

Langage de modélisation objet unifié

Introduction à l'Orienté Objet



Pr R.EL OUAHBI

Plan

- **Objet**
- **Classes d'objets**
- **Encapsulation**
- **Héritage**
- **Polymorphisme**
- **Composition**
- **Classe abstraite**
- **Interface**

L 'orientation Objet

C 'est une technique de modélisation de
système, associée à un langage de
programmation

Le système

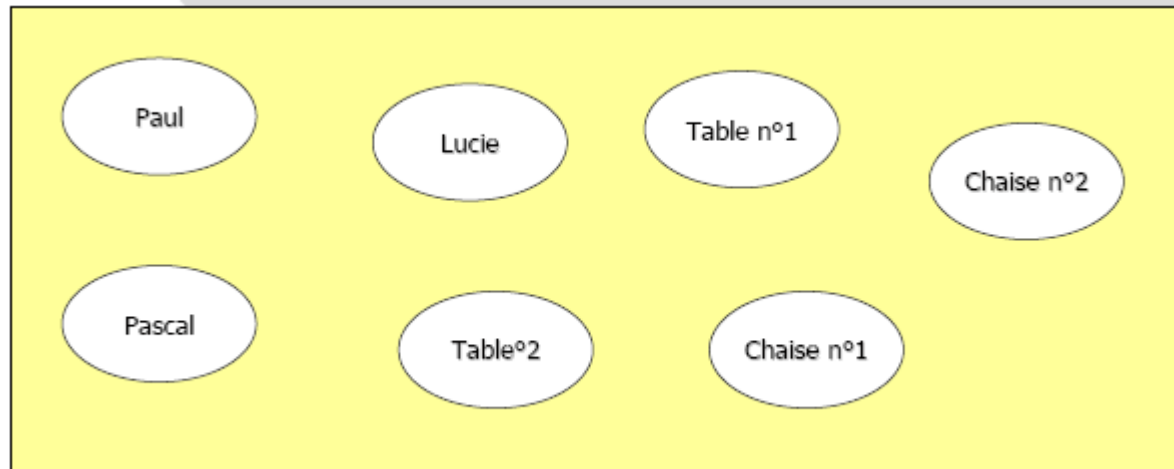
Un système est composé d 'objets en relation

Exemples :

- Une compagnie aérienne (avions, pilotes ...)
- Un amphithéâtre (élèves, professeur, tables ...)

Les objets

Les objets du système amphithéâtre



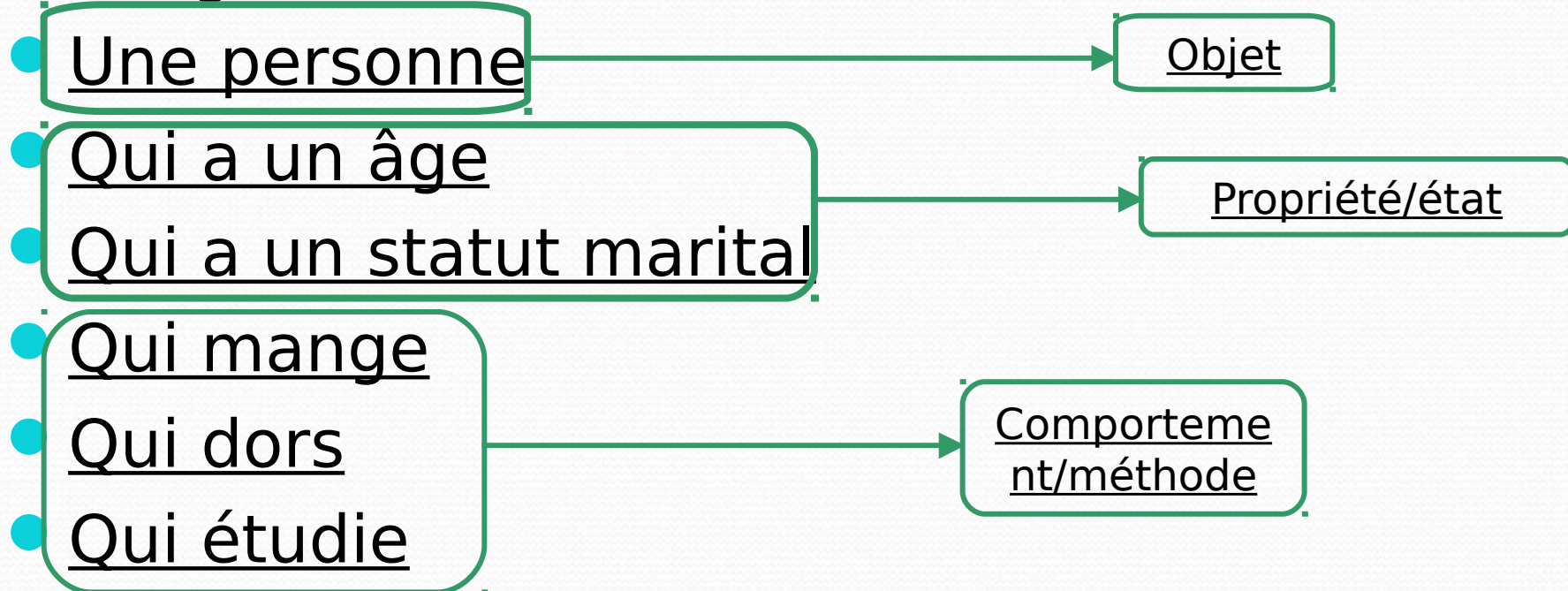
Les objets

Un **Objet** peut correspondre à :

- Un **objet concret du monde réel, ayant une réalité physique** (une personne, une voiture, un outil, un système mécanique, . . .) ;
- Un **concept abstrait** (un compte bancaire, une contrainte mécanique, une vitesse, . . .) ;
- Une **activité produisant des effets observables**
(un calcul numérique, un pilote d'impression, . . .) ;

