ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

02
Chapitre

Un paradigme : La Conception Orientée Objet

1I2AC1 : Génie logiciel et Conception orientée objet

Régis Clouard, ENSICAEN - GREYC

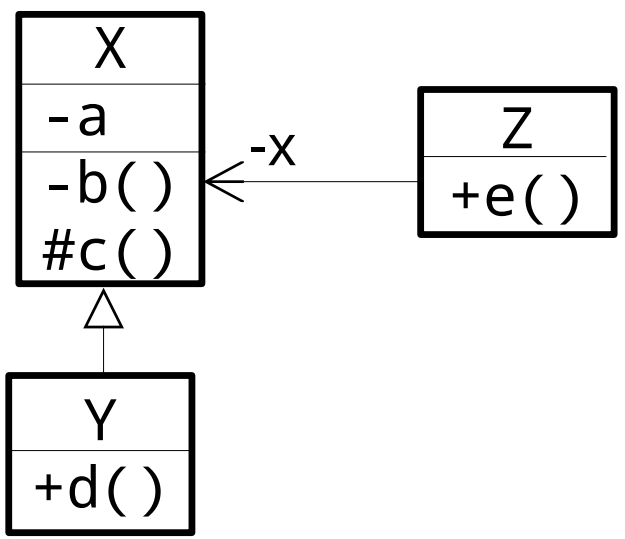
« N'importe quel programmeur peut écrire
du code que l'ordinateur comprend.
Les bons programmeurs écrivent du code
que les humains peuvent comprendre. »
Martin Fowler

(a) Visibilité

- Limiter l'accès aux membres des classes
  - attributs
  - méthodes
  - associations

	Notation UML	Accès aux membres de la classe par d'autres classes	Accès aux membres de la classe par des sous-classes
public	+	√	√
protected	#	X	√
private	-	X	X

Quiz



Soit trois variantes de la méthode Y::d(), les codes suivants sont-ils compilables ?

```
1. void d() { a=5; }
2. void d() { b(); }
3. void d() { c(); }
```

X
X
√

Soit quatre variantes de la méthode Z::e(), les codes suivants sont-ils compilables ?

```
4. void e() { x.a=5; }
5. void e() { x.b(); }
6. void e() { x.c(); }
7. void e() { x.d(); }
```

X
X
X
X

Le code de la méthode X::c() est-il compilable ?

```
8. void c() {
       X x = new X();
       x.b();
   }
```

√

Visibilité

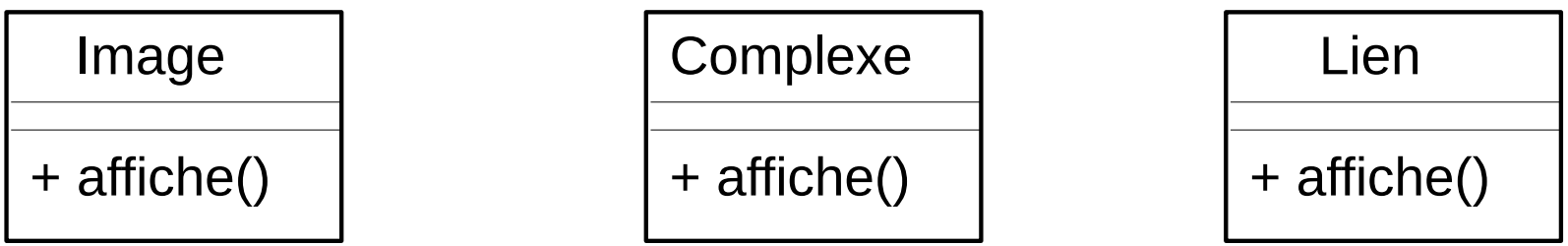
- Règle : restreindre le plus possible la visibilité
- Intention
  - Sécuriser la représentation interne des classes
  - Profiter du compilateur pour garantir l'encapsulation

