



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

Plan du chapitre

49

1
Le paradigme
objet

2
Les objets et le
principe
d'encapsulation)

3
Les classes

4
Associations
entre
classes

5
Héritage
et
polymorphisme

(a) Visibilité

50

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

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

Quiz sur la visibilité

51

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

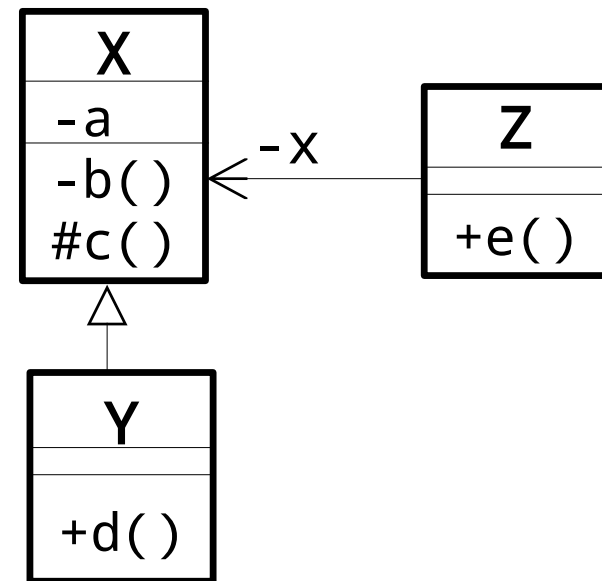
5. `void e() { x.b(); }` X

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();  
}
```



Objectif de la visibilité

52

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

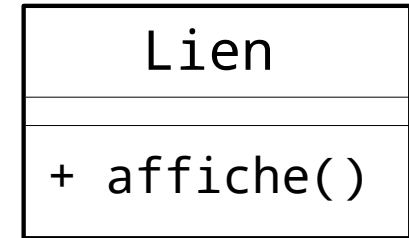
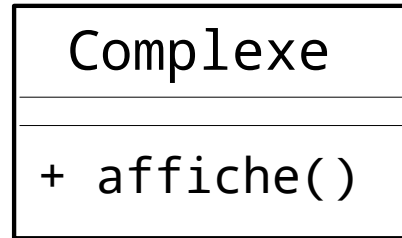
Règle intangible

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

(b) Portée des noms

53

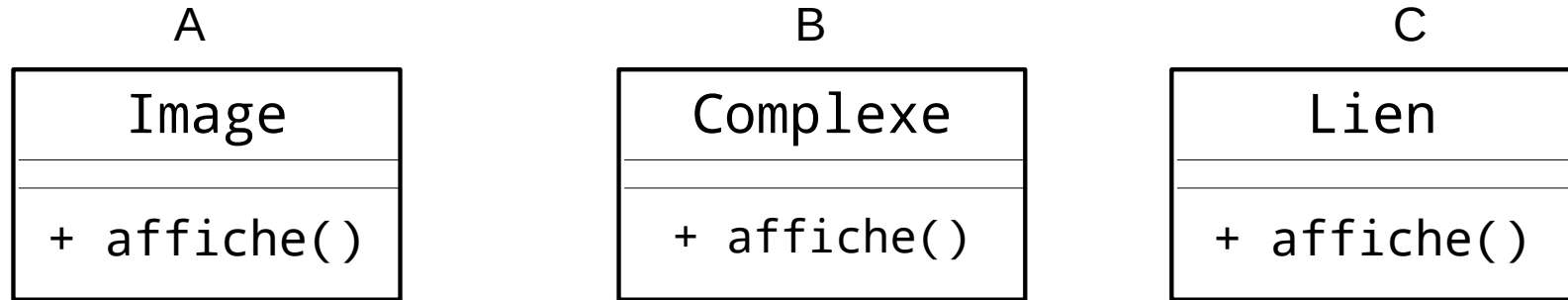
- Il est possible de donner un même nom à des méthodes de classes différentes



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

Quiz

54



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

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

A

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

B

(c) Surcharge

55

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

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

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

Quiz

56

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

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

