Informatique 1

13a. P.O.O
Héritage

Dr Pierre-André Mudry
pierre-andre.mudry@hevs.ch

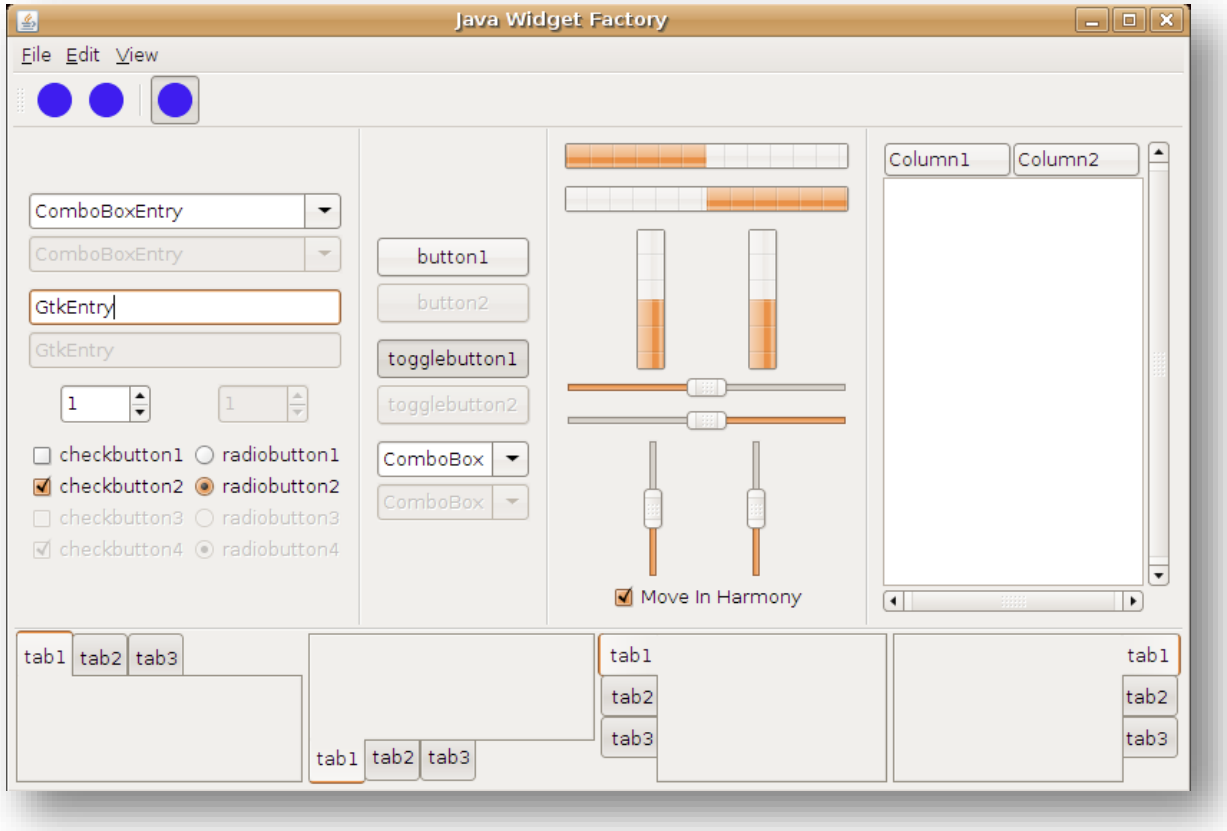
Rev 2.05

De la matière supplémentaire ?

- Programmation impérative ✓
- Orienté objet
  - Modéliser des objets :
    - Attributs ✓
    - Méthodes ✓
  - Créer des objets
    - Instances ✓

Il manque quelque chose…

- Relations entre les classes
  - *Héritage*
- «Obligations» pour les objets
  - *Interfaces et classes abstraites*
- Interfaces utilisateurs
  - Il faut le tout…



Objectifs de la thématique

Comprendre et pouvoir utiliser quelques principes avancés de POO

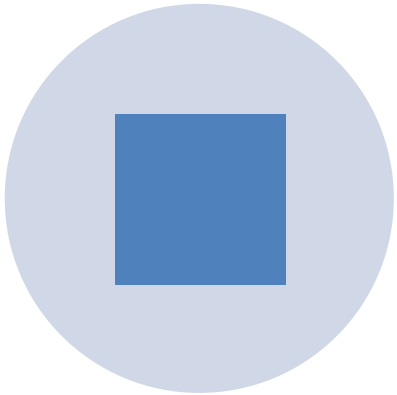
- Héritage
- Règles de visibilité
- Interfaces et classes abstraites

Passage des attributs et méthodes aux classes «enfant»