Règle intangible

Pour assurer l'encapsulation les attributs et associations sont TOUJOURS privés

(b) Portée des noms

- Il est possible de donner un même nom à des méthodes de classes différentes

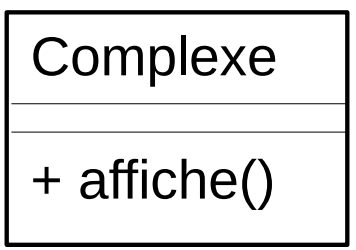


  - La portée des noms est locale
  - Le choix est levé à la compilation : **liaison statique**

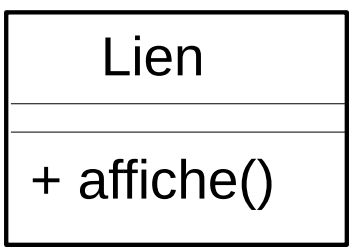
Quiz

A

B



C



Quelle méthode `affiche()` est exécutée (A, B ou C) ?

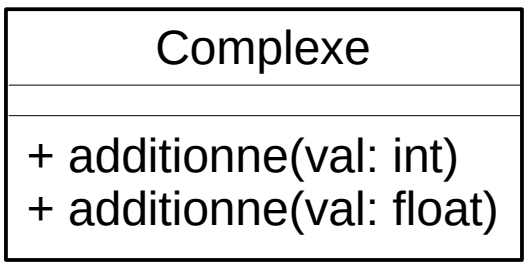
```
Image i = new Image();
i.affiche();
Complexe c = new Complexe();
c.affiche();
```

A

B

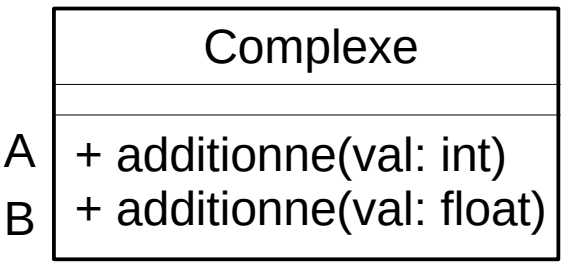
(c) Surcharge

- Dans une même portée, donner un même nom de méthode mais avec des signatures différentes
  - **Signature d'une méthode** :
    - nom + paramètres (ni le type de retour, ni les exceptions)



  - Le choix est levé à la compilation : **liaison statique**

Quiz



Quelle méthode `additionne()` est exécutée (A ou B) ?

```
Complexe c = new complexe();
c.additionne(5f);
c.additionne(5);
c.additionne(5d);
```

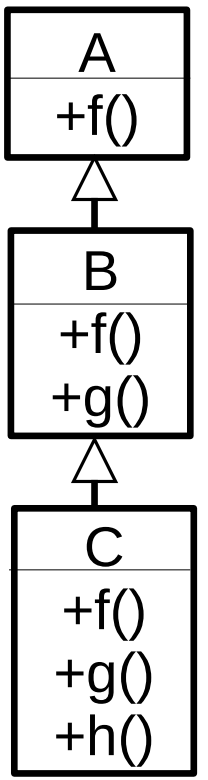
B

A

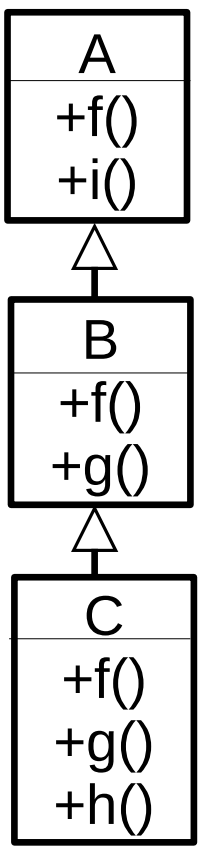
X

(d) Redéfinition et Polymorphisme

- Dans une même hiérarchie, donner la même signature à des méthodes de classes héritées
  - La méthode exécutée est celle qui est la plus proche de la **classe réelle** de l'objet appelant en remontant dans la hiérarchie.