Les objets

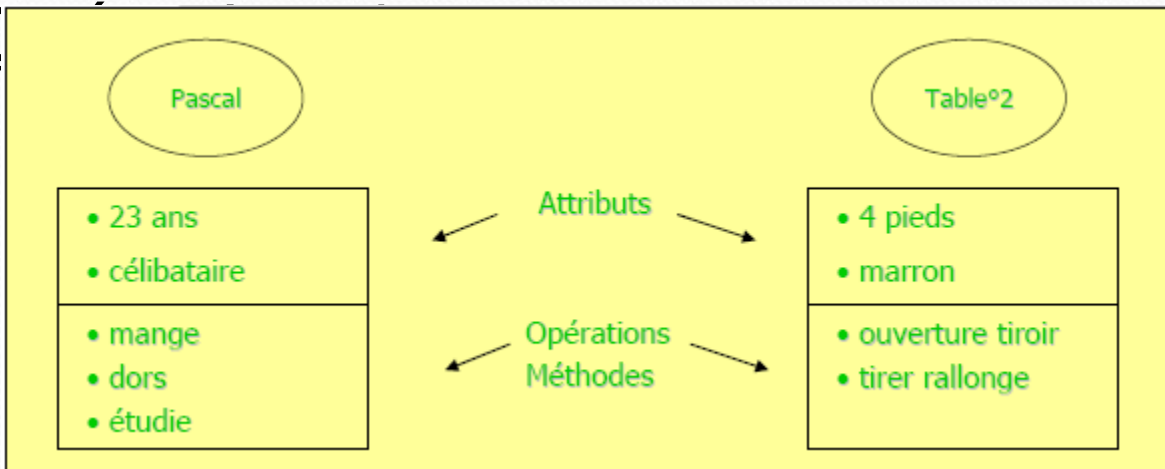
Analogie avec le monde réel



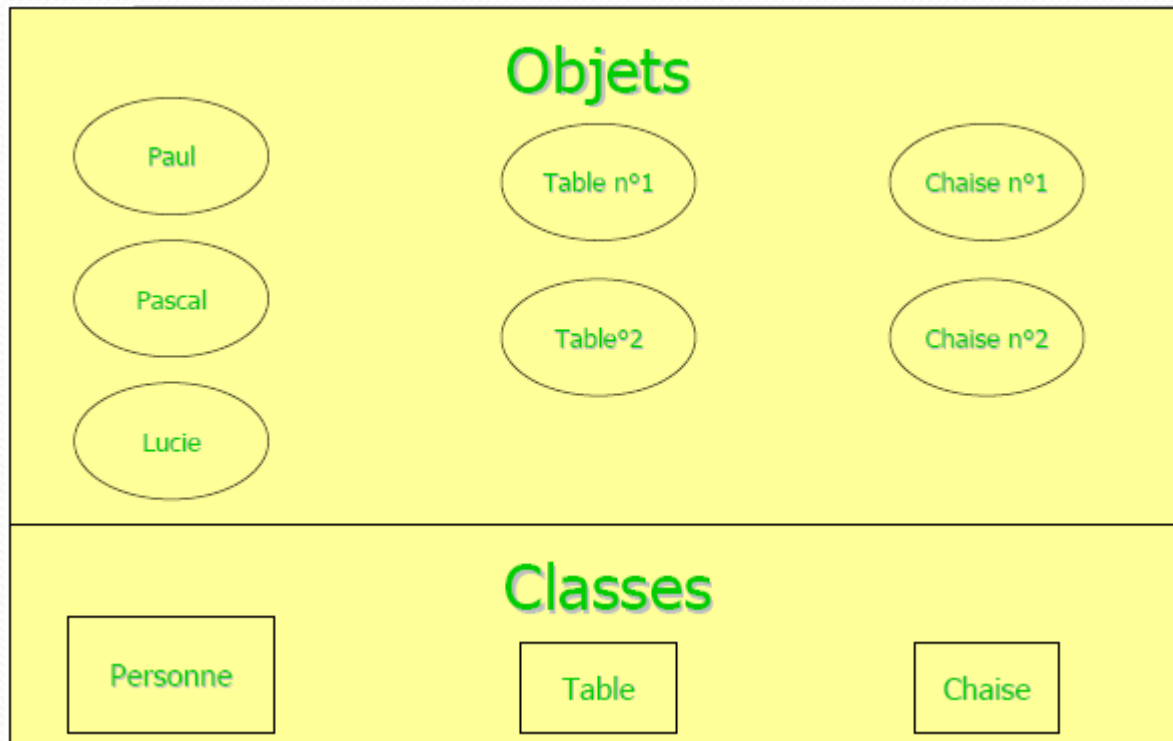
Les objets

Objet = attributs + Méthodes

(C



Les classes



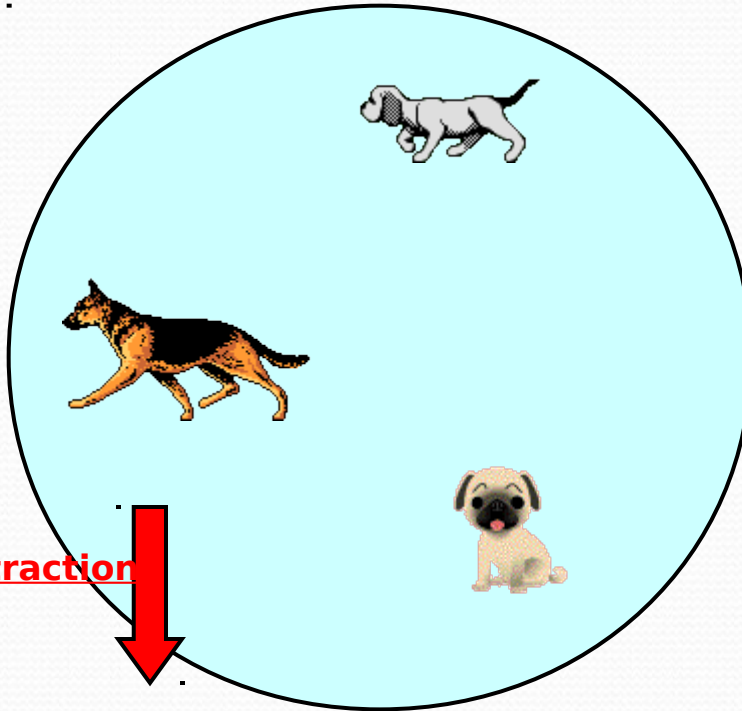
Les classes

- Représentation abstraite d'une catégorie d'objets.
- Elles regroupent les objets ayant des caractéristiques communes (informations/comportements).

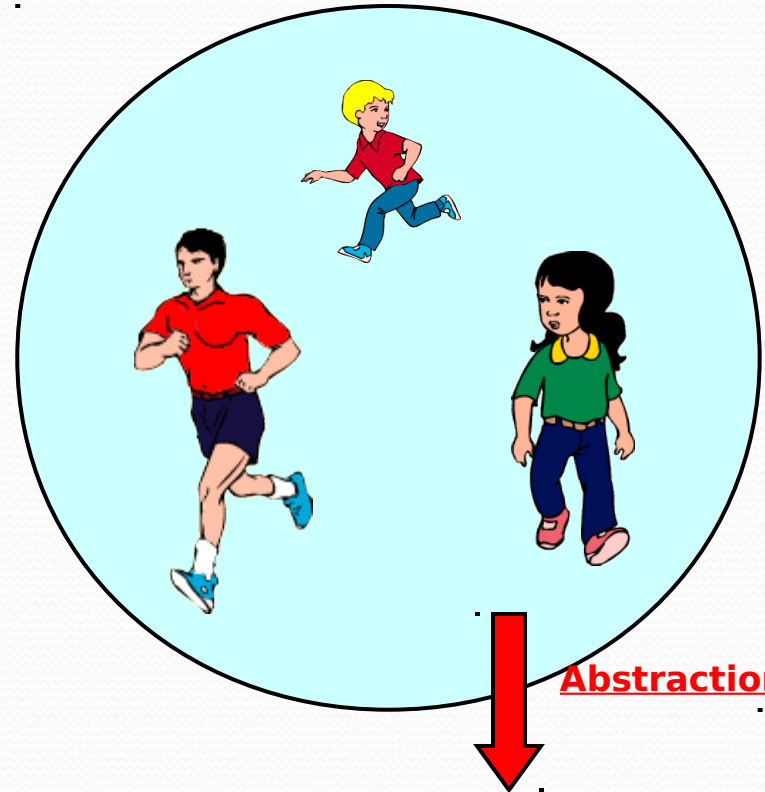
Abstraction

Elle consiste à retenir uniquement
les propriétés pertinentes d'un
objet pour un problème précis

Les classes



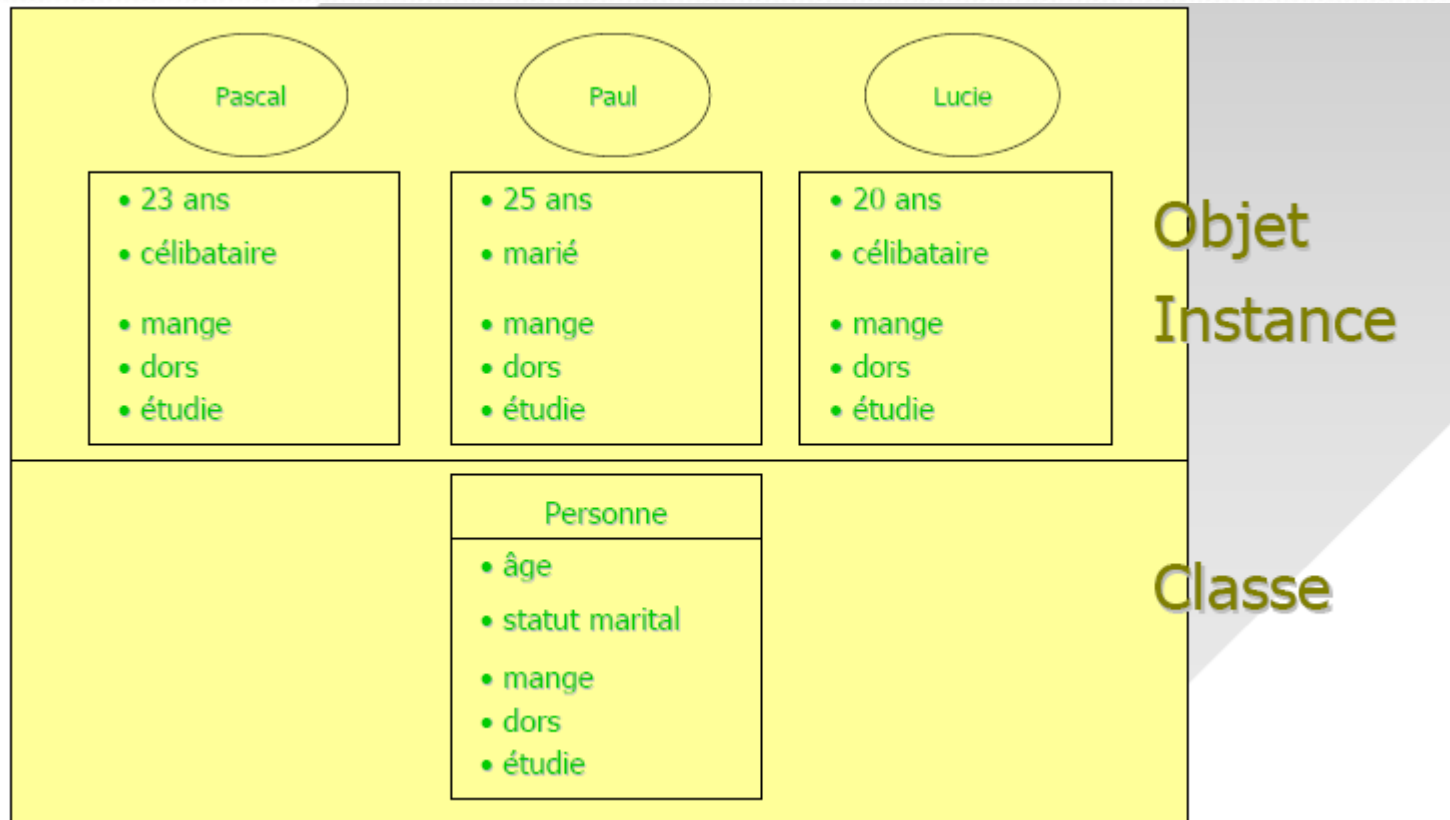
<u>Chien</u>
<u>age</u> <u>race</u>
<u>courir()</u> <u>aboyer()</u>



