Programmation Orientée Objet 2

Benoît Sonntag – sonntag@icps.u-strasbg.fr

Université Louis Pasteur – Strasbourg

September 22, 2008

Qui suis-je ?

Le projet Isaac : Technologie Objet à prototype

`http://isaacproject.u-strasbg.fr/`

ou

`http://IsaacOS.com`

- Développement d'un **Langage** à prototype : Lisaac.
- Développement d'un **compilateur** performant pour Lisaac.
- Développement d'un **système d'exploitation** : IsaacOS.

Modalité

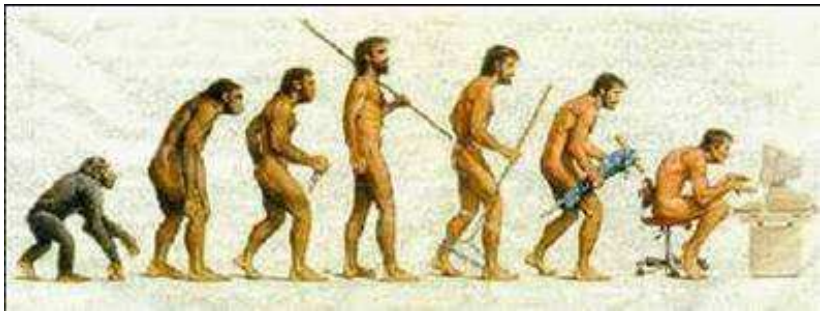
Volume horaire :

- Cours : 20h.
- TD : 16h.
- TP : 12h.

Contrôle des connaîssances :

- Première session
  - Contrôle continu sous forme de projet et TP noté : Coef. 1/3
  - Examen écrit en Janvier : Coef. 2/3 (durée 3h.)
- Seconde session
  - Examen écrit en Juin : Coef. 1 (durée 3h.) –Plus difficile–

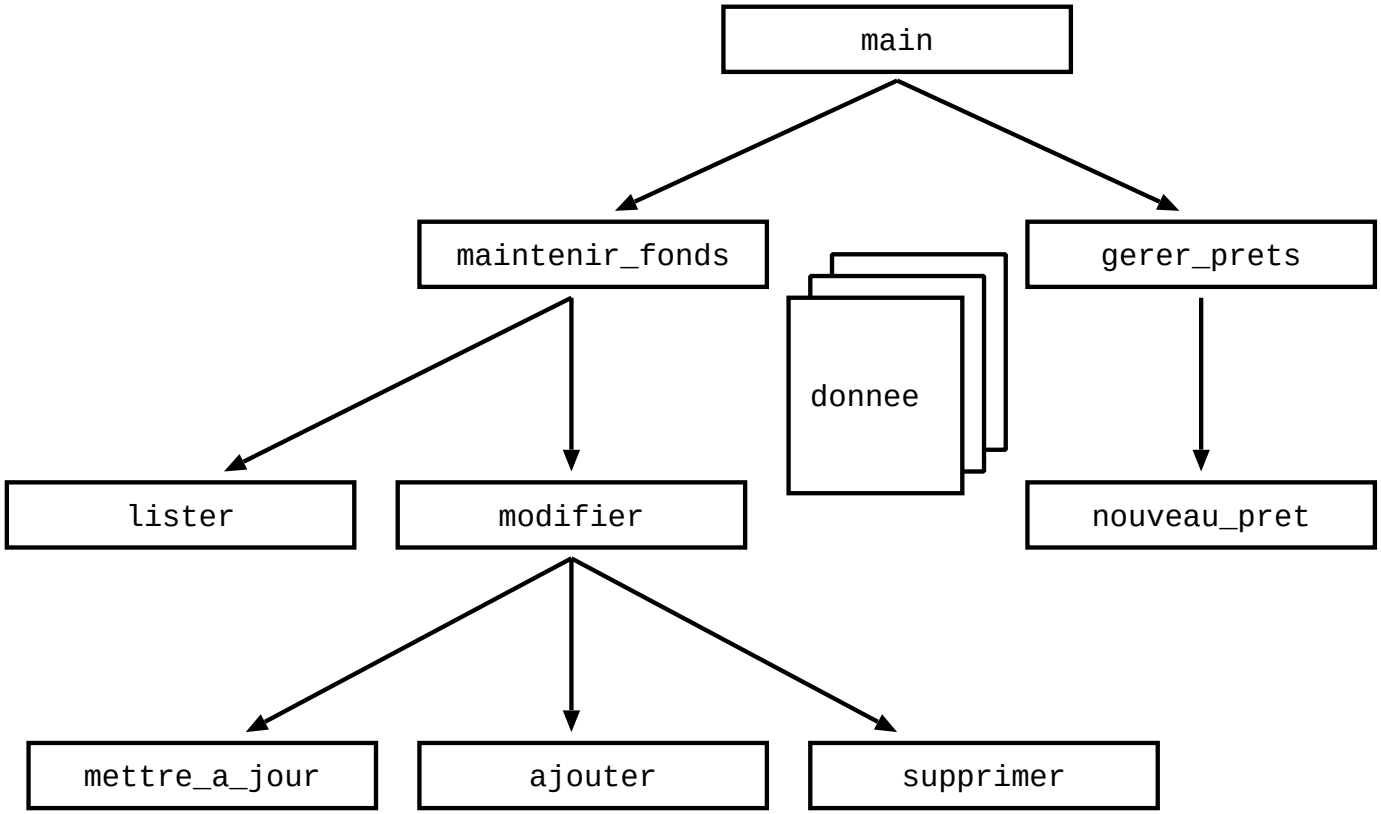
L'évolution des langages de programmation