13.1. HÉRITAGE

Motivation initiale

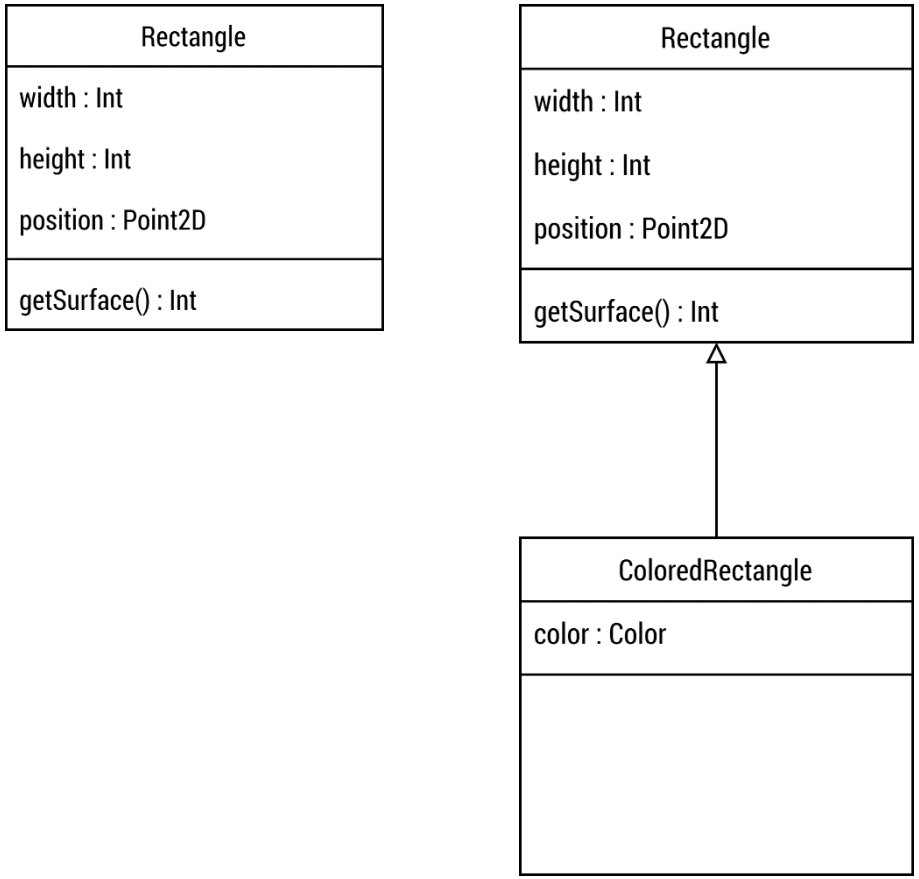
Rectangle



RECTANGLE

RECTANGLE
COLORÉ

Représentation UML



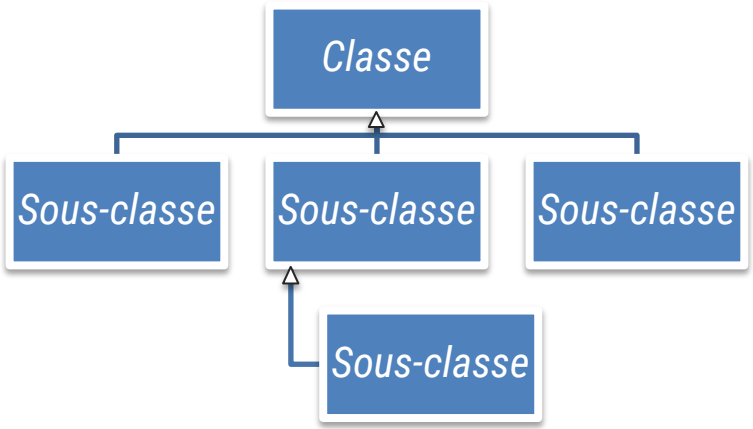

```
class Rectangle {
    int width;
    int height;
    Point2D position;

    public double getSurface() {
        return width * height;
    }
}
```

```
class ColoredRectangle extends Rectangle
{
    Color c;
}
```

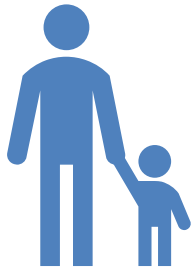
Héritage (2)

Héritage : technique permettant de créer des classes *spécialisées*, appelées sous-classes, à partir de classes existantes.

Définition



Héritage (3)



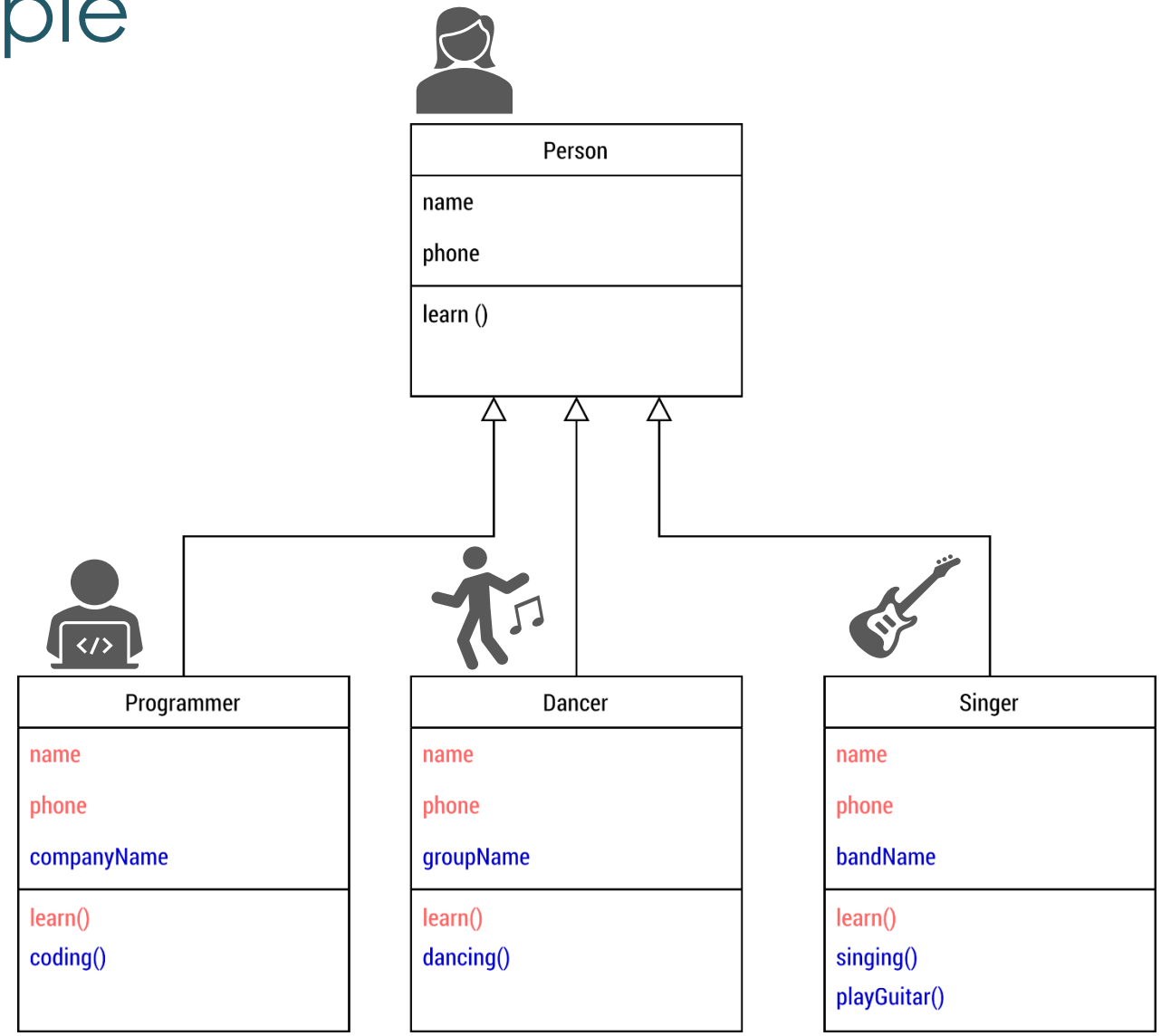
Par héritage, la sous-classe:

