Le concept d'héritage

Christelle CAILLOUET
(christelle.caillouet@unice.fr)

Sous-typage

- Idée de la redéfinition des méthodes *equals*, clone, *toString*, *getClass* …
  - Spécifier par rapport à l'objet réellement contenu dans la variable

- L'affichage d'un objet `Object` est différent de l'affichage d'un `Point`
  - … mais les 2 peuvent s'afficher
  - Ils disposent tous les deux de la méthode `toString()`

Plus généralement

- On voudrait avoir des **types**
  - Sur lesquels **un ensemble de méthodes est disponible**
  - Mais dont la définition exacte dépend du **sous-type**
  - La méthode finalement exécutée sera **la plus précise** possible

- Exemple :
  - toute figure géométrique a une surface, mais la surface d'un carré ne se calcule pas comme la surface d'un cercle…

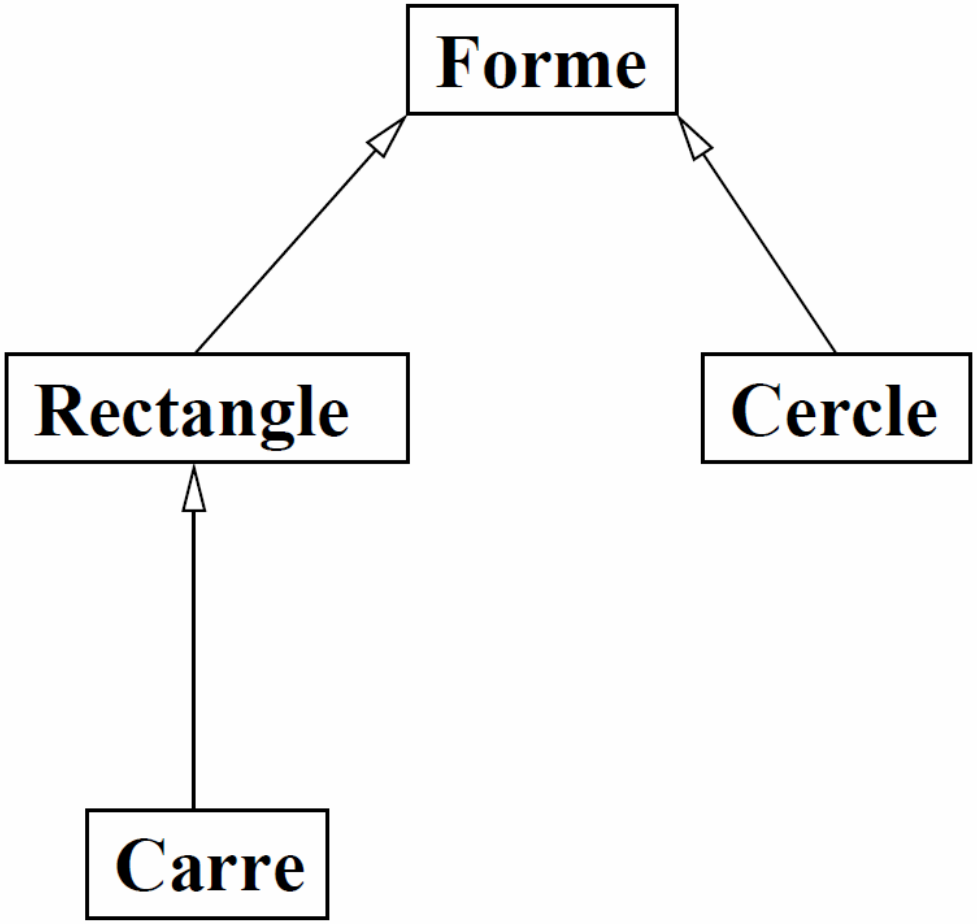


FIGURE 4.1 – Exemple de relations d'héritage

L'héritage

- 1 des fondements de la POO

- Idée :
  - Organiser les classes de manière hiérarchique
  - Définir des sous-types

- Favorise
  - le développement et le réutilisation de composants
  - La conception par spécialisation progressive

- Modélise la relation de spécialisation « ***is a*** »