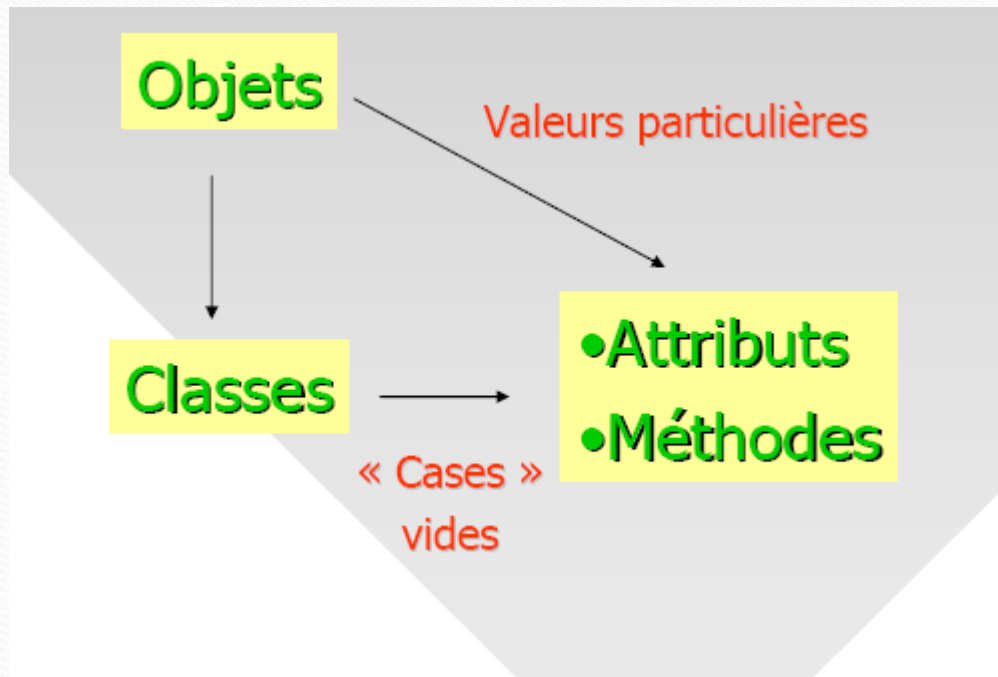
<u>Personne</u>
<u>age</u> <u>nationalité</u>
<u>se promener()</u> <u>crier()</u>

Les classes et les instances

Chaque instance est forcément associée à une classe



Résumé : Entités de l'approche orientée objet



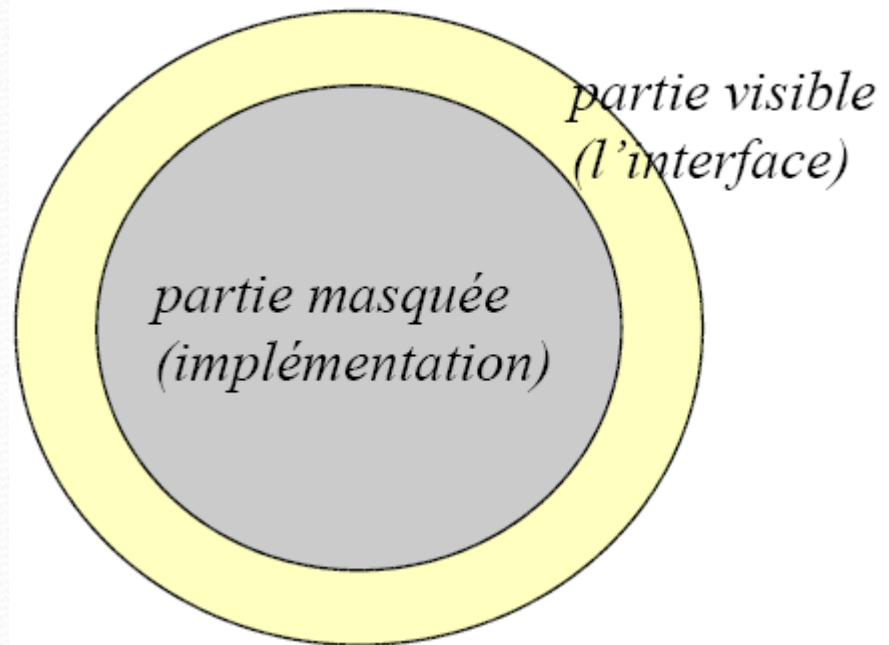
Notion d'encapsulation

- Masquer les détails d'implémentation d'un objet, en définissant une **interface**.
- **L'interface : vue externe d'un objet, elle définit les services accessibles** (offerts) aux utilisateurs de l'objet.

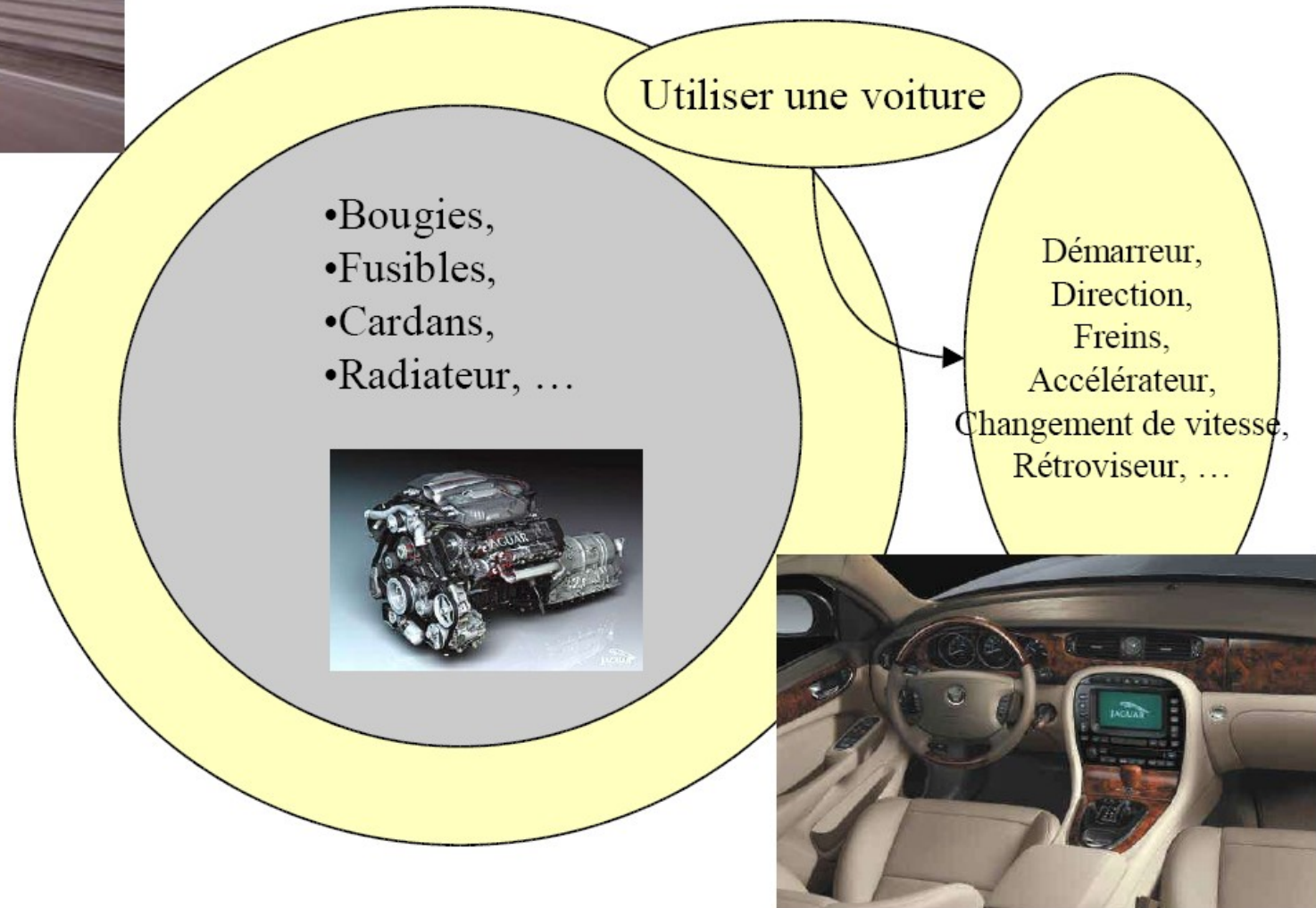
- Garantit l'intégrité et la sécurité des données
- Facile à maintenir (changements dans l'implémentation sans modifier l'interface)
- Concept de «boîte noire»