Quiz



Quelle méthode est exécutée (A, B ou C) ?

```
A a = new C();
a.f();
a.h();
a.g();
B b = (B)a;
b.f();
b.g();
b.h();
C c = (C)b;
c.h();
c.i();
```

C

X

X

C

C

X

C

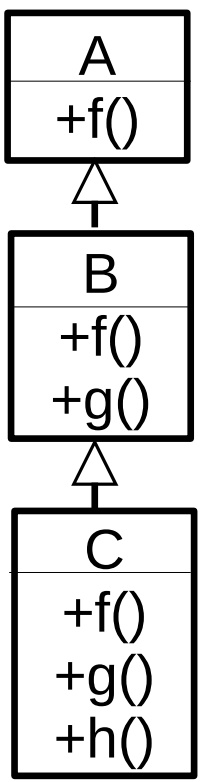
A

Polymorphisme

- **Liaison dynamique**
  - Le choix de la méthode ne peut pas être décidé à la compilation mais seulement à l'exécution
  - Quelle méthode `f()` est exécutée dans le code suivant ?

```
A getObject(int id) {
    switch(id) {
        case 1: return new B();
        case 2: return new C();
    }
}
A a = getObject(readInteger());
a.f();
```

?

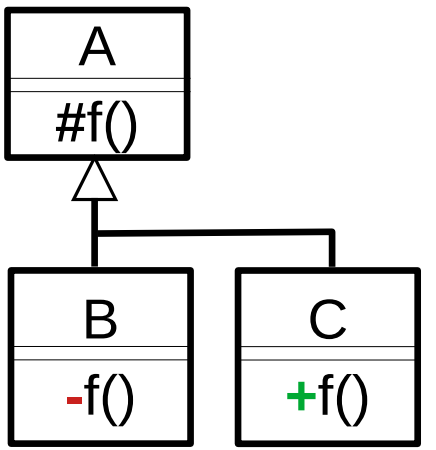


Polymorphisme

- Les classes dérivées ne peuvent pas diminuer la visibilité d'une méthode redéfinie
- Pourquoi ?

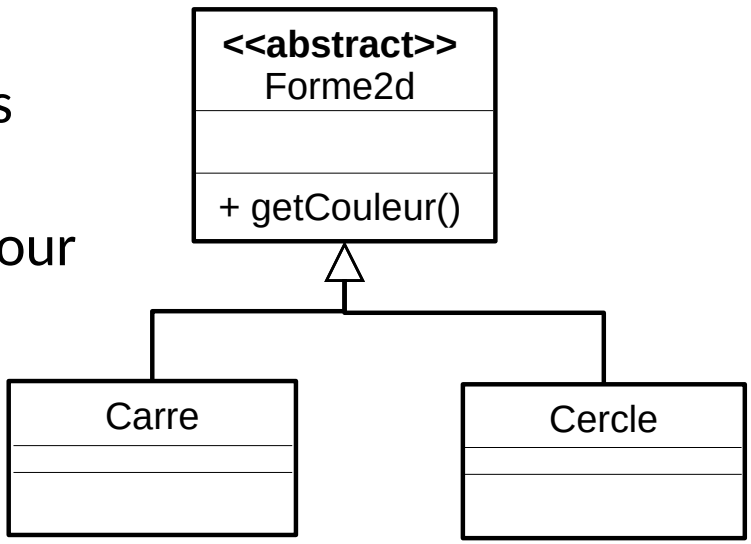
```
A getObject(int id) {
   switch(id) {
      case 1: return new B();
      case 2: return new C();
   }
}

A a = getObject(1);
a.f();
```



