- Module 1 -
Bases de l'orienté objet et notation UML

1re Technologie de l'informatique

1re Sécurité des systèmes

1.1. Orienté objet et procédural

Qu'est ce que l'orienté objet ?

Paradigme de programmation qui correspond à une organisation du code en fonction des données.

Comparons la programmation procédurale à celle orientée objet (OO)

Programmation procédurale :

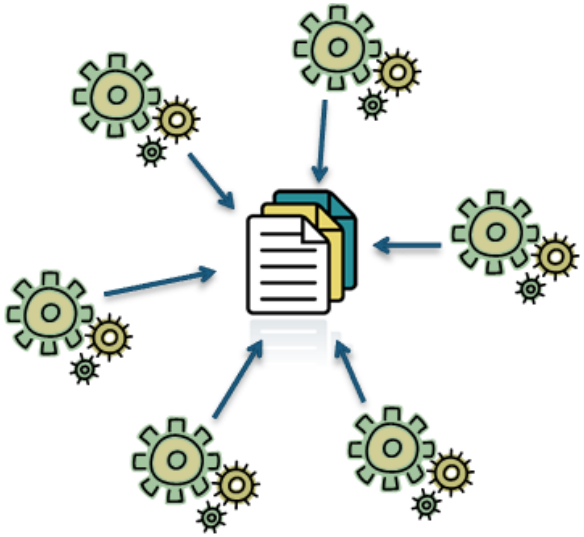
→ *on organise le code en fonction des actions à accomplir*

Programmation OO :

→ *on organise le code en fonction des (types de) données*

Programmation procédurale

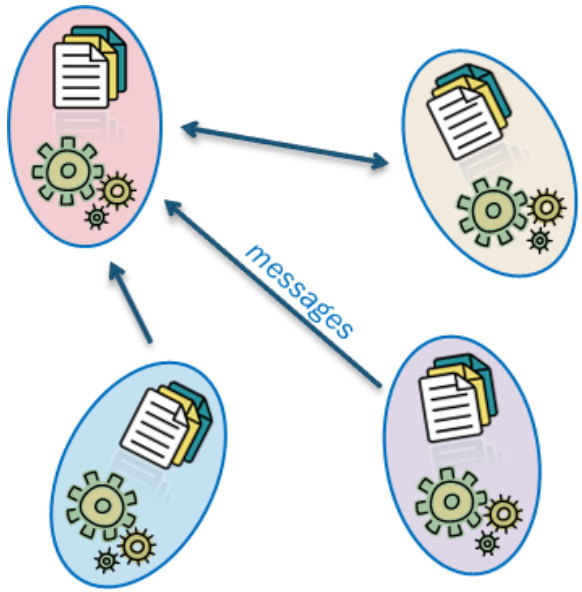
- En programmation procédurale…



- on définit des données "centrales" utilisées par tout le programme ;
- on définit des procédures/fonctions qui agissent sur ces données ;
- chaque procédure/fonction accomplit une tâche particulière : le code est découpé selon les actions/les fonctions désirées (découpe fonctionnelle).

Programmation OO

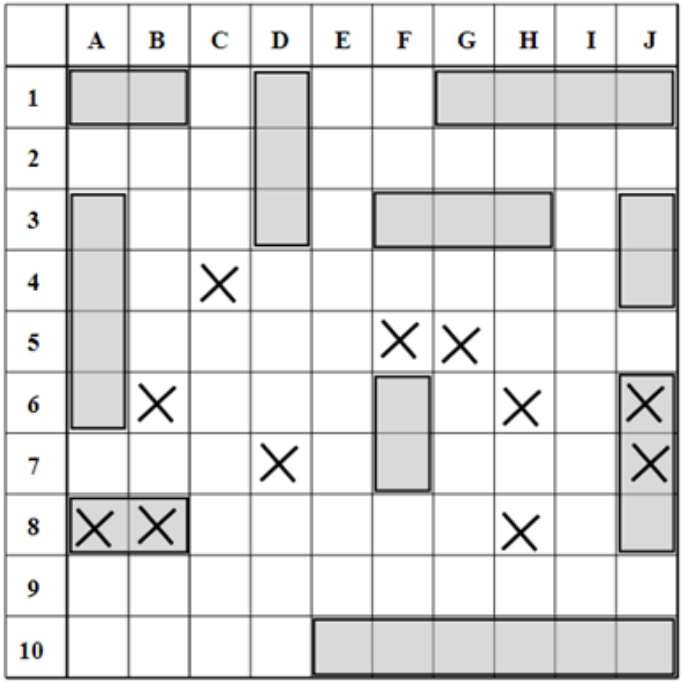
- En programmation OO…



  - on définit des "capsules" qui renferment une partie des données **et** le code qui travaille sur ces données ;
  - un objet contient donc à la fois des données et du code ;
  - ces "capsules" (= objets) peuvent s'envoyer des messages pour demander une action ou de l'information ;
  - le programme est découpé selon les objets / acteurs / types de données.

Exemple : touché-coulé

- deux joueurs, chacun ayant
- 10 navires
- placés sur une grille de 10×10
- dont certaines cases peuvent avoir été détruites par un missile



Approche procédurale

On définit des variables pour les données.

On définit des fonctions qui s'occupent des tâches.

Approche procédurale : constatations

- Le programme se découpe selon les **fonctionnalités**.

- Si on travaille à plusieurs,
  - on se met d'**accord sur la structure des données** puis
  - on peut se **répartir les modules** à construire (découpe fonctionnelle).

- Chaque module peut **accéder** à/modifier la plupart des **données** (sans restriction).

- Si on change la structure des données, il faut revisiter tout le code : le code est "**fortement couplé**" aux données.

