Encore les segments et les points !

L'objectif de ces exercices est de programmer en Java le modèle d'analyse présenté à la figure 1 en ayant comme contrainte de réalisation d'avoir un attribut dans la classe Segment pour stocker la longueur du segment.

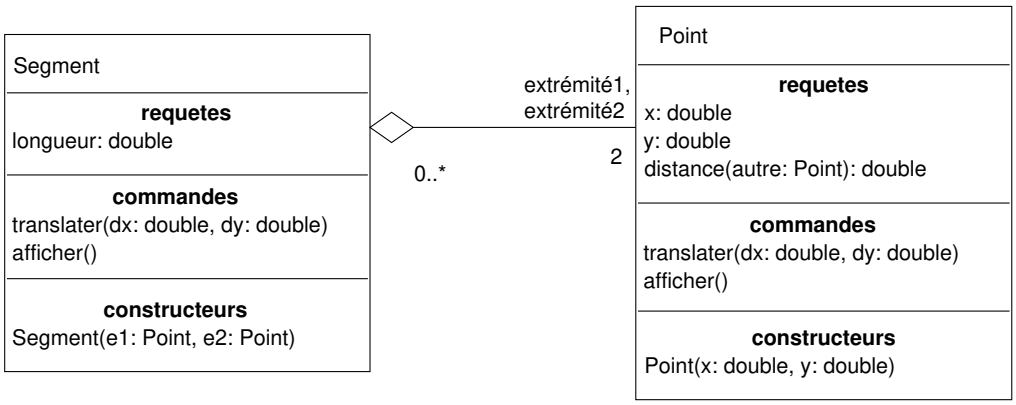

FIGURE 1 – Le diagramme de classe initial des classes Segment et Point

Exercice 1 : Préparer la phase de validation

Avant de commencer la programmation en Java du modèle d'analyse, commençons par spécifier les scénarios de test qui nous permettront de la valider. Chaque scénario permettra d'écrire un programme de test qui sera utilisé pour tester les classes de l'application écrite.

Nous ne décrivons ici que le premier scénario qui consiste à :

- créer un point p1 de coordonnées (0, 0) ;
- créer un point p2 de coordonnées (5, 0) ;
- créer un segment s à partir des points p1 et p2 ;
- afficher les coordonnées du point p2 ;
- afficher le segment s ;
- afficher la longueur du segment s ;
- translater le point p2 du vecteur (-2, 0) ;
- afficher les coordonnées du point p2 ;
- afficher le segment s ;
- afficher la longueur du segment s ;

1.1. Indiquer les résultats qui *devront* être affichés à l'écran par ce scénario de test.

1.2. Dessiner le diagramme de séquence qui correspond à ce scénario.

Exercice 2 : Première implantation

Une première version des classes Point (listing 3) et Segment (listing 4) a été écrite. Le programme de test correspondant au premier scénario de test (exercice 1) est donné au listing 1.

Listing 1 – La classe TestSegment1

```
/** Premier programme de Test de la classe Segment.  */
public class TestSegment1 {
    public static void main(String[] args) {
        Point p1 = new Point(0, 0);
        Point p2 = new Point(5, 0);
        Segment s = new Segment(p1, p2);

        System.out.print("p2 = ");  p2.afficher();  System.out.println();
        System.out.print("s = ");   s.afficher();   System.out.println();
        System.out.println("longueur de s = " + s.getLongueur());

        p2.translater(-2, 0);

        System.out.print("p2 = ");  p2.afficher();  System.out.println();
        System.out.print("s = ");   s.afficher();   System.out.println();
        System.out.println("longueur de s = " + s.getLongueur());
    }
}
```

2.1. Dessiner le diagramme de classe d'implantation correspondant à cette première solution.
2.2. Compléter le diagramme de séquence dessiné à la question 1.2.
2.3. Indiquer ce qu'affiche le programme de test. On pourra éventuellement dessiner l'état de la mémoire à la fin de l'exécution du programme.
2.4. Commenter les résultats obtenus.

Exercice 3 : Correction des classes Segment et Point

Nous allons maintenant corriger les classes Segment et Point pour qu'elles respectent le cahier des charges imposé. La solution proposée doit donc respecter les contraintes suivantes :

- la relation entre Segment et Point reste inchangée ;
- l'attribut `longueur` et la méthode `getLongueur()` de Segment restent inchangés.

3.1. Indiquer, sur le diagramme de séquence (question 2.2), les modifications à faire.
3.2. Compléter le diagramme de classe.
3.3. Écrire le code des méthodes modifiées et des nouvelles méthodes.

Exercice 4 : Gestion de la mémoire

La relation d'agrégation spécifiée entre Segment et Point indique qu'un même point peut être l'extrémité de plusieurs segments.
4.1. Cette propriété remet-elle en cause la solution proposée ? Donner les éventuelles modifications à apporter.
4.2. On considère le programme du listing 2. Indiquer combien de segments sont récupérés par le ramasse-miettes. Expliquer pourquoi et proposer une amélioration à la solution proposée.

Listing 2 – La classe TestSegment3

```
/** Programme de Test de la classe Segment. */
public class TestSegment3 {
```

```
   public static void main(String[] args) {
      Point p1 = new Point(1, 2);

      for (int i = 0; i < 100; i++) {
         Segment s = new Segment(new Point(i, i), p1);
      }

      System.out.println("Translation␣!");
      p1.translater(10, 0);
      System.out.println("Fin␣!");
}  }
```

Exercice 5 : Polygone et Point

La solution fonctionne correctement pour les classes Point et Segment. Considérons maintenant une classe Polygone. Un polygone est défini comme un ensemble de sommets. Comme dans le cas du segment, on décide de stocker le périmètre du polygone. Il doit donc être recalculé dès qu'un de ses sommets est translaté.

Adapter la classe Point pour prendre en compte les segments mais aussi les polygones.

Exercice 6 : Cercle et Point

Considérons maintenant un Cercle défini par son centre et un point de sa circonférence. On considère que le rayon du cercle est une information stockée et qu'il y a une relation d'agrégation entre le cercle et ses deux points. La translation de l'un de ces deux points a donc un impact sur le cercle :

— une translation du centre du cercle provoque une translation du point de la circonférence ;

— une translation du point de la circonférence change la taille du cercle (le centre reste inchangé) et nécessite donc de mettre à jour le rayon du cercle.

Adapter les classes pour prendre en compte les cercles.

Listing 3 – La classe Point

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

   /** Construction d'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 this.x;
   }

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

   /** Distance par rapport à un autre point. */
```

```
    public double distance(Point autre) {
       return Math.sqrt(Math.pow(autre.x - this.x, 2)
                + Math.pow(autre.y - this.y, 2));
    }
   /** 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) {
       this.x += dx;
       this.y += dy;
    }
    /** Afficher le point. */
    public void afficher() {
       System.out.print("(" + this.x + "," + this.y + ")");
    }
}
```

Listing 4 – La classe Segment

```
/** Définition d'un segment. */
public class Segment {
    private Point extremite1;
    private Point extremite2;
    private double longueur;

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

    /** Longueur du segment. */
    public double getLongueur() {
       return this.longueur;
    }

    /** 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) {
       this.extremite1.translater(dx, dy);
       this.extremite2.translater(dx, dy);
    }

    /** Afficher le segment. */
    public void afficher() {
       System.out.print("[");
       this.extremite1.afficher();
       System.out.print(";");
       this.extremite2.afficher();
       System.out.print("]");
    }
}
```