Programmation Objet Avancée - TD1

—o000o——o000o—

TP1 : Exercices de base sur les classes

1 Classe Point

Il s'agit de modéliser des points et des droites dans un espace à deux dimensions. Un point peut être modélisé grâce à ses coordonnées : **son abscisse et son ordonnée** tandis qu'une droite peut être représentée grâce au couple (a, b) de l'équation $\mathbf{y} = \mathbf{ax}+\mathbf{b}$ modélisant l'ensemble des coordonnées (x, y) des points qui appartiennent à cette droite.

1. Écrire la classe **Point** qui contient :

   - Deux champs privés pour représenter les coordonnées (x, y) du point.
   - *Un constructeur* qui prend comme paramètres les deux valeurs de type double des coordonnées (x, y) du point.
   - La méthode *double getAbscisse()* qui retourne l'abscisse du point.
   - La méthode *double getOrdonnee()* qui retourne l'ordonnée du point.
   - La méthode *boolean estConfondu(Point p)* qui indique si le point est confondu avec le point **p** passé en argument. Deux points sont théoriquement confondus s'ils ont les mêmes abscisses et ordonnées. En programmation, il n'est pas possible de garantir l'égalité parfaite entre 2 nombres réels, vous ajouterez donc un champ *static* privé nommé **Epsilon** initialisé à une très petite valeur et qui vous servira à spécifier la précision de vos comparaisons entre deux nombres réels.
   - La méthode *double distance(Point p)* qui retourne la distance du point avec le point **p** passé en argument. Rappel : la distance (euclidienne) entre deux points de coordonnées (x_1, y_1) et (x_2, y_2) est:
   $$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$
   - La méthode *String toString()* qui redéfinit la méthode toString() de la classe **Object** pour afficher les coordonnées du point.

   Solution :

```java
public class Point
{
  private double x;
  private double y;

    public Point(double x,double y) {this.x=x; this.y=y;}

  public double getAbscisse() {return x;}
  public double getOrdonnee() { return y;}
  public boolean  estConfondu(Point  p) {return (x==p.getAbscisse()
      && this.y==p.getOrdonnee());}
  public double  distance(Point  p) {return Math.sqrt(Math.pow(x-p.
     getAbscisse(), 2)+ Math.pow(y-p.getOrdonnee(),2)); }

  public Point translater(Vecteur v)
```

```java
  {
    Point p = new Point(x+v.getP().getAbscisse(), y+v.getP().
        getOrdonnee());
    return p;
  }
  public Point moyen1(Point p2)
  {
    double xx = (x+p2.x)/2;
    double yy = (y+p2.y)/2;
    Point pointMoyen = new Point(xx,yy);
    return pointMoyen;
  }
  public Point moyen2(Point p2)
  {
    Vecteur v= new Vecteur(this,p2);
    x=x/2;y=y/2;
    Point pointMoyen= translater(v);
    return pointMoyen;
  }
}
```

java/Point.java

2. Écrire la classe **Droite** qui contient trois champs et les méthodes suivantes :
   - Deux champs de type double privés **a** et **b** qui représente respectivement le coefficient directeur a et l'ordonnée à l'origine b (ou bias en anglais) dans l'équation $y = ax + b$.
   - Un champ privé static **Epsilon** pour réaliser des comparaisons numériques correctes.
   - *Un constructeur* qui prend comme arguments 2 points (de type Point) et qui initialise les valeurs du coefficient directeur et de l'ordonnée à l'origine de la droite. Pour rappel, si on considère la droite (AB) définie par les deux points A et B alors :

     $$a = (y_B - y_A)/(x_B - x_A)$$

     et

     $$b = y_A - a * x_A = y_B - a * x_B$$
   - L'accesseur *double getCoefficient()* qui retourne le coefficient directeur de la droite.
   - L'accesseur *double getBias()* qui retourne l'ordonnée a l'origine de la droite.
   - La méthode *boolean estParallele(Droite d)* qui indique si la droite **d** est parallèle à la droite. Deux droites sont parallèles si elles ont le même coefficient directeur. Attention ici à vos comparaison de nombres réels.
   - La méthode *boolean appartient(Point p)* qui indique si le point p appartient à la droite. Pour montrer qu'un point **p** appartient a la droite (AB), il suffit de vérifier les coordonnes du point p dans l'équation suivante :

     $$y_p = ax_p + b$$

     Attention à vos comparaisons de nombres réels.
   - La méthode *Point intersection(Droite d)* qui retourne le point d'intersection entre la droite **d** et la droite courante. On présupposera que les droites ne sont pas parallèles et on ne fera donc pas ce test dans la méthode. Pour rappel : deux droites non parallèles définie par les équations $y = a_1x + b_1$ et $y = a_2x + b_2$ s'intersectent au point de coordonnées suivant :

     $$(b_2 - b_1/a_1 - a_2; a_1 * (b_2 - b_1/a_1 - a_2) + b_1)$$
   - La méthode *String toString()* qui redéfinit la méthode toString() de la classe **Object**.

