Programmation avec des objets : Cours 7

Menu du jour: la notion d'héritage

1. Pourquoi?
2. Principes
3. Un exemple: figures du plan
4. Conclusion

A quoi sert l'héritage?

- Eviter les trop longs codes
- Organiser les composants d'un logiciel
- Elégance de certaines constructions
- Détermination lors de l'exécution de la fonction utilisée

Héritage : les classes

- Une classe `ClasseB` qui hérite d'une autre classe `ClasseA` contient les mêmes champs que `ClasseA` et éventuellement d'autres.
- Elle contient aussi les mêmes fonctions statiques, les mêmes méthodes, et éventuellement d'autres.
- Se déclare par

```
class ClasseB extends ClassA
```

Héritage : les objets

- Une classe `ClasseB` hérite d'une classe `ClasseA`
- Un objet `x` déclaré de la classe `B`, on peut lui appliquer les méthodes de la classe `A`.
- Si on applique la méthode `x.f()` alors ou bien la méthode est définie dans la classe `B`, elle est alors appliquée à `x`, sinon l'interpréteur examine si elle est définie dans la classe `A` si oui elle est appliquée à `x`
- On peut remonter à une classe dont hériterait la classe `A`

La hierarchie des classes

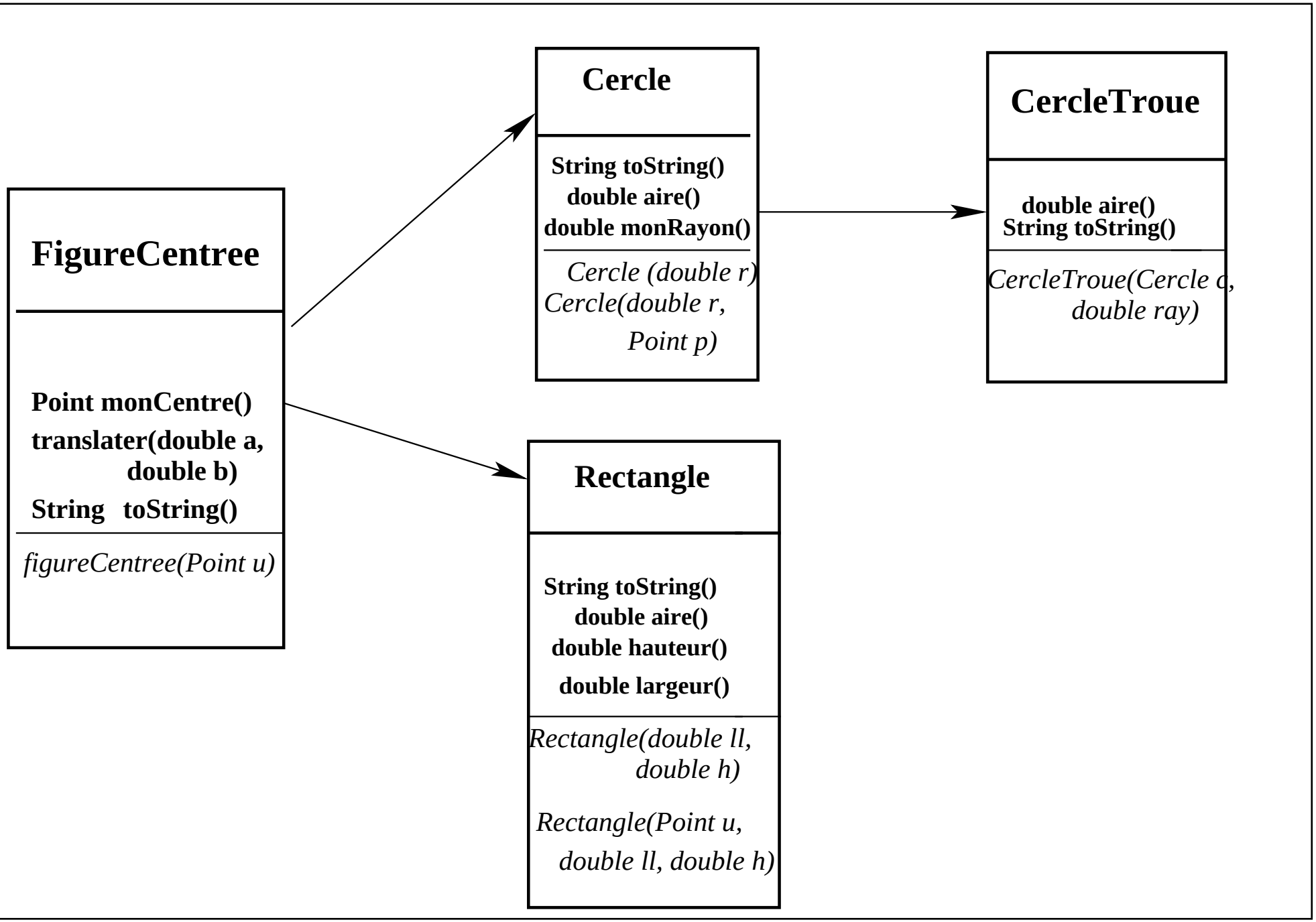
- Une classe en Java ne peut hériter directement que d'une seule classe (pas d'héritage multiple).
- L'héritage est transitif.
- Le sommet de la hierarchie est la classe: `Object`
- La mère d'une classe est notée: `super`

Quelle est la méthode qui s'applique à un objet?