Encapsulation

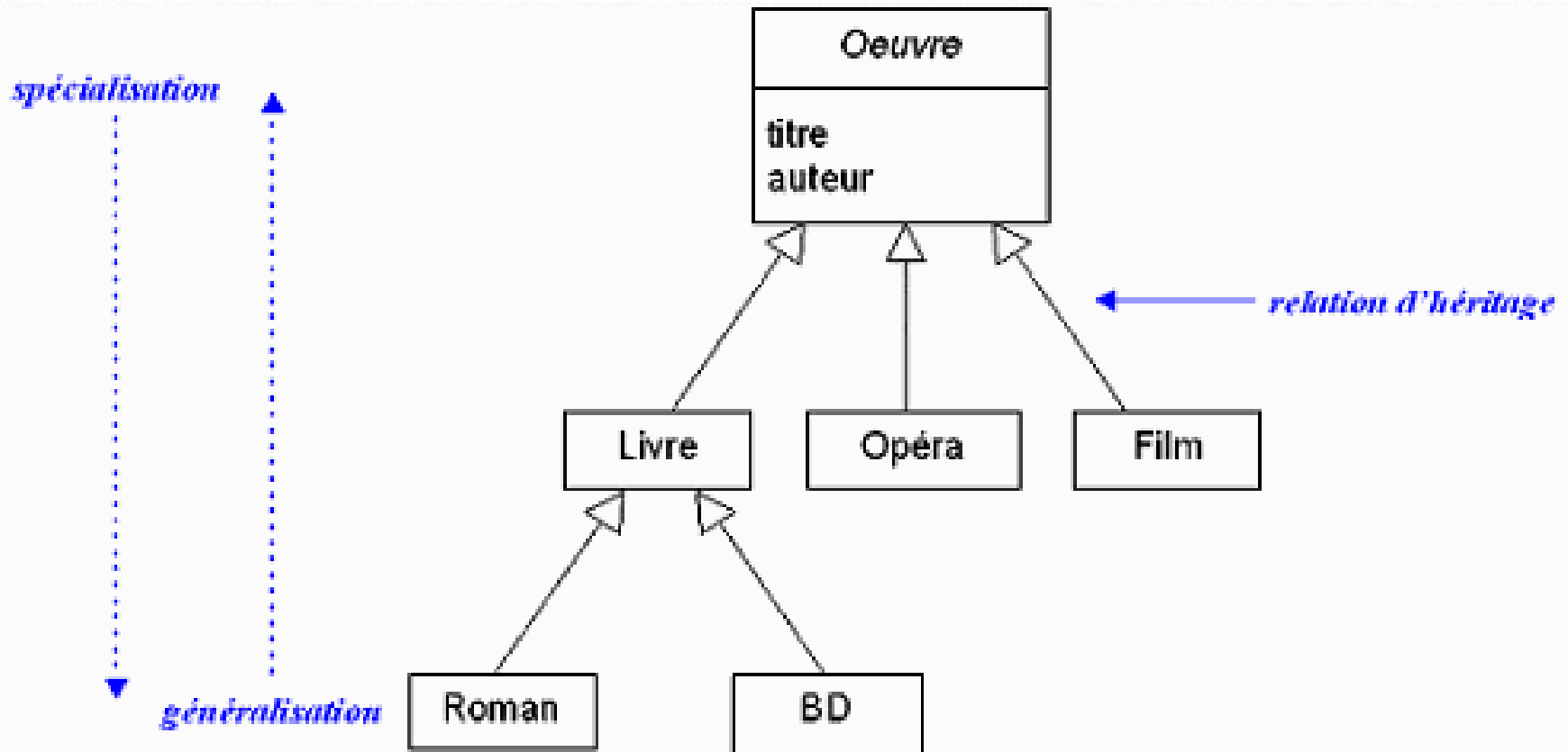
L'encapsulation permet de présenter uniquement l'abstraction à son observateur en lui permettant d'ignorer les détails de sa réalisation.



Ex. *Utiliser une voiture*



Notion d'héritage



Notion d'héritage

Généralisation/Spécialisation

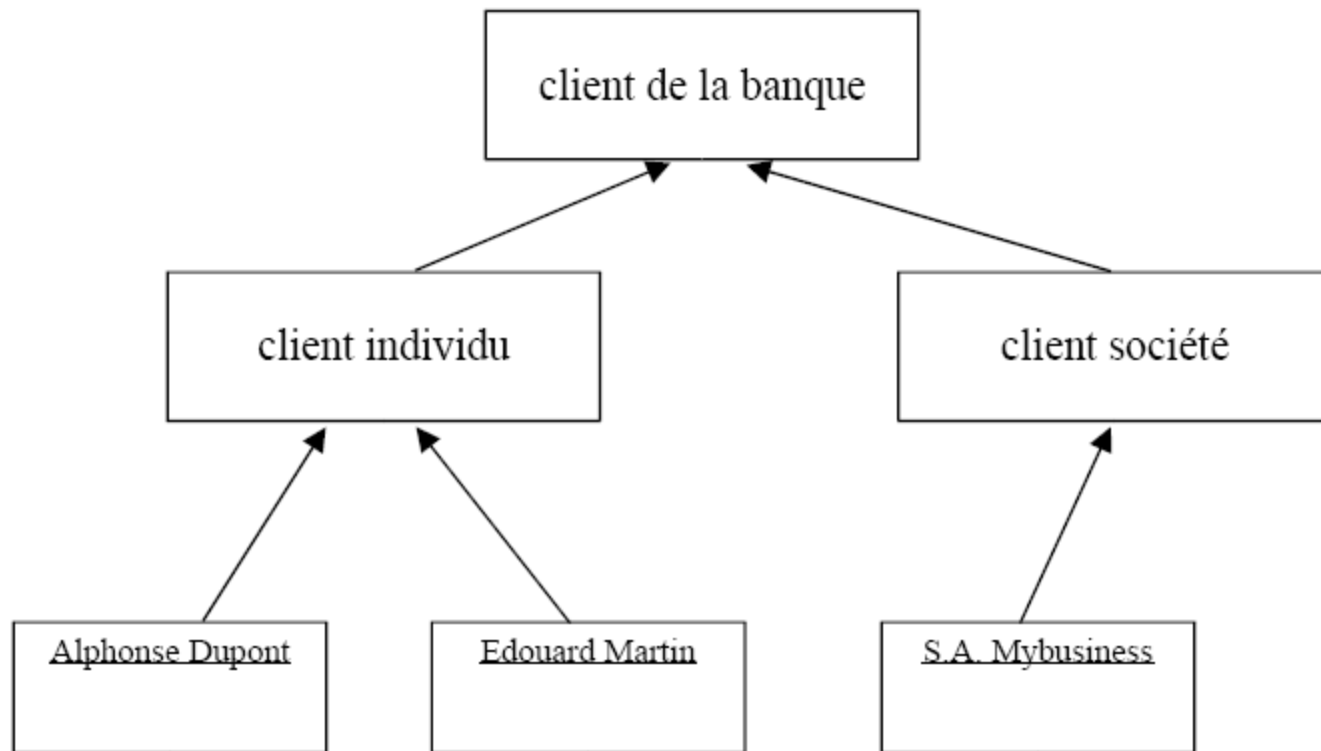
- Classement des abstractions en hiérarchie
- Lorsque des classes ont des caractéristiques communes, on peut mettre en commun ces informations/comportements dans une classe spécifique.
- Les classes d'origine hériteront des caractéristiques de celle-ci.

