

- ★ **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.

Réponse

Cet exercice est en partie (questions 1 et 2) à résoudre avec les élèves pas à pas. On peut tout à fait envisager de se brancher sur le vidéo projecteur et d'avancer pas à pas en même temps que les élèves.

Dans le cas où l'on résout l'exercice pas à pas, il ne faut pas suivre l'énoncé tel quel. Il vaut mieux :

- faire la carte CRC (question 1)
- puis, réaliser l'interface publique et l'implémentation de la classe méthode par méthode en écrivant un programme client (illustrant l'utilisation de la classe) en parallèle.
- enfin, écrire la classe de test (ou leur donner)

Fin réponse

▷ **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.

Réponse

Comportement attendu d'un point :

- consulter/modifier sont abscisse et son ordonnée
- déplacer le point (translation)

Remarque : on ne s'intéresse pas "à l'intérieur" de l'objet (boîte noire).

Point	
Responsabilités	Collaborateurs
<ul style="list-style-type: none"> – Conserve les coordonnées du point – Permet de consulter les coordonnées du point – Permet de modifier les coordonnées du point – Permet de traduire le point 	

Fin réponse

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

Réponse

Méthode de la classe Point.

```
1 public double getX()
2 public double getY()
3 public void setX(double newX)
4 public void setY(double newY)
5 public void traduire(double dx, double dy)
```

Remarque : peu importe si on représente le point sous la forme (x, y) ou $(norme, \theta)$.

Fin réponse

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

Réponse

```
1 public Point() // construit l'origine
2 public Point(double someX, double someY)
3 public Point(Point p) // copy-constructor
```

Fin réponse

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

Réponse

```

1 public class Point {
2     private double x;
3     private double y;
4
5     public Point() {
6         this.x = 0.;
7         this.y = 0.;
8     }
9
10    public Point(double someX, double someY) {
11        this.x = someX;
12        this.y = someY;
13    }
14
15    public Point(Point p) {
16        this.x = p.x;
17        this.y = p.y;
18    }
19
20    public double getX() {
21        return this.x;
22    }
23
24    public double getY() {
25        return this.y;
26    }
27
28    public void setX(double newX) {
29        this.x = newX;
30    }
31
32    public void setY(double newY) {
33        this.y = newY;
34    }
35
36    public void translater(double dx, double dy) {
37        this.x += dx;
38        this.y += dy;
39    }
40 }

```

Il faut insister sur beaucoup de choses : la syntaxe, les mot-clés (**public**, **void**, **this**), les conventions de nommage des classes et des méthodes, l'indentation (même si au tableau ce n'est pas simple).

Remarque : Ici on a choisi l'implémentation interne (x, y). On peut également demander aux élèves de réécrire l'implémentation en utilisant (*norme, θ*).

Fin réponse

► 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.

```

1 [ok] (new Point()).getX()
2 [ok] (new Point()).getY()
3 [ok] (new Point(3., 2.)).getX()
4 [ok] (new Point(3., 2.)).getY()
5 [ok] (new Point(new Point(3., 2.))).getX()
6 [ok] (new Point(new Point(3., 2.))).getY()
7 [ok] p.setX(4.).getX()
8 [ok] p.setY(7.).getY()
9 [ok] (new Point(2.,3.)).translater(4.,7.)
10 [ok] (new Point(2.,3.)).translater(4.,7.)

```

Réponse

- chaque méthode doit être testée (couverture des tests)
- le test doit comparer son résultat au résultat attendu (affichage du test, du résultat, de la valeur attendu; éventuellement on compte les tests ratés)
- notez que l'on aurait pu écrire la classe de test au moment de la définition de l'interface publique (base du développement orienté test)

```

1 public class TestPoint {
2
3     public static void main(String[] args) {
4         (new TestPoint()).run();
5     }
6 }