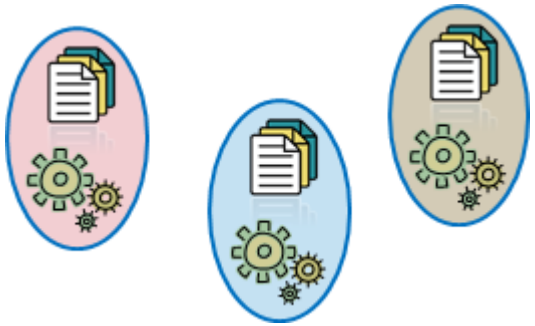
Approche OO

On identifie les (types d’) « acteurs ».

On leur associe des tâches (les acteurs peuvent faire appel les uns aux autres pour les réaliser).

Approche OO

Type d'objets :



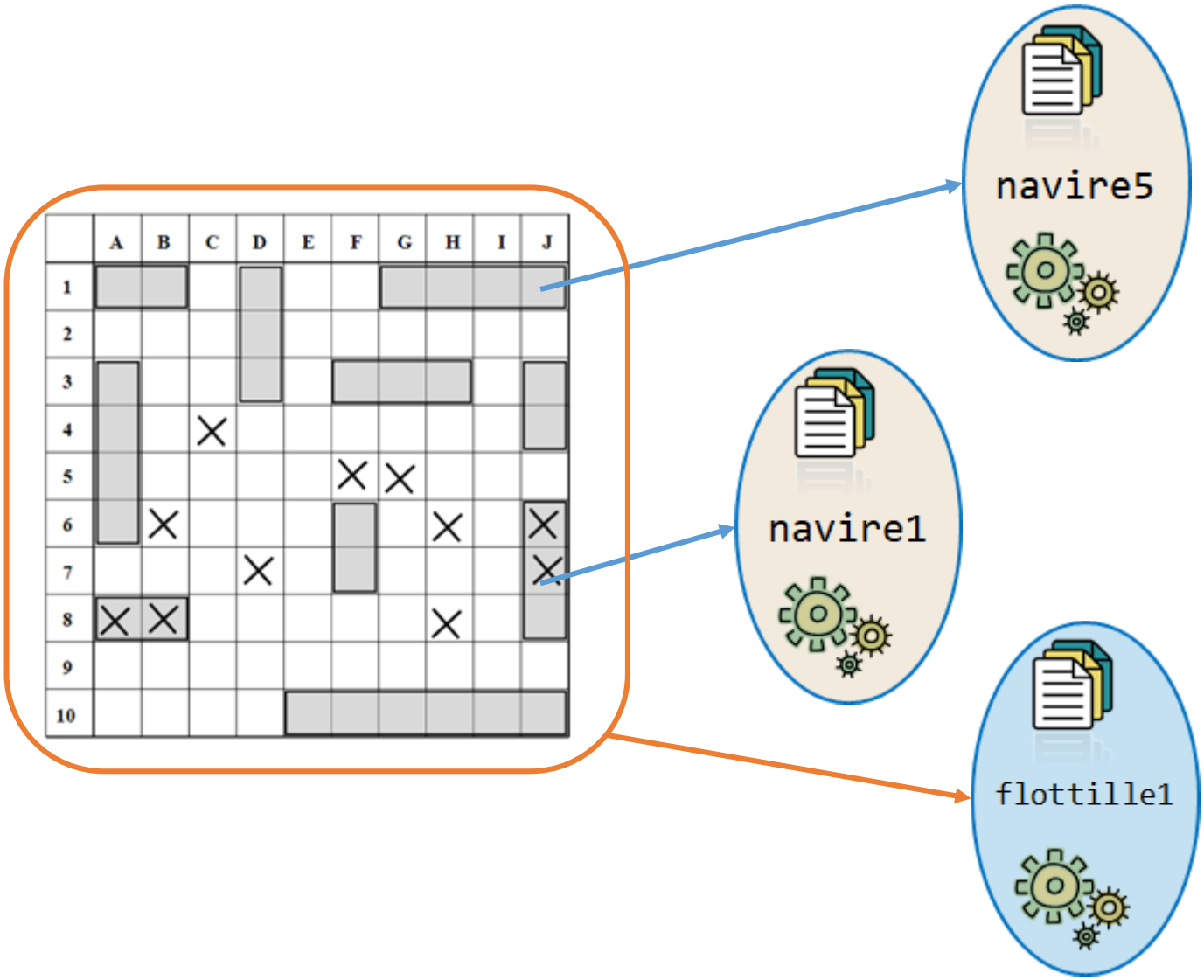
- Navire
  - = positionTête, direction, taille, casesDétruites
- Flottille
  - = liste de Navires
- Joueur
  - = nomJoueur, flottille

Approche OO

Actions associées :

- Navire
  - indiquer si le navire a été coulé / entièrement détruit
  - indiquer si une position donnée correspond à une case du navire
  - marquer une position comme détruite / gérer une attaque
  - afficher le navire sur la grille
- Flottille
  - afficher la grille et la position des navires
  - résoudre une attaque (dans l'eau, touché ou touché-coulé)
  - indiquer si une flottille a été entièrement détruite
- Joueur
  - effectuer un tour de jeu (demander la cible et gérer l'attaque)

Approche OO



Procédural vs OO

	Procédural	Orienté Objet
découpe		
Partage du travail		
Accès aux données		
Couplage		
Concepts		
Quantité de code		

1.2. Les objets et classes

Qu'est-ce qu'un objet ?

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

Un **objet**…

- est une entité **encapsulée**;
- regroupe à la fois :

- des **données** (= "valeurs"/"attributs") qui décrivent son état et
- 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**.