Hérite des méthodes du parent
Hérite des attributs du parent

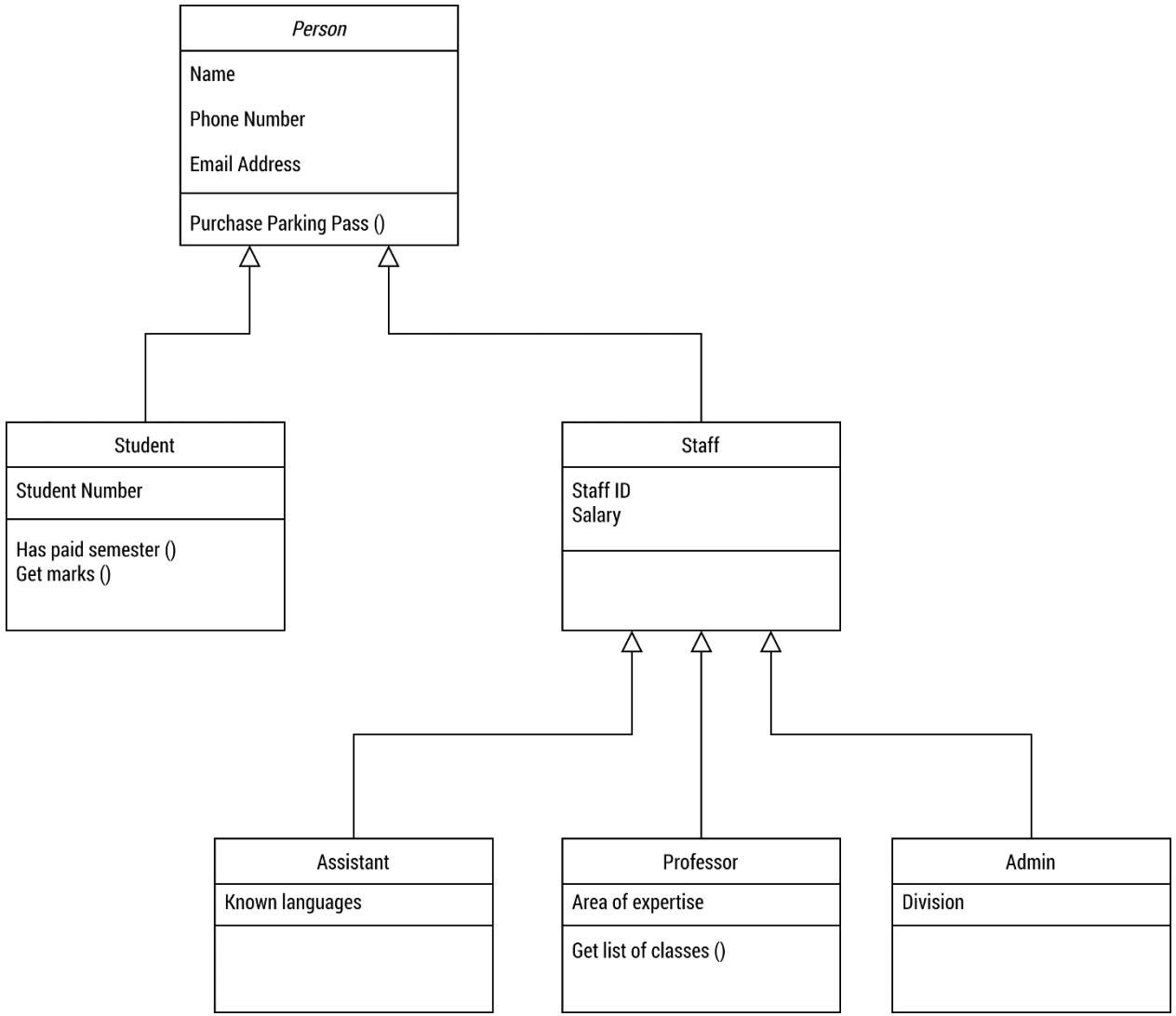
La sous-classe peut:

Ajouter de nouvelles méthodes
Ajouter de nouveaux attributs
Redéfinir des méthodes existantes