```
Complexe c = new complexe();
```

```
c.additionne(5f);      B
```

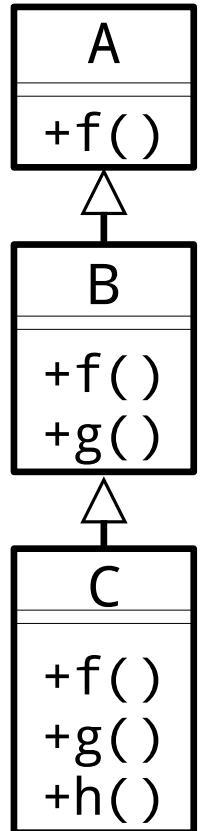
```
c.additionne(5);      A
```

```
c.additionne(5.2d);   X
```

(d) Redéfinition et Polymorphisme

57

- 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

58

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

A **a = new C();**

a.f(); **C**

a.h(); **X**

a.g(); **X**

B **b = (B)a;**

b.f(); **C**

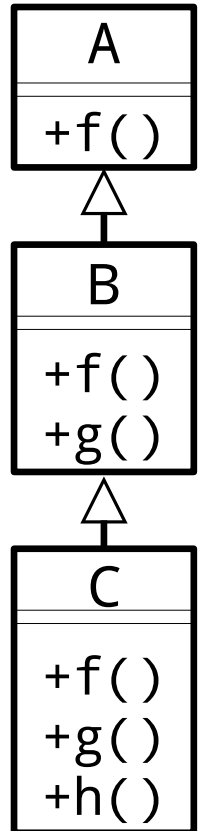
b.g(); **C**

b.h(); **X**

C **c = (C)b;**

c.h(); **C**

c.i(); **A**



Polymorphisme

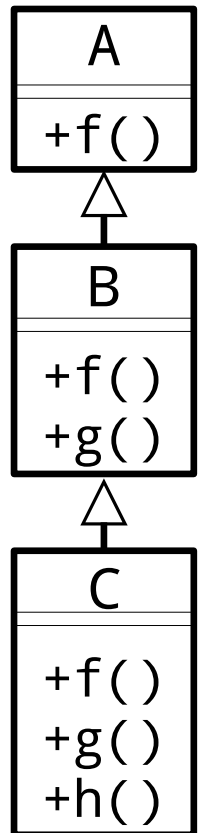
59

■ 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(); ?
```