L'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

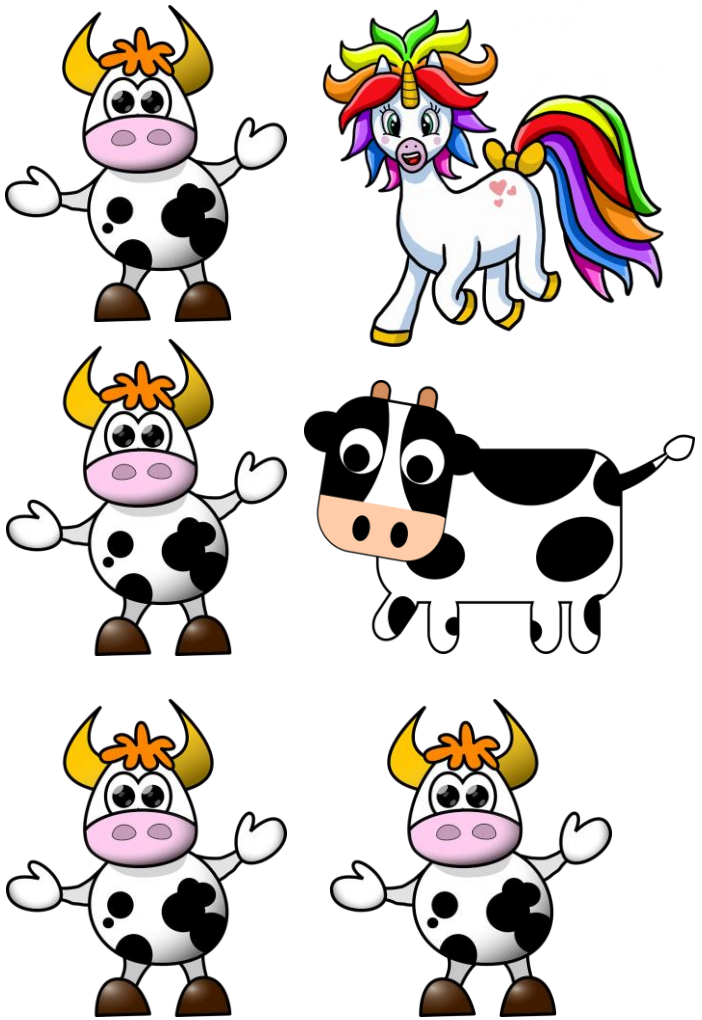
Qu'est ce qui va où?

- Un objet est composé d'un état et un comportement
- Un objet possède une partie privée et une partie publique

➔Qu'est ce qui va où?

	Partie Publique	Partie Privée
Etat	Généralement **vide** *(pour empêcher l'accès direct aux données)*	En règle générale (= 99% des cas), **les données sont privées**.
Comportement	Méthodes **accessibles depuis l'extérieur** *= comment répondre aux messages que d'autres objets peuvent envoyer*	Méthodes « **internes** » *= gestion interne de l'objet, pas directement accessible depuis l'extérieur*

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



Comportement, valeur et identité

ETAT

- valeurs des **ATTRIBUTS** à un moment donné
- valeur constante (exemple : nom) ou en évolution (exemple : âge)

COMPORTEMENT

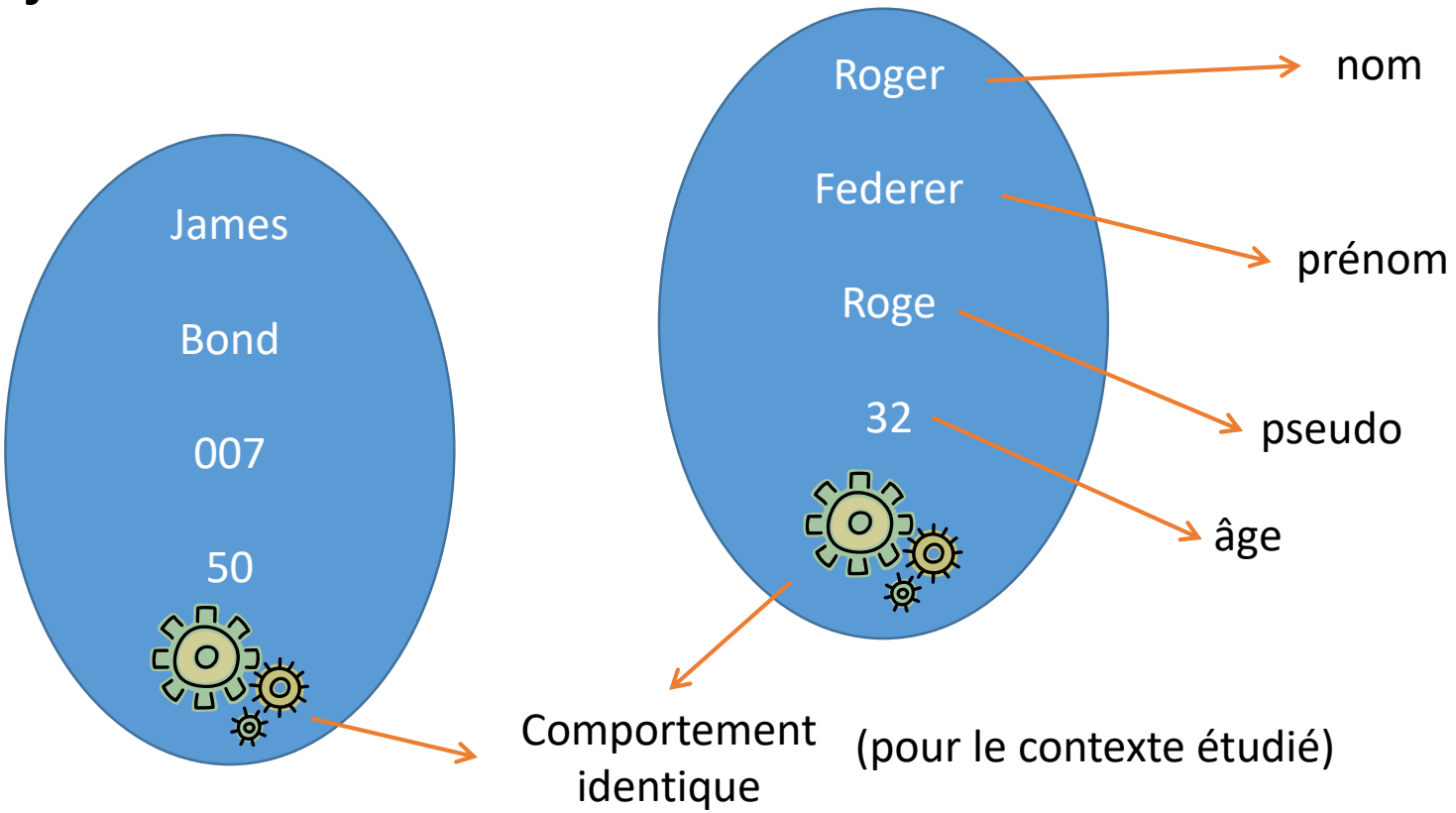
- ensemble de messages compréhensibles par l'objet
- "actions" déclenchées suite à la réception d'un message envoyé par un (autre) objet
- (ces actions peuvent modifier l'état !)

