



02

Chapitre

Un paradigme : 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

Plan du chapitre

2

1
Le paradigme
objet

2
Les objets

3
Les classes

4
Relations
entre
classes

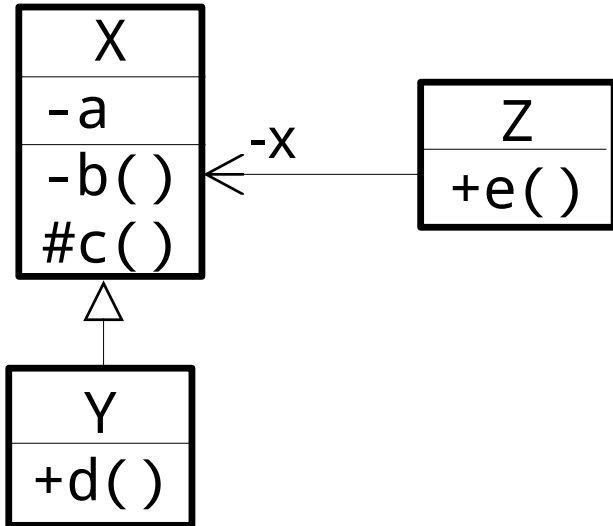
5
Héritage
et
polymorphisme

(a) Visibilité

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

	Notation UML	Classe elle-même	Sous-classe	Autres classes
public	+	+	+	+
protected	#	+	+	-
private	-	+	-	-

Quiz



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

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

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

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

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

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

 ✓

Visibilité

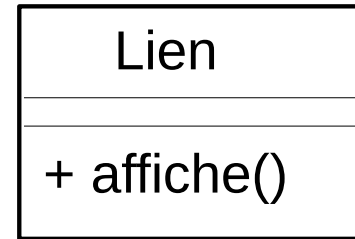
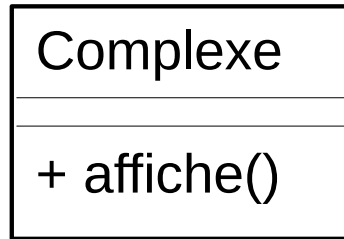
- Règle : restreindre le plus possible la visibilité
- Intention
 - Sécuriser la représentation interne des classes
 - Profiter du compilateur pour garantir l'encapsulation

Règle intangible

**Les attributs et associations
sont TOUJOURS privés**

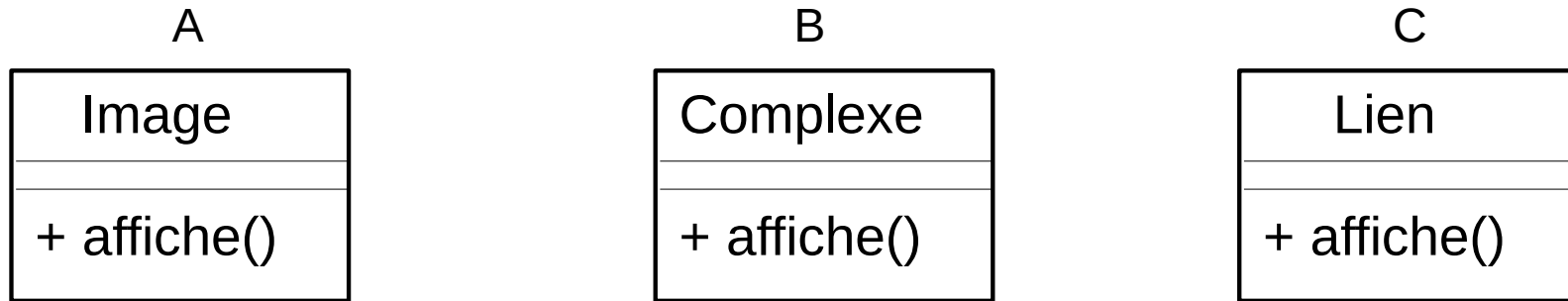
(b) Portée des noms

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



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

Quiz



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

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

(c) Surcharge

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

Complexe
+ additionne(val: int)
+ additionne(val: float)