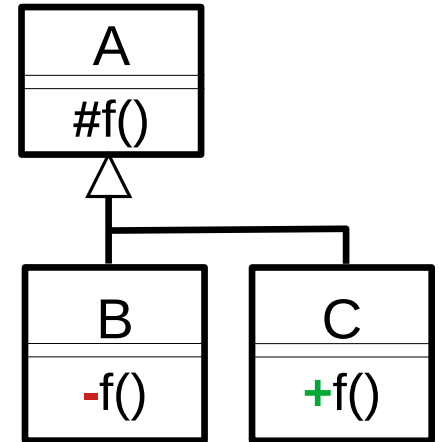
Polymorphisme

60

- 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();
```

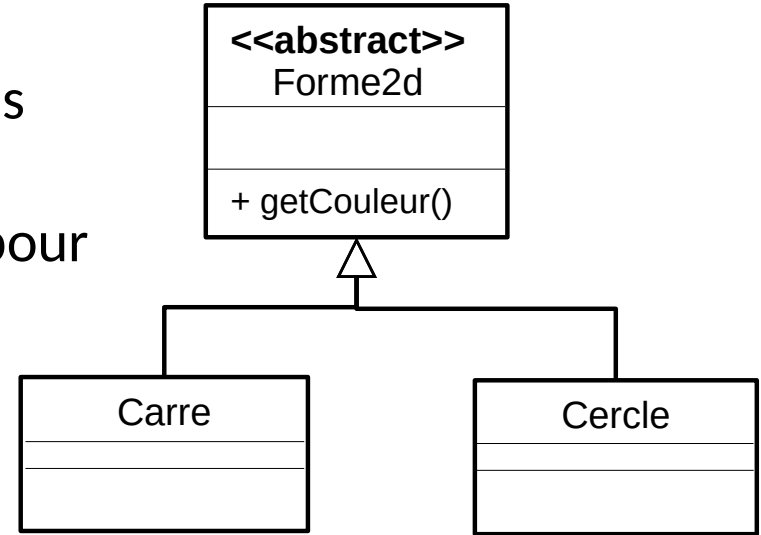


(e) Classe abstraite

61

■ 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

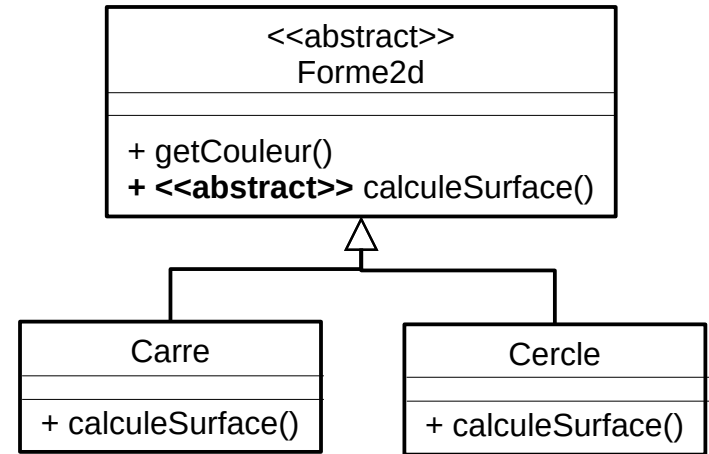
- Mot clé **abstract** devant la classe
- Code Java pour l'exemple

(f) Méthode abstraite

62