IDENTITE

- pour distinguer les copies (clones) d'objets identiques

Exemple d'objets

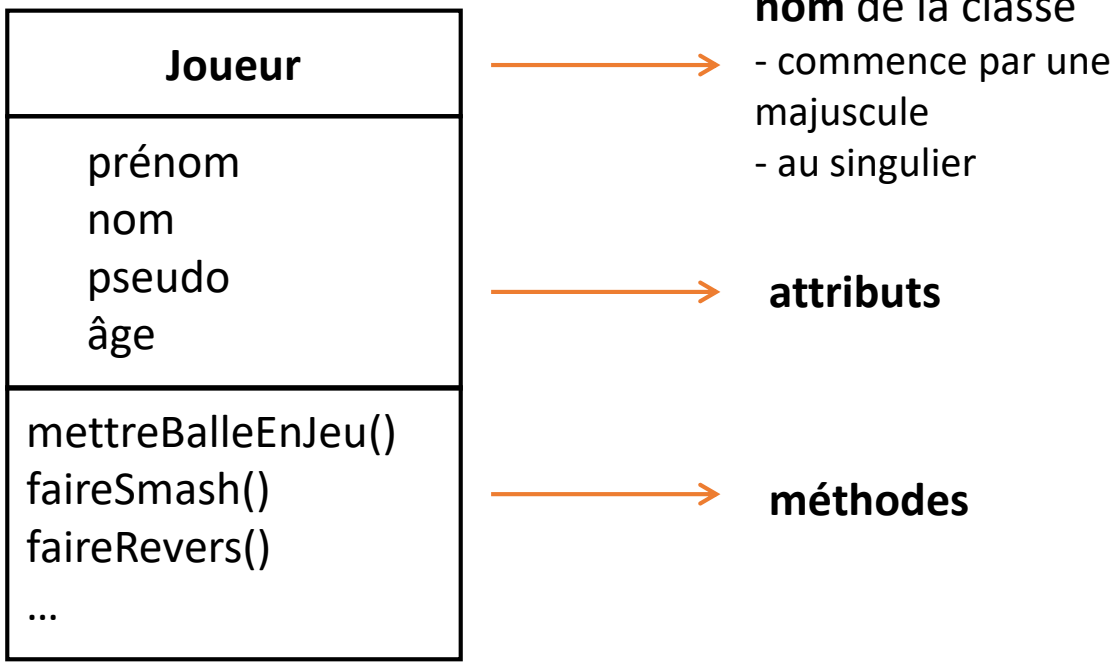
Deux joueurs de tennis :



Réflexion : les attributs de cet exemple sont-ils bien choisis ? Discutez selon les cas.

Exemple de classe

- Quels sont les points communs entre ces objets ?



Qu'est-ce qu'une classe ?

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

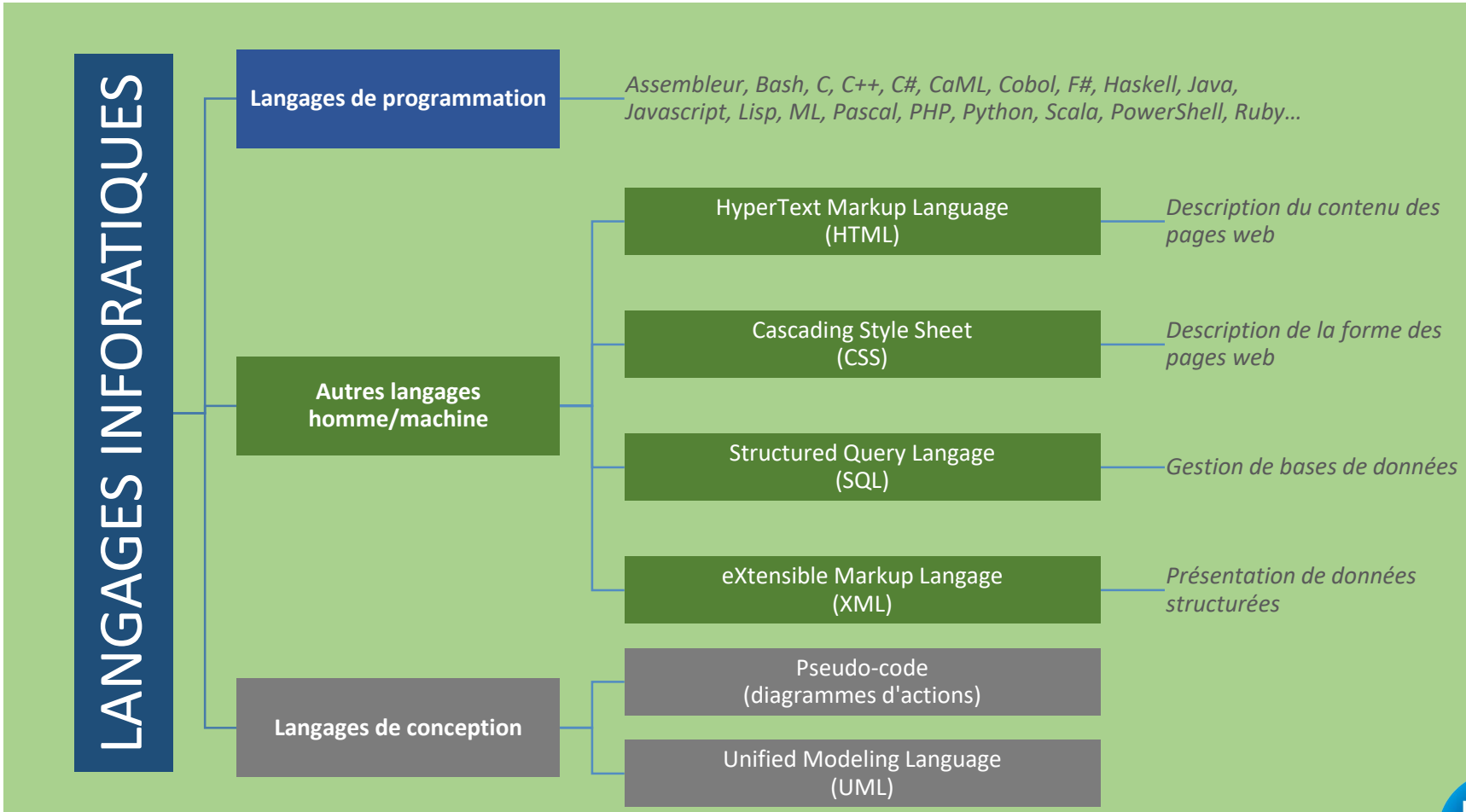
- Description des attributs et des méthodes de toute une famille d'objets
- Les objets d'une même classe auront des attributs de même type mais des valeurs (peut-être) différentes.