- 1954 : Fortran, langage des mathématiciens
- 1960 : **Simula**, premier langage objet à classe
- 1972 : C, programmation système, puis généraliste
- 1972 : **SmallTalk**, premier langage interprété à objet pur
- 1980 : **C++**, la couche objet pour C
- 1986 : **Eiffel**, L'apogée des langages objet à classes et le premier ayant la programmation par contrat
- 1995 : **Java**, Langage objet à classe pour Internet
- 2002 : **Lisaac**, premier langage à prototype compilé

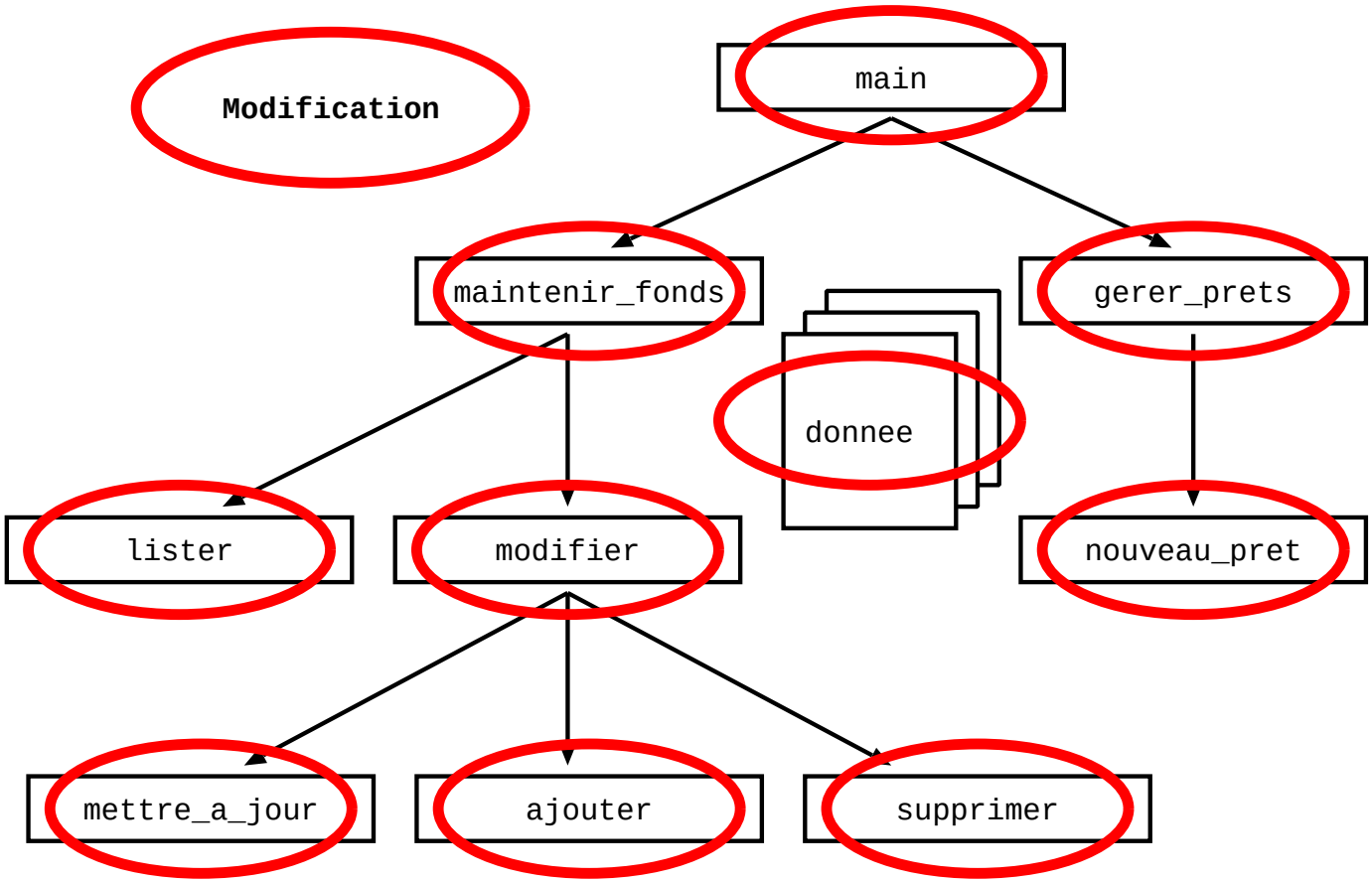
Approche fonctionnelle (1/6)

