



Chapitre 2

Objets, classes et formalisme UML

Plan

- Encapsulation
- Objet
- Classe
- Représentation UML
- Attribut
- Méthode

Encapsulation

En orienté objet, la brique de base est l'objet

Un objet...

- est une entité encapsulée (= elle forme un tout et peut cacher/rendre inaccessibles certaines parties)
- regroupe à la fois
 - des données (= "valeurs d'attributs") qui décrivent son état
 - des modules (appelés "méthodes") qui indiquent ce qu'il peut faire en réponse à des messages et constitue son comportement



En général, on ne peut accéder aux valeurs de l'objet que via ses méthodes. C'est ce qu'on appelle l'encapsulation

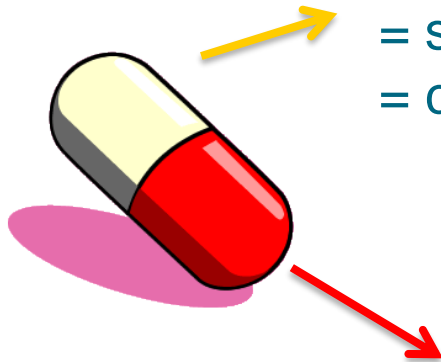
Encapsulation

Comme une gélule médicinale (= capsule en anglais), un objet présente...

une partie transparente dont on peut voir le contenu depuis l'extérieur

= son interface, ses éléments de visibilité **publique**

= ce à quoi le monde extérieur a accès



une partie opaque dont le contenu est caché



= sa partie **privée**, interne

= ce à quoi le monde extérieur ne peut pas accéder directement

Encapsulation

Relations entre les deux découpes :

- état et comportement
- partie publique et partie privée

	Partie publique	Partie privée
État 	<i>généralement vide</i> (pour empêcher l'accès direct aux données)	<i>En règle générale (= 99% des cas), les données sont privées</i>
Comportement 	<i>Méthodes accessibles depuis l'extérieur</i> = comment répondre aux messages que d'autres objets peuvent envoyer	<i>Méthodes "internes"</i> = gestion interne de l'objet, pas directement accessible depuis l'extérieur

Encapsulation

L'encapsulation permet...

- une découpe facile du travail (**modularité**)

Comme on ne peut accéder à la partie privée d'un objet que via ses méthodes publiques, les codes externes et internes peuvent être construits de manière indépendantes.

- Une **sécurité** renforcée

*Le monde extérieur ne peut pas modifier les valeurs privées directement, mais seulement via des méthodes. Ces méthodes peuvent appliquer des **filtres** et donc **refuser les modifications indésirables**.*

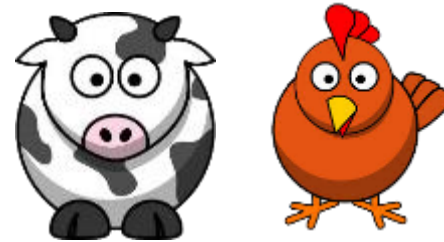
- un (accès plus aisé à un) **couplage faible**

Comme les codes internes et externes sont indépendants, on peut en modifier un sans devoir changer l'autre (tant que l'interface reste inchangée).

Objet

Qu'est-ce qui peut distinguer deux objets différents ?

(1) des comportements différents



(2) *même comportement mais...*
des valeurs d'attributs différentes



(3) *même comportement et*
mêmes valeurs d'attributs mais...
des identités différentes



