TP N° 3

Exercice 1

1. Ecrire une classe « **Volaille** ». Une volaille est un oiseau élevé dans une ferme, caractérisée par un « **numéro** » et un « **poids** ». Ajouter à cette classe un **constructeur** et une méthode « **ChangePoids** » qui permet de modifier son poids. Ecrire **deux autres méthodes**, l'une retourne son **prix**, et l'autre si la volaille est assez grosse pour être abattue. Seulement, le prix et l'estimation de la grosseur dépend du type de la volaille (Poulet, canard, dinde, etc.).

2. Ecrire une classe « **Poulet** » qui spécifie la classe « Volaille ». Tous les poulets ont le même **prix de vente au kilo**, et le même **poids d'abattage**. Cependant, pour des raisons de marché, le prix de vente et le poids d'abattage peuvent changer.

3. **Testez** la classe « Poulet ».

Exercice 2

1. On souhaite disposer d'un ensemble de classes permettant de manipuler des formes tridimensionnelles. Pour cela on propose la hiérarchie de classes de la figure ci-dessous.

2. Créer la classe **Point3D** ayant les coordonnées **x, y et z,** disposant d'une méthode **déplacer** qui permet de déplacer un point dans l'espace et d'une méthode **toString** qui retourne la représentation textuelle du point sous la forme (si par exemple, x=10.0, y=4.0 et z=3.0) :
   [Point3D x :10.0 , y : 4.0, z : 3.0].

Les contraintes sur ces classes sont les suivantes :

- La classe Forme **ne peut être instanciée**.
- La classe Cube **ne peut être dérivée**.
- Chaque forme possède un attribut de type Point3D qui représente son **centre de gravité** et un attribut réel représentant sa **densité**.
- Un objet de type Brique est caractérisé aussi, par son centre de gravité, sa densité, une **largeur**, une **longueur** et une **hauteur**.
- Un objet de type Cube est une brique pour laquelle : **largeur = longueur = hauteur**.
- Les formes disposent d'une méthode **déplacer** prenant comme paramètres trois réels représentant les composantes x, y et z d'un vecteur de translation et des méthodes **calculerSurface**, **calculerVolume** et **calculerPoids** calculant respectivement la surface, le volume et le poids (volume x densité) de la forme.
- De plus, toute forme est capable de donner sa représentation sous la forme d'une chaîne de caractères contenant le nom de sa classe et la description textuelle de chacun de ses attributs (**getClass()** est une méthode de Object qui renvoie le nom de classe).

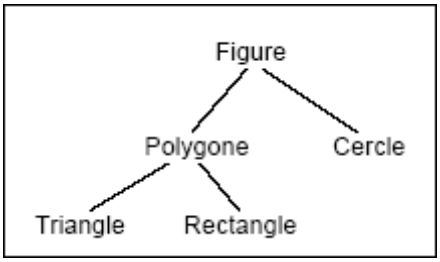
  Exemple : la chaîne de caractères produite pour un objet de classe Brique :
  [Brique
  centre de gravité : [Point3D x :10.0 , y : 4.0, z : 3.0], densité : 1.2, largeur : 10.5, longueur : 14.3, hauteur : 4.6]

NB. : Surface d'une "brique" : 2x(largeur x longueur + largeur x hauteur + longueur x hauteur).
Volume d'une "brique" : largeur x hauteur x longueur;

3. Ecrire la classe **Test** Permettant de :
   a) Créer **deux cubes**.
   b) Afficher la représentation de chaque forme.

Afin de vérifier si les deux cubes sont égaux, ajouter une méthode **estEgalA (Object o)** dans la classe Cube. (Deux cubes sont égaux s'ils sont de la même nature (des cubes) et s'ils ont les mêmes densités et surfaces).

Exercice 3



Soit l'arbre d'héritage de classes d'objets géométriques.
On vous demande d'implémenter ces différentes classes.

1. Pour cela vous allez commencer par définir une classe **Point** qui servira dans les autres classes. La classe **Point** possède

- Deux attributs réels **abscisse** et **ordonnée**.
- Un **constructeur** possédant **deux paramètres réels** x et y.
- Un constructeur possédant un paramètre p de type **Point**.
- Une méthode **statique** qui retourne la distance entre deux points.