Exemple 1 : Gestion **d'une Bibliothèque**



Approche fonctionnelle (2/6)

Exemple 1 : Gestion d'une Bibliothèque **et d'une médiathèque**



Approche fonctionnelle *VS* approche objet (3/6)

Exemple 1 : Gestion d'une **Bibliothèque et d'une médiathèque**

Fonctionnelle

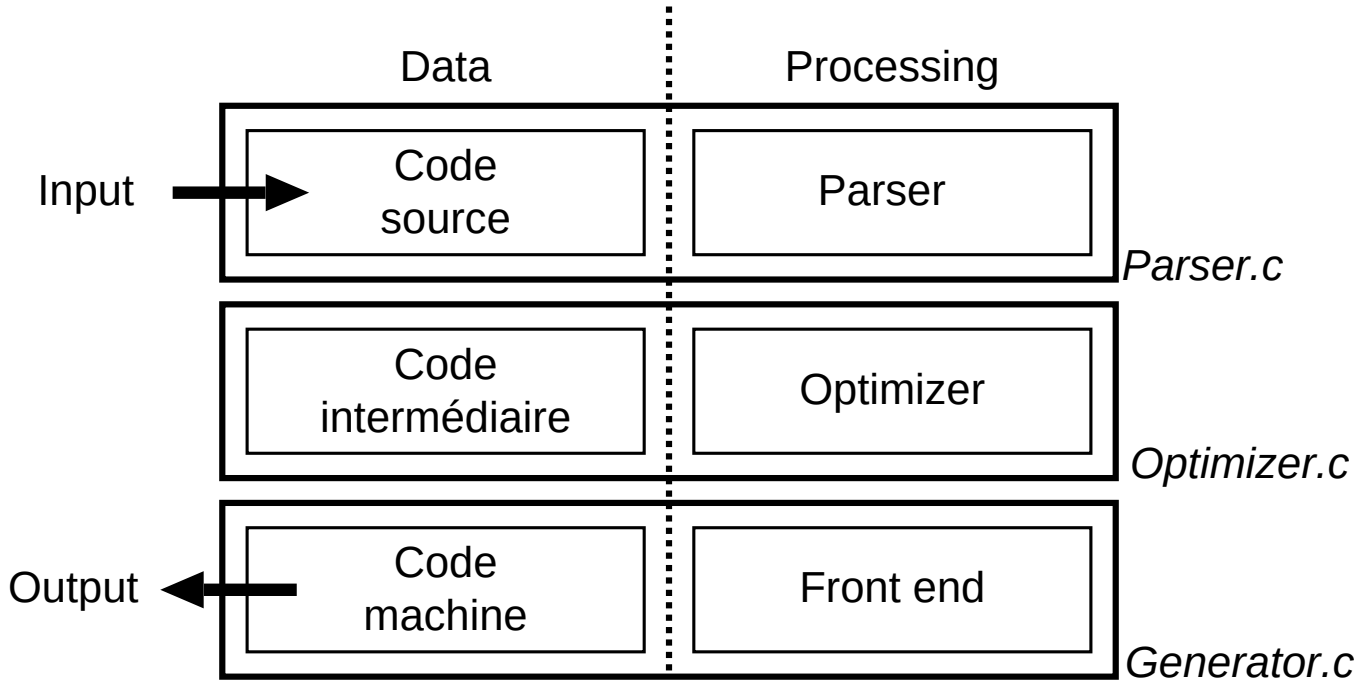
```
switch (DOC[ref].type) {
case LIVRE:
        maj_livre(DOC[ref]);
        break;
case CASSETTE_VIDEO:
        maj_k7(DOC[ref]);
        break;
case CD_ROM:
        maj_cd(DOC[ref]);
        break;
}
```