(e) Classe abstraite

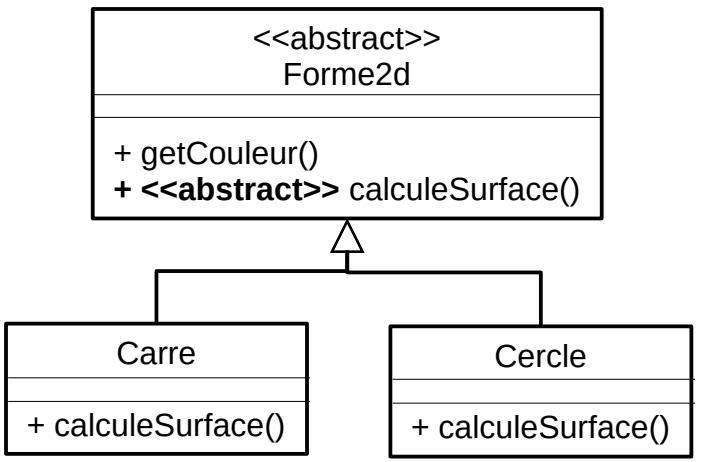
- Classe sans instance
  - Interdire la création d'instances parce qu'elles n'ont pas de sens
  - La classe abstraite n'est pas assez complète pour être instanciée



- En Java
  - Mot clé **abstract** devant la classe
  - Code Java pour l'exemple

(f) Méthode abstraite

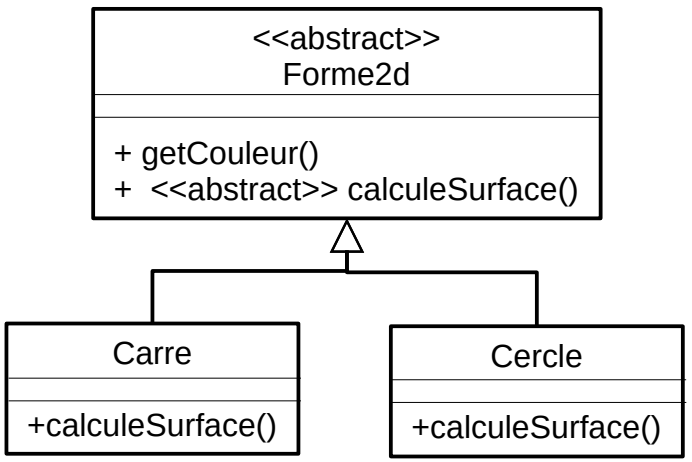
- Méthode sans code
  - Obliger les sous-classes à définir le code de la méthode
  - Profiter du polymorphisme pour exécuter la bonne méthode
  - Exemple :
    - `Forme2d f = new Carre();`
    - `double s = f.calculeSurface();`



- En Java
  - Le mot clé **abstract** devant la méthode
  - Pas de code dans le corps de la méthode
  - Code Java pour l'exemple

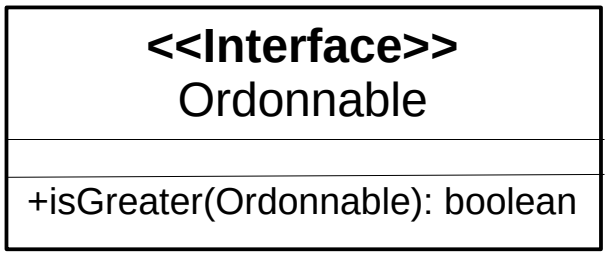
Classe et méthode abstraites

- Une classe avec une méthode abstraite est forcément abstraite
- Une classe abstraite peut ne contenir que des méthodes concrètes