Remarques :

- Soit deux points $M_1(x1,y1)$ et $M_2(x2,x2)$. La distance entre M_1 et M_2 est : $\sqrt{(x2-x1)^2+(y2-y1)^2}$
- `Math.sqrt(a) est une méthode qui retourne la r`acine carré de a en java

2. Définissez la classe **Figure** constituée de :
   - Un attribut **Point** statique, nommé **zéro** qui a pour coordonnées (0,0) ;
   - Un attribut **Point,** nommé **origine** ;
   - Un constructeur **sans paramètre** où **origine** est initialisé par **zéro**
   - Un deuxième constructeur avec un paramètre **Point** p ;
   - Deux méthodes abstraites **afficher()** et **périmètre()**.

3. Définissez la classe **Cercle** qui dérive de la classe Figure qui possède :
   - Deux attributs **centre** et **rayon** ;
   - Un constructeur avec **deux paramètres : le centre et le rayon** ;
   - Complétez cette classe afin qu'elle puisse instancier des objets Cercle.

Remarque : Périmètre d'un cercle est : 2*Math.PI* rayon

4. La classe **Polygone** dérive de la classe Figure. Elle se caractérise par :
   - Un tableau de Point nommé sommets ;
   - Un entier **nbs** qui représente le nombre de ces sommets ;
   - Un constructeur où **nbs** est initialisé à 0 ;
   - Un constructeur **Polygone(Point[] m, int n),** n désigne le nombre de sommets ;
   - Complétez cette classe afin qu'elle puisse instancier des objets Polygone.

5. La classe **Triangle** est une classe élémentaire dérivée de la classe **polygone.**
6. La classe **Rectangle** dérive de la classe Polygone. Elle est définie avec :
   - Ses **caractéristiques** mathématiques classiques, c'est à dire la longueur et la largeur de ses cotés et un de ses sommets, le sommet inférieur gauche.
   - Un constructeur **Rectangle(Point m, double long, double larg)**

7. Ecrire un programme principal géométrie, qui crée un cercle, un triangle et un rectangle et qui les affiche (Spécifiez afficher() afin qu'elle visualise toutes les caractéristiques de chaque forme).

Exercice 1

```
abstract class Volaille{
        double poids;
        int identite;
        Volaille ( double p , int i ){ poids = p; identite = i ;}
        void changePoids ( double np ){poids = np ;}
        abstract double prix ( ) ;
        abstract boolean assezGrosse ( ) ;
}

class Poulet extendsVolaille{
        static double prixAuKilo = 1.0 ;
        static double poidsAbattage = 1.2 ;
        Poulet ( double p , int i ){super ( p , i ) ;}
        static void changePrix ( double x ){prixAuKilo = x ;}
        static void changePoidsAbattage ( double x ){poidsAbattage = x ;}
        double prix ( ) {return poids * prixAuKilo ;}
        boolean assezGrosse ( ) {return poids >= poidsAbattage ;}
}
```

Exercice 2

```
abstract class Forme {
    Point3D centre;    double d;
    public Forme(Point3D c, double r){    centre=c;    d=r;   }
    public void deplacer(double v1,double v2,double v3){centre.deplacer(v1,v2,v3)}
    abstract public double calculerSurface();
    abstract public double calculerVolume();
    public double calculerPoids(){     return d*calculerVolume();    }
    public String toString(){
     return "["+getClass()+" :\n centre de gravite : "+centre.toString()+
            "\n densite : "+      d+"\n]\n";     } }

class Brique extends Forme{
    double largeur,longueur, hauteur;
    public Brique(Point3D p, double d, double l, double L, double h){
      super(p,d);     largeur=l;    longueur=L;    hauteur=h;    }
    public double calculerSurface(){
      return 2*(largeur*longueur + largeur*hauteur + longueur*hauteur);    }
    public double calculerVolume(){   return largeur*hauteur*longueur;    }
    public String toString(){
      return "[Brique :\n centre de gravite : "+centre.toString()+"\n densite : "+
            d+"\n largeur : "+largeur+"\n longueur : "+longueur+"\n hauteur : "+
            hauteur+"\n]\n";
    }
  }
```

```
final class Cube extends Brique{
    public Cube(Point3D p, double d, double l){    super(p,d,l,l,l);    }
    public String toString(){
     return "[Cube :\n centre de gravite : "+centre.toString()+"\n densite : "+
            d+"\n      cote : "+largeur+"\n]\n";  }
```

```
    public boolean estEgaleA (Object o){
       if (!(o instanceof Cube) return false;
       else {     Cube c=(Cube)o,
       return    (this.densite==c.densite)&&
                 (this.calculerSuraface()==c. calculerSuraface());   }
     }
}
```

```
class Point3D {
      double x,y,z;
      public Point3D(double x, double y, double z){this.x=x;this.y=y;this.z=z; }
     public void deplacer(double vx, double vy, double vz){x=x+vx; y=y+vy; z=z+vz; }
      public String toString(){
            return "[Point3D x :"+x+" , y :"+y+", z :"+z+"]\n";    }
}
```

```
public class Test {
    public static void main(String arg[]){
      Cube c1=new Cube(new Ponit3d(1,1,1),5,1) ;
      Cube c2=new Cube(new Ponit3d(0,0,0),5,1) ;
      System.out.println(c1.toString());
      System.out.println(c2.toString());
      if (c1.estEgaleA(c2))   System.out.println("Les deux cubes sont égaux) ;
      else  System.out.println("Les deux cubes sont inégaux) ;}}
```

