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 RelationSegmentPoint

```
/** Programme illustrant la relation entre les classes Segment et Point
 * (agrégation ou composition ?). */
public class RelationSegmentPoint {

   public static void main(String[] args) {
      // Créer trois points
      Point p1 = new Point(3, 2);
      Point p2 = new Point(6, 9);
      Point p3 = new Point(11, 4);

      // Créer deux segments à partir de ces trois points
      // (le point p2 est utilisé pour constuire s12 et s23)
      Segment s12 = new Segment(p1, p2);
      Segment s23 = new Segment(p2, p3);

      // Afficher les deux segments
      System.out.println("s12 = " + s12);
      System.out.println("s23 = " + s23);

      // Translater le point p2 qui a servi à initialiser les segments
      System.out.println("> p2.translater(5, -10);");
      p2.translater(5, -10);

      // Afficher les deux segments
      System.out.println("s12 = " + s12);
      System.out.println("s23 = " + s23);
} }
```

1.2 Comment faire pour obtenir l'autre relation ?

Listing 2 – La classe Point

```
/** Définition d'un point avec ses coordonnées cartésiennes. */
public class Point {
   private double x;  // abscisse
   private double y;  // ordonnée

   /** Construire un point à partir de son abscisse et de son ordonnée.
    * @param x abscisse
    * @param y ordonnée */
   public Point(double x, double y) {
      this.x = x;
      this.y = y;
   }

   /** Abscisse du point */
   public double getX() {
      return x;
   }

   /** Ordonnée du point */
   public double getY() {
      return y;
   }

   /** Changer l'abscisse du point
    * @param x la nouvelle abscisse */
   public void setX(double x) {
      this.x = x;
   }

   /** Changer l'ordonnée du point
    * @param y la nouvelle ordonnée */
   public void setY(double y) {
      this.y = y;
   }

   @Override public String toString() {
      return "(" + x + "," + y + ")";
   }

   /** Distance par rapport à un autre point */
   public double distance(Point autre) {
      double dx2 = Math.pow(autre.x - x, 2);
      double dy2 = Math.pow(autre.y - y, 2);
      return Math.sqrt(dx2 + dy2);
   }

  /** Translater le point.
   * @param dx déplacement suivant l'axe des X
   * @param dy déplacement suivant l'axe des Y */
   public void translater(double dx, double dy) {
      x += dx;
      y += dy;
} }
```

Listing 3 – La classe Segment

```
/** Un segment caractérisé par ses deux points extrémités. */
public class Segment {
   private Point extremite1;
   private Point extremite2;

   /** Construire un Segment à partir de ses deux points extrémités
    * @param ext1 le premier point extrémité
    * @param ext2 le deuxième point extrémité */
   public Segment(Point ext1, Point ext2) {
      extremite1 = ext1;
      extremite2 = ext2;
   }

  /** Translater le segment.
   * @param dx déplacement suivant l'axe des X
   * @param dy déplacement suivant l'axe des Y */
   public void translater(double dx, double dy) {
      extremite1.translater(dx, dy);
      extremite2.translater(dx, dy);
   }

   /** Longueur du segment */
   public double getLongueur() {
      return extremite1.distance(extremite2);
   }

   @Override public String toString() {
      return "[" + extremite1 + "-" + extremite2 + "]";
} }
```


Exercice 2 : Composition et polymorphisme

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 (listing 4) ? Pourquoi ?

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

```
// Créer trois points
PointNomme p1 = new PointNomme("p1", 3, 2);
PointNomme p2 = new PointNomme("p2", 6, 9);
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 Voyons comment 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.

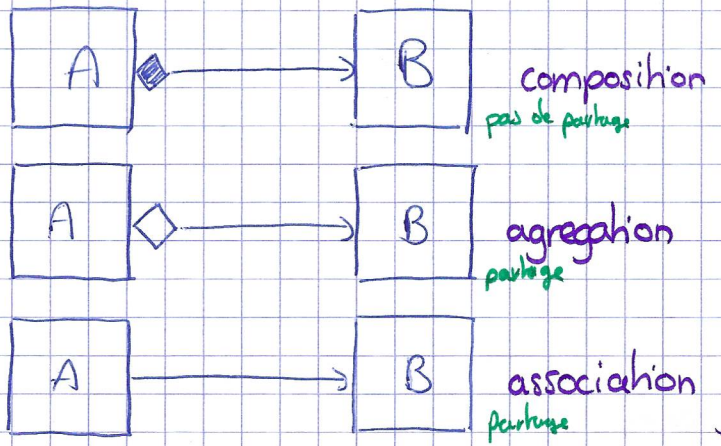
2.3.2 Proposer une autre solution.

Listing 4 – La classe PointNomme