```

```

6
7 public void run() {
8     Point p = new Point();
9     double expected = 0.;
10    double value = p.getX();
11    check("(new Point()).getX()", value == expected, "expected="+expected+" value="+value);
12
13    value = p.getY();
14    check("(new Point()).getY()", value == expected, "expected="+expected+" value="+value);
15
16    p = new Point(3.,2.);
17
18    expected = 3.;
19    value = p.getX();
20    check("(new Point(3., 2.)).getX()", value == expected, "expected="+expected+" value="+value);
21
22    expected = 2.;
23    value = p.getY();
24    check("(new Point(3., 2.)).getY()", value == expected, "expected="+expected+" value="+value);
25
26    Point p2 = new Point(p);
27
28    expected = 3.;
29    value = p2.getX();
30    check("(new Point(new Point(3., 2.)).getX()", value == expected, "expected="+expected+" value="+value);
31
32    expected = 2.;
33    value = p2.getY();
34    check("(new Point(new Point(3., 2.)).getY()", value == expected, "expected="+expected+" value="+value);
35
36    p.setX(4.);
37
38    expected = 4.;
39    value = p.getX();
40    check("p.setX(4.).getX()", value == expected, "expected="+expected+" value="+value);
41
42    p.setY(7.);
43
44    expected = 7.;
45    value = p.getY();
46    check("p.setY(7.).getY()", value == expected, "expected="+expected+" value="+value);
47
48    p = new Point(2.,3.);
49    p.translater(4.,7.);
50    double exp1 = 2.+4.;
51    double exp2 = 3.+7.;
52    double v1 = p.getX();
53    double v2 = p.getY();
54
55    check("(new Point(2.,3.).translater(4.,7.)", v1 == exp1, "exp1="+exp1+" v1="+v1);
56    check("(new Point(2.,3.).translater(4.,7.)", v2 == exp2, "exp2="+exp2+" v2="+v2);
57 }
58
59
60 public void check(String message, boolean condition, String debug) {
61     if (! condition) {
62         System.out.print("[failed]\\t"+debug+" - ");
63     } else {
64         System.out.print("[ok]\\t");
65     }
66     System.out.println(message);
67 }
68
69 }

```

Fin réponse

▷ **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.

Réponse

Segment	
Responsabilités	Collaborateurs
<ul style="list-style-type: none"> – Conserve les coordonnées du segment – Permet de consulter les coordonnées du segment – Permet de modifier les coordonnées du segment – Permet de traduire le segment 	<ul style="list-style-type: none"> – Point (ou pas)

Fin réponse

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

Réponse

Méthode de la classe **Segment**.

```

1 public double getX1()
2 public double getY1()
3 public double getX2()
4 public double getY2()
5
6 public void setX1(double newX1)
7 public void setY1(double newY1)
8 public void setX2(double newX2)
9 public void setY2(double newY2)-
10
11 public void traduire(double dx, double dy)
12
13 // on pourrait ajouter (remplacer) par
14
15 public Point getPoint1()
16 public Point getPoint2()
17 public void setPoint1(Point p1)
18 public void setPoint2(Point p2)

```

Remarque : peu importe si on représente le segment avec 2 points ou 4 coordonnées.

Fin réponse

(c) Spécifiez les profils des constructeurs de la classe **Segment**.

Réponse

```

1 public Segment() // construit à l'origine un segment de longueur 0
2 public Segment(double x1, double y1, double x2, double y2)
3 public Segment(Point p1, Point p2) // ou pas
4 public Segment(Segment s) // copy-constructor

```

Fin réponse

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

Réponse

On peut procéder en deux étapes :

- (a) sans ré-utiliser la classe **Point**. On utilise donc 4 variables d'instance de type **double**.
- (b) en ré-utilisant la classe **Point**. Notez la ré-utilisation du code et la clarté du code obtenu.

```

1 public class Segment {
2     private Point p1;
3     private Point p2;
4
5     public Segment() {
6         this.p1 = new Point();
7         this.p2 = new Point();
8     }
9
10    public Segment(double x1, double y1, double x2, double y2) {
11        this.p1 = new Point(x1, y1);
12        this.p2 = new Point(x2, y2);
13    }
14
15    public Segment(Point p1, Point p2) {
16        this.p1 = p1;

```

```

17     this.p2 = p2;
18 }
19
20 public Segment(Segment s) {
21     this.p1 = new Point(s.p1);
22     this.p2 = new Point(s.p2);
23 }
24
25 public double getX1() {
26     return this.p1.getX();
27 }
28
29 public double getY1() {
30     return this.p1.getY();
31 }
32
33 public double getX2() {
34     return this.p2.getX();
35 }
36
37 public double getY2() {
38     return this.p2.getY();
39 }
40
41 public void setX1(double newX1) {
42     this.p1.setX(newX1);
43 }
44
45 public void setY1(double newY1) {
46     this.p1.setY(newY1);
47 }
48
49 public void setX2(double newX2) {
50     this.p2.setX(newX2);
51 }
52
53 public void setY2(double newY2) {
54     this.p2.setY(newY2);
55 }
56
57
58 public void translater(double dx, double dy) {
59     this.p1.translater(dx,dy);
60     this.p2.translater(dx,dy);
61 }
62 }

```

Fin réponse

▷ **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**

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

Réponse

Remarque : Surtout pour occuper les plus rapides.

Fin réponse

★ Exercice 2.