Objet

État



= valeurs des **attributs** à un moment donné

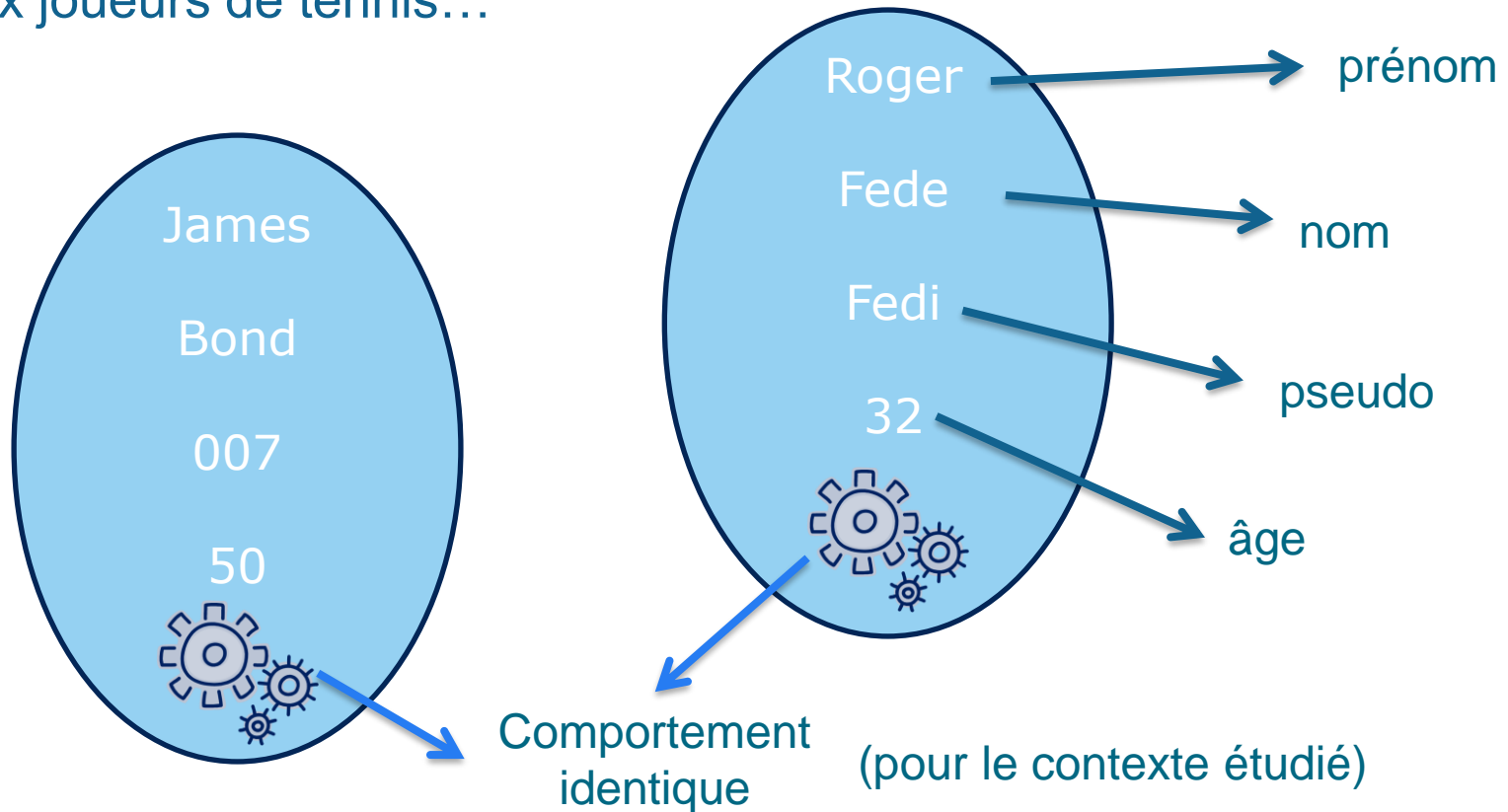
↳ valeur constante (exemple : nom)
ou en évolution (exemple : nombre d'enfants)

Comportement = ensemble de messages compréhensibles par l'objet

= "actions" déclenchées suite à la réception d'un message et définies par les **méthodes**
(ces actions peuvent modifier l'état de l'objet !)

Objet

Deux joueurs de tennis...



Réflexion : les attributs de cet exemple sont-ils bien choisis ?

Discutez selon les cas

Objet

L'inconvénient de stocker l'âge comme propriété ?

