3   *  @author     Xavier Crégut (cregut@enseeiht.fr)
4   *  @version    1.2
5   */
6  public class Segment
7  {
8      private Point extrémité1;
9      private Point extrémité2;
10
11     /**  Construire un Segment à partir de ses deux points extrémités
12       *  @param ext1    le premier point extrémité
13       *  @param ext2    le deuxième point extrémité
14       */
15     public Segment(Point ext1, Point ext2) {
16         extrémité1 = ext1;
17         extrémité2 = ext2;
18     }
19
20    /** Translater le segment.
21      * @param dx_ déplacement suivant l'axe des X
22      * @param dy_ déplacement suivant l'axe des Y
23      */
24     public void translater(double dx, double dy) {
25         extrémité1.translater(dx, dy);
26         extrémité2.translater(dx, dy);
27     }
28
29     /** Longueur du segment */
30     public double getLongueur() {
31         return extrémité1.distance(extrémité2);
32     }
33
34     /** Afficher le segment.  Le segment est affiché sous la forme :
35       * <PRE>
36       *         [extrémité1-extrémité2]
37       * </PRE>
38       */
39     public void afficher() {
40         System.out.print("[");
41         extrémité1.afficher();
42         System.out.print("-");
43         extrémité2.afficher();
44         System.out.print("]");
45     }
46 }
```

Dans l'exercice 1 nous avons proposé une manière de réaliser en Java la relation de composition. Nous allons en voir les limites et proposer une meilleure solution.

2.1 Peut-on créer un segment à partir de points et points nommés ? Pourquoi ?

2.2 On modifie le programme du listing 1 pour créer les points p1 et p2 comme des points nommés. La déclaration des trois points devient alors :

```
1 // Créer trois points
2 PointNommé p1 = new PointNommé(3, 2, "p1");
3 PointNommé p2 = new PointNommé(6, 9, "p2");
4 Point p3 = new Point(11, 4);
```

2.2.1 Indiquer ce qui se passe quand on exécute ce programme avec la classe Segment proposée à la question 1.2 (celle identifiée comme réalisant la relation de composition). Expliquer.

2.2.2 Indiquer ce qui se passe quand on exécute ce programme avec la classe Segment version initiale de la classe Segment (celle identifiée comme réalisant la relation d'agrégation).

2.3 Nous allons voir comment faire pour avoir à la fois la notion de composition (les extrémités du segment sont indépendantes des points ayant servi à initialiser le segment) et le polymorphisme (lorsqu'on affiche un segment construit à partir de points nommés, le nom des points nommés apparaît).

2.3.1 Proposer une solution s'appuyant sur la surcharge et la commenter.

2.3.2 Proposer une autre solution.