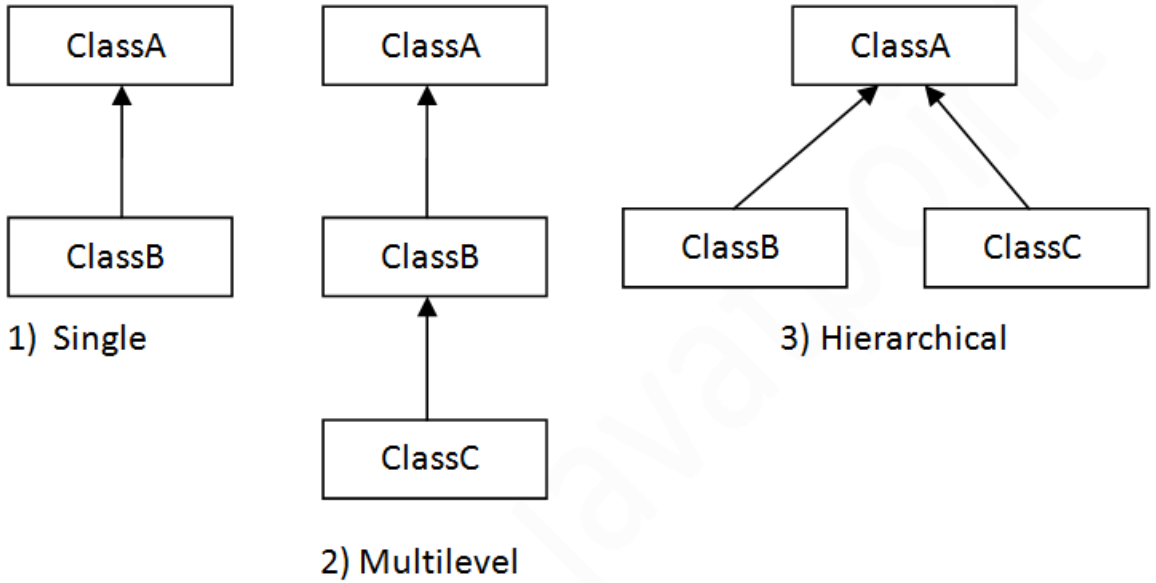
Autre exemple

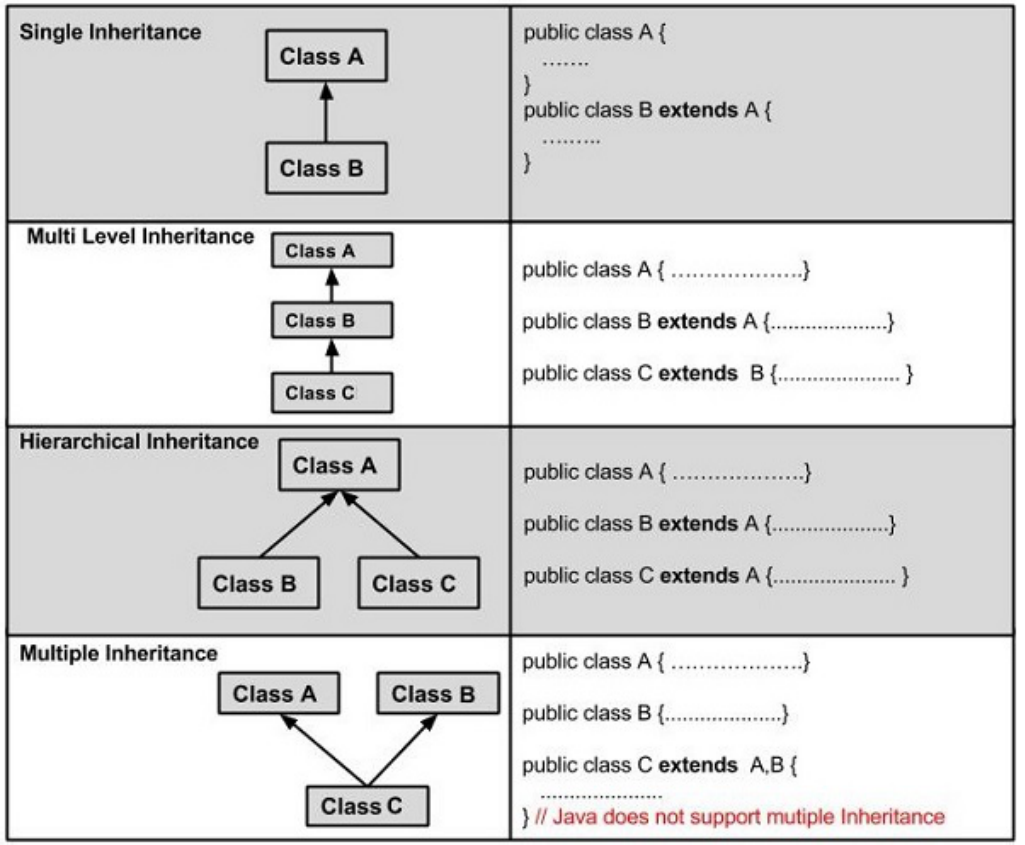


Hiérarchie plus profonde



Types d'héritage et implémentation





Surcharge de méthodes

```
class Adult {
    String name;

    String talk() {
        return "Hello. My name is " + name;
    }
}

class Child extends Adult {
    String talk() {
        return "Hey! I'm a child and my name is " + name;
    }
}
```

Exemple Surcharge

Constructeur superclasse

- Le constructeur de la sous-classe doit toujours comporter un appel **`super`** (sauf si la superclasse a un constructeur par défaut).
- Doit être la première instruction du constructeur de la sous-classe.