■ 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();`



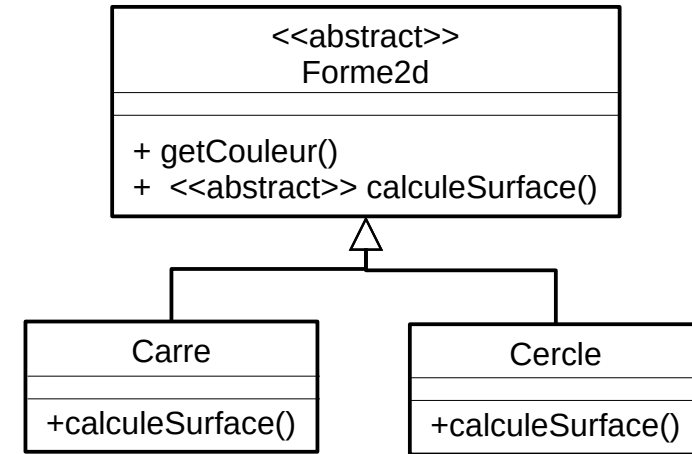
■ En Java

- 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

63

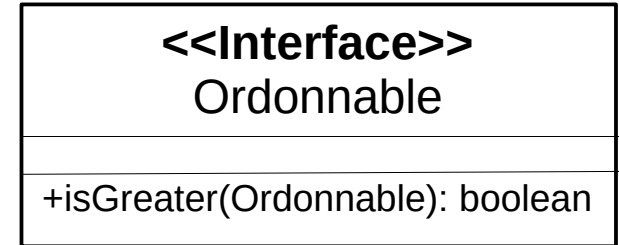
- Une classe avec une méthode abstraite est forcément abstraite
- Une classe abstraite peut ne contenir que des méthodes concrètes



(g) Interface

64

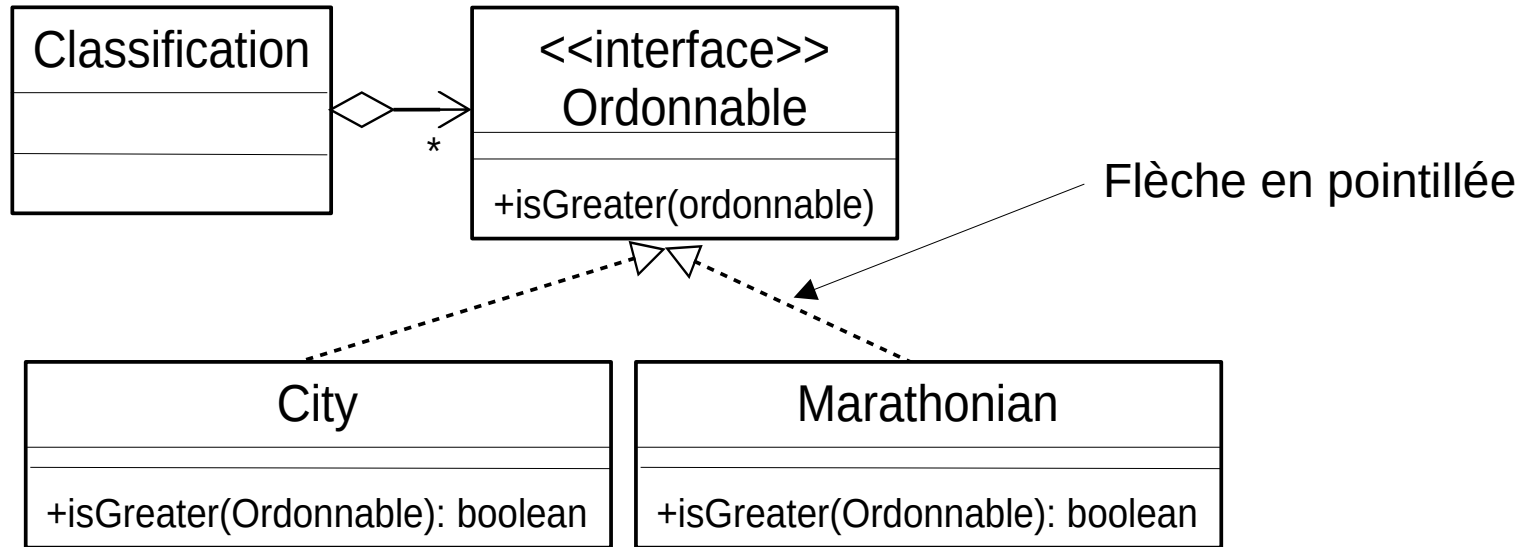