L'âge varie

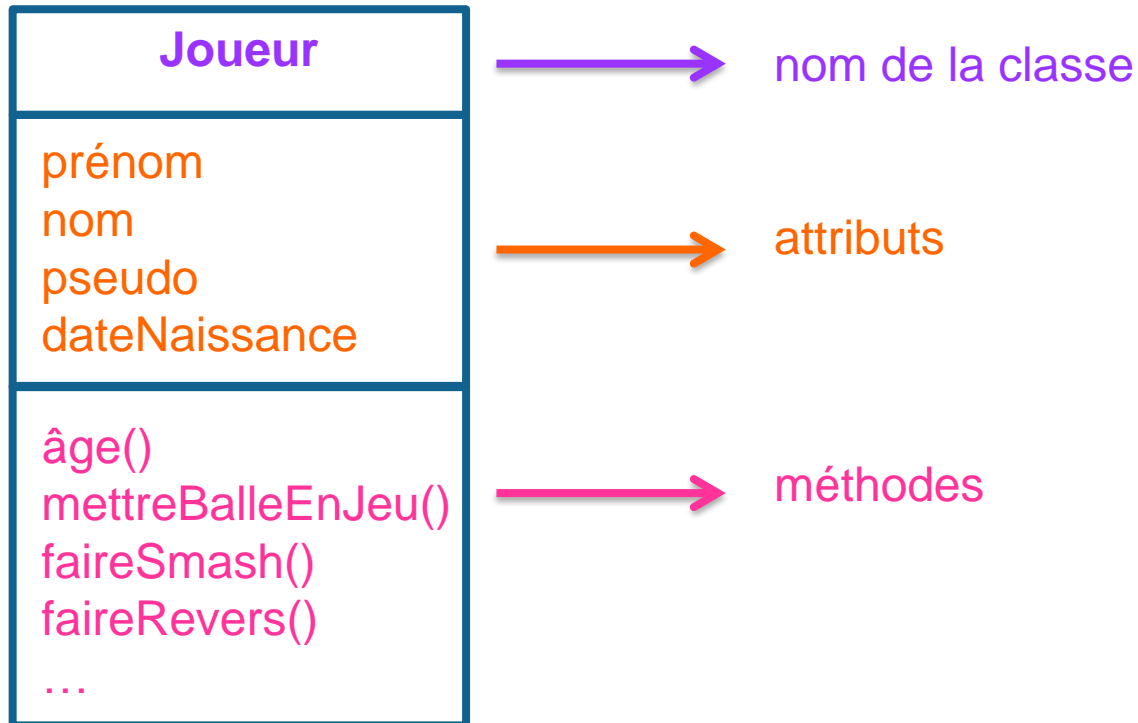
Or, il peut être calculé à partir de la date de naissance

⇒ ~~Attribut âge~~

⇒ Attribut dateNaissance + méthode âge() car calculable

Classe

Quels sont les points communs entre ces objets ?



Classe

Moule de l'objet ou description générale de l'objet

- Description des **attributs** et des **méthodes** de toute une famille d'objets
- Les objets d'une même classe auront des attributs de même type mais des valeurs (peut-être) différentes

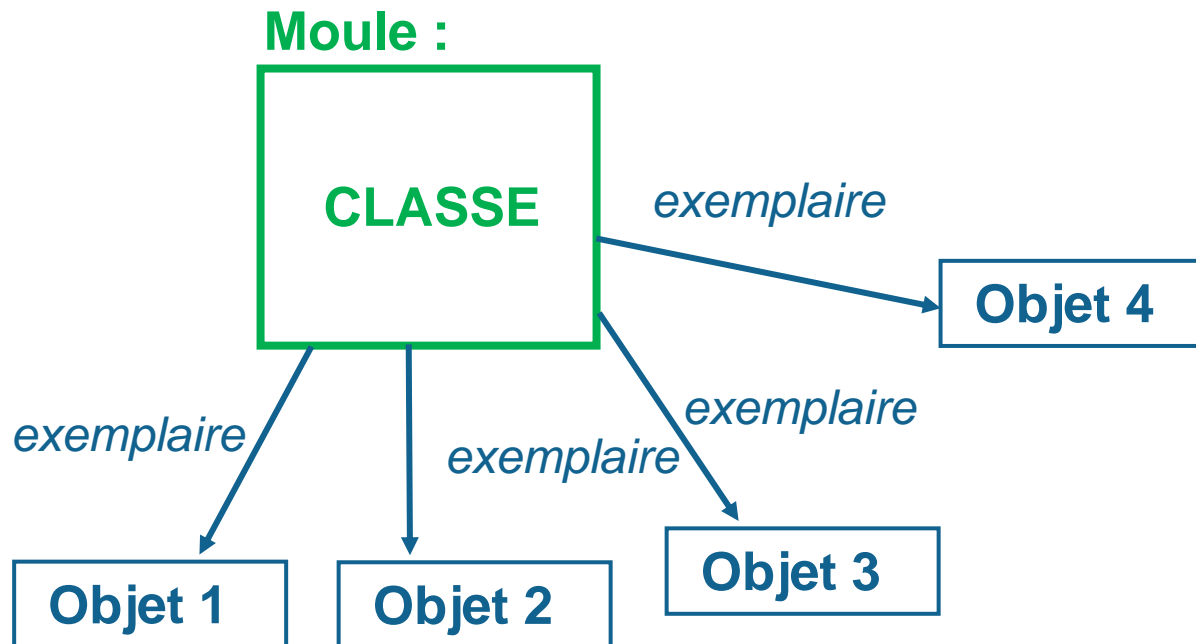
Classe

Fabrique d'objets

Instanciation

- Création d'un objet à partir de sa classe
 - Donner des valeurs aux attributs, c'est-à-dire déterminer son état
- On dit aussi :
 - Un objet est une **instance** de sa classe