Instanciation :

- Création d'un objet à partir de sa classe
  - Donner des valeurs aux attributs : préciser son état
- On dit aussi :
  - « **Un objet est une instance de sa classe**. »

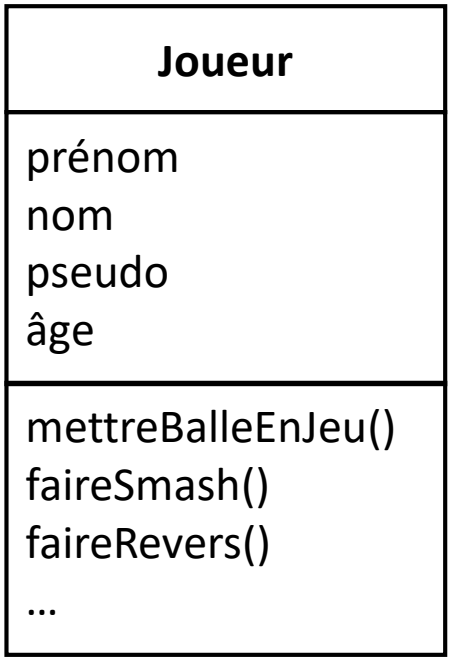
1.3. Notation UML

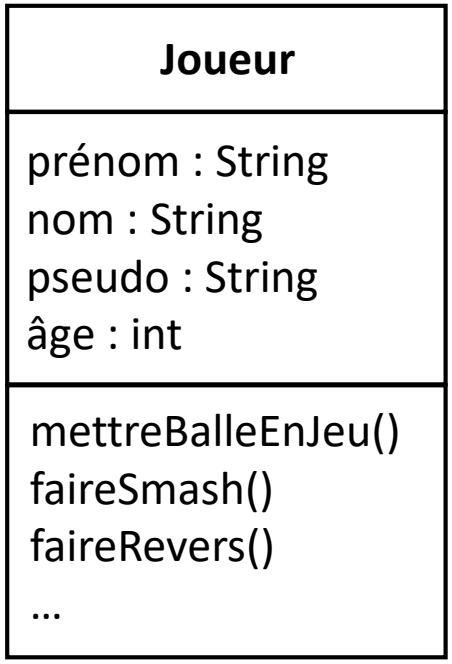
Les langages informatiques

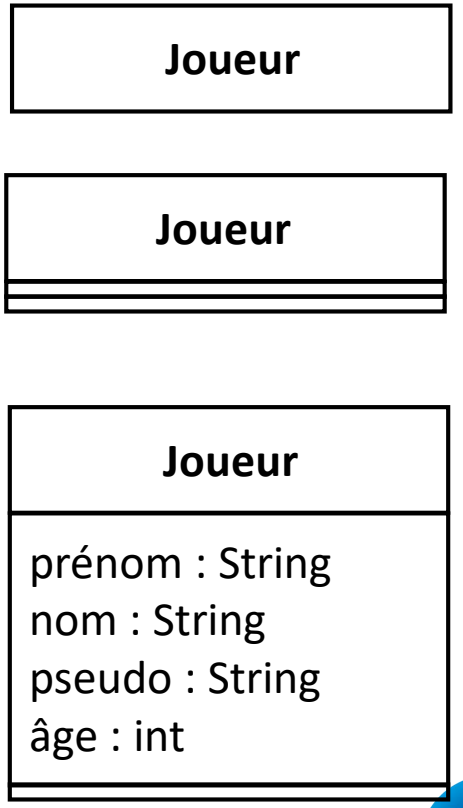


Représentation UML

UML permet différents niveaux de granularité (précision).