Exercice 3

1)

```
public class Point
{
  double abscisse;
  double ordonnee;
  Point(double x, double y) {abscisse=x; ordonnee=y;}
  Point(Point p){abscisse=p.abscisse; ordonnee=p.ordonnee;}
  static double distance(Point p, Point q) {
     double dx=p.abscisse-q.abscisse;
     double dy=p.ordonnee-q.ordonnee;
     return Math.sqrt(dx*dx+dy*dy);
   }
}
```

2)

```
abstract class Figure {
   private static final Point zero=new Point(0,0);
   Point origine;
   Figure(){origine=zero;}
   Figure(Point p){origine=new Point(p);}
   abstract double perimetre();
   abstract void affiche();
}
```

3)

```
class Cercle extends Figure {
   private static final double pi=3.141592;
   double rayon;
   Cercle(Point centre, double r){super(centre); rayon=r;}
   double perimetre(){return 2*pi*rayon;}
   void affiche() {
      System.out.println("Cercle");
      System.out.println("rayon : " + rayon +" et centre : " +"(" + origine.abscisse +
      "," + origine.ordonnee + ") ");
   }
}
```

4)

```
class Polygone extends Figure {
   Point sommet[]= new Point[100];
   int nbs;
   Polygone(){nbs=0;}
   Polygone(Point[] m, int n) {
      super(m[0]);nbs=n;
      for (int i=0; i<n; i++)  sommet[i]=m[i];
   }
   double lcote(int i) {
      if (i<nbs)
         return Point.distance(sommet[i-1], sommet[i]);
      else
         return Point.distance(sommet[i-1], sommet[0]);
   }
   double perimetre() {
      double somme=0;
      for (int i=1; i<=nbs; i++)
      somme += lcote(i);
      return somme;
   }
   void affiche() {
      System.out.println("Polygone");
      for (int i=0; i<nbs; i++)
         System.out.print("(" + sommet[i].abscisse + "," + sommet[i].ordonnee + ") ");
         System.out.println();
      }
}
```

5)

```
class Triangle extends Polygone {
   Triangle(Point[] m) { super(m,3); }
}
```

6)

```
class Rectangle extends Polygone {
   double largeur;
   double longueur;
   Rectangle(Point m, double lo, double la) {
      // Appel implicite du constructeur de Polygone sans parametre
      // L'appel du constructeur de Polygone avec parametres ne peut se faire
      //car il faut d'abord construire le tableau a lui transmettre
      //qui ne peut se faire qu'apres l'appel explicite ou implicite de super
      Point P1= new Point(m.abscisse+ lo, m.ordonnee);
      Point P2= new Point(m.abscisse, m.ordonnee+ la);
      Point P3= new Point(m.abscisse+ lo, m.ordonnee+ la);
      Point mr[]={m, P1, P3, P2};
      sommet=mr;
      nbs=4;
      largeur= la;
      longueur= lo;
   }
}
```

7)

```
class Geometrie {
   public static void main(String args[]) {
      Point P1= new Point(3,4); Point P2= new Point(4,4);
      Point P3= new Point(0,0);Point P4= new Point(1,0);
      Point P5= new Point(0,1);Point [] TabP={P3, P4, P5, P2};
      Cercle c= new Cercle(P1,2);
      Rectangle r= new Rectangle(P2,5,2);
      Triangle t= new Triangle(TabP);
      Figure f; //autorise, mais pas new Figure !
      System.out.println("perimetre cercle");
      f=c; f.affiche(); // appel de affiche de Figure puis de sa forme derivee de cercle
      System.out.println("perimetre : " + f.perimetre());
      f=r; f.affiche();
      System.out.println("perimetre : " + f.perimetre());
      f=t; f.affiche();
      System.out.println("perimetre : " + f.perimetre());
   }
}
```