Objet

```
DOC[ref].maj();
```

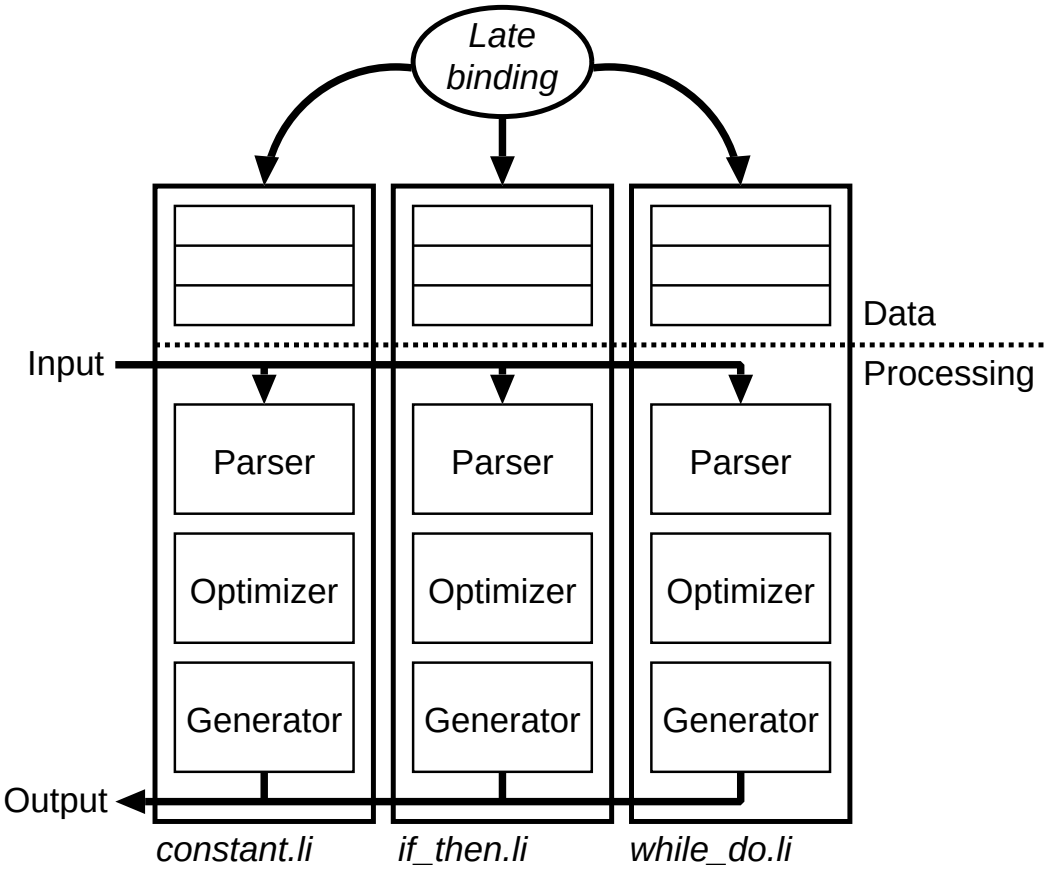
Approche fonctionnelle : structuration horizontale (4/6)