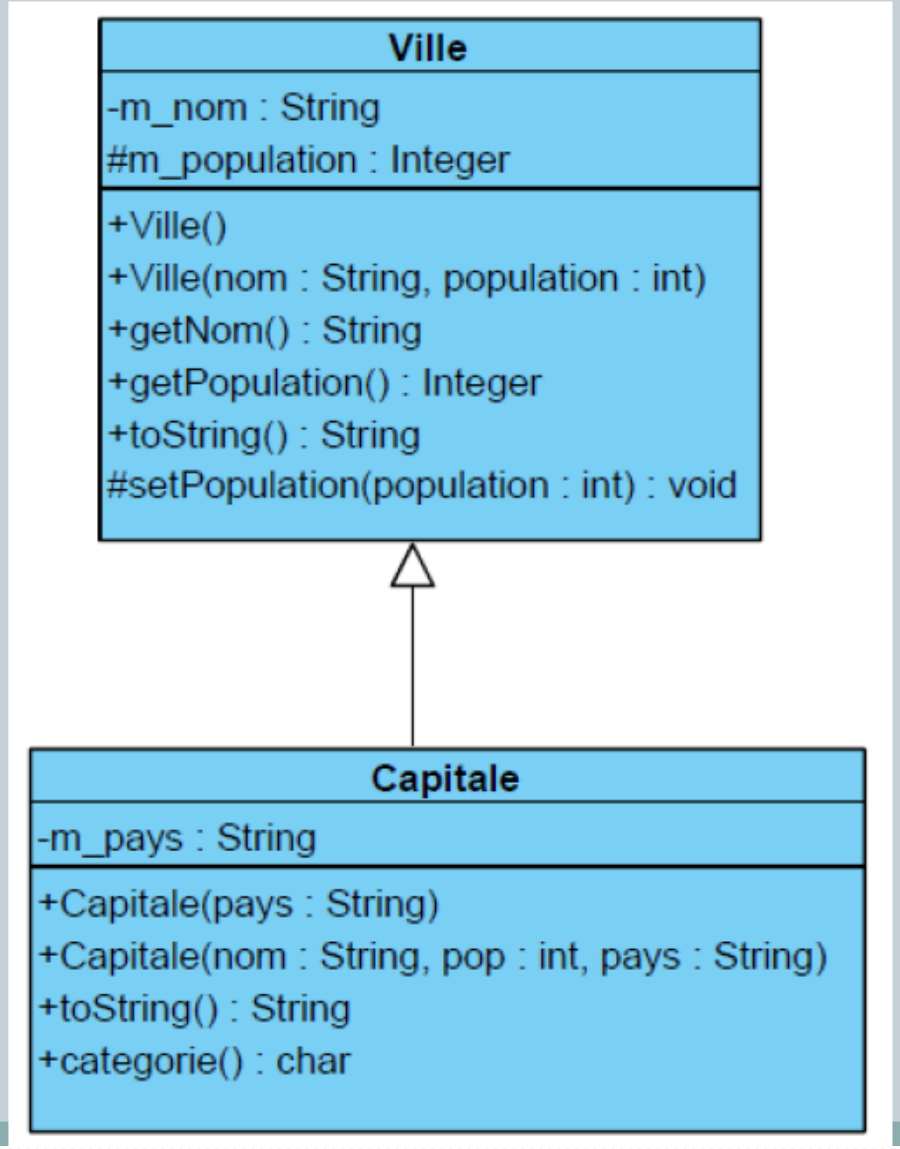
L'héritage

- Vocabulaire :
  - Si une classe **B** *hérite* d'une classe **A**
  - **B** est une *sous-classe/classe dérivée/spécialisation/cas particulier* de **A** qui est appelée *classe de base/super-classe*.
  - Les objets instanciant **B** instancient également **A**

- La classe **B** hérite d'emblée des fonctionnalités de **A**

➔ Développer des nouveaux outils en se fondant sur certains acquis

spécialisation



généralisation

Exemple : la classe Point

```
public class Point {
        private double abscisse;
        private double ordonnee;

        public Point(){
                abscisse = 0.0;
                ordonnee = 0.0;
        }

        public Point(double x, double y){
                abscisse = x;
                ordonnee = y;
        }

        public double getAbscisse(){ return abscisse; }
        public double getOrdonnee(){ return ordonnee; }

        public double distanceP(Point p){
                double x1=abscisse, x2=p.abscisse;
                double y1=ordonnee, y2=p.ordonnee;
                return Math.sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
        }

        public String toString(){
                return "("+abscisse+" , "+ordonnee+")";
        }
}
```