- Une classe avec uniquement des méthodes **abstraites pures**
 - Ni attribut, ni association
- Intention : définir un type
 - Imposer la liste des méthodes publiques que doivent, au moins, posséder toutes les classes qui implémentent l'interface



Interface

65

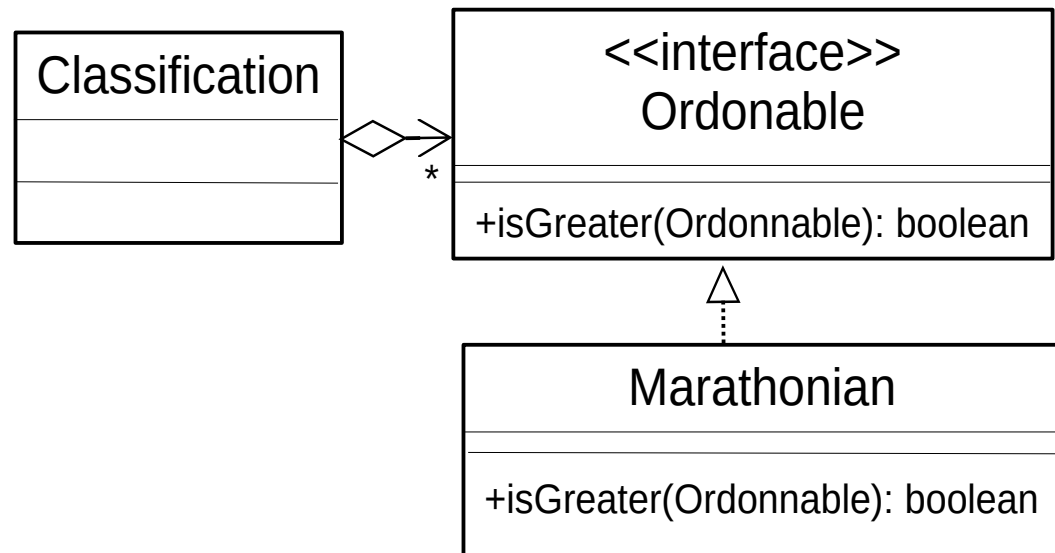
- Contrat entre deux classes :
 - Une classe fournit un service
 - Une classe utilise le service



Interface en Java