Solution :

```java
public class Droite {

    private double a, b;
    private static final double Epsilon = 1/10000000.;

    public Droite(Point p1, Point p2) {
      this.a = (p2.getOrdonnee() - p1.getOrdonnee()) / (p2.getAbscisse() - p1.getAbscisse());
        this.b = (p1.getOrdonnee() - a  * p1.getAbscisse());
    }

    public String toString() {
      return "Droite : y = " + a + " x + " + b;
    }

    public double getCoefficient() {
     return a;
    }

    public double getBias() {
     return b;
    }

  public boolean appartient(Point  p)
  {
    double diff = Math.abs(p.getOrdonnee() - (a * p.getAbscisse() + b));
    return diff <= Epsilon;
  }


  public boolean estparallele(Droite d) {
    return (a-d.getCoefficient()) <= Epsilon;
  }

  public Point intersection(Droite d) {
    double x = (d.getBias()- b) / (a - d.getCoefficient());
    double y = a * x + b;
    return new Point(x,y);
  }

}
```

java/Droite.java

3. Écrire la classe **Cercle** représentée par son point central et un rayon. Cette classe contient les méthodes suivantes :
   - *double surface()* qui retourne la surface du cercle. Pour rappel, la surface du cercle est calculé par $surface = \pi * rayon * rayon$
   - La méthode *String toString()* qui redéfinit la méthode toString() de la classe **Object** pour afficher l'équation de la droite $y = ax + b$.

Solution :

```java

public class Cercle {
  private Point centre;
  private double rayon;
  private static final double Epsilon = 1/10000000.;

  public Cercle(Point c, double r) {
    centre = c;
    rayon = r;
  }

  public Point getCentre() {return centre;}
  public double getRadius() {return rayon;}

  public String toString() {
    return "Cercle : centre = " + centre + " ; rayon = " + rayon;
  }

  public double surface() {
    return Math.PI * rayon * rayon;
  }
}
```

java/Cercle.java

2 Classe Vecteur

1. Écrire une classe **Vecteur** qui est définit par son nom et ses coordonnées en x et y nommée **dx** et **dy** respectivement. Ajouter les méthodes nécessaires pour créer et gérer les objets du Vecteur et notamment les accesseurs (getters) et les mutateurs (setters) et une méthode d'affichage.

2. Ajouter des méthodes pour calculer:
   - l'addition de deux vecteurs (le nom du vecteur résultant sera la concaténation des deux noms).
   - la multiplication du vecteur par un scalaire.
   - la norme du vecteur qui est calculé selon la formule suivante:

   $$\sqrt{x^2 + y^2}$$

3. Implémenter une comparaison de deux vecteurs. Pourquoi tester l'égalité des valeurs entre deux vecteurs n'est pas recommandé ? Que faire à la place ? On utilisera pour cela une méthode *boolean equals()* qui surcharge la méthode *equals()* de la classe **Object**.

4. Ajouter une méthode pour appliquer une rotation selon un angle passé en paramètre.

   Pour la rotation d'angle θ, il faut appliquer la matrice de rotation au vecteur qui produit les nouvelles coordonnées comme suit :

   $$(rx = \cos(\theta)dx - \sin(\theta)dy; ry = \sin(\theta)dx + \cos(\theta)dy)$$

5. Tester si deux vecteurs sont colinéaires en notant que deux vecteurs **v1** et **v2** sont colinéaires si on observe $\frac{dx_1}{dx_2} = \frac{dy_1}{dy_2}$. Attention aux comparaisons de valeurs numériques.

6. Tester si deux vecteurs sont orthogonaux en calculant leur produit scalaire. Le produit scalaire est $\langle v_1, v_2 \rangle = dx_1 \times dx_2 + dy_1 \times dy_2$ et vaut 0 si les vecteurs sont orthogonaux.

3 Classe Vecteur et Point

Ajouter les méthodes qui suivent à la classe Point ou à la classe Vecteur.

1. Dans la classe Vecteur, surcharger le constructeur afin de construire un vecteur à partir de deux points : on considérera que le vecteur va du premier au second point.
2. Dans la classe Point, définir une méthode translater qui renvoie un point translaté selon un vecteur.
3. Dans la classe Point, ajouter une méthode qui renvoie le point moyen entre deux points en essayant de deux manières différentes :
   - à l'aide des attributs de points
   - en utilisant la méthode de la translation du vecteur.