Licence Informatique — **UE J1IN4W01**
Université de Bordeaux — **Programmation 2**
2015-2016 — **Travaux Dirigés - feuille 2**

FORME CANONIQUE DE CLASSE. DONNÉES ET MÉTHODES MEMBRES. SURCHARGE ET ENCAPSULATION.

Exercice 1. Dans cet exercice, vous allez créer une classe `Point2DCartesien` qui permet la représentation d'un point dans un plan grâce à des coordonnées cartésiennes (`abscisse` et `ordonnee`).

1. Écrire un premier constructeur `public Point2DCartesien()` qui crée un point placé au centre du repère.
2. Écrire un constructeur plus général `public Point2DCartesien(double x, double y)`, permettant de construire un point de coordonnées x et y.
3. Ajouter les accesseurs en lecture `public double getAbscisse()` et `public double getOrdonnee()` qui retournent respectivement l'abscisse et l'ordonnée de l'objet dans un repère cartésien.
4. Ajouter les accesseurs en écriture `public void setAbscisse(double x)` et `public void setOrdonnee(double y)` qui modifient les valeurs de l'abscisse et de l'ordonnée de l'objet.
5. Ajouter une méthode `public boolean equals(Point2DCartesien p)` qui teste l'égalité entre l'objet auquel s'applique la méthode et l'objet passé en paramètre.
6. Ajouter une méthode `public Point2DCartesien clone()` qui réalise une copie de l'objet auquel s'applique la méthode.
7. Ajouter une méthode `public String toString()` qui donne la transformation en chaîne de caractères de l'objet auquel s'applique la méthode.
8. Ajouter une méthode `public void translation(double dx, double dy)` qui applique au point une translation d'un vecteur de coordonnées dx et dy.
9. Ajouter la méthode `public void homothetie(double facteur)` qui applique au point une homothétie en utilisant comme centre le centre du repère.
   Soit $M(x, y)$ un point dans un repère cartésien. Soit $M'(x', y')$ l'image de M par l'homothétie de centre $C(x_c, y_c)$ et de facteur k, alors
   $$x' = k * (x - x_c) + x_c$$
   $$y' = k * (y - y_c) + y_c$$
10. Ajouter la méthode `public void rotation(double alpha)` qui applique au point une rotation d'angle `alpha` en utilisant comme centre le centre du repère.
    Soit $M(x, y)$ un point dans un repère cartésien. Soit $M'(x', y')$ l'image de M par la rotation de centre $C(x_c, y_c)$ et d'angle α, alors
    $$x' = \cos\alpha * (x - x_c) - \sin\alpha * (y - y_c) + x_c$$
    $$y' = \sin\alpha * (x - x_c) + \cos\alpha * (y - y_c) + y_c$$
11. Ajouter une méthode `distance` qui retourne la distance entre deux points.
12. Soit $M(x, y)$ un point dans un repère cartésien. La représentation de M dans un repère polaire est $M(r, \phi)$ avec le module $r = \sqrt{x^2 + y^2}$ et l'argument ϕ :
    $$\phi = \begin{cases} \arctan\frac{y}{x}, & x > 0 \\ signe(y) * \pi + \arctan\frac{y}{x}, & x < 0 \\ signe(y) * \frac{\pi}{2}, & x = 0, y \neq 0 \\ 0, & x = 0, y = 0 \end{cases}$$
    Ajouter les accesseurs en lecture `public double getModule()` et `public double getArgument()` qui retournent respectivement le module et l'argument du point dans un repère polaire.

Exercice 2. Dans cet exercice, vous allez créer une classe `Complexe` qui représente un nombre complexe. Un nombre complexe s'écrit sous la forme $z = a + ib$, où a est la partie réelle et b la partie imaginaire, i étant le nombre dont le carré vaut -1.

1. Écrire les méthodes qui définissent la forme canonique de la classe `Complexe`.
2. Écrire une méthode `calculerModule` qui renvoie le module du complexe.
3. Écrire une méthode `ajouter` qui ajoute au complexe courant (i.e., l'objet auquel s'applique la méthode) un complexe passé en paramètre. S'agit-il d'une méthode d'instance ou de classe ?
4. Écrire une méthode `addition` qui retourne un nouveau complexe représentant l'addition des complexes passés en paramètres. S'agit-il d'une méthode d'instance ou de classe ?
5. Écrire une méthode `multiplication` qui retourne un nouveau complexe correspondant au produit des complexes passés en paramètres.

Exercice 3.

On souhaite créer une classe `Rectangle` qui modélise des rectangles alignés avec les axes. Un rectangle est représenté par son `coin inférieur gauche` et son `coin supérieur droit` (qui sont des objets de type `Point2DCartesien`), ainsi que par sa `surface`. N'oubliez pas de vérifier que les deux points définissent un rectangle valide.

1. Proposer plusieurs constructeurs pour la classe `Rectangle`. Comment appelle-t-on l'action de créer plusieurs méthodes du même nom ?
2. Proposer des implémentations des accesseurs en lecture et en écriture là où c'est nécessaire.
3. Proposer des implémentations des autres méthodes qui font partie de la forme canonique de classe.
4. On se place dans le cas où les coordonnées d'un point, sommet d'un rectangle, sont modifiées. Ces modifications sont-elles répercutées dans la définition du rectangle ? Comment gère-t-on la surface ?

Exercice 4. Dans cet exercice, vous allez créer une classe `PolygonePoint2DCartesien` qui permet la représentation d'un polygone grâce à un tableau de points 2D. On utilisera la classe `Point2DCartesien`.