- Déterminer la classe de l'objet!
- La classe (le type) d'un objet est définie une fois pour toute lors de sa construction: celle du constructeur qui l'a créé.
- La méthode qui s'applique à un objet est celle qui se trouve dans la classe décrivant son type sinon dans la classe mère, sinon dans la mère de sa mère ...

Un exemple: figures du plan

- Figures avec un centre
- Rectangles, carrés. losanges
- Cercles, cercles troués


Cercle
String toString()
double aire()
double monRayon()
Cercle (double r)
Cercle(double r,
Point p)
CercleTroue
double aire()
String toString()
CercleTroue(Cercle c,
double ray)
FigureCentree
Point monCentre()
translater(double a,
double b)
String toString()
figureCentree(Point u)
Rectangle
String toString()
double aire()
double hauteur()
double largeur()
Rectangle(double ll,
double h)
Rectangle(Point u,
double ll, double h)

La classe Point

```
class Point{
    private double abs;
    private double ord;
    Point (double x, double y)
    Point() { this(0,0);}
    public double abscisse () { return abs;}
    public double ordonnee() {return ord;}
    public void translater(double a, double b){
        abs += a;
        ord += b; }
    public String toString(){
        String a = " mon abscisse est: " + abs;
        String b = "   mon ordonnee est: " + ord +"\n";
        return a + b;}
}
```

Figure centrée

```
public class FigureCentree{
    Point centre;
    FigureCentree(Point u){
        centre = u;
    }
    Point monCentre() {
        return centre;
    }
    public String toString() {
        String f = centre.toString();
        return "Je suis une figure centree,
                mon centre est: \n" + f;
    }
```

Figure centrée (suite)

```
    void translater (double a, double b){
        centre.translater(a,b);
    }
    public static void main(String[] args){
        Point p = new Point(2,5);
        FigureCentree u = new FigureCentree(p);
        System.out.println(u);
    }
}
```

Rectangle

```
class Rectangle extends FigureCentree{
    Point coin;
    double largeur; double hauteur;
    Rectangle (double ll , double h){
        super(new Point(0,0));
        this.largeur = ll; this.hauteur = h;
        coin = new Point(-ll/2, -h/2);}
    }
    Rectangle (Point u, double ll, double h){
        super(u);
        this.largeur = ll; this.hauteur = h;
        double x = u.abscisse()-ll/2, y = u.ordonnee()-h/2;
        coin = new Point(x,y);}
```

Rectangle (suite)

```
double aire (){
    return this.hauteur * this.largeur;
}
public String toString(){
    Point p1 = coin;
    double a2 = p1.abscisse() + largeur;
    double b2 = p1.ordonnee();
    Point p2 = new Point(a2, b2);
    double a3 = a2;
    double b3 = b2 + hauteur;
    Point p3 = new Point(a3, b3);
    Point p4 = new Point(p1.abscisse(),b3);
    return p1.toString()  +  p2.toString() +
           p3.toString()   +  p4.toString() ;
}
}
```

Cercle

```
class Cercle extends FigureCentree{
    double rayon;
    Cercle (double r){
        super(new Point(0,0));
        rayon = r;
    }
    Cercle (double r, Point u){
        super(u);
        rayon = r;
    }
    Cercle (Cercle c){
        super(c.centre);    rayon = c.rayon;
    }
```

Cercle (suite)

```
    double monRayon(){
        return rayon;
    }
    double aire() {
        return Math.PI*rayon*rayon;
    }
    public String toString() {
         String u = " je suis un cercle de centre: "
              + centre.toString()
                   + "de surface : " + aire();
         return u;
    }
 }
}
```

CercleTroue

```
public class CercleTroue extends Cercle{
    Cercle trou;
    CercleTroue (Cercle c, double ray) {
        super (c.rayon, c.centre);
        trou = new Cercle(ray, super.centre);
    }
   double aire() {
        return super.aire() - trou.aire();}
    public String toString() {
         String u = " je suis un cercle troue de centre: ("
              + centre.toString()
                    + ")" + "de surface : " + aire();
         return u;}
}
```

Utilisation

```
public class Utilise{
    public static void main(String[] args){
        FigureCentree[] tab = new FigureCentree[5];
        Cercle c = new Cercle(5);
        Point p = new Point(5,3);
        tab[0] = c;
        tab[1] = new Rectangle(p, 2, 5);
        tab[2] = new Rectangle(3,6);
        tab[3] = new CercleTroue(c, 2);
        tab[4] = new Cercle(4,p);
        for (int i = 0; i < 5; ++i)
            System.out.println(tab[i]);
        }
    }
```

Conclusion

- Utiliser `extends`
- Attention à la fonction utilisée à l'exécution
- Problème de spécialisation/ surcharge