# Séquence 19

# Classe abstraite

## 1 Notion de classe abstraite

Une classe abstraite est une classe qui déclare certaines de ses méthodes sans les définir. Ces méthodes s'appellent des méthodes abstraites. La classe donne leur entête mais pas leur corps. Les déclarations des classes abstraites et des méthodes abstraites sont précédées du mot clé abstract.

Une classe abstraite ne peut pas être instanciée avec un new. On ne peut pas créer d'objet de cette classe. A quoi sert-elle alors ? Elle sert si plusieurs autres classes héritent d'elle. La classe abstraite donne alors un type qui peut être utilisé pour réunir les instances de toutes les sous-classes. La classe abstraite est un noeud de plus dans l'arbre d'héritage.

Prenons un exemple : celui des figures géométriques dans le plan. Ces figures ont en commun d'avoir un type (triangle, cercle, etc) et une surface. Mais chaque type de figure a son calcul particulier pour la surface.

```
abstract class Figure{
    abstract public String type();
    abstract public double surface();
}
```

Si l'on essaye d'instantier la classe avec `new Figure()`, on obtient l'erreur de compilation :

```
Figure.java:5: error: Figure is abstract; cannot be instantiated
        new Figure();
        ^
1 error
```

Lorsqu'on crée une sous-classe de la classe abstraite, il faut définir toutes les méthodes abstraites.

```
class Point{
    double x, y;
    Point (double xi, double yi){
        x = xi;
        y = yi;
    }
    static double distance(Point p1, Point p2){
        return Math.sqrt((p1.x-p2.x)*(p1.x-p2.x) + (p1.y-p2.y) *(p1.y-p2.y));
    }
}
```

```
class Triangle extends Figure{
    private Point p1, p2, p3;
    Triangle(Point p11, Point p22, Point p33){
        p1=p11; p2=p22; p3=p33;
    }
    public String type(){
        return "triangle";
    }
    public double surface(){
        double a = Point.distance(p1,p2);
        double b = Point.distance(p1,p3);
        double c = Point.distance(p2,p3);
        double demiper = (a+b+c)/2;
        return Math.sqrt(demiper*(demiper-a)*(demiper-b)*(demiper-c));
    }
}
class Cercle extends Figure{
    private Point centre;
    private double rayon;
    Cercle(Point ctr, double r){
        centre = ctr;
        rayon = r;
    }
    public String type(){
        return "cercle";
    }
    public double surface(){
        return Math.PI *rayon *rayon;
    }
}
class Exemple{
    public static void main(String[] args){
        Figure fig1;
        Point p1, p2, p3;
        p1=new Point(10,10);
        p2=new Point(5,5);
        p3=new Point(0,3);
        fig1=new Triangle(p1,p2,p3);
        fig1=new Cercle(p1,3.0);
    }
}
```

Dans la méthode main, on utilise le nom de la classe abstraite comme type pour la variable `fig` et on lui affecte successivement un triangle et un cercle.

Toutes les méthodes d'une classe abstraite ne sont pas nécessairement abstraites : il peut y avoir des méthodes abstraites et d'autres pas. Celles qui ne sont pas abstraites doivent être définies avec leur code. Une classe abstraite peut ne pas avoir de méthodes abstraites. La déclarer abstraite quand même permet d'interdire son instanciation.

```
abstract class FigureColoree{
    int[] couleur;
    FigureColoree(int r, int g, int b){
        couleur = new int[3];
```

```
            couleur[0]=r;
            couleur[1]=g;
            couleur[2]=b;
        }
        abstract public String typeForme();
        abstract public double surface();
        public int getRed(){
            return couleur[0];
        }
        public int getGreen(){
            return couleur[1];
        }
        public int getBlue(){
            return couleur[2];
        }
        public void eclaircir(int x){
            for (int i=0; i<3; i++){
                couleur[i]=Math.min(255,couleur[i]+x);
            }
        }
}
class Point{
        double x, y;
        Point (double xi, double yi){
            x = xi;
            y = yi;
        }
        static double distance(Point p1, Point p2){
            return Math.sqrt((p1.x-p2.x)*(p1.x-p2.x) + (p1.y-p2.y) *(p1.y-p2.y));
        }
}
class TriangleColore extends FigureColoree{
        private Point p1, p2, p3;
        TriangleColore(Point p11, Point p22, Point p33, int r, int g, int b){
            super(r,g,b);
            p1=p11; p2=p22; p3=p33;
        }
        public String typeForme(){
            return "triangle";
        }
        public double surface(){
            double a = Point.distance(p1,p2);
            double b = Point.distance(p1,p3);
            double c = Point.distance(p2,p3);
            double demiper = (a+b+c)/2;
            return Math.sqrt(demiper*(demiper-a)*(demiper-b)*(demiper-c));
        }
}
class CercleColore extends FigureColoree{
        private Point centre;
        private double rayon;
        CercleColore(Point ctr, double r, int re, int g, int b){
            super(re,g,b);
```

```
        centre = ctr;
        rayon = r;
    }
    public String typeForme(){
        return "cercle";
    }
    public double surface(){
        return Math.PI *rayon *rayon;
    }
}
class Exemple{
    public static void main(String[] args){
        FigureColoree fig1;
        Point p1, p2, p3;
        p1=new Point(10,10);
        p2=new Point(5,5);
        p3=new Point(0,3);
        fig1=new TriangleColore(p1,p2,p3,200,0,0);
        fig1.eclaircir(10);
        fig1=new CercleColore(p1,3.0,0,200,150);
        System.out.println(fig1.surface());
    }
}
```

## 2 Classe abstraite et interface

Les classes abstraites et les interfaces sont deux mécanismes d'abstraction qui créent des types sans instance propres, qui contiennent les instances d'autres classes. Ces nouveaux types permettent une programmation polymorphe (une variante de la généricité), pour un ensemble de classes qui ont des caractéristiques communes.

Dans certains cas, on pourra choisir entre l'un ou l'autre mécanisme mais dans d'autres cas, un des deux sera préférable.

Les diférences sont les suivantes :

— les interfaces peuvent être implémentées indépendamment des relations d'héritage. Le type correspondant est l'union de plusieurs sous-arbres de l'arbre d'héritage. Une classe abstraite est un type qui contient uniquement les sous-classes (directes et indirectes) de la classe abstraite.

— une classe abstraite peut contenir des variables et des méthodes concrètes, ce qui n'est pas le cas d'une interface. La classe abstraite permet de partager du code et la représentation de certaines données. Elle est donc bien indiquée lorsqu'une représentation commune et/ou des traitements comuns pourront être définis et partagés par tous les objets.

— une classe abstraite peut contenir des constructeurs, pas une interface.

— une classe abstraite peut contenir des méthodes non publiques (par exemple protected) ce qui n'est pas le cas des interfaces dont toutes les composantes sont publiques.

— une classe peut implémenter plusieurs interfaces, mais elle a exactement une super-classe, que cette super-classe soit abstraite ou non.