Exemple constructeur superclasse

```
class Rectangle {
    int width;
    int height;
    Point2D position;

    Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    Rectangle(int side) {
        this(side, side);
    }
}
```

```
class ColoredRectangle extends Rectangle {
    Color c;

    public ColoredRectangle(int width, int height) {
        super(width, height);
    }
}
```

Restrictions et permissions

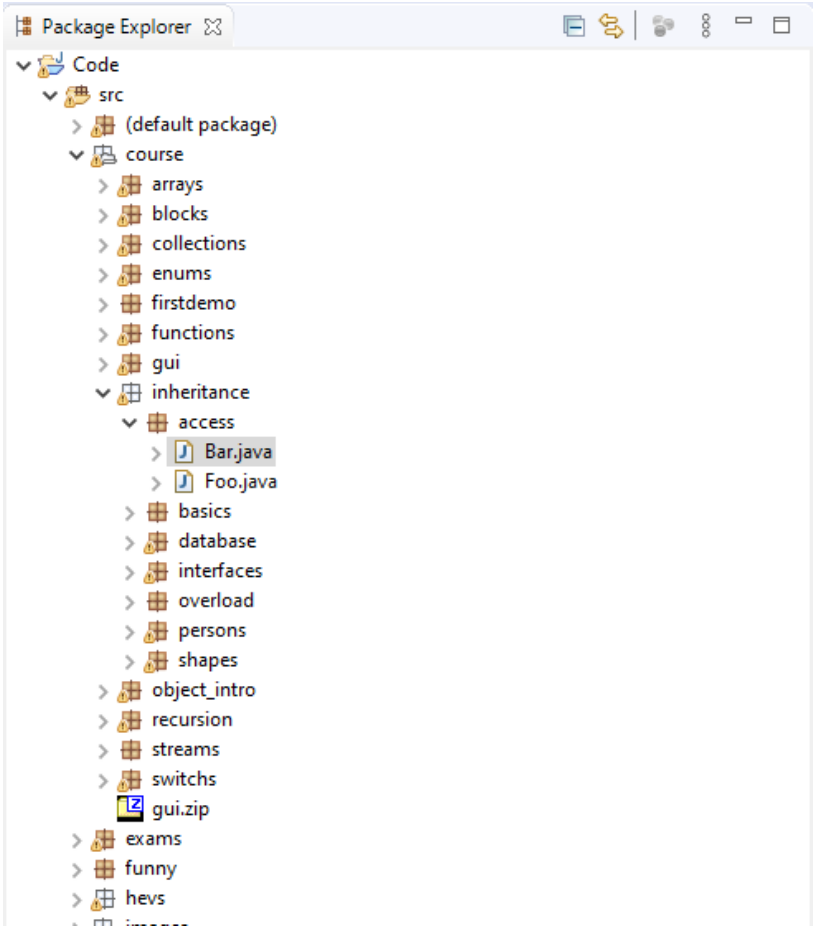
13.2. RÈGLES DE VISIBILITÉ ET D'HÉRITAGE

Règles de visibilité

- **`public`**
  - Visible extérieur classe + toute hiérarchie
- **`private`**
  - Visible uniquement dans la classe
- **`protected`** (*nouveau mot-clé*)
  - Visible dans les sous-classes de la hiérarchie mais pas à l'extérieur
- Rien
  - Visible dans le package

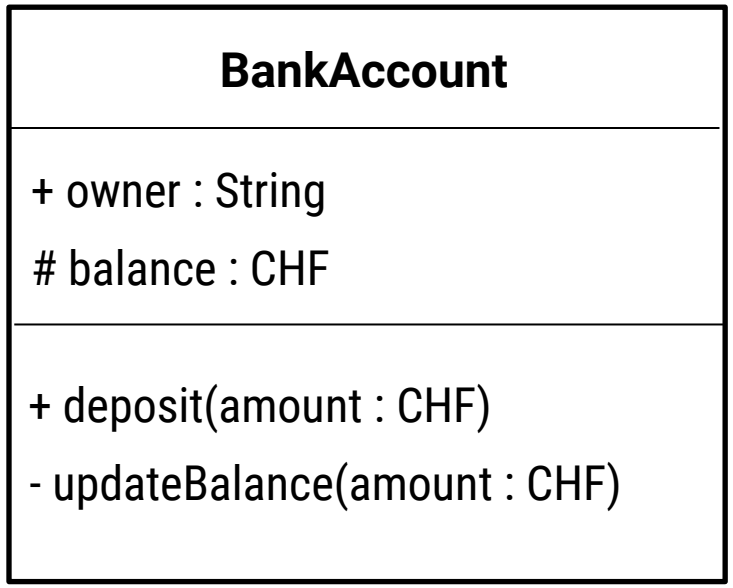
Exemple
Règles de visibilité, Rectangle

Accès et héritage (attribut et méthodes)



	Class	Sub class	Package	World
Public	Y	Y	Y	Y
Protected	Y	Y	Y	N
Private	Y	N	N	N
No modifier	Y	Y	Y	N

Notation UML pour la visibilité



Contrôle des attributs : getters / setters

- Lorsque l'on souhaite un contrôle sur les accès aux attributs :
  -
  -
  -

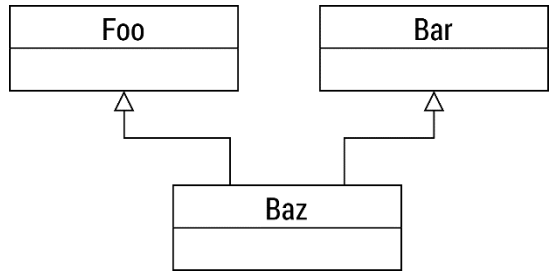
Exemple
Getters et setters

Code final

```
class Rectangle {
    protected int width;
    protected int height;
    protected Point2D position;

    public Point2D getPosition() {
        return position;
    }

    public void setHeight(int h) {
        height = Math.abs(h);
    }

    public void setWidth(int w) {
        width = Math.abs(w);
    }

    Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    public double getSurface() {
        return width * height;
    }
}
```