Classe



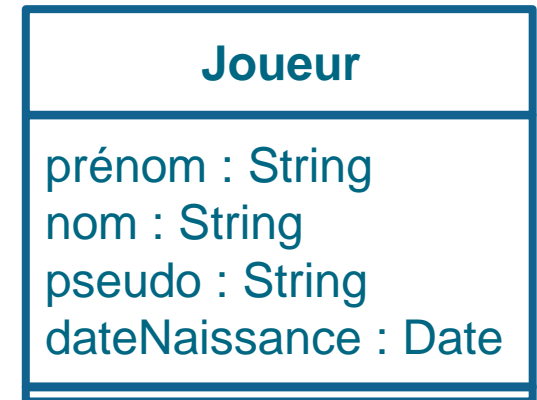
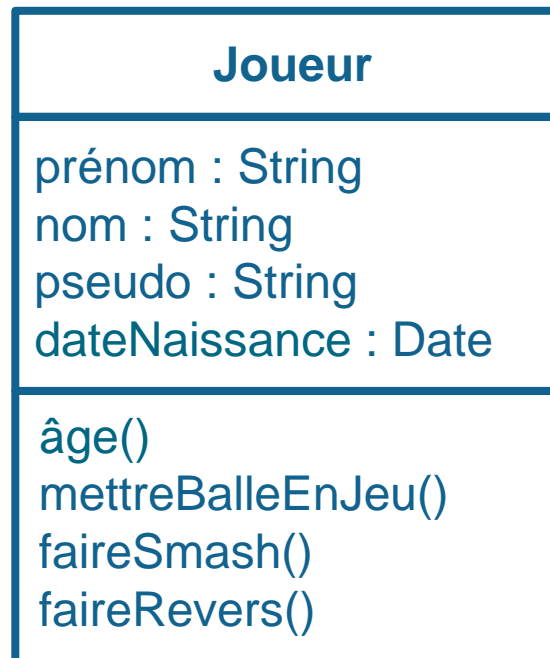
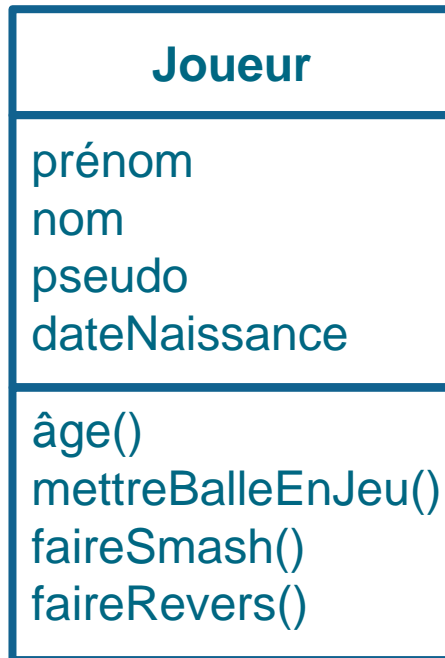
Classe

Peut être vue comme une **boîte à outils**

⇒ reprend toutes les fonctions/méthodes (= outils) qu'on peut appeler sur les objets de la classe

Représentation UML

UML permet différents niveaux de **granularité** (précision)



préciser les types... ou pas...

préciser les attributs/méthodes... ou pas...

Représentation UML

UML permet différents niveaux de **granularité** (précision)

Joueur

prénom : String
nom : String
pseudo : String
dateNaissance : Date

Principe de la granularité : ce n'est pas parce que quelque chose n'est pas spécifié que cette chose n'existe pas...

Exemples :

- *On ne cite aucune méthode...
ça ne veut pas dire qu'il n'y en a pas !*
- *On cite 4 méthodes qu'on juge importantes...
ça ne veut pas dire qu'il n'y en a pas d'autres !*
- *On cite une méthode sans préciser de paramètres...
ça ne veut pas dire qu'il n'y en a pas !*

Représentation UML

Convention dans les exercices :

On ne précise pas les types des attributs sauf les attributs booléens

Représentation UML

Diagramme de classes

= Ensemble de classes interagissant entre elles

Chaque classe définit les (types des) attributs et les méthodes (= messages auxquels ses objets peuvent réagir) qui lui sont propres.