66

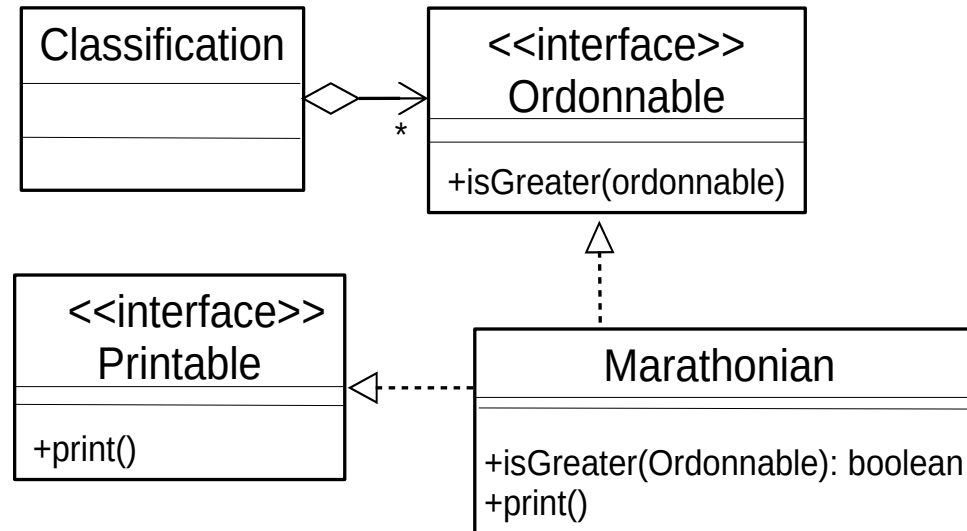
- La notation Java utilise le mot clé Interface
 - Code Java pour l'exemple



Interface

67

- 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();`~~

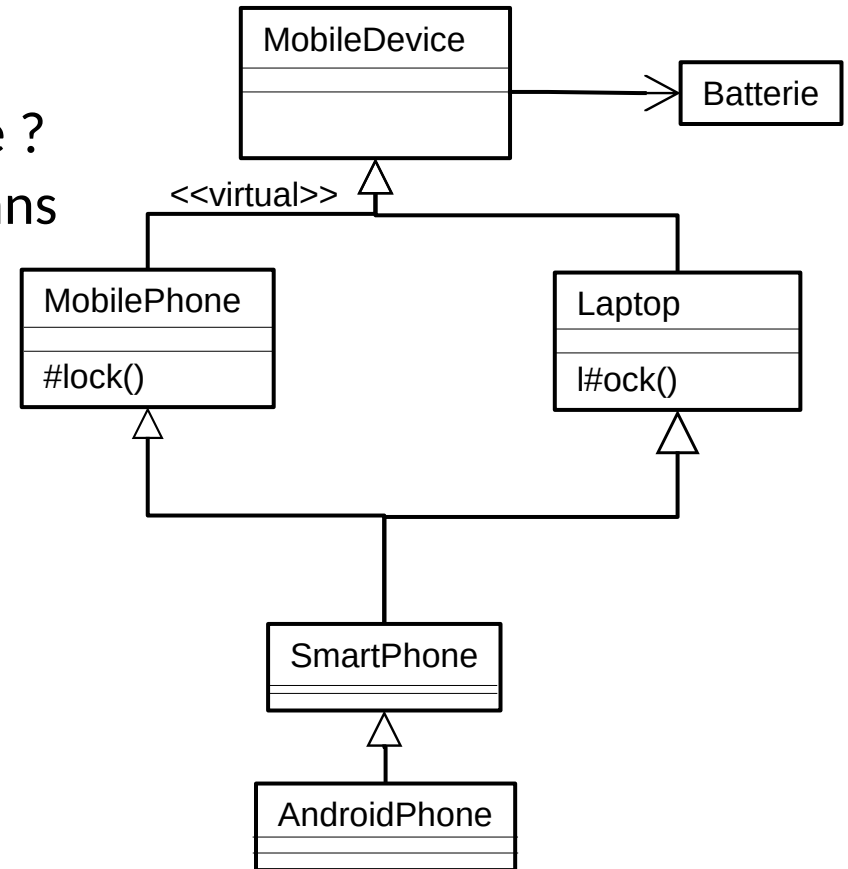
Cas de l'héritage multiple

68

■ Questions

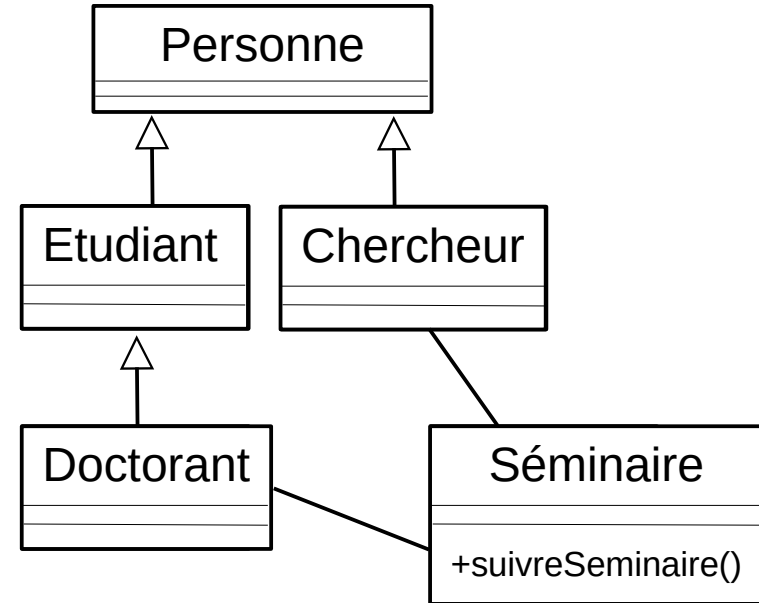
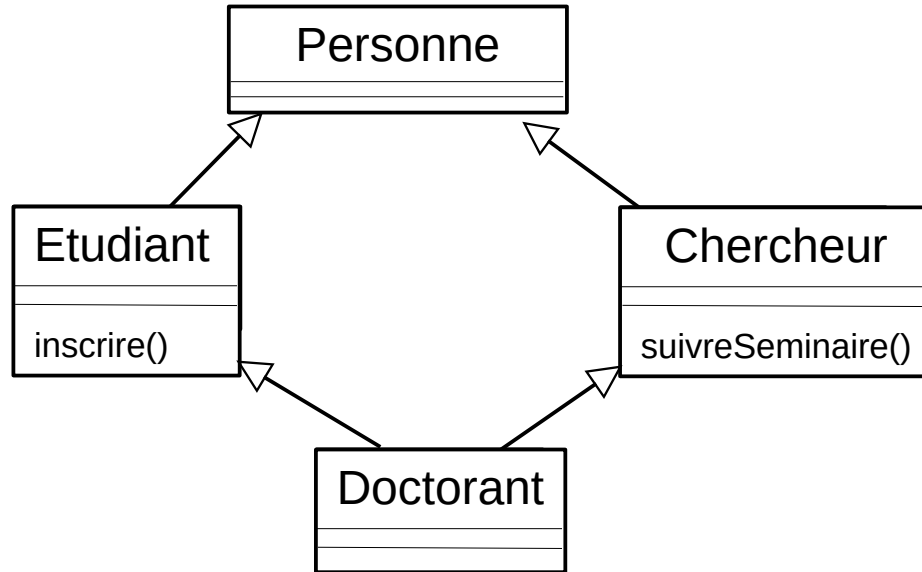
- Combien de batteries pour un téléphone ?