Exemple 2 : Modélisation d'un **compilateur**



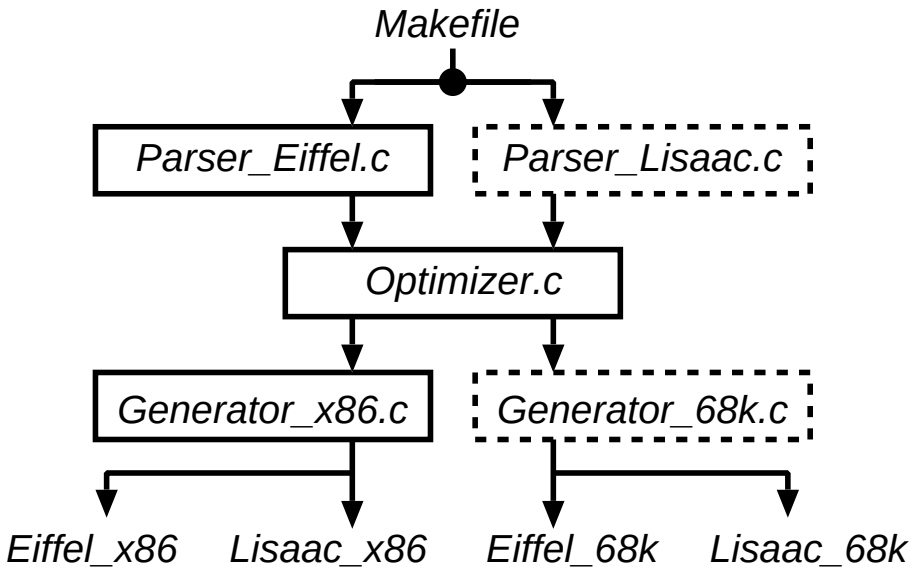
Approche objet : structuration verticale (5/6)

Exemple 2 : Modélisation d'un **compilateur**

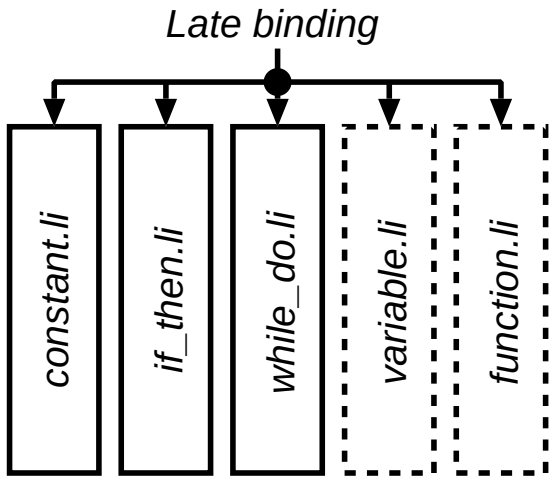


fonctionnelle *vs* objet : flexibilité (6/6)

Exemple 2 : Modélisation d'un **compilateur**



Processing flexibility



Data flexibility

Définition d'un objet

Définition

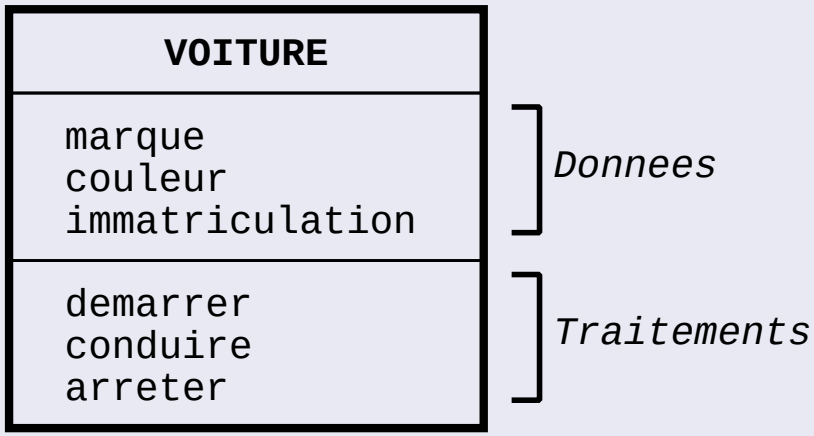
Un **objet** : Une entité autonome, qui regroupe un ensemble de **propriétés** cohérentes et de **traitements** associés.

En résumée. . .