Ce sont ces classes et ces interactions que le diagramme UML va représenter !

La plupart des exercices de PPOO consisteront à traduire un énoncé parfois imprécis en un diagramme UML reprenant les classes permettant de le modéliser.

Attribut : types et notations UML

Convention pour le cours :

indiquer explicitement les attributs/variables booléens

Film

titre
duréeEnMinutes
annéeSortie
estEnCouleurs : booléen

Valeurs simples

livreInspiration [0..1] : Livre

Attribut facultatif :
soit 0 livre soit 1 livre

Référence vers un **objet**
de type **Livre**
On ne recopie pas tout le
contenu de l'objet ; on garde
juste une référence
(adresse, pointeur) vers cet
objet.

Livre

titre
annéeSortie
langue
auteurs []

Attribut multiple
(tableau par exemple) ;
on peut aussi préciser
la taille min/max : [2..5]

/nbAuteurs

Attribut dérivable (calculable à partir
des autres, généralement codé sous
forme de méthode et dont la valeur
n'est pas retenue en mémoire)

Attribut

Ne pas confondre attribut booléen et facultatif !



Attribut booléen : valeur soit true, soit false

Notation nomAttribut : **booléen**



Attribut facultatif : peut ne pas avoir de valeur

Notation : nomAttribut**[0..1]**

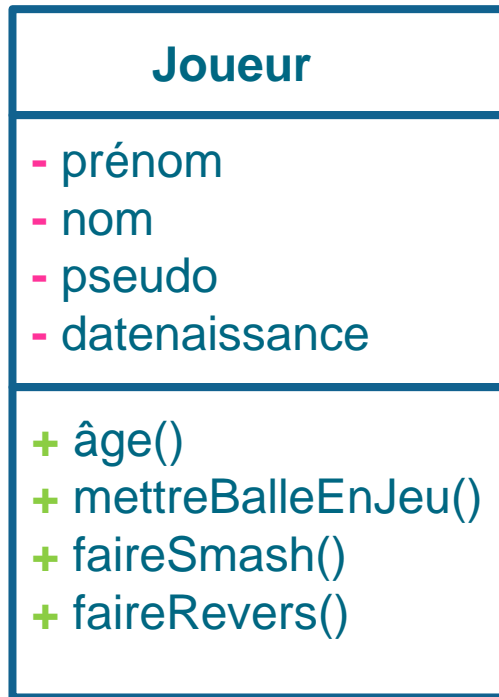
Attribut

Pas d'attribut qui joue le rôle d'identifiant



Schéma entité-association (ERA)

Attribut



On peut également préciser la visibilité des attributs et des méthodes :

- pour privé (= **pas accessible** de l'extérieur)
- + pour public (= **accessible** de l'extérieur)

(d'autres symboles/visibilités existent également)

Méthode

Le code se construit à l'aide de **messages** (déclenchant l'exécution de **méthodes**)

Ex : `navire8.marqueDetruite(10,8);`
`boolean caseDansNavire8 = navire8.contientCase(3,5);`

Chaque message comporte

- un **destinataire** : l'objet auquel il est adressé
- une **demande** : le nom de la méthode à exécuter
- des **paramètres** : les arguments à fournir à la méthode

et peut induire une **réponse** (quand la méthode est une fonction plutôt qu'une procédure) = **type de retour**

Méthode

Chaque classe définit ses propres méthodes
= modules permettant de répondre aux messages reçus

On distingue 4 grands types de méthodes "standards" qu'on retrouve dans la majorité des classes :

- 2 types de méthodes liés à la création/destruction d'objets ;
- 2 types de méthodes liés à l'aspect privé des valeurs des attributs.

+ Autres méthodes :

À côté de ces méthodes "standards", on trouve toutes les autres méthodes propres au fonctionnement de la classe.

Méthode : catégories de méthodes "standard"

- **Constructeur**
 - Crée des objets en mémoire
 - Se charge de garnir les attributs avec des valeurs (initialisation de l'état)
- **Destructeur**
 - Efface des objets en mémoire
 - *(Quasiment jamais utilisé dans les langages orientés objets modernes)*
- **Sélecteur (accesseur, "getter")**
 - Permet de consulter/lire les données d'un objet
 - Renvoie des informations sur l'état d'un objet
- **Modificateur (mutateur, "setter")**
 - Permet de modifier les données d'un objet
 - Change tout l'état ou une partie de l'état d'un objet