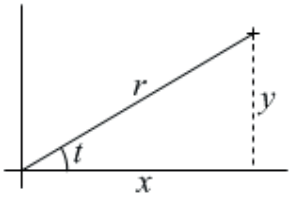
Feuille de TP N°03

(Agrégation et délégation)

Exercice 1 *La classe Point*

Définissez une classe `Point` pour représenter les points du plan rapporté à une origine fixée. Les coordonnées d'un point sont ici deux nombres flottants `x`, `y`, mémorisés dans deux variables d'instance privées (par exemple de type `double`).
La classe `Point` comportera au moins les méthodes suivantes, toutes publiques :

- `Point()`, constructeur d'un point qui initialise l'abscisse et l'ordonnée à `0`,
- `Point(double x, double y)`, constructeur d'un point à partir de ses coordonnées cartésiennes,
- `Point(Point p)`, constructeur d'un point à partir des coordonnées d'un point passé en paramètre. On peut donc créer un point de trois manières :
  `Appel au premier constructeur : Point p1 = new Point();`
  `Appel au second constructeur : Point p2 = new Point(2, 3);`
  `Appel au troisième constructeur : Point p3 = new Point(p2);`
- `double getX()`, `double getY()` qui renvoient les coordonnées cartésiennes du point,
- `double getR()`, `double getT()`, qui renvoient les coordonnées polaires du point,
- `String toString()`, qui renvoie une expression du point sous forme de texte,
- `boolean equals(Object o)`, comparaison de points,
- `void translation(double dx, double dy)`, qui applique au point une translation de vecteur `(dx, dy)`
- `void rotation(double a)`, qui applique au point une rotation de centre `(0, 0)` et d'angle `a`.
- `void afficher()` qui affiche à l'écran l'abscisse et l'ordonnée du point,
- `static Point milieu(Point a, Point b)`, qui renvoie un nouveau `Point` situé au milieu du segment ayant pour extrémités les points `a` et `b`. Cette méthode fera appel au constructeur précédemment défini.



Regarder maintenant le code des deux fonctions `dupliquer1` et `dupliquer2`.

```
static Point dupliquer1(Point a) {
        Point b = new Point(a);
        return b;
    }
```

```
static Point dupliquer2(Point a) {
    Point b = a;
    return b;
}
```

En quoi ces deux fonctions sont-elles différentes ? Dans le `main`, créer un point `p1`, appeler `p2 = dupliquer1(p1)`, modifier `p1` et regarder l'effet produit sur `p2`. De même, modifier `p2` et regarder l'effet produit sur `p1`. Faites de même avec `dupliquer2`. Que constatez vous ?

Exercice 2 *La classe Rectangle*

Un rectangle est défini par la donnée de deux points : `orig`, qui correspond au coin inférieur gauche du rectangle et `diag`, qui correspond au coin supérieur droit exprimé relativement à `orig`.

Définissez une classe `Rectangle` ayant deux variables d'instance privées `orig` et `diag` et, au moins, les méthodes publiques suivantes :
- `Rectangle(double x, double y, double dx, double dy)`, constructeur d'un rectangle ayant `(x,y)` pour origine et `(dx,dy)` pour diagonale,
- `double aire()`, qui renvoie l'aire du rectangle,
- `String toString()`, qui renvoie une expression du rectangle sous forme de texte,
- `boolean contient(Point p)` qui répond à la question "*ce rectangle contient-il le point p ?*"

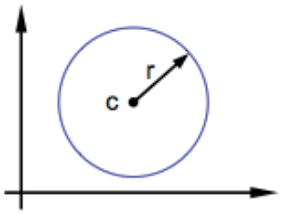
Exercice 3 *La classe LignePolygonale*

Une ligne polygonale est définie par un certain nombre de points, ses **sommets**.