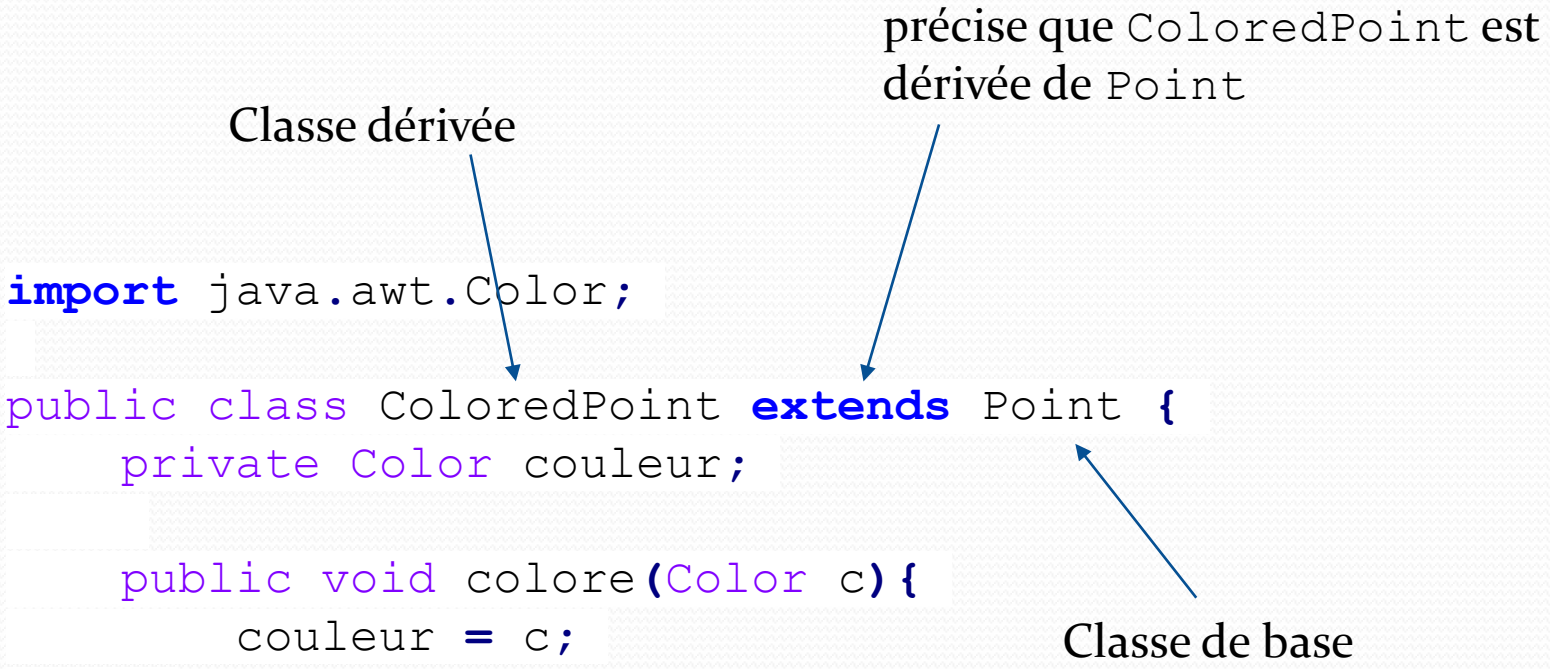
Une classe dérivée : le Point coloré

- Manipuler les points colorés d'un plan
- Une telle classe peut manifestement disposer des mêmes fonctionnalités que la classe `Point`
- On pourrait y ajouter, par exemple, une méthode *colore* chargée de définir la couleur

➔ Classe `ColoredPoint` dérivée de la classe `Point`

Définition d'une classe dérivée

- Utilisation du mot-clé **extends**

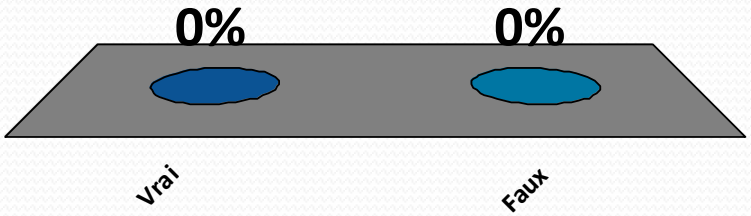


Tout objet `ColoredPoint` est avant tout un objet `Point` ?

✓ A. Vrai

B. Faux

20



En Java, l'héritage multiple est interdit. Qu'est-ce que cela signifie ?