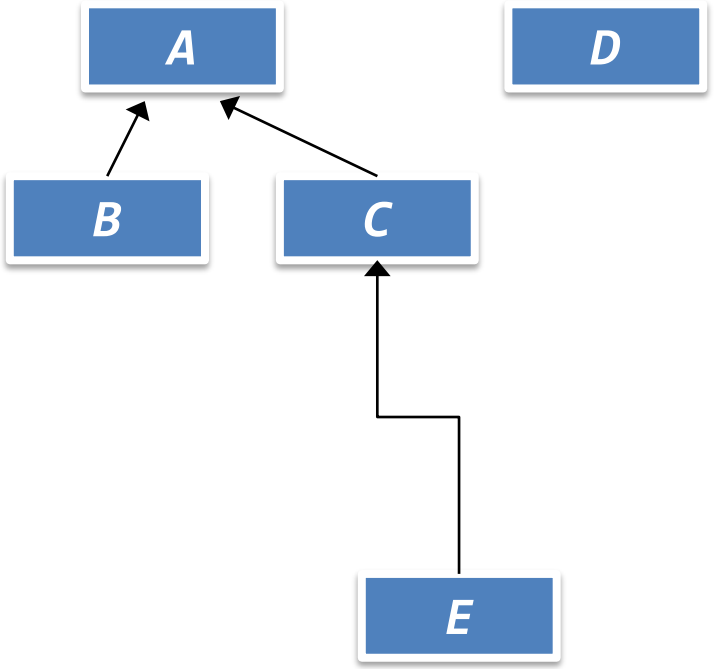
Remarques

- En Java, une classe hérite que d'une seule classe (pas d'héritage multiple)
- Parfois délicat…



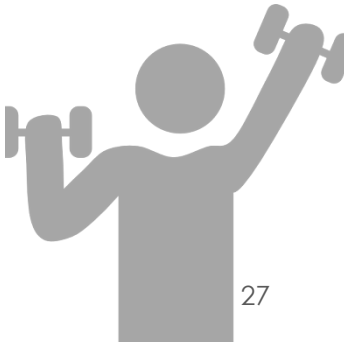
Exercice – Une variable

- Déclarée `public` dans A, est-elle
  - Accessible dans E ?
  - Accessible dans D ? Accessible dans C ?
- Déclarée `protected` dans A, est-elle
  - Accessible dans B ?
  - Accessible dans E ?
- Déclarée `private` dans C
  - Accessible dans E ?
- Déclarée `protected` dans C
  - Accessible dans E ?
  - Accessible dans D ?
  - Accessible dans A ?



Faire quiz en ligne chez vous

Exercice Building



Différences et similitudes

13.3. INTERFACES ET CLASSES ABSTRAITES

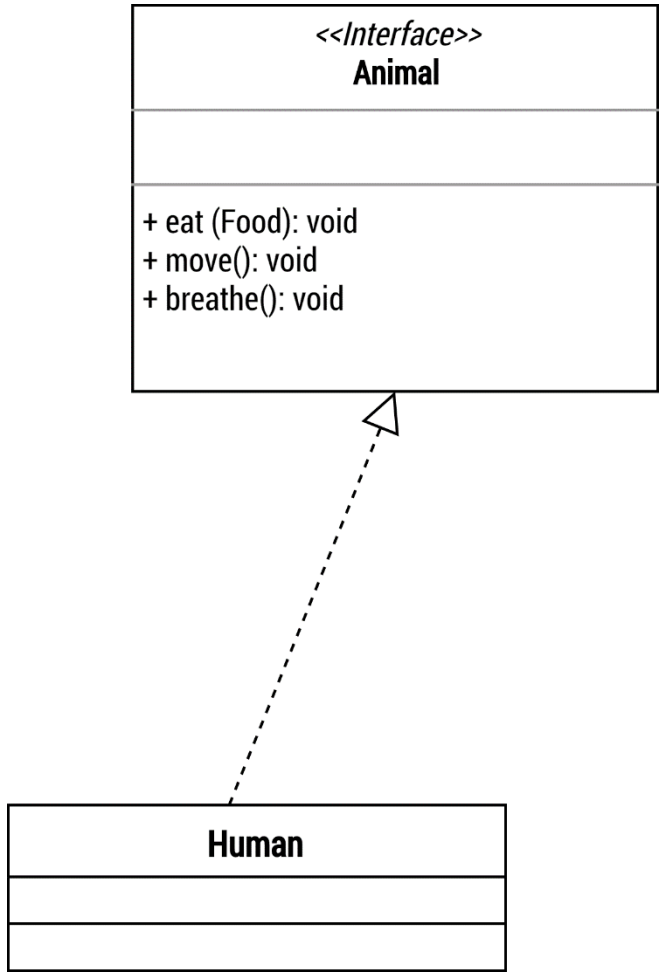
Interface

- Permet d'indiquer ce que les classes doivent posséder comme méthodes
  - Interface ≅ voici à quoi ressemblera toutes les classes qui implémentent cette interface
- Seulement prototype des méthodes

```
interface Animal {
  void eat(Food f);
  void move();
  void breathe();
}
```

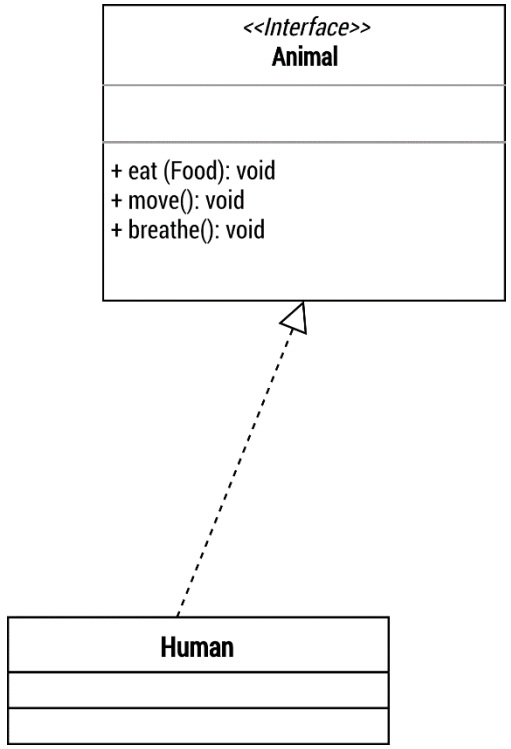
Implémentation d'une interface

- Mot-clé implements
- Une classe peut implémenter une ou plusieurs interfaces → ≈ héritage multiple
- Ce mot clé s'utilise dans la classe utilisant l'interface.