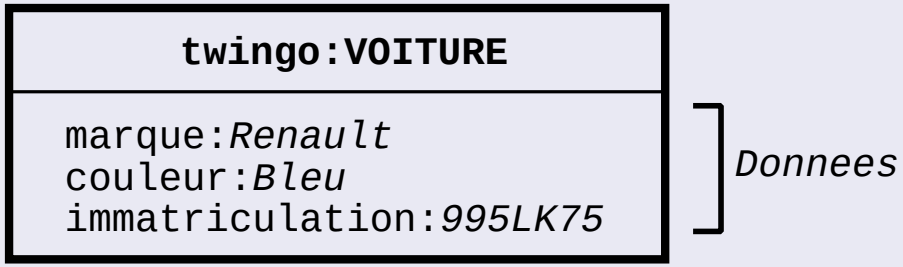
- Approche fonctionnelle : Dirigé par le traitement
- Approche objet : Dirigé par le type des données

Classes & Objets

Classe : regroupement de données & de traitements



Objet : instance d'une classe



Analogie avec le monde réel (1/2)

Une entité

- Un chien
- qui s'appelle Prosper
- qui est de couleur blanche
- qui à faim
- qui va chercher une balle
- qui aboie

Analogie avec le monde réel (2/2)

Objet

- Un chien

Propriété / Etat (= données)

- qui s'appelle Prosper
- qui est de couleur blanche
- qui à faim

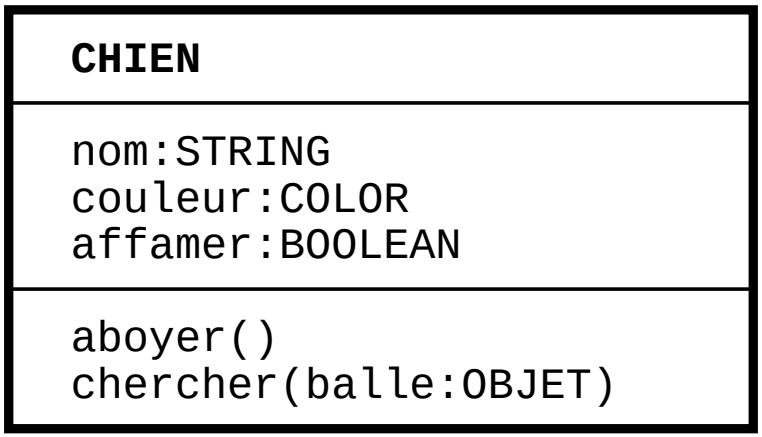
Comportement (= méthodes)

- qui va chercher une balle
- qui aboie

Classe d'objet

Definition

Classe : Représentation abstraite d'une cathégorie d'objets.
La classe d'un objet est aussi appelé son **type**.



Instance d'une classe : un objet

Definition

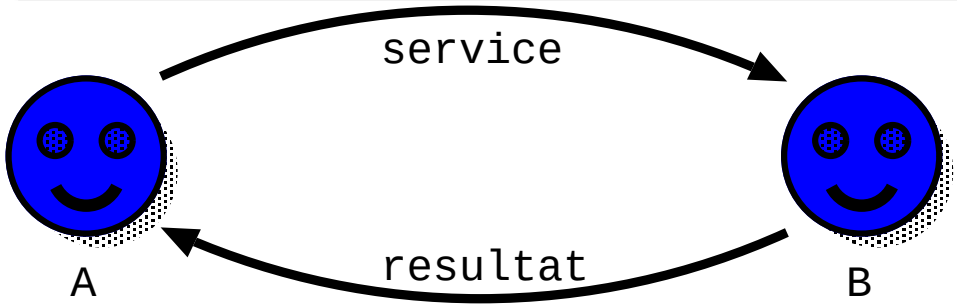
Objet : Représentation en mémoire d'une entité physique.
Un objet appartient à une classe.

CHIEN:*monchien*
nom:STRING := "Prosper" **couleur:COLOR := blanc** **affamer:BOOLEAN := TRUE**
aboyer() chercher(balle:OBJET)

Communication inter-objets (1/3)

Envoi de message sur un objet

Objet : Données + méthodes (= fonctions ou procédures).
Service/message : Donnée ou méthode.



Dans A : **B ● service**

Point de **liaison dynamique**
(late binding)

Communication inter-objets (2/3)

Appel

Dans A : `b.service`

Vocabulaire : *pour avoir les idées claires...*

b est l'appelé.
a est appelant.
a est client de *b*.
b réalise un service pour *a*.
a envoi un message à *b*.