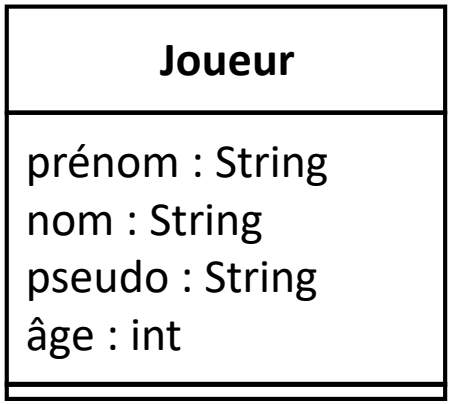






préciser les types… ou pas…
préciser les attributs/méthodes… ou pas…

Granularité



Principe de la granularité : ce n'est pas parce que quelque chose n'est pas spécifié qu'il n'existe pas…

Exemples :

- *On ne cite aucune méthode…*
  ça ne veut pas dire qu'il n'y en a pas !
- *On cite 3 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 !

Conventions entre nous :

- On ne cite que les attributs, pas les méthodes (sauf cas spécifiques).

Diagramme UML

= **Ensemble de classes interagissant entre elles.**

- Chaque **classe** définit les attributs et les méthodes qui lui sont propres.
- Si un objet doit **interagir** avec un autre objet, cela se fait sous la forme d'un message.
- Ce sont ces classes et ces interactions que le diagramme UML va représenter !

Attributs

Indique un attribut facultatif

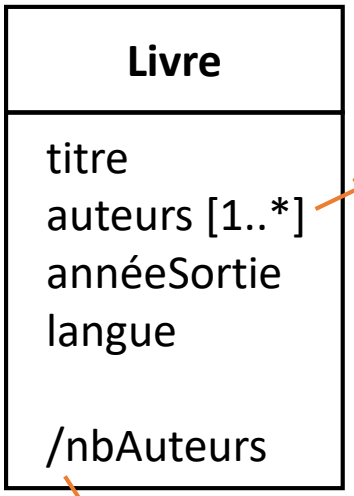
Valeurs simples

Convention pour le cours : indiquer explicitement les arguments booléens

Référence vers un objet

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

Attributs

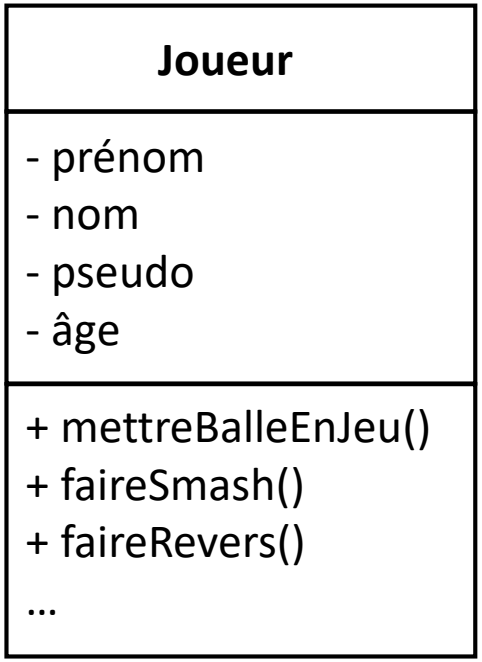


Indique un attribut multiple (liste ou autre séquence)

- *Minimum 1*
- *Maximum plusieurs*

Indique un attribut dérivable (calculable à partir des autres, généralement codé sous forme de méthode)

Attributs



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 existent également)

Méthodes

Chaque classe **définit ses propres méthodes**
= définir les réponses que les objets de la classe donneront lorsqu'ils recevront des messages.

- 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.

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

Méthodes

Constructeurs

- Création des objets en mémoire
- Se charger de mettre en place les données (initialisation de l'état)

Destructeurs

- Effacer des objets en mémoire
- (Quasiment jamais utilisés dans les langages orientés objet modernes)

Sélecteurs (accesseurs, "getters")

- Méthodes de consultation / lecture
- Retour de tout l'état ou d'une partie de l'état d'un objet

Modificateurs (mutateurs, "setters")

- Méthodes d'écriture / de modification
- Modifier tout l'état ou une partie de l'état d'un objet

1.4. Relation entre les classes

Association

Agrégation

Composition

Dépendance

Types de relations

Le diagramme (de classes) UML va présenter les classes d'un projet **et les relations qui les lient** en se basant sur certaines conventions de représentation.

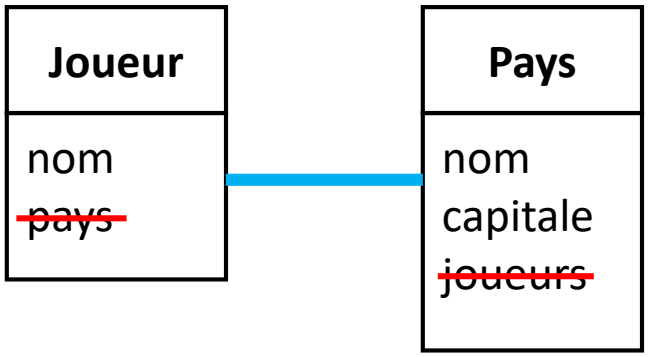
On va s'intéresser principalement à 5 types de relations :

- **Association** : un attribut est une instance d'une autre classe
  - **Agrégation** : association où une classe est plus importante que l'autre
    - **Composition** : association de type partie/tout ou composant/composite
- **Dépendance** : une classe utilise de temps en temps une autre
- **Héritage** : une classe est un cas particulier d'une autre classe (traité dans un chapitre suivant)

Association

Association entre deux classes

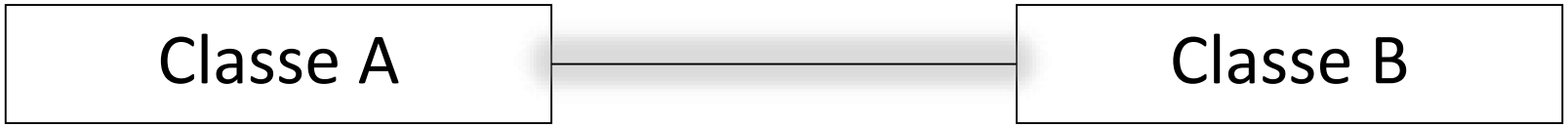
- (au moins) une des classes a un attribut instance de l'autre
- lien durable entre les objets des deux classes



- **Représentation en UML** : ligne pleine entre les classes.
- **Note** : on ne note pas dans les classes les attributs concernés par l'association (ils sont représentés par la ligne).