```
/** Un point nommé est un point avec un nom. */
public class PointNomme extends Point {
   private String nom;

   /** Construire un point nommé. */
   public PointNomme(String nom, double x, double y) {
      super(x, y);
      this.nom = nom;
   }

   /** Nom du point nommé */
   public String getNom() {
      return nom;
   }

   /** Changer le nom du point nommé
    * @param nom le nouveau nom */
   public void setNom(String nom) {
      this.nom = nom;
   }

   @Override public String toString() {
      return nom + ":" + super.toString();
} }
```

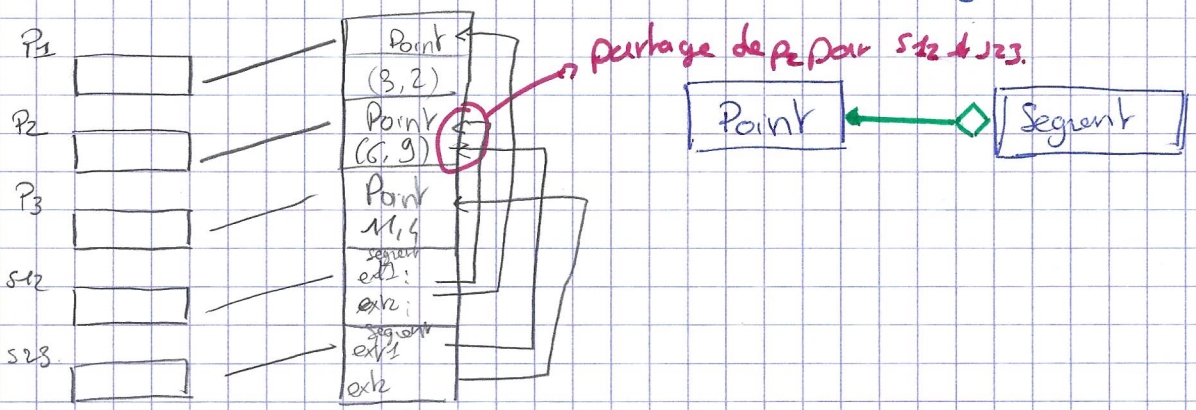
TD 6.1
TOB

Variations autour de la relation Segment-Point.



en JAVA : une seule forme class A { ... }

Exercice 1 : Caractériser une relation entre Segment et Point.



S12 = [(3,2) - (6,9)]
S23 = [(6,9) - (11,4)].
> p2.translater(5, -10);
S12 = [(3,2) - (11,-1)].
S23 = [(11,-1) - (11,4)].

```
class Segment {
public Segment(Point ext1, Point ext2)
public
    extremite1 = new Point(ext1.getX(), ext1.getY());
    extremite2 = new Point(ext2.getX(), ext2.getY());
```

Exercice 2 : p2 = new PointNomme

```
PointNomme p2 = new PointNomme()
Point p3 = new (Point
Segment s23 = new (Segment (p2, p3)
```

p2 est un PointNomme, mais aussi un point → Polymorphisme

Segment (PointNone ext1, PointNone ext2)

~~if (ext1 instance of Point~~

if (ext1 instance of PointNone)
extremité1 = new PointNone

} else {

if (ext1 instance of PointNone)
extremité1 = new PointNone

class Point { implement CloneAble

public object clone() {

return new Point (this.x, this.y);

};

super.clone()

catch CloneNotSupported()

class PointNone (...) {

public Point clone() {

return new PointNone ();

}

Segment (...)

ext1 = ext1.clone()
ext2 = ext2.clone()