TP1 : Classes et classes de test.

POO : Programmation Orientée Objet

Première année

★ **Exercice 1.** L'objectif de ce premier exercice est de concevoir et réaliser les classes de bases d'un programme manipulant des points, des segments, des triangles et d'autres formes géométriques.

▷ **Question 1. L'objet `Point`**

(a) Décrivez (en langage naturel) le comportement que l'on peut attendre d'un objet `Point`. Réalisez la carte CRC correspondante.

(b) Spécifiez l'interface publique d'un objet `Point`.

(c) Spécifiez les profils des constructeurs de la classe `Point`.

(d) Implémentez la classe `Point` en écrivant le corps des méthodes de l'interface publique.

▷ **Question 2. La classe de test `TestPoint` de la classe `Point`**

Écrivez une classe de test qui teste le comportement réel de la classe `Point` et le compare au comportement attendu. Ci-dessous, vous trouverez un exemple de sortie d'une classe de test.

```
[ok]     (new Point()).getX() == 0.
[ok]     (new Point()).getY() == 0.
[ok]     (new Point(3., 2.)).getX() == 3.
[ok]     (new Point(3., 2.)).getY() == 2.
[ok]     (new Point(new Point(3., 2.))).getX() == 3.
[ok]     (new Point(new Point(3., 2.))).getY() == 2.
[ok]     (new Point()).setX(4.).getX() == 4.
[echec]  (new Point()).setY(7.).getY() == 7. -- attendu: 7.  obtenu: 8.
[ok]     (new Point(2.,3.)).translater(4.,7.)
[ok]     (new Point(2.,3.)).translater(4.,7.)
```

▷ **Question 3. L'objet `Segment`**

(a) Décrivez (en langage naturel) le comportement que l'on peut attendre d'un objet `Segment`. Réaliser la carte CRC correspondante.

(b) Spécifiez l'interface publique d'un objet `Segment`.

(c) Spécifiez les profiles des constructeurs de la classe `Segment`.

(d) Implémentez la classe `Segment` en écrivant le corps des méthodes de l'interface publique.

▷ **Question 4. La classe de test `TestSegment` de la classe `Segment`**

Écrivez une classe de test qui teste le comportement réel de la classe `Segment` et le compare automatiquement au comportement attendu.

▷ **Question 5. L'objet `Triangle` (optionnelle)**

Suivez la même procédure que précédemment pour réaliser la classe `Triangle`.

★ **Exercice 2.**

▷ **Question 6.** Concevez une classe `FractionImpl` permettant de représenter des nombres rationnels. Cette classe doit permettre :

- de consulter et de modifier le numérateur et le dénominateur du nombre,
- de multiplier un nombre rationnel par un coefficient entier,
- d'ajouter, de soustraire, de multiplier, et de diviser deux nombres rationnels,
- d'inverser un nombre rationnel,
- de comparer deux nombres rationnels,
- de réduire un nombre rationnel.

Vous trouverez ci-dessous l'interface `Fraction` que doit implémenter la classe `FractionImpl`. Ce code java est également disponible dans le fichier `Fraction.java`.

```
/**
 * Definit une fraction qui représente un nombre rationnel<br/>
 * Attention : le signe est au numerateur<br/>
 * Attention : le denominateur sera toujours different de 0<br/>
 * @see 01-tp-basic-objects-enonce.pdf
 */
public interface Fraction {
```

```
8
9       /**
10       * Calcule le pgcd de deux entiers
11       * @param a un des deux entiers
12       * @param b un des deux entiers
13       * @return le pgcd des deux entiers
14       */
15      //public static int pgcd(int a, int b) ;
16
17      /**
18       * Constructeur d'une fraction
19       * @param num numerateur
20       * @param den denominateur
21       */
22      //public Fraction(int num, int den) ;
23
24      /**
25       * Retourne le numerateur de la fraction
26       * @return le numerateur
27       */
28      public int getNumerateur() ;
29
30      /**
31       * Retourne le denominateur de la fraction
32       * @return le denominateur
33       */
34      public int getDenominateur() ;
35
36      /**
37       * Change la valeur du denominteur
38       * @param nd le nouveau denominateur
39       */
40      public void setDenominateur(int nd) ;
41
42      /**
43       * Change la valeur du numerateur
44       * @param nd le nouveau numerateur
45       */
46      public void setNumerateur(int nn) ;
47
48      /**
49       * Test si la fraction est egale a une autre fraction
50       * @param f fraction a comparer
51       * @return vrai si les deux fractions sont egales
52       */
53      public boolean egaleA(Fraction f) ;
54
55      /**
56       * Ajoute une fraction a la fraction
57       * @param f fraction a ajouter
58       */
59      public void ajoute(Fraction f) ;
60
61      /**
62       * reduit/simplifie la fraction
63       */
64      public void reduire() ;
65
66      /**
67       * Inverse la fraction<br>
68       * Le signe est au numerateur.
69       */
70      public void inverse() ;
71
72      /**
73       * Multiplie la fraction par un coefficient
74       * @param i coefficient multiplicateur
75       */
76      public void multiplierParCoeff(int i) ;
77
78  }
```

Pour réaliser la méthode `public void reduire()`, vous utiliserez la méthode donnée ci-dessous (recopiez la définition de cette méthode dans votre classe `FractionImpl`) :

```
1  public static int pgcd(int a, int b) {
2    int c;
3    while (b != 0) {
4      c = a % b;
5      a = b;
6      b = c;
7    }
8    return a;
9  }
```

Conseil : Procédez par étapes :

(a) écrivez une méthode de la classe `FractionImpl`,

(b) décommentez uniquement les tests correspondants à cette méthode dans la classe `TestFraction` qui vous est fournie. (cf. fichier `TestFraction.java`),

(c) puis continuez avec la prochaine méthode que vous devez implémenter.

▷ **Question 7.** Complétez la classe de test `TestFraction` afin de tester le comportement des méthodes `multiplie(Fraction f)`, `soustrait(Fraction f)` et `divise(Fraction f)`.