- Quelle méthode `lock()` est appelée dans le bout de code suivant ?

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



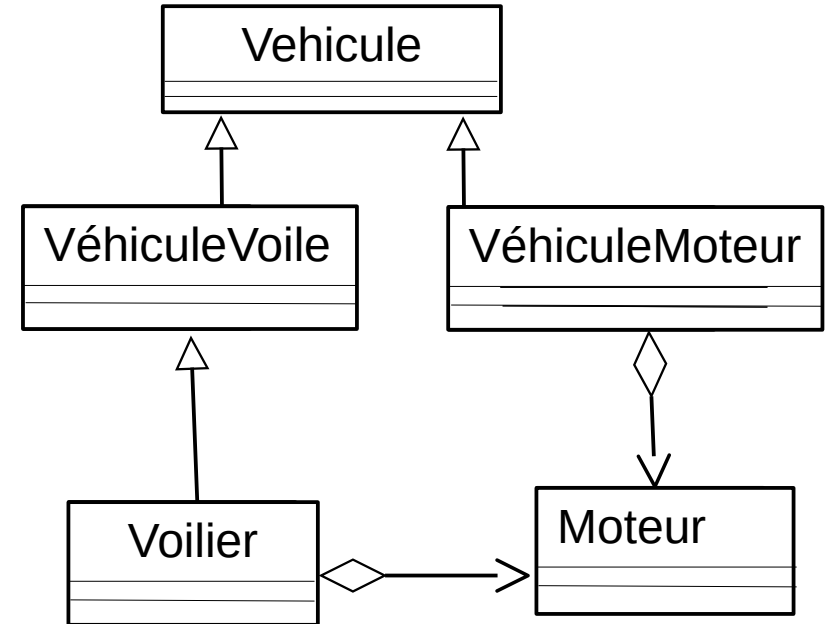
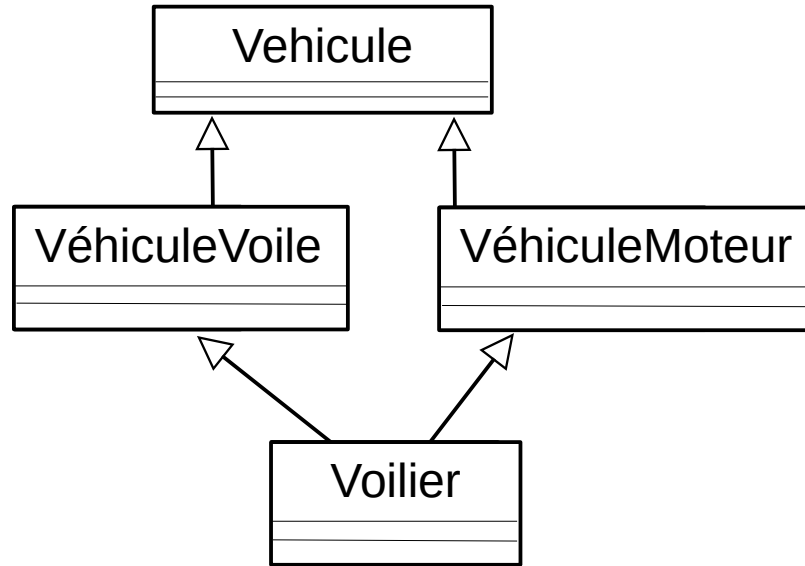
Avons nous besoin de l'héritage multiple ?

69



Avons nous besoin de l'héritage multiple ?

70



Que retenir de ce chapitre ?

71

- 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.