Notion d'héritage

- La spécialisation et la généralisation permettent de construire des hiérarchies de classes. L'héritage peut être **simple ou multiple.**

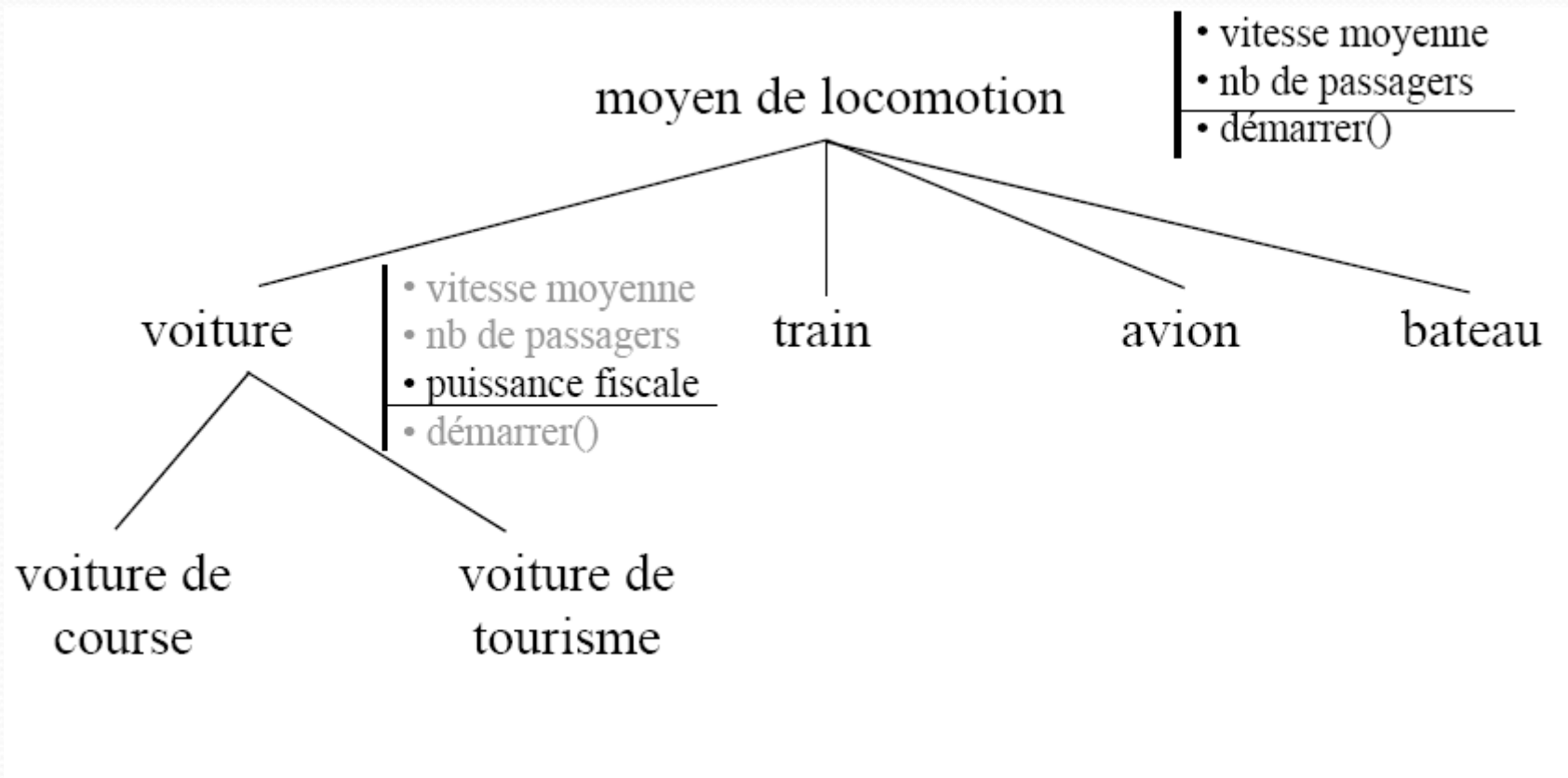
L'héritage évite la duplication et encourage la réutilisation.

Notion d'héritage



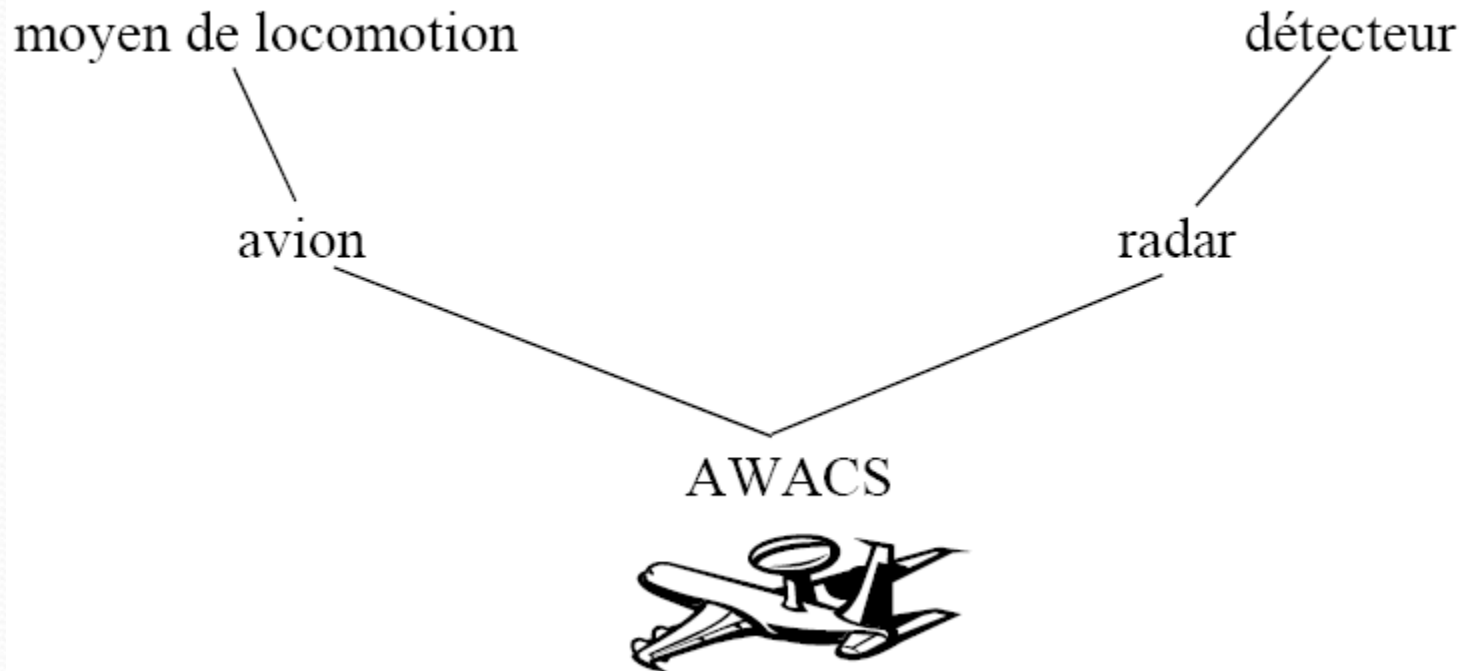
Notion d'héritage

- héritage simple : «est un»