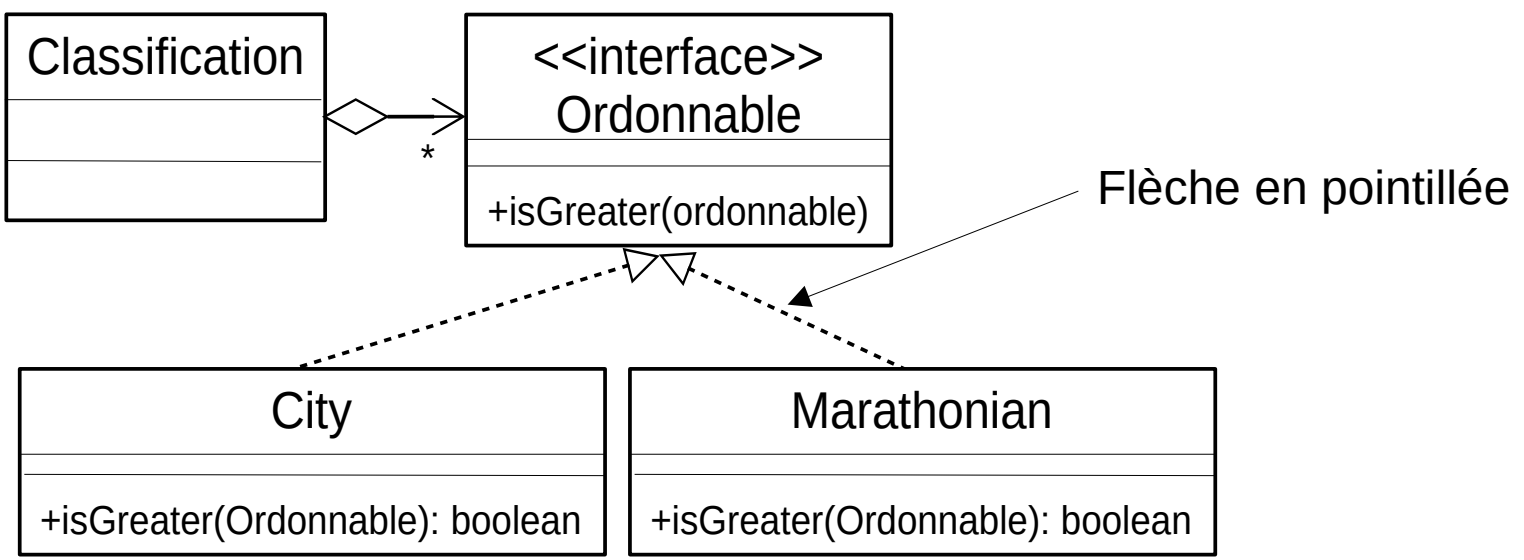
(g) Interface

- Une classe avec uniquement des méthodes **abstraites pures**
  - Ni attribut, ni association
- Intention : définir un type
  - Imposer la liste des méthodes publiques que doivent, au moins, posséder toutes les classes qui implémentent l'interface



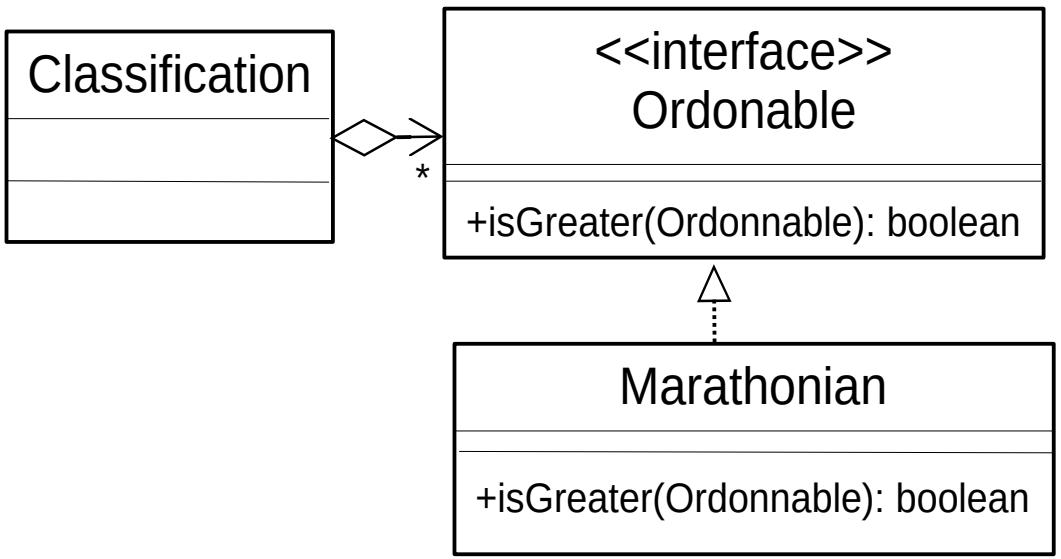
Interface

- Contrat entre deux classes :
  - Une classe fournit un service
  - Une classe utilise le service