Communication inter-objets (3/3)

Exemple d'invocation de message en un point de liaison dynamique

```
mon_chien.aboyer();
mon_chien.chercher(ma_balle);
System.out.println(mon_chien.nom);
```

En langage objet pur (type SmallTalk ou Lisaac)

```
mon_chien.aboyer;
mon_chien.chercher ma_balle;
mon_chien.nom.print;
(mon_chien.nom).print;
```

Encapsulation *(aussi appelé boîte noire)* (1/3)

Définition

Encapsulation : Cacher l'information d'un objet (son implantation) et ne proposer que des services disponibles pour le client.

Solutions

1. Limiter des accès avec `private`, `public`, ...
2. Interdire l'accès en écriture par un client *(pas comme Java)*
3. Ne pas diffèrencier méthodes et données pour le client *(pas comme Java)*

Notion de Encapsulation (2/3)

Avantages

- **Facilite l'évolution** d'une application : On peut modifier l'implantation d'un objet sans modifier sont utilisation.
- **Garantie l'intégrité des données** : l'encapsulation permet d'interdire l'accès en écriture d'une donnée. L'objet reste maître de la valeur de ses données.

Notion de Encapsulation (3/3)

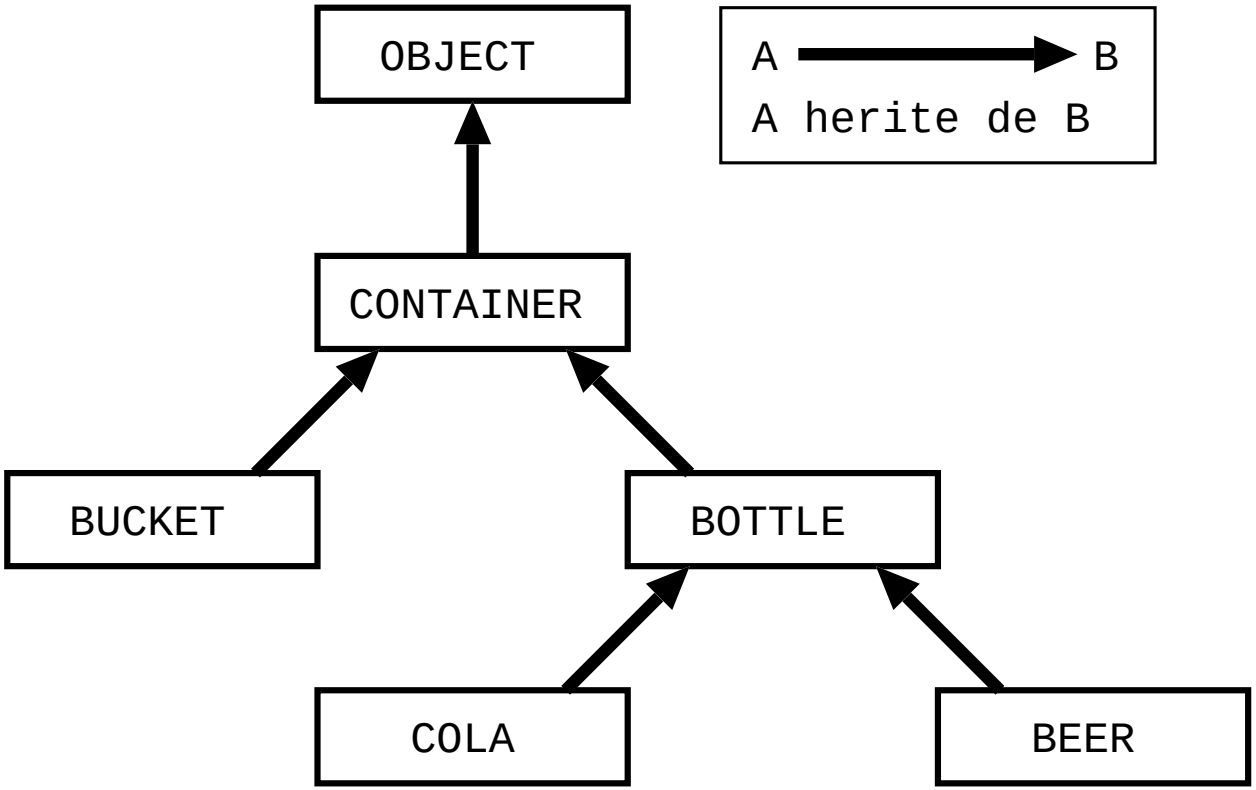
Remarque

En Java ou C++, le langage ne garantit pas l'encapsulation.
Ce problème exige :

- la déclaration de l'ensemble des données en `private` (ou `protected`)
- l'écriture de méthode *getter* et *setter*, pour lire ou écrire dans une donnée.