Association : notation

- **Notation** : ligne pleine

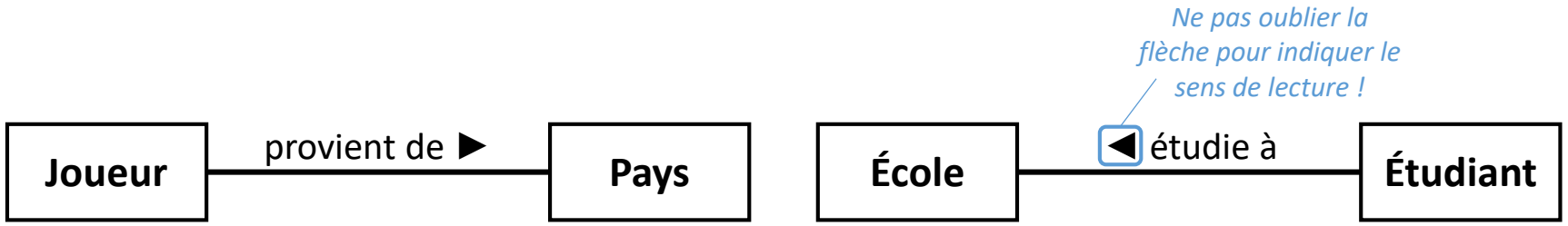


- **Signification** :
  - (A) contient une référence à (B) sous la forme d'un attribut et vice versa.

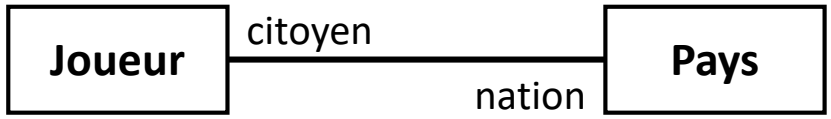
Association : décoration

Préciser la signification d'une relation en la **décorant/documentant**…

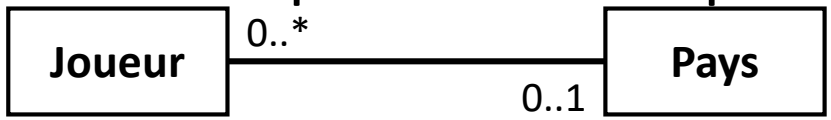
- en indiquant la nature de la relation (verbe + flèche)



- en indiquant les rôles des classes associées (noms)



- en indiquant les multiplicités/cardinalité.

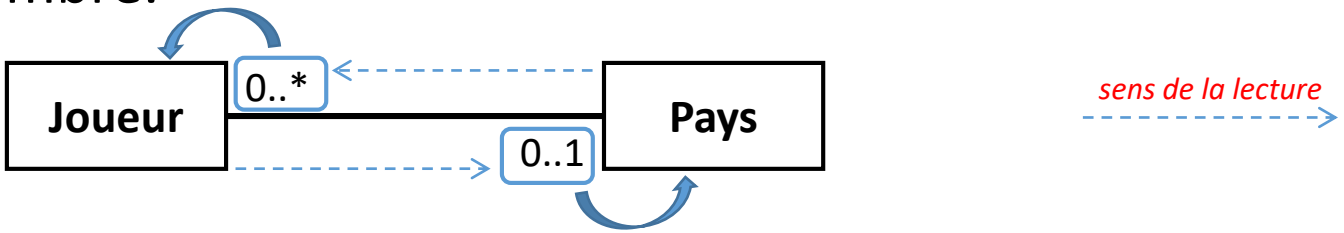


Association : cardinalité

Les **cardinalités** précisent le nombre d'objets pouvant participer à une relation.

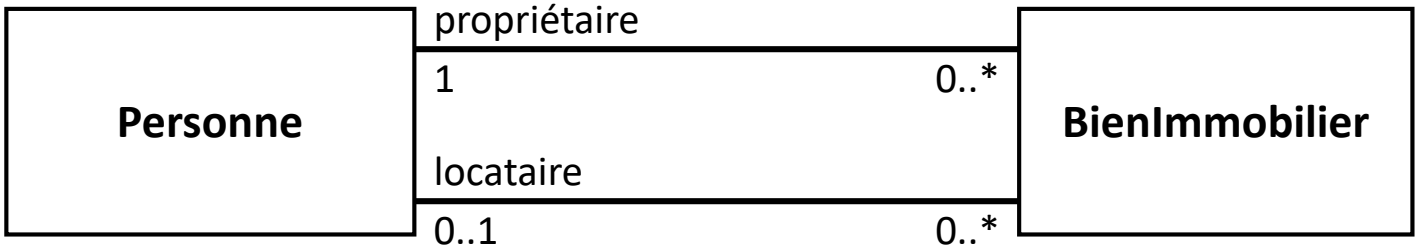
1	un et un seul
0..1	zéro ou un
1..*	un à plusieurs
0 ..* ou *	0 à plusieurs
M..N	de M à N (entiers naturels)

- Elles se placent à côté de la classe dont elles indiquent le nombre.



  - *Un joueur a entre 0 et 1 pays.*
  - *Un pays peut avoir n'importe quel nombre de joueurs.*

Association : exemple



Réflexions (qui montrent toute l'importance du contexte !)

- Si un propriétaire ne possède qu'un seul bien immobilier et le vend,
  conserve-t-on ses informations en mémoire ?
- Plusieurs personnes peuvent-elles posséder un bien en copropriété ?
- Un bien immobilier peut-il être loué par un groupe de locataires (collocation) ?