- Le choix peut être fait à la compilation : **liaison statique**

Quiz

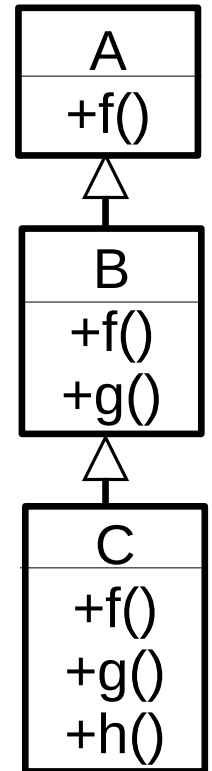
Complexe	
A	+ additionne(val: int)
B	+ additionne(val: float)

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

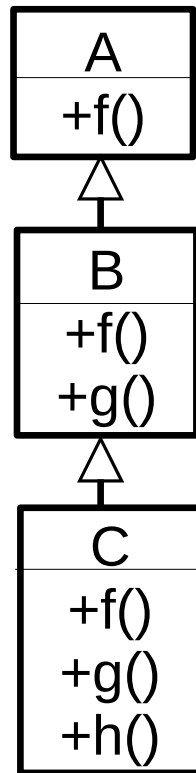
```
Complexe c = new complexe();  
c.additionne(5);  
c.additionne(5f);  
c.additionne(5d);
```

(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 `a = new C();`

a. `f();`

a. `g();`

a. `h();`

B `b = (B)a;`

b. `f();`

b. `g();`

b. `h();`

C `c = (C)b;`

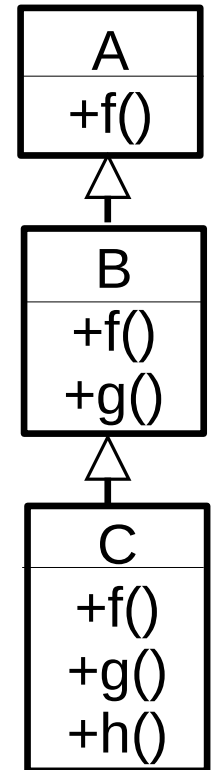
c. `h();`

Polymorphisme

■ Liaison dynamique

- Le choix de la méthode ne peut pas être décidé à la compilation mais seulement à l'exécution
- Exemple :