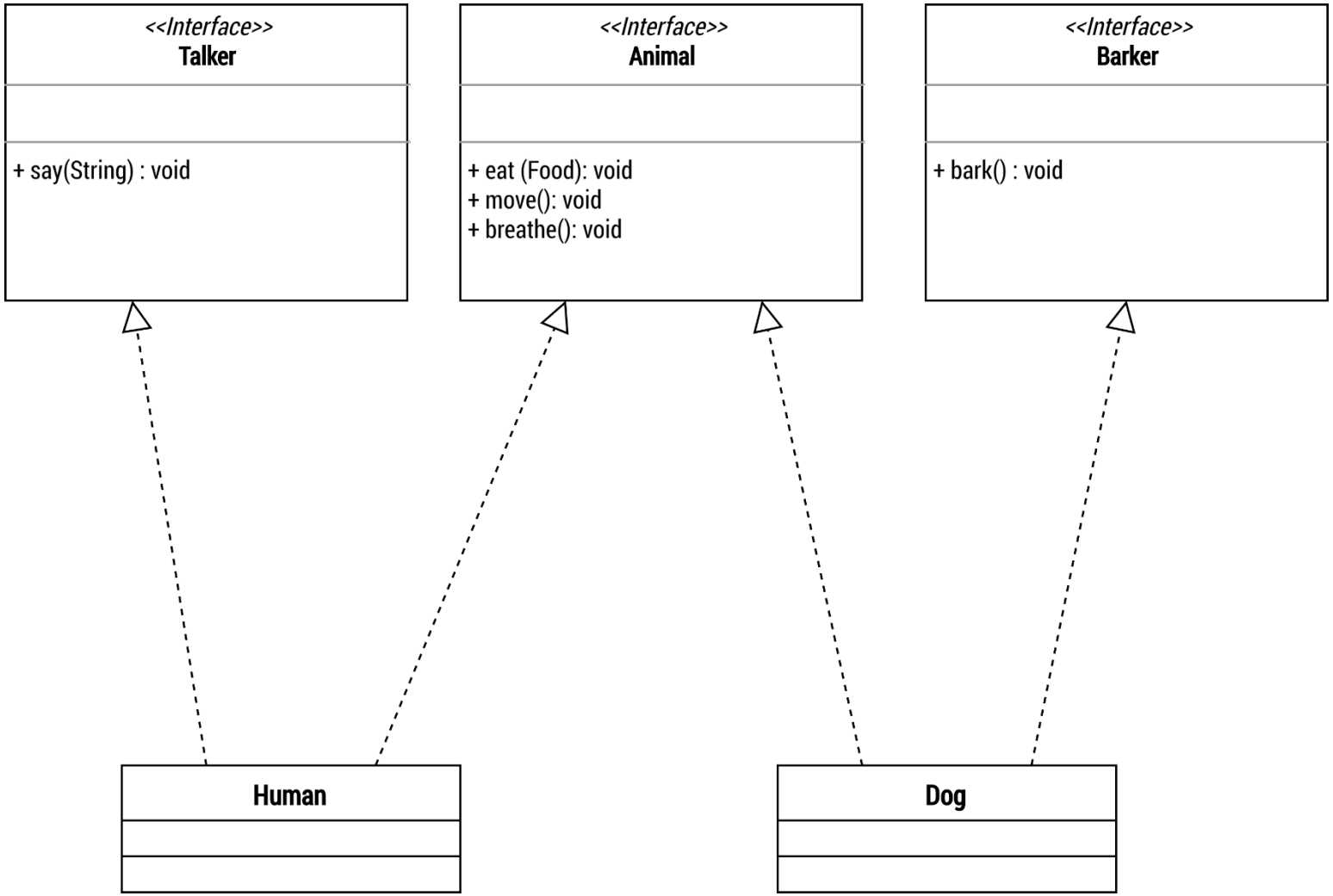
Exemple
Human avec interfaces

Code de l'exemple




```
interface Animal {
    public void eat(Food x);
    public void move();
    public void breathe();
}

class Human implements Animal {
    public void breathe() {
        System.out.println("Inspire, expire");
    }

    public void eat(Food x) {
        System.out.println("I am eating " + x + ". This is
good.");
    }

    public void move() {
        System.out.println("I move here and there");
    };
}
```

