Licence Informatique **UE J1IN4W01**
Université de Bordeaux **Programmation 2**
2015-2016 **Travaux Dirigés - feuille 10**

La délégation pour simuler l'héritage multiple

Soit les interfaces `Point2D` et `Color` définies de la manière suivante.

```
interface Point2D  {
        Object clone();
        boolean equals(Object o);
        String toString();
        double getX();
        double getY();
        void setX(double x);
        void setY(double y);
        void translate(double dx, double dy);
}
interface Color {
        Object clone();
        boolean equals(Object o);
        String toString();
        int getR();
        int getG();
        int getB();
        int getA();
        void set(int r, int g, int b, int a);
        void setR(int r);
        void setG(int g);
        void setB(int b);
        void setA(int a);
}
```

Prenons maintenant deux classes abstraites qui implémentent ces interfaces, respectivement `Point2DAbstract` et `ColorAbstract`.

```
abstract class Point2DAbstract  implements Point2D {
        abstract public Object clone();
        public boolean equals(Object o) {
                if ( o instanceof Point2D ) {
                        Point2D p = (Point2D) o;
                        return getX() == p.getX() && getY() == p.getY();
                }
                else
                        return false;
        }
        public String toString() {
                StringBuffer sb = new StringBuffer();
           sb.append("(").append(getX());
           sb.append(",").append(getY()).append(")");
                return sb.toString();
        }
        public void translate(double dx, double dy) {
                setX(getX() + dx);
                setY(getY() + dy);
        }
}
abstract class ColorAbstract  implements Color {
        abstract public Object clone();
        public boolean equals(Object o) {
                if ( o instanceof Color )
                {
```

```
                        Color c = (Color) o;
                        return getR() == c.getR() && getG() == c.getG()
                                && getB() == c.getB() && getA() == c.getA();
                }
                else
                        return false;
        }
        public String toString() {
                 StringBuffer sb = new StringBuffer();
           sb.append("(").append(getR());
           sb.append(",").append(getG());
           sb.append(",").append(getB());
           sb.append(",").append(getA()).append(")");
                return sb.toString();
        }
        public void set(int r, int g, int b, int a) {
                setR(r);
                setG(g);
                setB(b);
                setA(a);
        }
}
```

Les classes `Point2DXY`, `Point2DArray`, ainsi que `ColorInt` et `Color4Int`, héritent respectivement de `Point2DAbstract` et `ColorAbstract`.

```
class Point2DXY extends Point2DAbstract {
        public Point2DXY(double x, double y) {
                this.x = x;
                this.y = y;
        }
        //Copie
        public Object clone() {
                return new Point2DXY(x, y);
        }
        //Accesseurs
        public double getX() {
                return x;
        }
        public double getY() {
                return y;
        }
        public void setX(double x) {
                this.x = x;
        }
        public void setY(double y) {
                this.y = y;
        }
        private double x;
        private double y;
}

class Point2DArray extends Point2DAbstract {
        public Point2DArray(double x, double y) {
                coords = new double[2];
                coords[0] = x;
                coords[1] = y;
        }
        //Copie
        public Object clone() {
                return new Point2DArray(coords[0], coords[1]);
        }
```

```
        //Accesseurs
        public double getX() {
                return coords[0];
        }
        public double getY() {
                return coords[1];
        }
        public void setX(double x) {
                coords[0] = x;
        }
        public void setY(double y) {
                coords[1] = y;
        }
        private double coords [];
}
class ColorInt extends ColorAbstract {
        public ColorInt(int r, int g, int b, int a){
                set(r, g, b, a);
        }
        public Object clone() {
                return new ColorInt(getR(), getG(), getB(), getA());
        }
        public int getR() {
                return (int)((_col & 0xFF000000) >> 24);
        }
        public int getG() {
                return (int)((_col & 0x00FF0000) >> 16);
        }
        public int getB() {
                return (int)((_col & 0x0000FF00) >> 8);
        }
        public int getA() {
                return (int)((_col & 0x000000FF));
        }
        public void setR(int r) {
                _col = _col & 0x00FFFFFF;
                _col = _col | ((int)b) << 24;
        }
        public void setG(int g) {
                _col = _col & 0xFF00FFFF;
                _col = _col | ((int)b) << 16;
        }
        public void setB(int b) {
                _col = _col & 0xFFFF00FF;
                _col = _col | ((int)b) << 8;
        }
        public void setA(int a) {
                _col = _col & 0xFFFFFF00;
                _col = _col | ((int)b);
        }
        private int _col;
}
class Color4Int extends ColorAbstract {
        public Color4Int (int r, int g, int b, int a){
                _r = r;
                _g = g;
                _b = b;
                _a = a;
        }
        public Object clone() {
                return new Color4Int(_r, _g, _b, _a);
        }
```

```
        public int getR() {
                return _r;
        }
        public int getG() {
                return _g;
        }
        public int getB() {
                return _b;
        }
        public int getA() {
                return _a;
        }
        public void setR(int r) {
                _r = r;
        }
        public void setG(int g) {
                _g = g;
        }
        public void setB(int b) {
                _b = b;
        }
        public void setA(int a) {
                _a = a;
        }
        private int _r, _g, _b, _a;
}
```

On veut maintenant créer des `Point2D` colorés. On aimerait définir un type `PointWithColor` qui hérite en même temps, par exemple, de `Point2DArray` et `ColorInt`. L'héritage multiple n'étant pas supporté par Java, nous allons le simuler de trois manières différentes. Pour cela, on peut créer une classe `PointWithColor` qui implémente les deux interfaces `Point2D` et `Color`.

Exercice 1. Dans un premier temps, créez la classe `PointWithColor_1` qui respecte les spécifications suivantes.

```
class PointWithColor_1 implements Color, Point2D {
        ...
        private double coords [];
        private int _col;
}
```

Que pouvez dire de cette solution ? Précisez ses inconvénients.

Exercice 2. Pour résoudre en partie ces problèmes, on propose une deuxième solution qui est basée sur le mécanisme dit de *délégation*. On va maintenant créer une classe `PointWithColor_2` qui contient une donnée membre de type `Point2DArray` et une donnée membre de type `ColorInt`.

```
class PointWithColor_2 implements Color, Point2D {
        ...
        private Point2DArray p;
        private ColorInt c;
}
```

Que pouvez vous dire de cette deuxième solution ? Quelles sont ses limites ?

Exercice 3. On aimerait maintenant créer des `Point2D` colorés qui soient de type `Point2DArray` ou bien `Point2DXY`, avec des couleurs définies par `ColorInt` ou `Color4Int`. Comment transformer la classe `PointWithColor_2` pour répondre à ces spécifications ? Implémenter votre proposition sous la forme d'une troisième classe `PointWithColor_3`.