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

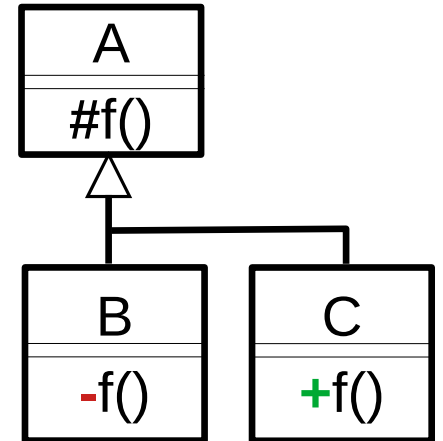


Polymorphisme

- 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();
```



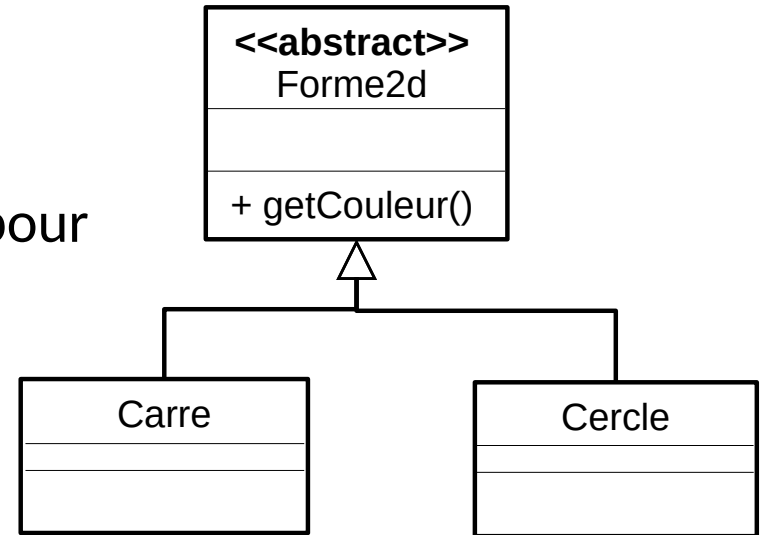
(e) Classe abstraite

- Classe sans instance

- Interdire la création d'instances puisqu'elles n'ont pas de sens
- La classe abstraite n'est pas assez complète pour être instanciée