Exemple plus complexe



Exemple
Human et Dog avec interfaces

Exemple

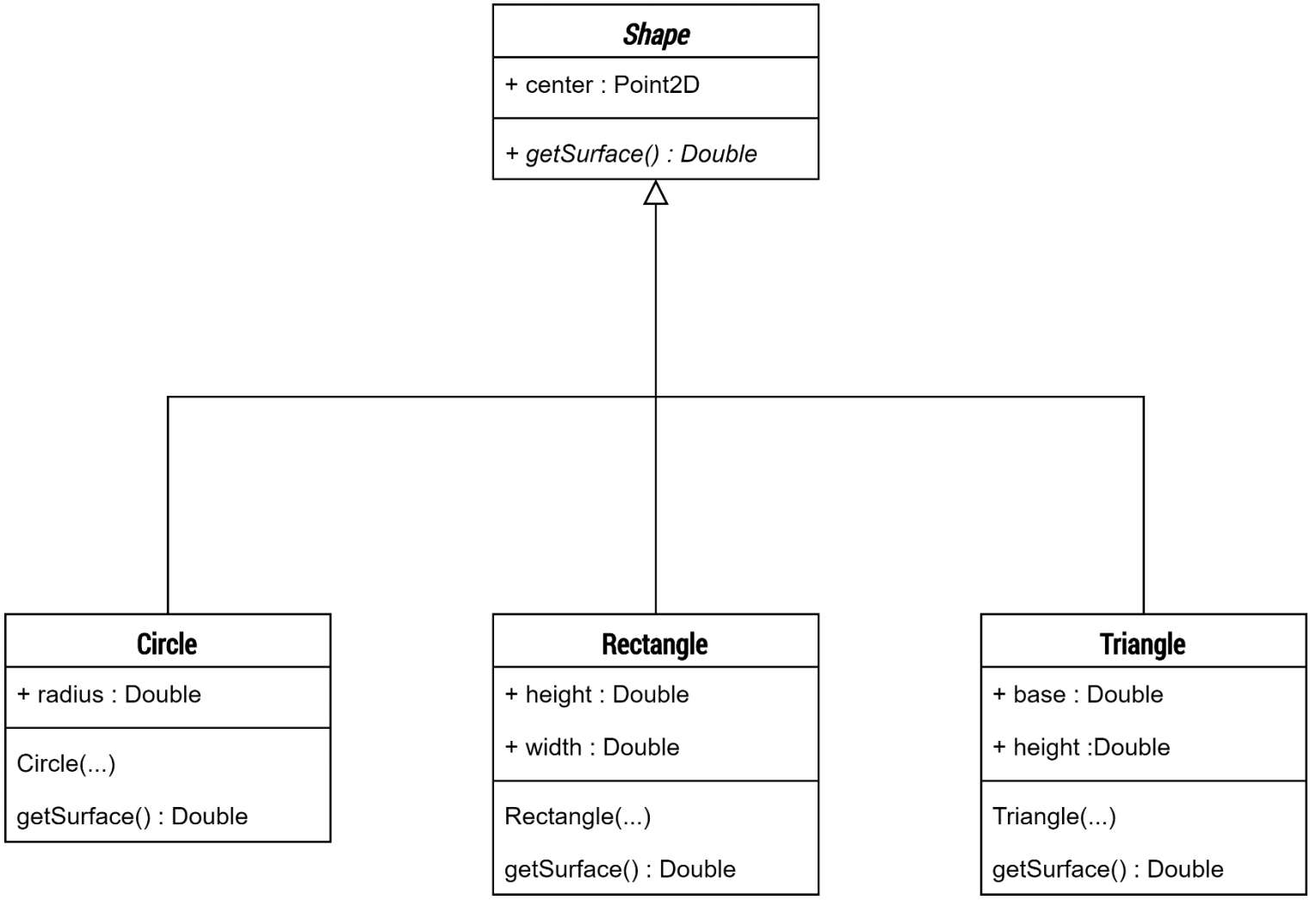
```
public class Dog implements Animal, Barker
{
    public void bark() {
        System.out.println("Woof woof");
    }

    public void breathe() {
        System.out.println("Wu haha");
    }

    public void eat(Food food) {
        System.out.println("Gnorfl");
    }

    public void move() {
        System.out.println("Tip tap");
    }
}
```

```
public class HumanComplete implements Animal, Talker {
    public void say(String text) {
        System.out.println("I say : \"" + text + "\"");
    }

    public void breathe() {
        System.out.println("Inspire, expire");
    }

ublic void eat(Food x) {
    System.out.println("I am eating " + x + ". This is
);


blic void move() {
    System.out.println("I move here and there");
```

Classe abstraite

- Mot-clé abstract
  - Devant la classe, signifie peut pas être instanciée.
  - Classe abstraite peut avoir des attributs.
  - Aussi devant méthode → méthode abstraite
- Pourquoi ?

Classe abstraite, exemple



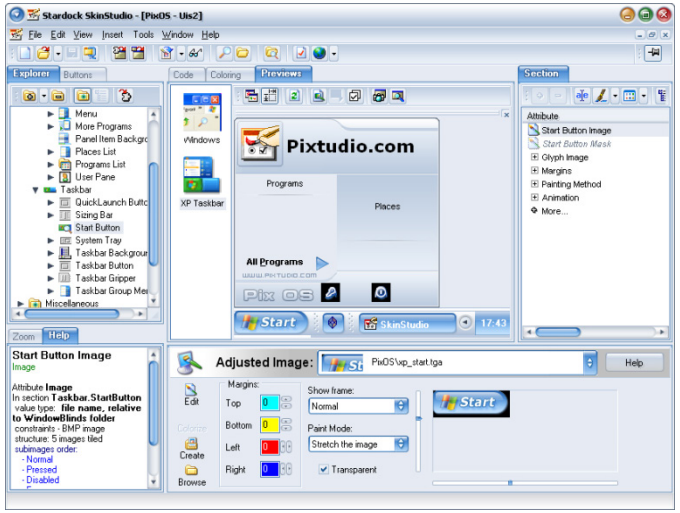
Exemple Formes

Classe abstraite VS interface

- Classe abstraite
  - ✓ Méthodes implémentées ou non
  - ✓ Possible d'avoir des attributs
  - X Pas d'héritage multiple
- Interface
  - ✓ Une classe peut implémenter plusieurs interfaces
  - X Pas d'attributs
  - X Que des prototypes de méthodes

Conclusion

- *Héritage* : moyen d'éviter de réécrire du code
- *Interfaces* : moyen de spécifier ce que doit faire une classe
- *What's next ?*
  - Interfaces graphiques !



Exercice
Classe abstraite Vehicle