Notion d'héritage

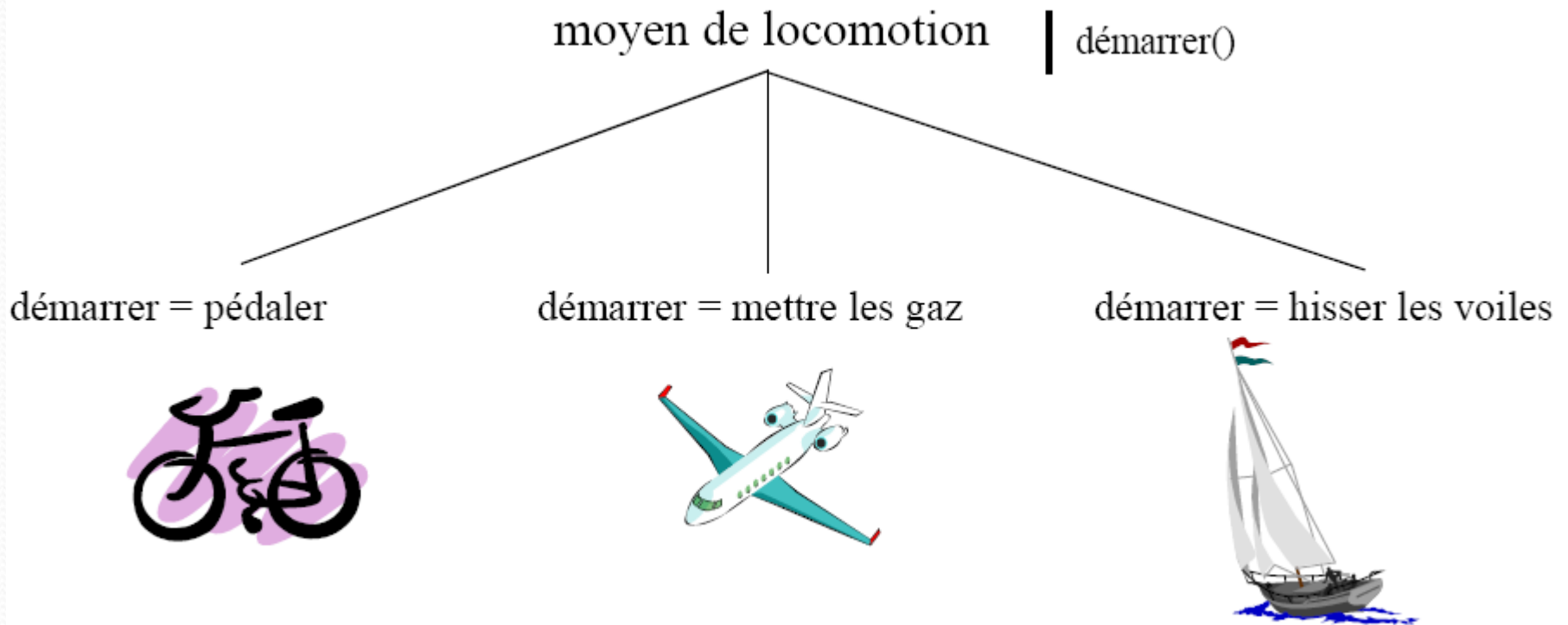
- héritage multiple : «partie de»



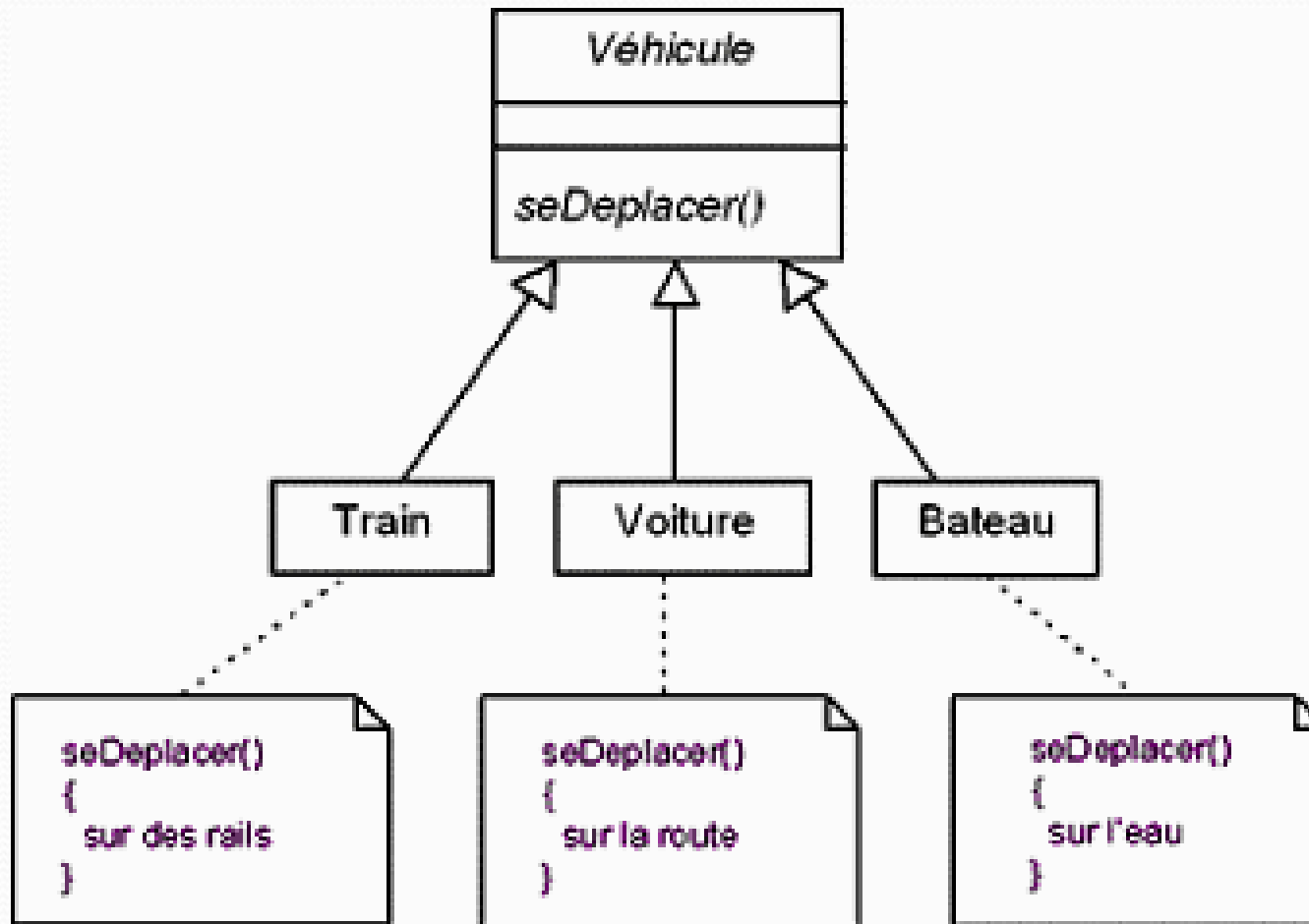
Notion de polymorphisme

- Capacité des objets d'une même hiérarchie à répondre différemment à la même demande
 - Une méthode peut être interprétée de différentes façons
- ➔ Le polymorphisme augmente la généricité du code