- En Java

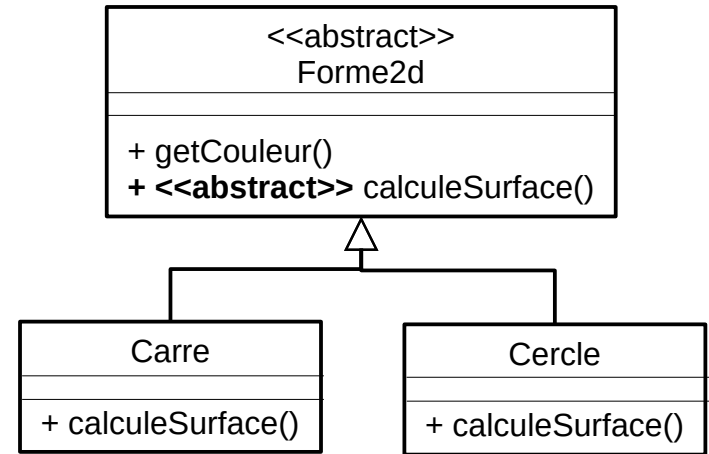
- Mot clé **abstract** devant la classe



(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();`

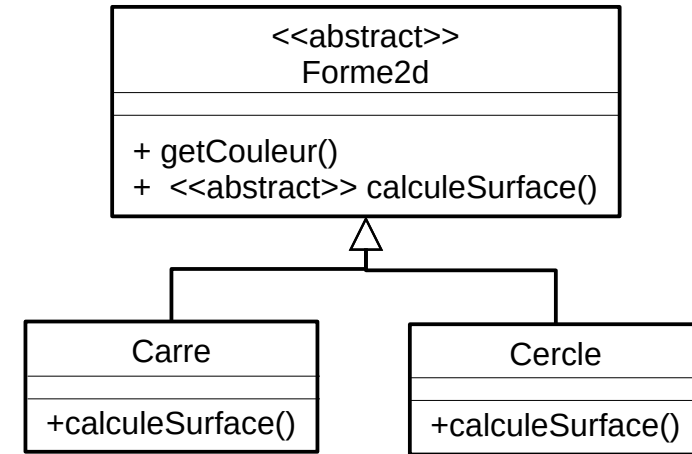


■ En Java

- Le mot clé **abstract** devant la méthode
- Pas de code dans le corps de la méthode

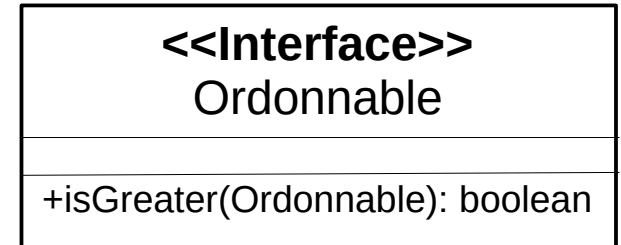
Classe et méthode abstraites