Définissez une classe `LignePol` pour représenter de tels objets. Les sommets `y` seront mémorisés sous forme de tableau de points (il en découle que le nombre de sommets d'une ligne polygonale ne pourra pas augmenter au cours de la vie de celle-ci, mais cela n'est pas un problème dans cet exercice).
Cette classe comportera au moins les méthodes publiques suivantes :
- `LignePol(int n)`, constructeur d'une ligne polygonale à `n` sommets (initialement indéterminés),
- `LignePol(Point[] sommets)`, constructeur d'une ligne polygonale ayant les sommets indiqués,
- `Point getSommet(int i)`, obtention du $i^{\text{ème}}$ sommet,
- `void setSommet(int i, Point p)`, définition du $i^{\text{ème}}$ sommet,
- `String toString()`, qui renvoie une expression de la ligne polygonale sous forme de texte,
- `Object clone()`, qui renvoie une copie entièrement disjointe d'une ligne polygonale,
- `void translation(double dx, double dy)`, qui applique à chaque sommet de la ligne polygonale une translation de vecteur `(dx, dy)`
- `void rotation(double a)`, qui applique à chaque sommet de la ligne polygonale une rotation de centre (0, 0) et d'angle `a`
- Pour tester les différentes classes lignes polygonales, les points et leurs transformations, écrivez un programme simple qui crée une ligne de `10` sommets aléatoires (utilisez la méthode `java.lang.Math.random()`), l'affiche, lui applique une ou plusieurs transformations et l'affiche à nouveau, ...

Exercice 4 *La classe Cercle*

On souhaite réaliser une classe `Cercle` disposant des fonctionnalités suivantes :
- un constructeur recevant en argument les coordonnées du centre du cercle et son rayon
- `void translation(double dx, double dy)` pour déplacer les coordonnées du centre du cercle (incréments `dx` et `dy`)
- `void changerRayon(double rayon)` pour modifier le rayon du cercle
- `Point getCenter()` qui fournit en résultat un objet de type `Point` correspondant au centre du cercle.
- Redéfinir la méthode `toString()` afin qu'elle retourne une chaîne de caractères contenant les coordonnées du centre du cercle ainsi que son rayon.
- `afficher()` qui affiche les coordonnées du centre du cercle et son rayon
- Deux cercles sont égaux s'ils ont même centre de gravité et même rayon. Écrivez le code à ajouter à la classe `Cercle` pour implémenter cette fonctionnalité de test d'égalité (redéfinition de la méthode `equals`). `Point`).



Implémenter deux variantes de cette classe `Cercle` :

a) comme classe dérivée (sous-classe) de `Point` ⇒ **Héritage**

b) avec un attribut de type `Point` ⇒ **Composition**

Rappel de formules :

- Les deux coordonnées polaires r et t peuvent être converties en coordonnées cartésiennes x et y en utilisant les fonctions trigonométriques *sinus* et *cosinus* :
  $x = r \times \cos(t)$,
  $y = r \times \sin(t)$,
- Deux coordonnées cartésiennes x et y permettent de calculer la première coordonnée polaire r par :
  $r = \sqrt{x^2 + y^2}$,
- Pour déterminer la seconde (l'angle t) dans l'intervalle $[0, 2\pi[$, on utilise les formules suivantes (*arctan* désigne la réciproque de la fonction tangente) :

$$t = \begin{cases} \arctan(\frac{y}{x}) & \text{si } x > 0 \text{ et } y \geq 0 \\ \arctan(\frac{y}{x}) + 2\pi & \text{si } x > 0 \text{ et } y < 0 \\ \arctan(\frac{y}{x}) + \pi & \text{si } x < 0 \\ \frac{\pi}{2} & \text{si } x = 0 \text{ et } y > 0 \\ \frac{3\pi}{2} & \text{si } x = 0 \text{ et } y < 0 \end{cases}$$

- la rotation de centre $(0, 0)$ et d'angle t d'un point de coordonnées cartésiennes x et y est donnée par :
  $x = x \times \cos(a) - y \times \sin(a)$
  $y = x \times \sin(a) + y \times \cos(a)$