A. Une classe de base ne peut pas avoir + d'une classe dérivée.

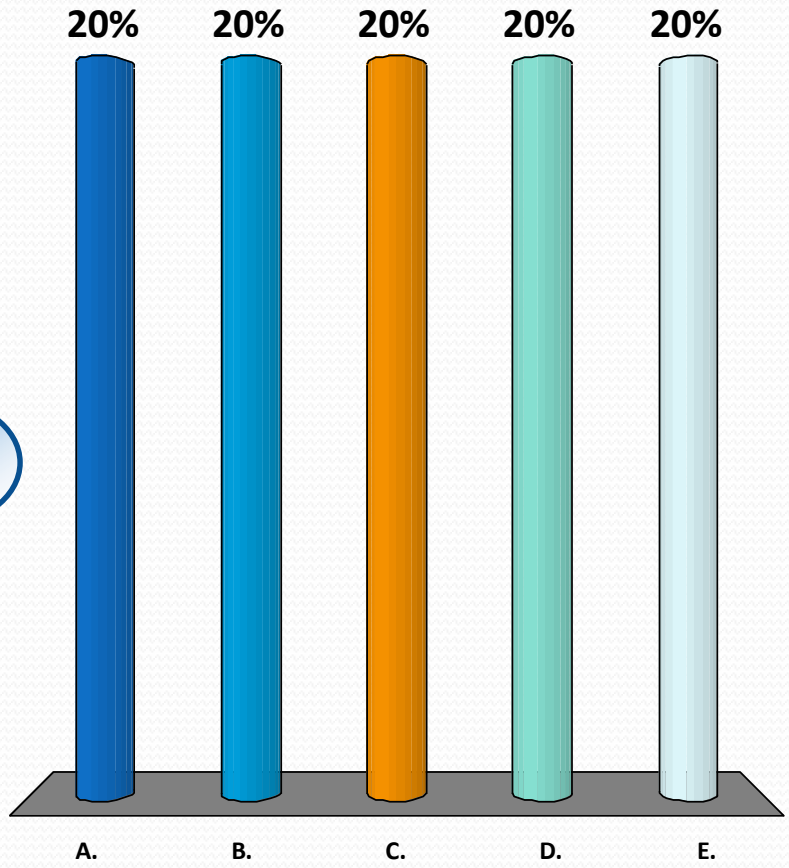
✓ B. Une classe dérivée ne peut pas avoir + d'une classe de base.

C. Une classe dérivée ne peut pas dériver elle-même.

D. 2 classes dérivées ne peuvent pas dériver de la même classe.

E. Un programme ne peut pas contenir + d'une classe dérivée.

60



Combien d'attributs possède un objet `ColoredPoint` ?

```
public class Point {
        private double abscisse;
        private double ordonnee;
        …
public class ColoredPoint extends Point {
    private Color couleur;
    …
```

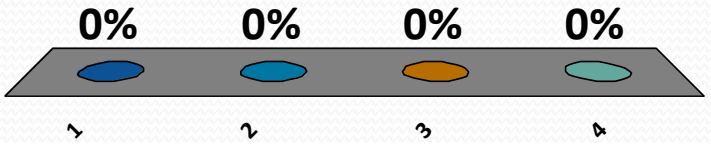
A. 1

B. 2

C. 3

D. 4

20



Tous les champs sont hérités

- **Une classe dérivée hérite des attributs et méthodes de sa classe de base**… mais ne peut pas forcément y accéder !

- Ils peuvent être manipulés directement si leur accessibilité le permet
  - Si `abscisse` n'est pas *private* dans `Point`, on peut y accéder directement par `this.abscisse` dans `ColoredPoint`
  - Sinon, on passe par sa méthode *accesseur*, ou un constructeur

Modificateur du membre	`private`	*défaut*	`protected`	`public`
Accès depuis la classe	Oui	Oui	Oui	Oui
Accès depuis une classe du même package	Non	Oui	Oui	Oui
Accès depuis une sous-classe	Non	Non	Oui	Oui
Accès depuis tout autre classe	Non	Non	Non	Oui

Accessibilité des méthodes

- Un objet de type `ColoredPoint` :
  - Peut faire appel aux méthodes de la classe `ColoredPoint` (ici *colore*)
  - Mais aussi aux méthodes publiques de la classe `Point`: accesseurs, *distanceP*, *toString*
- Bytecode partagé entre classe de base et dérivée
- Possibilité de redéfinition et surcharge des méthodes par la classe dérivée (cf. prochain cours)

Les constructeurs ne se dérivent pas

- En Java, le constructeur de la classe dérivée doit prendre en charge l'intégralité de la construction de l'objet (attributs spécifiques et hérités)

- On peut appeler les constructeurs de la classe de base à l'aide du mot-clé ***super*** afin d'initialiser les attributs privés par exemple

- ***super***, c'est ***this*** vu avec le type de la classe de base

Construction et initialisation d'objets dérivés

```
import java.awt.Color;

public class ColoredPoint extends Point {
    private Color couleur;

    public ColoredPoint(){
        // Appel par défaut à super()
        couleur = Color.black;
    }

    public ColoredPoint(double x, double y, Color c){
        super(x,y); // Appel au constructeur de la classe Point
        couleur = c;
    }

    public void colore(Color c){
        couleur = c;
    }
}
```

Construction et initialisation d'objets dérivés

- **Remarques** :
  - L'instruction ***super*** doit être la 1ère instruction du constructeur de la classe dérivée
  - Si aucun appel ***super*** n'est fait, appel implicite au constructeur vide de la classe de base (comme si ***super()*** était présent)
    Si aucun constructeur vide n'est présent dans la classe de base → erreur de compilation

Interdire l'héritage

- Rendre une classe non accessible à sa classe dérivée
  `private class Point{…}`
  ou classes non publiques dans des packages différents

- Mot-clé **final** associé à la classe
  `public final class Point{…}`

➔ Aucune classe ne peut dériver d'une classe *privée* ni d'une classe *finale*