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



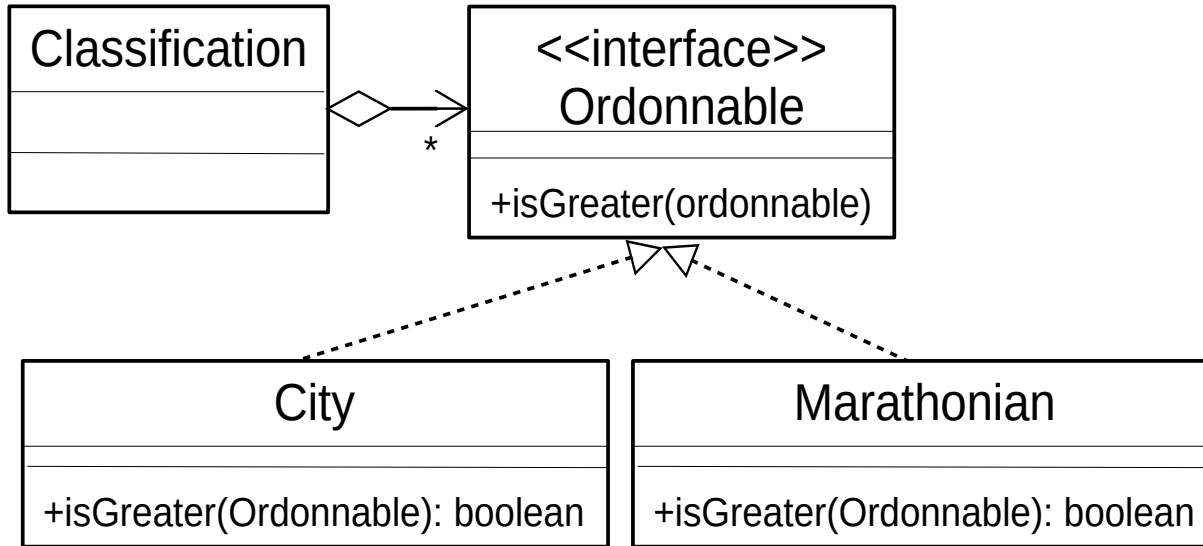
(g) Interface

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



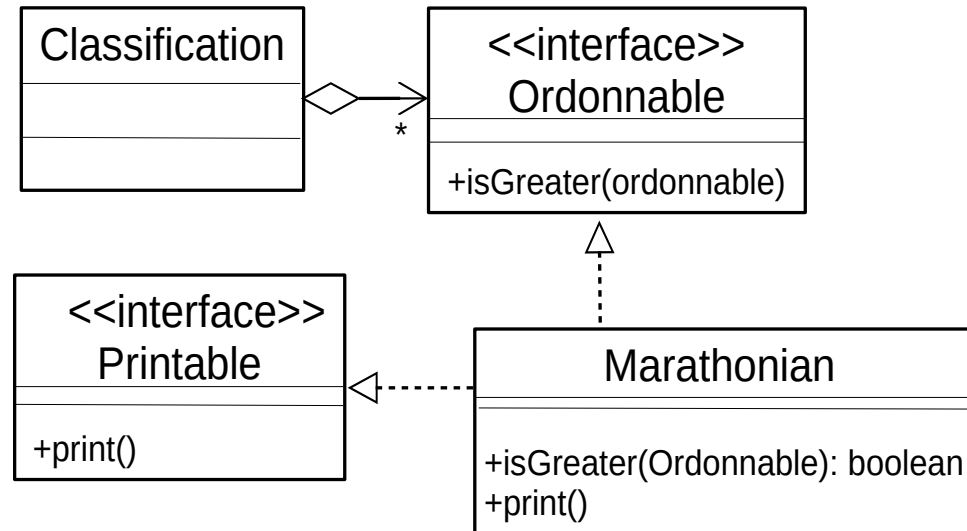
Interface

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



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();`~~

Interface en Java

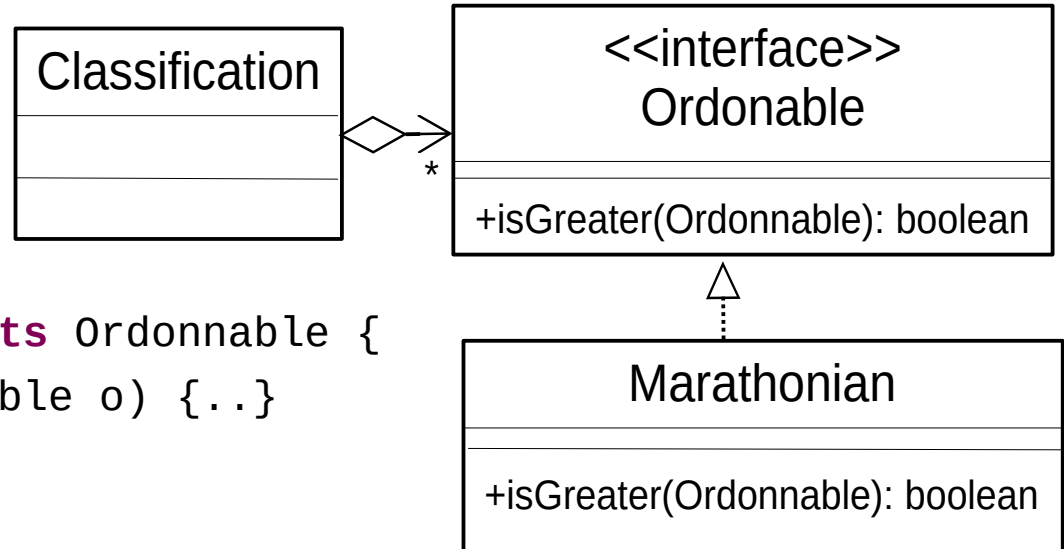
20

■ Notation Java :