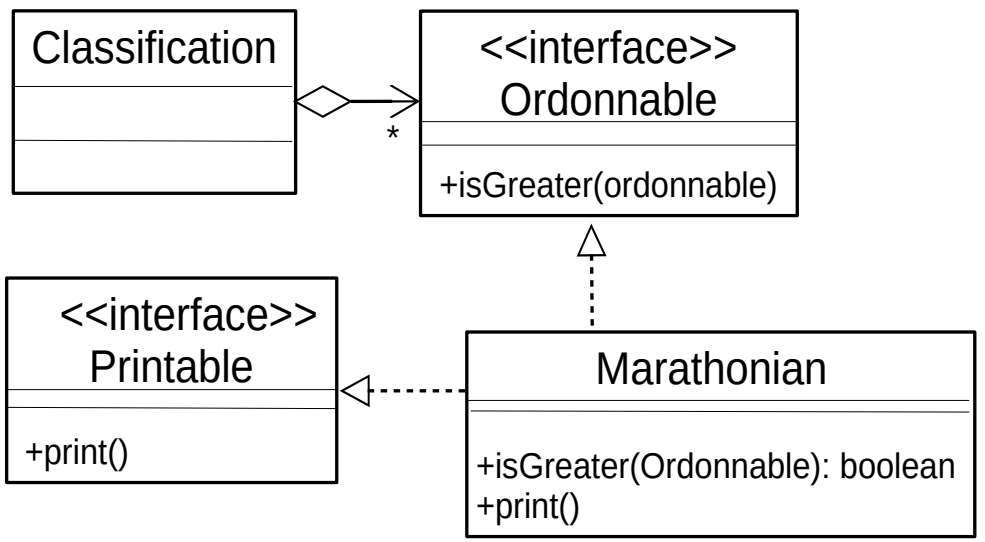
Interface en Java

- La notation Java utilise le mot clé `Interface`
  - Code Java pour l'exemple



Interface

- Une classe peut implémenter plusieurs interfaces

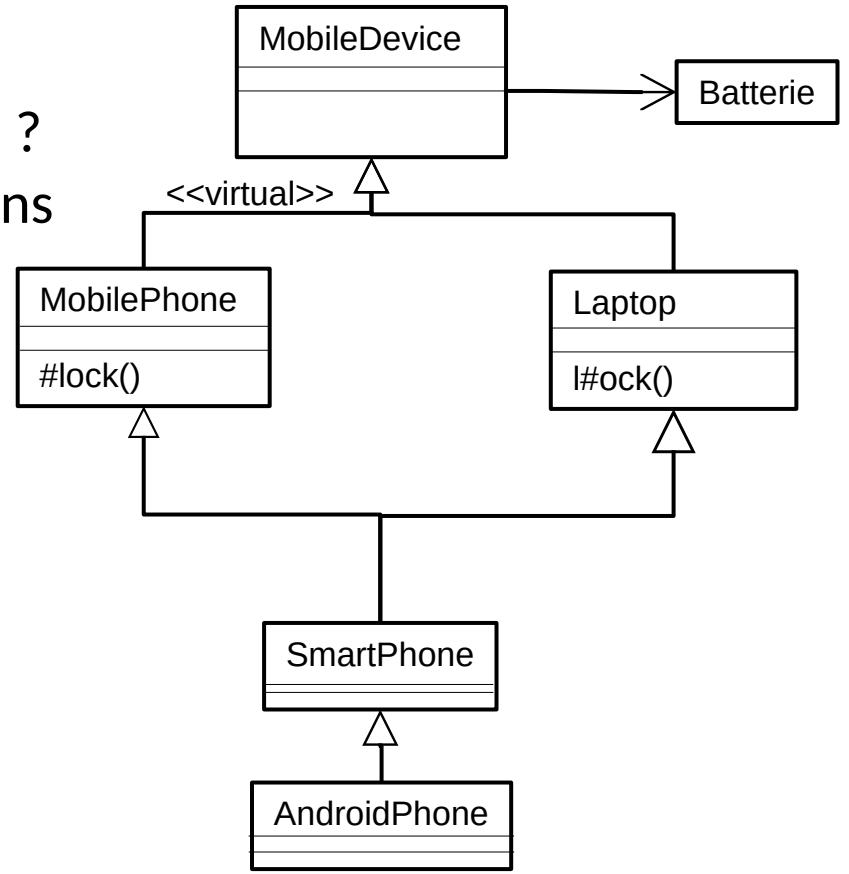


- On peut définir des références d'interface
  - `Ordonnable o = new Marathonian();`