Les règles d'encapsulation sont ainsi présentes, au prix d'une lourdeur d'écriture et d'un accès systématique à une méthode pour le client *(En bref, des '()' qui polluent le code ! :-))*.

Notion d'héritage (1/4)



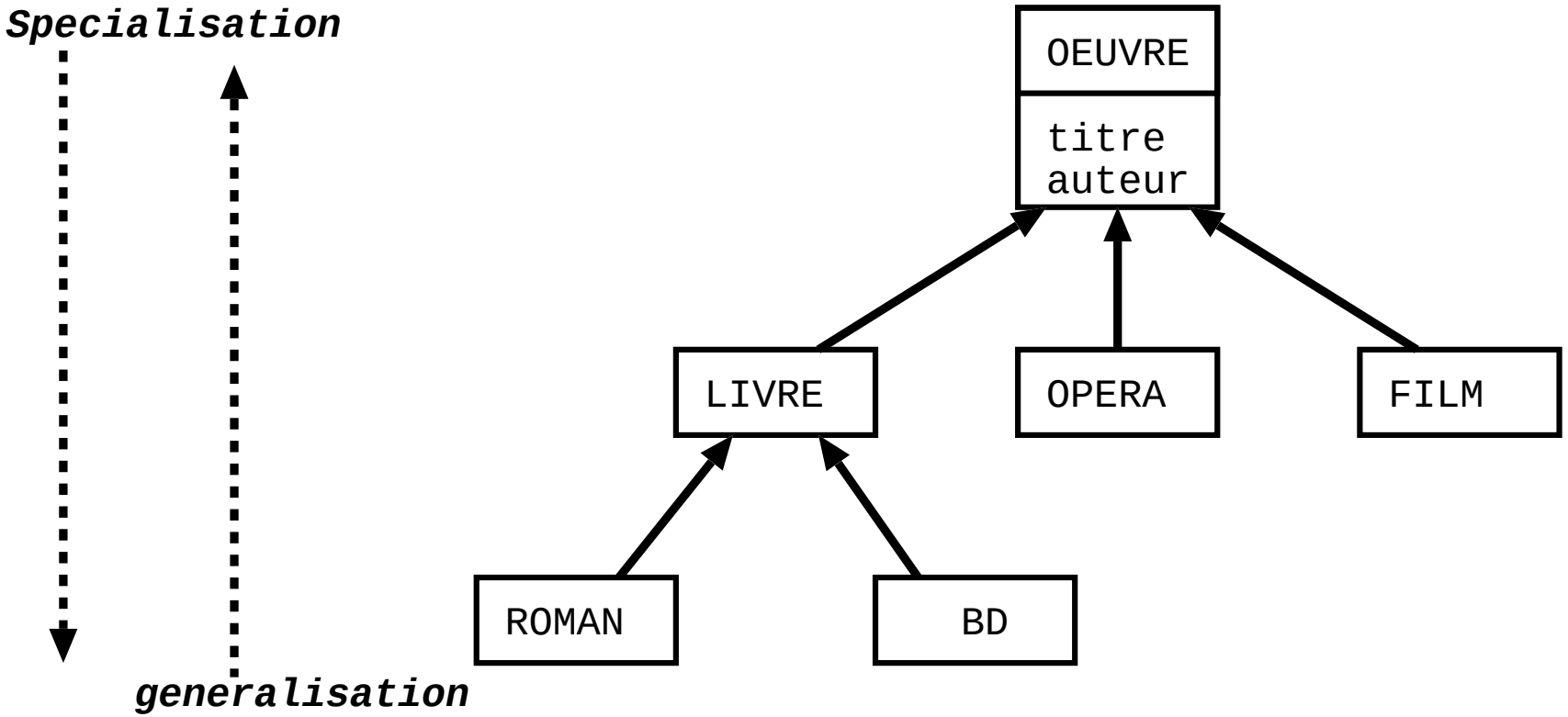
A ⟶ B

A herite de B

Definition General

Definition specifique

Notion d'héritage (2/4)



Notion d'héritage (3/4)