```
public interface Ordonnable {  
    public boolean isGreater();  
}
```

```
public class Marathonian implements Ordonnable {  
    public boolean isGreater(Ordonnable o) {...}  
    // méthodes spécifiques  
}
```

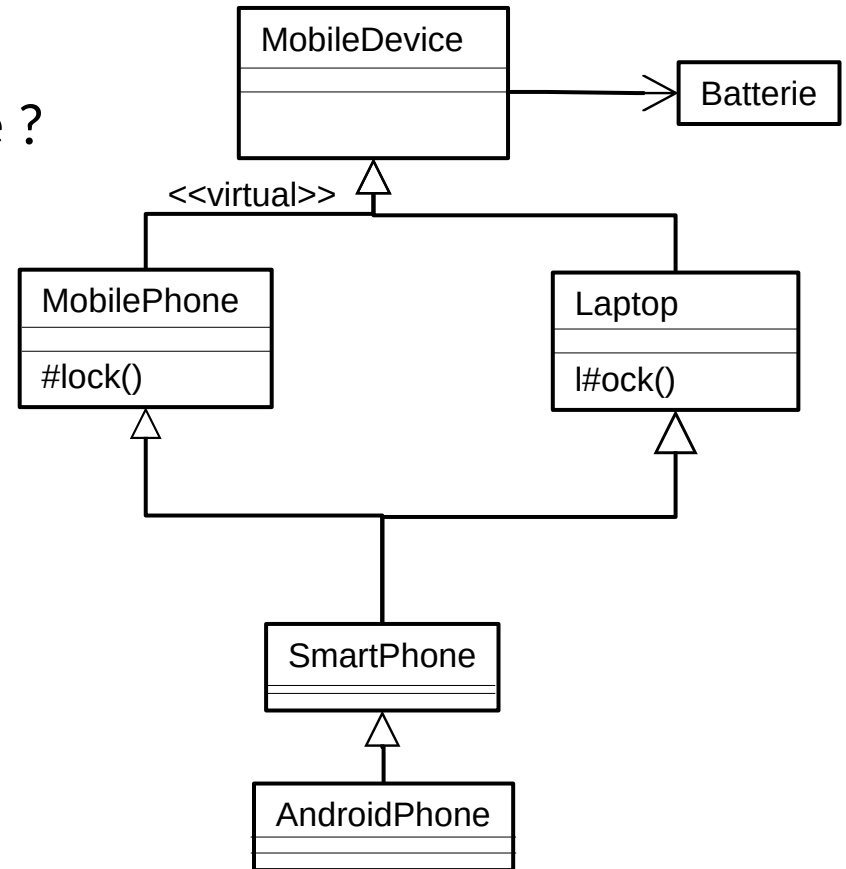
```
public class Classification {  
    ArrayList<Ordonnable> coureurs;  
    ...  
    coureurs.get(1).isGreater(coureurs.get(2));  
}
```



Cas de l'héritage multiple

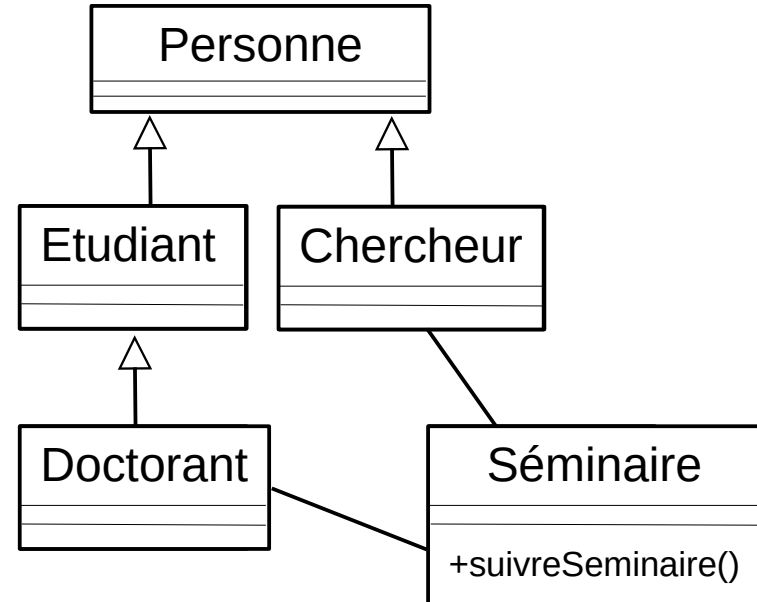
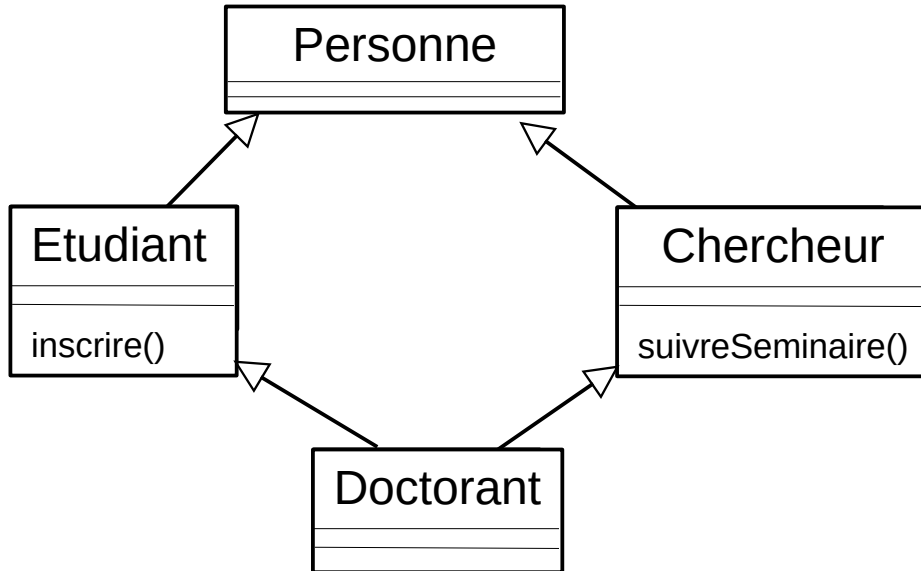
■ Questions

- Combien de batteries pour un téléphone ?