▷ **Question 6.** Concevez une classe **Fraction** 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 publique de la classe **Fraction** à implémenter.

```

1  /**
2  * Calcule le pgcd de deux entiers
3  * @param a un des deux entiers
4  * @param b un des deux entiers
5  * @return le pgcd des deux entiers
6  */
7  public static int pgcd (int a, int b) ;
8
9  /**
10 * Constructeur d'une Fraction
11 * @param num Numerateur
12 * @param den Denominateur
13 */
14 public Fraction(int num, int den) ;
15
16 /**
17 * Retourne le numerateur de la Fraction
18 * @return le numerateur
19 */
20 public int getNumerateur() ;
21
22 /**
23 * Retourne le denominateur de la Fraction
24 * @return le denominateur
25 */
26 public int getDenominateur() ;
27
28 /**
29 * Change la valeur du denominteur
30 * @param nd le nouveau denominateur
31 */
32 public void setDenominateur(int nd) ;
33
34 /**
35 * Change la valeur du numerateur
36 * @param nd le nouveau numerateur
37 */
38 public void setNumerateur(int nn) ;
39
40 /**
41 * Test si la Fraction est egale a une autre Fraction
42 * @param f Fraction a comparer
43 * @return vrai si les deux Fractions sont egales
44 */
45 public boolean egaleA(Fraction f) ;
46
47 /**
48 * Ajoute une Fraction a la Fraction
49 * @param Fraction a ajouter
50 */
51 public void ajoute(Fraction f) ;
52
53 /**
54 * Reduit/Simplifie la Fraction
55 */
56 public void reduire() ;
57
58 /**
59 * Inverse la Fraction<br>
60 * Le signe est au numerateur.
61 */
62 public void inverse() ;
63
64 /**
65 * Multiplie la Fraction par un Coefficient
66 * @param i Coefficient multiplicateur
67 */
68 public void multiplierParCoeff(int i) ;

```

Réponse

```

1  /**
2  * Definie une fraction qui représente un nombre rationnel<br>
3  * Attention : le signe est au numerateur<br>
4  * Attention : le denominateur sera toujours different de 0<br>
5  * @author Nicolas EISEN <Nicolas.Eisen@gmail.com>
6  */
7
8  public class Fraction {
9
10     /**
11     * Calcule le pgcd de deux entiers

```

```

12     * @param a un des deux entiers
13     * @param b un des deux entiers
14     * @return le pgcd des deux entiers
15     */
16     public static int pgcd(int a, int b) {
17         int c;
18         while (b != 0) {
19             c = a % b;
20             a = b;
21             b = c;
22         }
23         return a;
24     }
25
26     /**
27     * Numerateur
28     */
29     private int n;
30
31     /**
32     * Denominateur
33     */
34     private int d;
35
36     /**
37     * Constructeur d'une Fraction
38     * @param num Numerateur
39     * @param den Denominateur
40     */
41     public Fraction(int num, int den) {
42         n = num;
43         if (den == 0)
44             d = 1;
45         else
46             d = den;
47     }
48
49     /**
50     * Retourne le numerateur de la Fraction
51     * @return le numerateur
52     */
53     public int getNumerateur() {
54         return n;
55     }
56
57     /**
58     * Retourne le denominateur de la Fraction
59     * @return le denominateur
60     */
61     public int getDenominateur() {
62         return d;
63     }
64
65     /**
66     * Change la valeur du denominteur
67     * @param nd le nouveau denominteur
68     */
69     public void setDenominateur(int nd) {
70         if (nd != 0)
71             d = nd;
72     }
73
74     /**
75     * Change la valeur du numerateur
76     * @param nd le nouveau numerateur
77     */
78     public void setNumerateur(int nn) {
79         n = nn;
80     }
81
82     /**
83     * Test si la Fraction est egale a une autre Fraction
84     * @param f Fraction a comparer
85     * @return vrai si les deux Fractions sont egales
86     */
87     public boolean egaleA(Fraction f) {
88         return n * f.getDenominateur() == f.getNumerateur() * d;
89     }
90
91     /**
92     * Ajoute une Fraction a la Fraction
93     * @param Fraction a ajouter
94     */
95     public void ajoute(Fraction f){
96         n = f.getNumerateur() * d + n * f.getDenominateur();

```

```

97     d = d * f.getDenominateur();
98 }
99
100 /**
101  * Reduit/Simplifie la Fraction
102  */
103 public void reduire() {
104     int denomCommun = Fraction.pgcd(n,d);
105
106     n = n / denomCommun ;
107     d = d / denomCommun ;
108
109     if ( d < 0 ) {
110         d *= -1;
111         n *= -1;
112     }
113 }
114
115 /**
116  * Inverse la Fraction<br>
117  * Le signe est au numérateur.
118  */
119 public void inverse() {
120     if ( n != 0 ) {
121         int temp;
122         temp = n;
123         n = d;
124         d = temp;
125     }
126 }
127
128 /**
129  * Multiplie la Fraction par un Coefficient
130  * @param i Coefficient multiplicateur
131  */
132 public void multiplierParCoeff(int i) {
133     n = n * i;
134 }
135 }

```

Fin réponse

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

```

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 :

- écrivez une méthode de la classe `Fraction`,
- décommentez uniquement les tests correspondants à cette méthode dans la classe `TestFraction` qui vous est fournie (cf. `/home/depot/1A/P00/TP1/TestFraction.java`),
- 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)`.