

# **Le Polymorphisme**

# Polymorphisme

- Le polymorphisme est étroitement associé à l'interaction entre l'héritage et la liaison dynamique
- Des objets « similaires » peuvent réagir différemment à un même message
- Exemple :
  - `class Polygon { ... float getArea\(\) { ... } ... }`
  - `class Rectangle extends Polygon { ... float getArea\(\) { ... } ... }`
  - `class Square extends Rectangle { ... float getArea\(\) { ... } ... }`

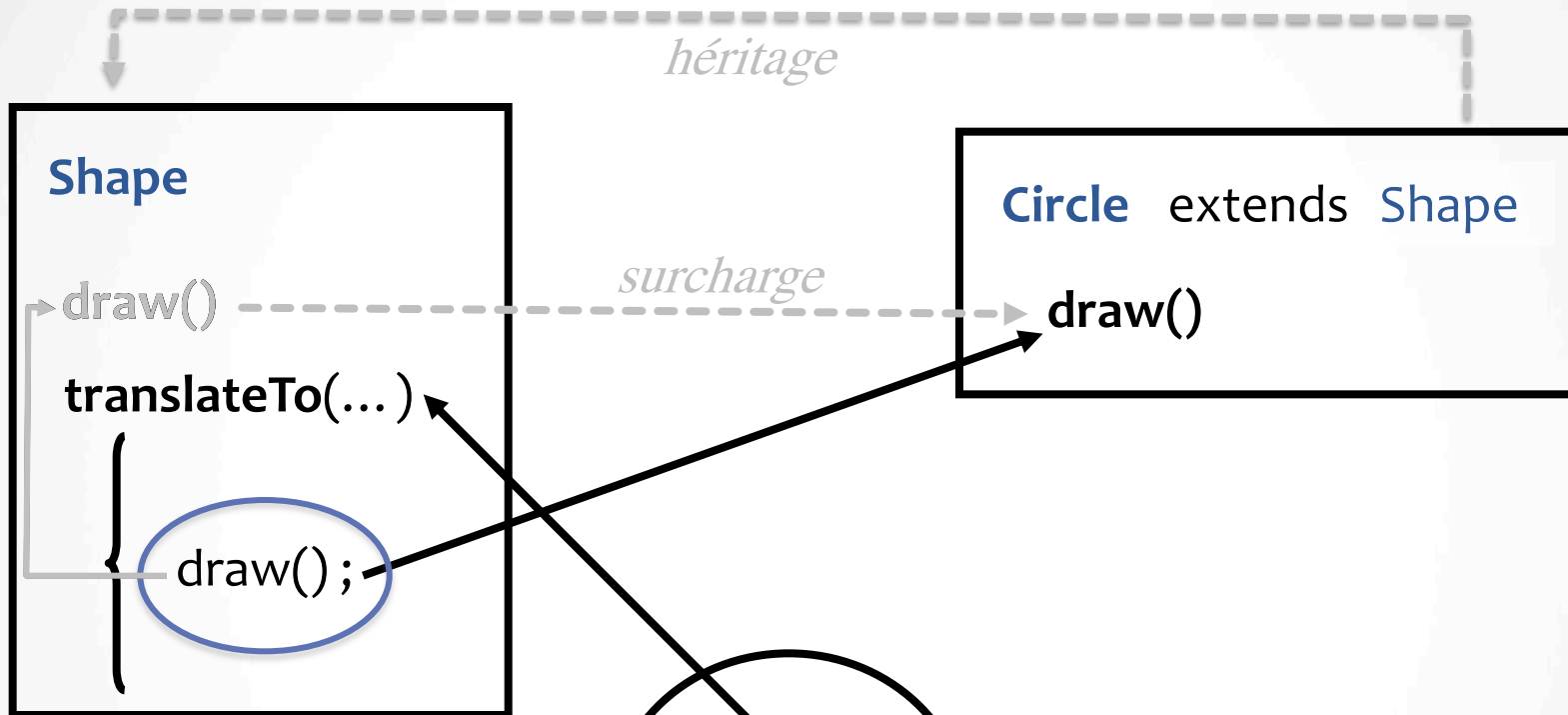
# Polymorphisme

- Exemple plus complexe :
  - toute forme géométrique sait se traduire
    - affichages successifs en faisant varier le point d'origine
      - methode `translateTo(...)` dans **Shape**
      - methode `draw(...)` dans **Shape**
  - des formes géométrique se dessinent différemment
    - class **Polygon** extends **Shape**
      - surcharge `draw(...)` dans **Polygon**
    - class **Circle** extends **Shape**
      - surcharge `draw(...)` dans **Circle**

# Polymorphisme

- On peut définir une variable de type **Shape** et lui affecter une instance de **Polygon** ou de **Circle** (car ce sont des **Shape** !)
- Lors d'un appel à **translateTo**, la fonction définie au niveau de **Shape** va faire appel au **draw** spécifique au sous-type : **Polygon** ou **Circle**

# Polymorphisme



```
Shape shape ;  
shape = new Circle() ;  
shape.translateTo(...) ;
```

# **CLASSE ABSTRAITE ET INTERFACE**

# Les classes abstraites

- Une classe abstraite est une classe ayant au moins une méthode abstraite
- Une méthode abstraite ne possède pas de définition
- Une classe abstraite ne peut pas être instanciée ( **new** )
- Une classe dérivée d'une classe abstraite ne redéfinissant pas toutes les méthodes abstraites est elle-même abstraite

# Les classes abstraites

```
abstract class Shape {  
    protected Point origin ;  
    public abstract double perimeter();  
    public abstract void draw() ;  
    public void translateTo(int x, int y){ ... draw() ; ... }  
}
```

```
class Circle extends Shape {  
    ...  
    public double perimeter() { return 2 * Math.PI * r ; }  
    public void draw() { ... }  
}
```

```
class Rectangle extends Shape {  
    ...  
    public double perimeter() { return 2 * (height + width) ; }  
    public void draw() { ... }  
}
```

```
Shape[] shapes = { new Circle(2), new Rectangle(2,3), new Circle(5) } ;  
double sum_of_perimeters = 0 ;  
for(int i=0; i<shapes.length; i++){  
    sum_of_perimeters += shapes[i].perimeter() ;  
}
```

# Les interfaces

- Une interface correspond à une classe où toutes les méthodes sont abstraites
- Une interface peut contenir des méthodes abstraites et des attributs statiques et finaux
- Une classe peut implémenter ( **implements** ) une ou plusieurs interfaces, tout en héritant ( **extends** ) d'une seule classe
- Une interface peut hériter ( **extends** ) de plusieurs interfaces

# Les interfaces

```
interface Colored{  
    public void setColor(Color color);  
    public Color getColor();  
}
```

```
class Circle extends Shape implements Colored, Serializable{  
    private int red, green, blue, alpha;  
    ...  
    public void setColor (Color color) { ... };  
    public Color getColor(){ return new Color(red, gree, blue, alpha); }  
}
```

```
Class Car implements Colored{  
    private Color color;  
    ...  
    public void setColor (Color color) { ... };  
    public Color getColor(){ return color.clone(); }  
}
```

```
Colored[] colored= { new Circle(2), new Car(), ... };  
for(int i=0 ; i<colored.length ; i++){  
    colored[i]. setColor(Color.BLUE);  
}
```