- Quelle méthode est appelée ?
 - ▶ `t = new AndroidPhone()`
 - ▶ `t.lock();`

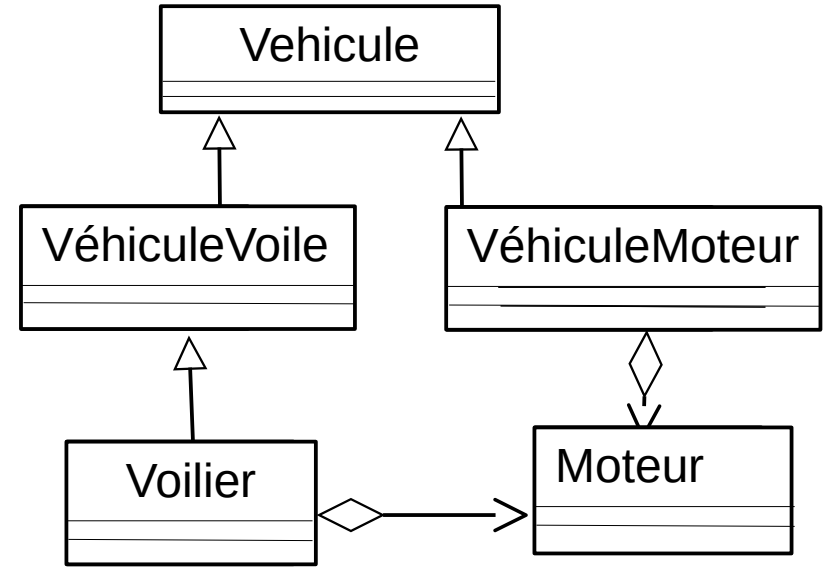
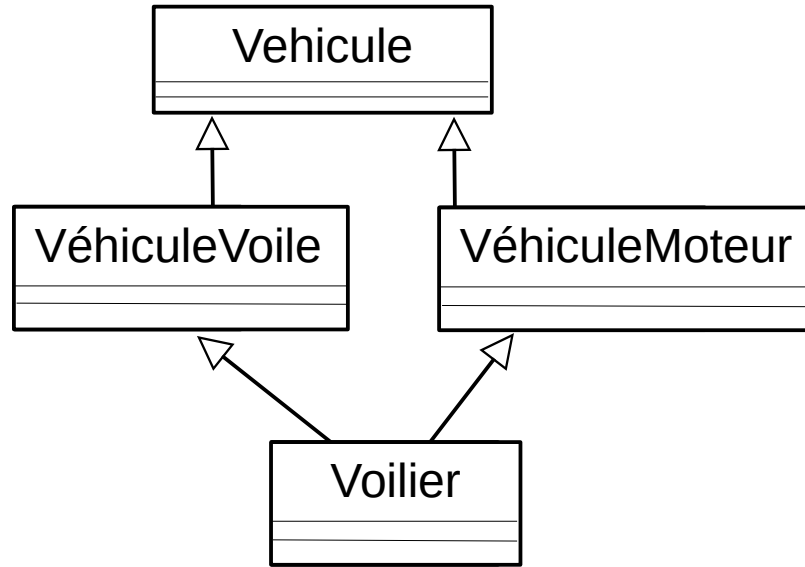


Avons nous besoin de l'héritage multiple ?

22



Avons nous besoin de l'héritage multiple ?



Que retenir de ce chapitre ?

- Le paradigme objet définit plusieurs concepts importants :
 - Classe et objet
 - Encapsulation
 - Relations
 - ▶ Association
 - Standard, Agrégation, Composition
 - ▶ Héritage et polymorphisme
 - ▶ Interface et type

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.