1. Créer la forme canonique de la classe `PolygonePoint2DCartesien` en s'inspirant du code donné ci-dessous.

```
public class PolygonePoint2DCartesien {
        public PolygonePoint2DCartesien(Point2DCartesien[] points){
          ...
        }

        public Point2DCartesien[] getSommets(){
          ...
        }

        public void setSommets(Point2DCartesien[] points){
          ...
        }

        public boolean equals(PolygonePoint2DCartesien poly) {
          ...
        }

        public PolygonePoint2DCartesien clone() {
          ...
        }

        public String toString(){
          ...
        }
```

```
        private Point2DCartesien[] points;
}
```

2. Écrire les méthodes nécessaires au calcul du périmètre d'un polygone, de son centre de gravité et de sa boîte englobante.
   Le périmètre d'un polygone est défini comme la somme des longueurs de ses côtés.
   Le centre de gravité est défini comme le point ayant comme abscisse la moyenne des abscisses de ses sommets, et comme ordonnée la moyenne des ordonnées de ses sommets.
   La boîte englobante est définie comme un objet du type `Rectangle` dont le coin inférieur gauche est le point ayant comme abscisse et ordonnée respectivement l'abscisse minimale et l'ordonnée minimale des sommets du polygone, et dont le coin supérieur droit est le point ayant comme abscisse et ordonnée respectivement l'abscisse maximale et l'ordonnée maximale des sommets du polygone.
3. Compléter la classe `PolygonePoint2DCartesien` avec les méthodes d'instance suivantes :

   ```
   public void homothetie(double rapport);
   public void rotation(double angle);
   ```

   L'homothétie et la rotation du polygone consistent à appliquer la transformation à chaque sommet en prenant comme centre le centre du repère.

Exercice 5. Soit $E = \{0, 1, \ldots, n\}$.
Une relation binaire R sur E est un sous-ensemble de $E \times E$ constitué des couples (i, j) tels que iRj.

On peut représenter une relation binaire R par une matrice carrée de booléens de dimension $n+1$, notée `relation`, telle que :

`relation`$[i][j]$ a la valeur *true* si iRj, *false* sinon.

1. Une relation R est réflexive si et seulement si $\forall x \in E, xRx$.
2. Une relation R est symétrique si et seulement si $\forall x, y \in E$, $xRy \Rightarrow yRx$.
3. Une relation R est transitive si et seulement si $\forall x, y, z \in E$, xRy et $yRz \Rightarrow xRz$.
4. R et S étant 2 relations binaires sur E, la composée V de ces 2 relations est la relation définie par : $xVz \Leftrightarrow \exists\, y \in E | xRy$ et ySz
5. L'image de `i` pour la relation R est définie par : $image_R(i) = \{j \in E | iRj\}$.

Si $n = 10$, quel tableau de booléens permet de représenter la relation *i est multiple de j* ? Est-ce une relation réflexive ? symétrique ? transitive ? Même question pour la relation *i est congru à j modulo 2*.

Ecrire une classe `Relation` qui permet de tester si une relation est réflexive, transitive et symétrique. Cette classe contiendra également une méthode qui calcule l'image d'un entier par une relation ainsi qu'une méthode qui retourne la composée de deux relations.

En TP, on testera les méthodes au fur et à mesure de leur écriture à l'aide de la classe `TestRelation` fournie (voir extrait ci-dessous).

```
public class TestRelation{
    private static final int N = 11;

    private static void resultatTest(boolean condition, String nomMethode){
        if (condition)
            System.out.println("Test␣" + nomMethode + "␣passe␣avec␣succes");
        else
            System.out.println("Erreur␣test␣"+ nomMethode);
    }

    public static void main(String[] arg){

        boolean[][] tMultipleDe = new boolean[N][N];
        tMultipleDe[0][0] = true;
```

```
        for (int i=1; i<N; ++i){
            tMultipleDe[i][0] = false;
            for (int j=1; j<N; ++j)
                tMultipleDe[i][j] = (i%j==0);
        }

        Relation multipleDe = new Relation(tMultipleDe);

        resultatTest (multipleDe.reflexive(), ``reflexive'');

    }
}
```

Exercice 6. On considère l'application `StringDemo` dont les sources sont fournis. À l'aide de la documentation des classes `String` et `StringBuffer`, analysez le source de cette application.
Que fait cette application ? Exécutez-la. Expliquez le résultat obtenu.

```
public class StringDemo {
  public static long badLongString(int n) {
    long start = System.currentTimeMillis();
    String result = "";
    for (int i = 0; i < n; ++i) {
      result += 'A';
    }
    long end = System.currentTimeMillis();
    return end - start;
  }

  public static long goodLongString(int n) {
    long start = System.currentTimeMillis();

    StringBuffer result = new StringBuffer();
    for (int i = 0; i < n; ++i) {
      result.append('A');
    }
    long end = System.currentTimeMillis();
    return end - start;
  }

  public static void main(String args[]) {
    for (int i = 1024; i <= 65536; i *= 2) {
      System.out.print("␣i␣=␣" + i + "\t␣constructing␣goodLongString␣took␣");
      System.out.println(goodLongString(i) + "ms");
    }
    for (int i = 1024; i <= 65536; i *= 2) {
      System.out.print("␣i␣=␣" + i + "\t␣constructing␣badLongString␣took␣");
      System.out.println(badLongString(i) + "ms");
    }
  }
}
```