- Mais, on ne peut pas créer d'instance d'interface
  - ~~`Ordonnable o = new Ordonnable();`~~

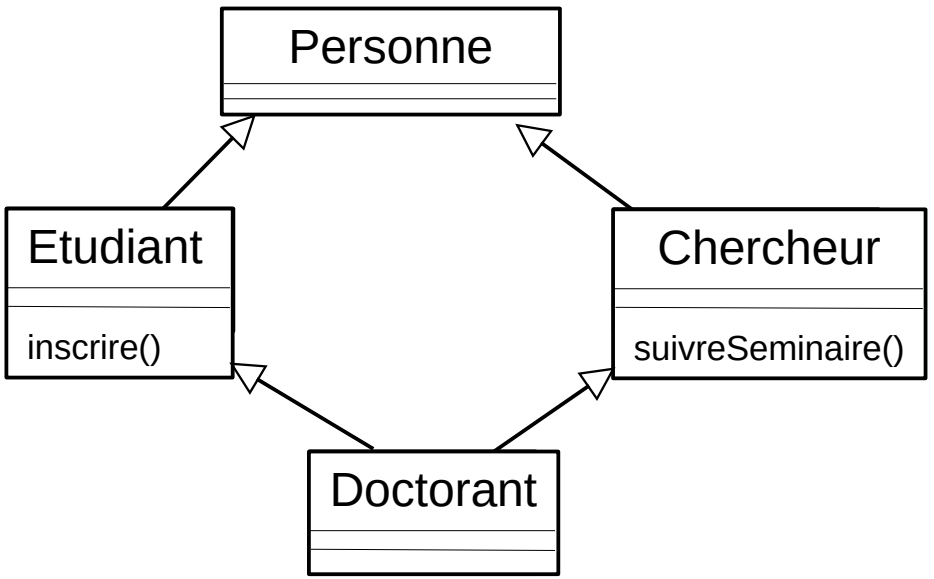
Cas de l'héritage multiple

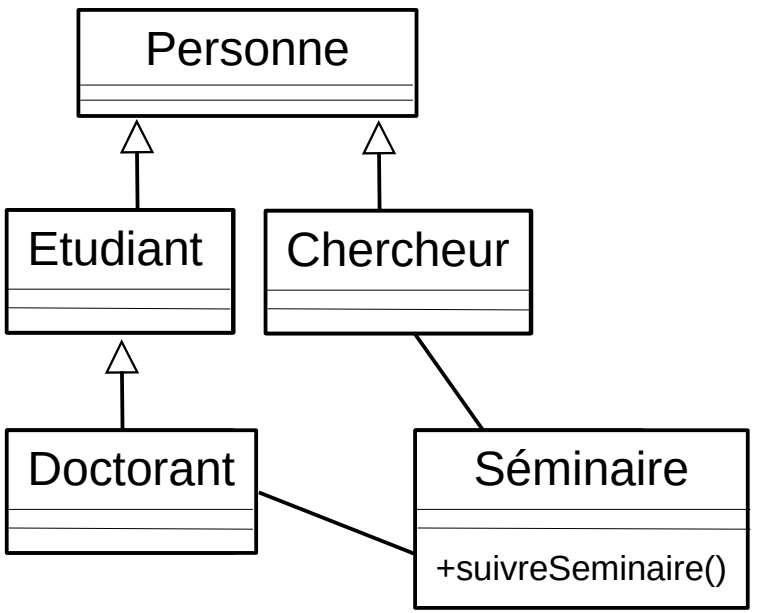
- Questions
  - Combien de batteries pour un téléphone ?
  - Quelle méthode `lock()` est appelée dans le bout de code suivant ?

```
t = new AndroidPhone()
t.lock();
```