Ex de polymorphisme



Ex de polymorphisme

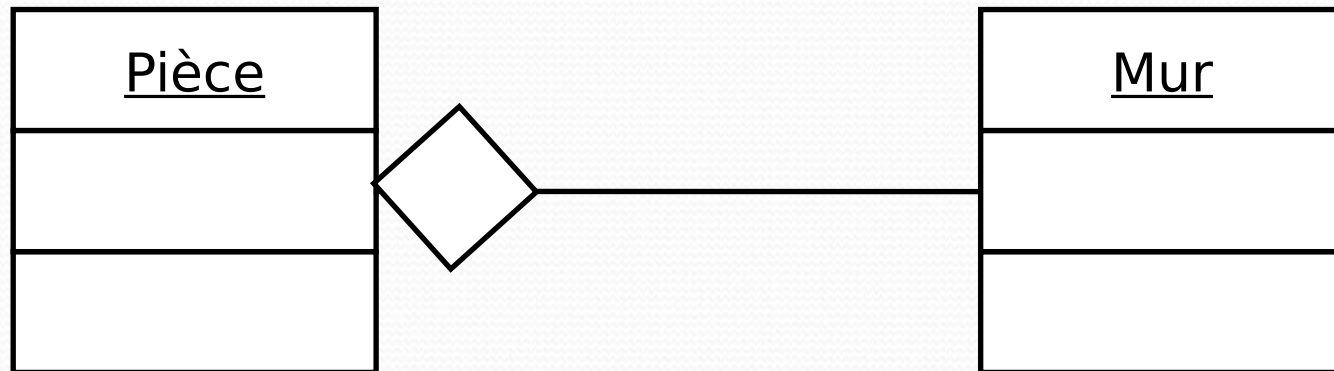


Notion d'Agrégation

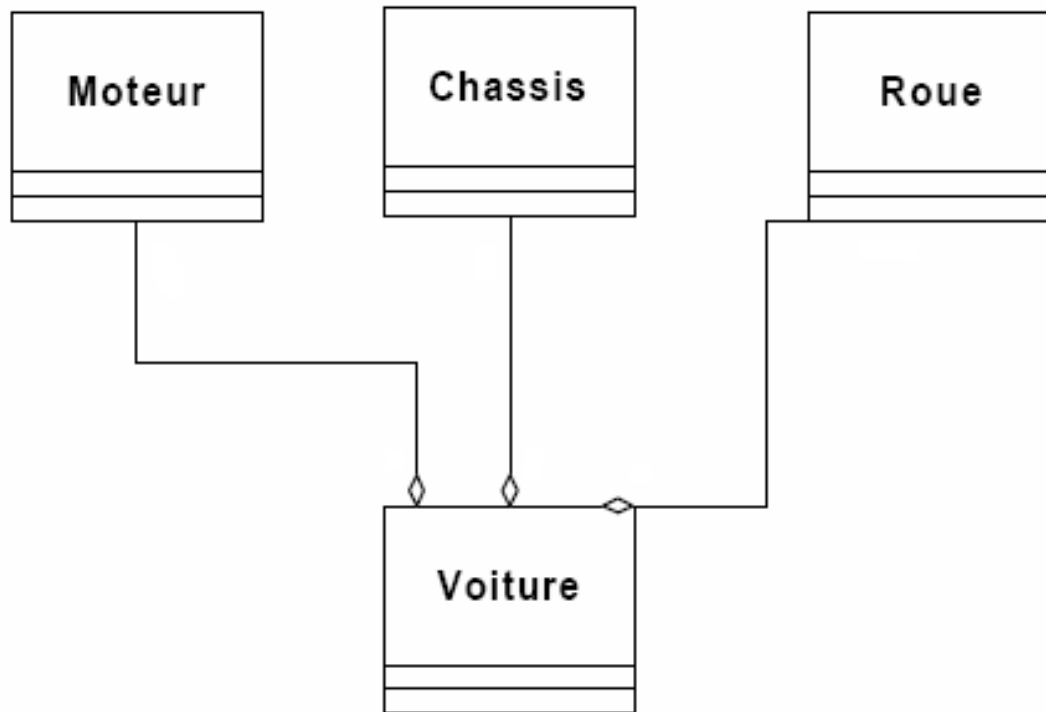
- Il s'agit d'une relation entre deux classes, spécifiant que les objets d'une classe sont des composants de l'autre classe.
- Une relation d'agrégation permet donc de définir des objets composés d'autres objets.
- L'agrégation permet d'assembler des objets de base, afin de construire des objets plus complexes.

l'objet contient, regroupe,
possède

Ex d'Agrégation



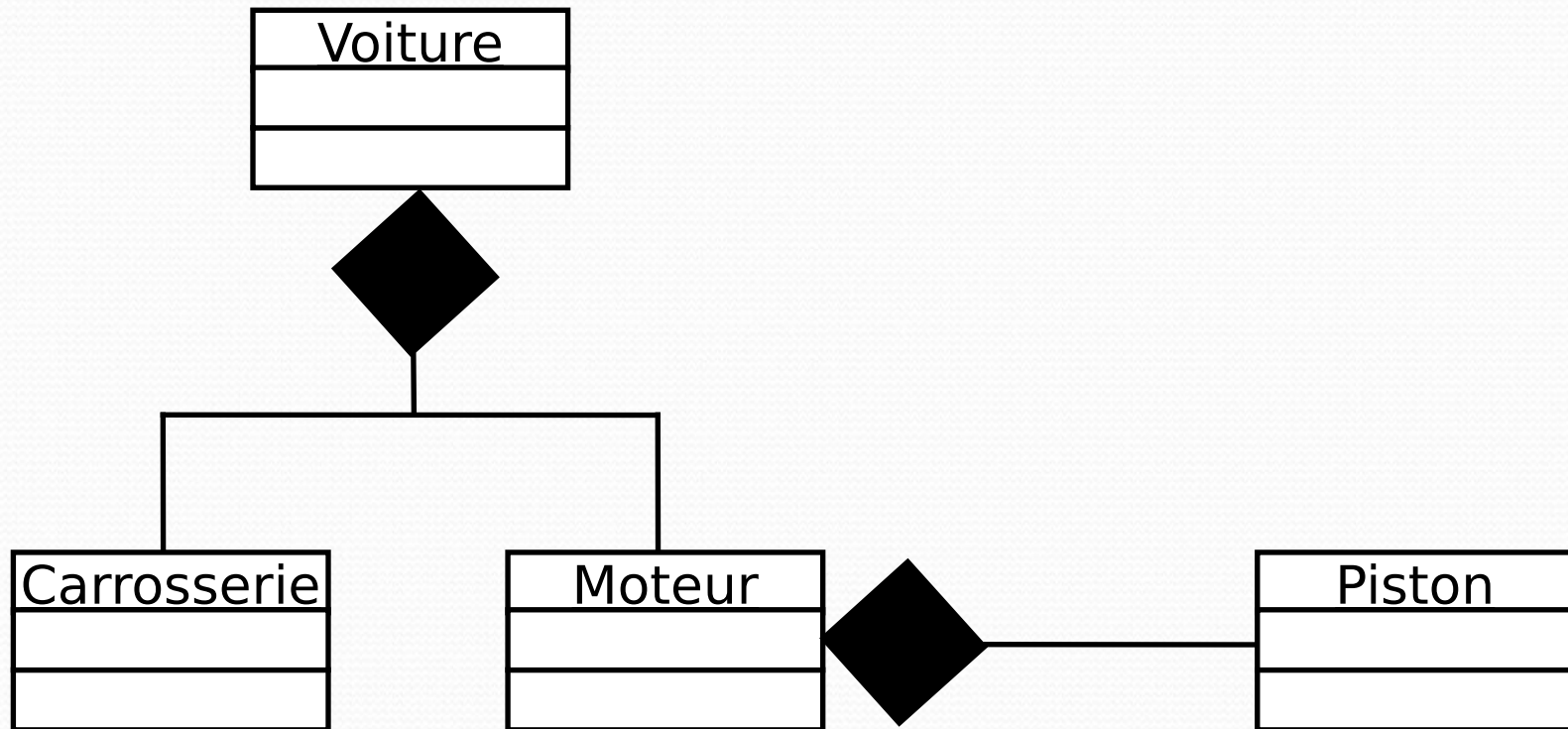
Ex d'Agrégation



Composition

- Cas particulier d'agrégation : **contenance physique**
- Représente une relation de type "**composé / composant**"
- Les **cycles de vies** des composants et du composé sont **liés** : si le composé est détruit (ou copié), ses composants le sont aussi
- Le composant **ne peut pas appartenir simultanément** à plusieurs agrégats