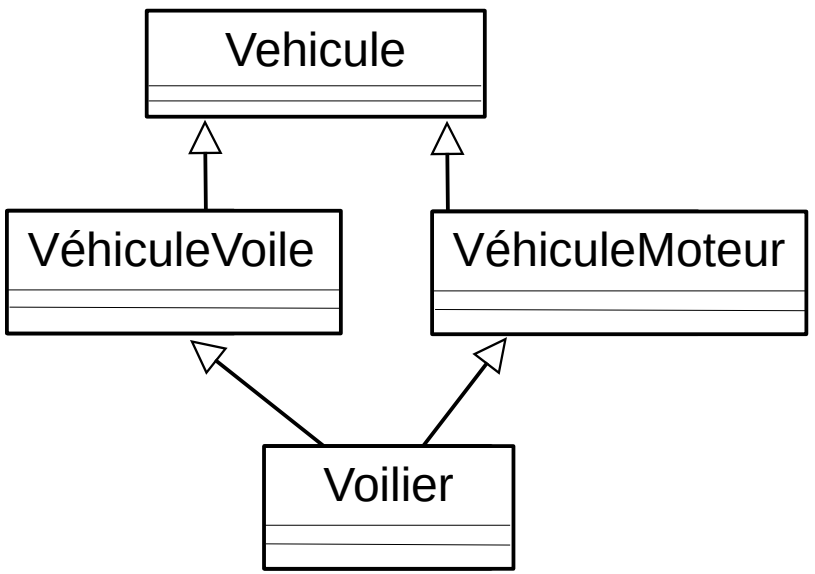


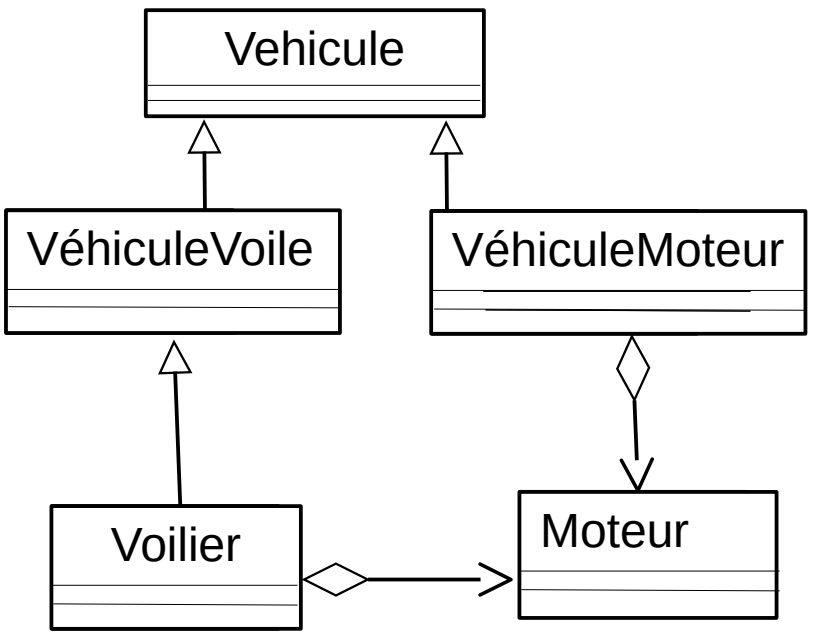
Avons nous besoin de l'héritage multiple ?





Avons nous besoin de l'héritage multiple ?





Que retenir de ce chapitre ?

- Dans l'utilisation de ces concepts, le développeur doit respecter deux principes fondamentaux :
  - Restreindre le plus possible la visibilité des membres pour respecter le principe d'encapsulation et la sécurité.
    - Ainsi les attributs sont TOUJOURS privés.
  - Lever les choix avant la programmation.