Ex de Composition

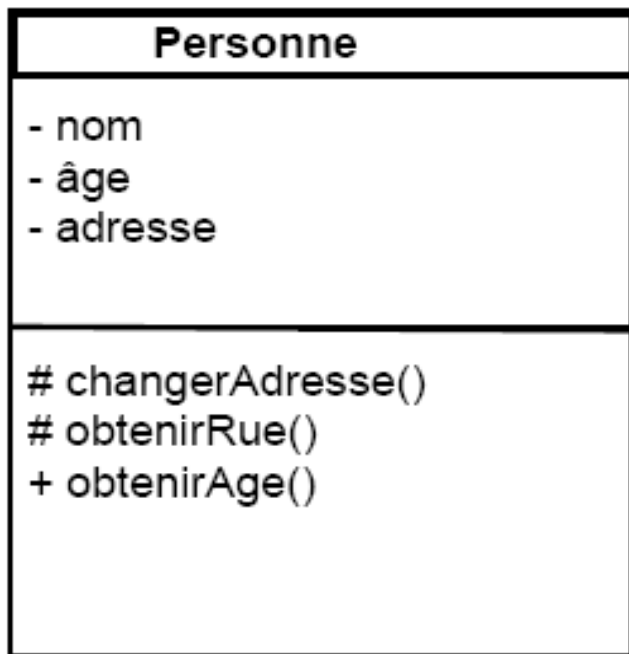


Protection des attributs et des opérations

- Les attributs sont en général inaccessibles (secret).
 - Ils sont alors qualifiés de :
 - "**protected**" (notation UML : #)
 - accessibles seulement à la classe elle-même et aux classes dérivées
 - ou "**private**" (notation UML: -)
 - accessibles seulement par la classe elle-même
- Leur lecture ou modification n'est possible qu'au travers de certaines opérations (accesseurs : *changerAdresse()*, *obtenirAge()*, etc.)
- Les opérations sont en général accessibles (publiques) :
 - Elles sont alors qualifiées de :
 - "**public**" (notation UML: +)
- Certaines opérations peuvent cependant être privées et certains attributs peuvent être publics (non souhaitable / principe d'encapsulation)

Protection des attributs et des opérations

CLASSE

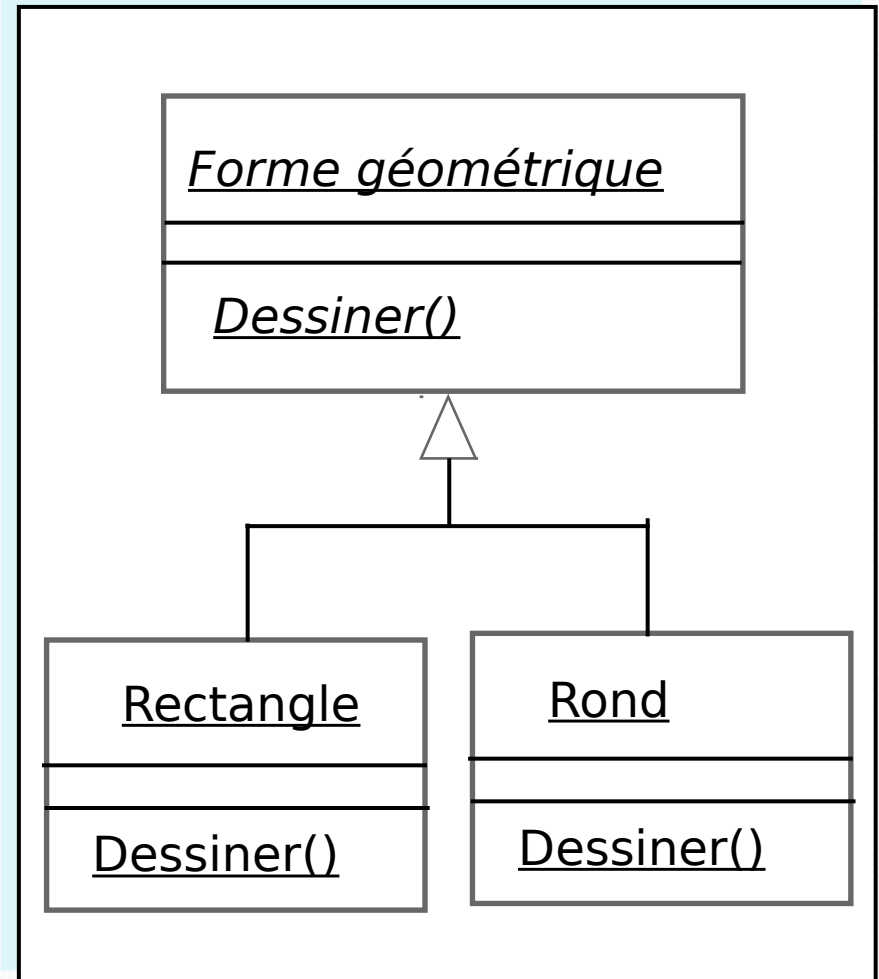


+ : attribut public
: attribut protected
- : attribut private

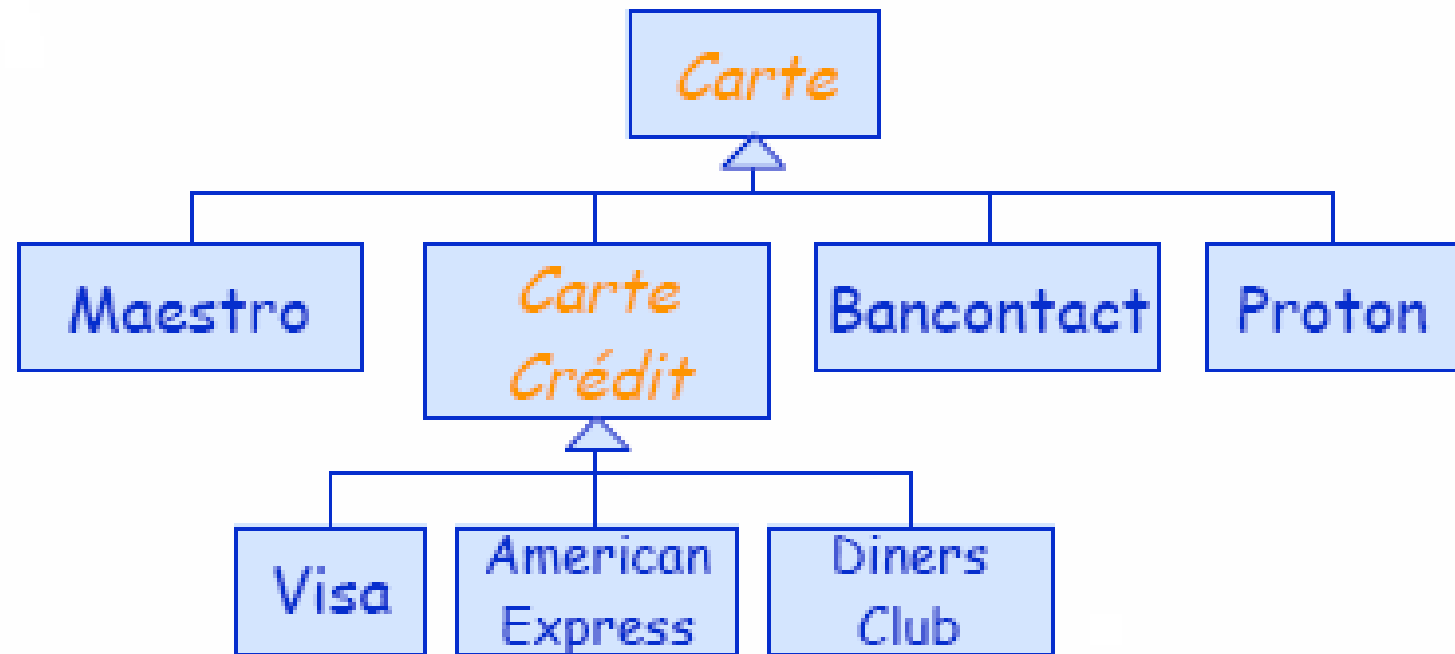
+ : opération public
: opération protected
- : opération private

Classe et opération abstraites

- Une **classe abstraite** est une classe pour laquelle il n'est pas possible de créer d'instances directement
- Une **opération abstraite** d'une classe A est une opération ne possédant pas d'implémentation dans A mais qui doit obligatoirement être implémentée dans les sous-classes de A
- Toute classe contenant au moins une opération abstraite est abstraite



Ex de Classe abstraites



Interface

- Une **interface** permet de décrire le **comportement** d'une entité (classe, paquetage ou composant), c'est à dire un savoir faire sous la forme d'une **liste d'opérations**.
- Une interface ne peut donner lieu à **aucune implémentation**.
- Une interface est équivalente à une classe abstraite sans attributs où toutes les méthodes sont abstraites.
- **Une classe** peut déclarer qu'elle **implémente une interface**. Elle doit alors implémenter toutes les opérations de cette interface. Elle peut ensuite être utilisée partout où ce comportement est exigé.

Ex d'Interface

```
<<interface>>  
Comparable
```

```
    estEgal()  
    estDifférent()  
    estPlusPetit()  
    estPlusGrand()
```