Agrégation

Cas particulier d'association assymétrique où une classe **fait partie** d'une autre

- Par exemple :

- Notation : ligne pleine avec losange du côté de l'agrégat

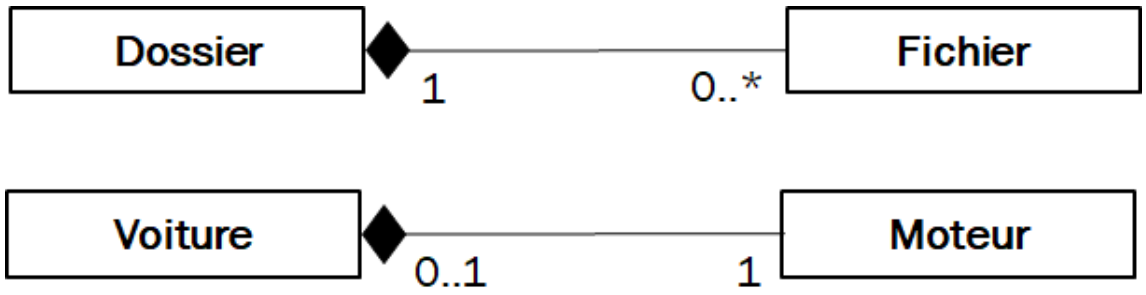
- Signification :
  - (A) et (B) sont en association, mais (A) joue un rôle plus important.
  - (B) pourrait être une partie / un esclave de (A).

Composition

Cas particulier d'agrégation où

- La multiplicité du composite <=1
- Si le composant meurt, le(s) composite(s) aussi

- Par exemple :

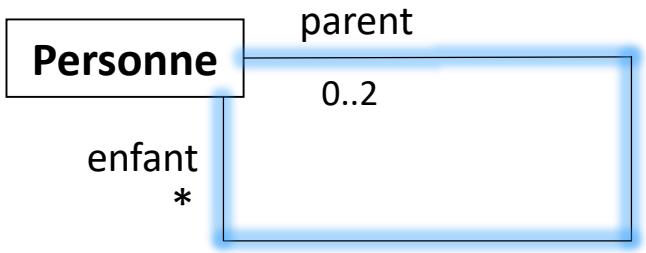


- Notation : ligne pleine avec losange plein du côté de l'agrégat

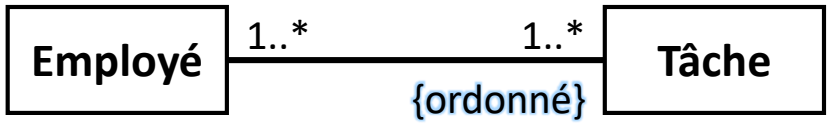
D’autres associations

Il existe plein d’autres associations…

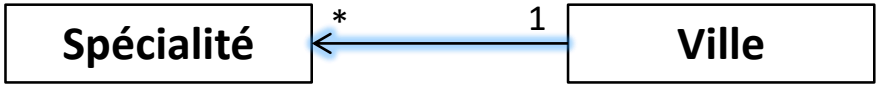
- réflexive



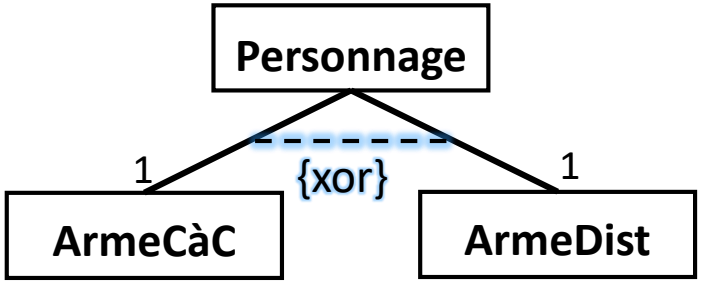
- ordonnée



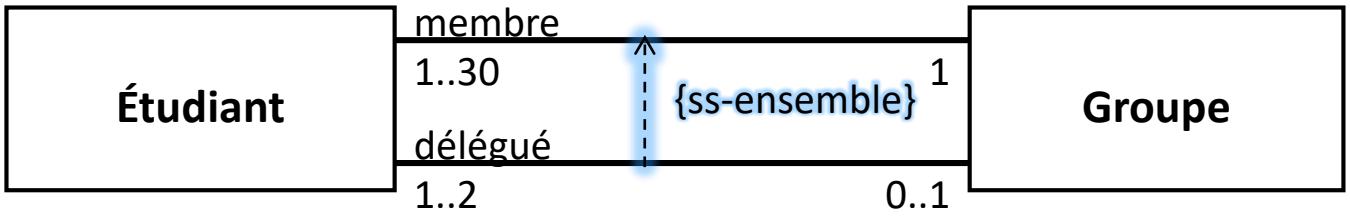
- unidirectionnelle



- disjointe



- sous-association



Dépendance

La classe A dépend/utilise la classe B si, dans son code, elle fait appel aux méthodes de la classe B.

→ Un élément A (le "client") dépend d'un élément B (le "fournisseur").

→ A utilise les services de B.

Exemple :

On doit pouvoir imprimer une facture. Dans la classe Facture, on trouve une méthode imprime(Imprimante imp) qui utilise les services de la classe Imprimante.

Contrairement à l'association, la dépendance est directionnelle.
Contrairement à l'association, la dépendance décrit un lien temporaire.

Dépendance

- Notation : flèche en pointillés

- Signification :
  - (Client) dépend / a besoin de / utilise (Fournisseur)
  - Il s'agit d'une relation temporaire, brève, pas permanente.
  - (Client) n'a pas d'attribut de type (Fournisseur).
  - (Fournisseur) est utilisé comme argument/type de retour d'une méthode de (Client).
  - Si (Fournisseur) est modifiée, (Client) devra peut-être changer

1.5. Modélisation

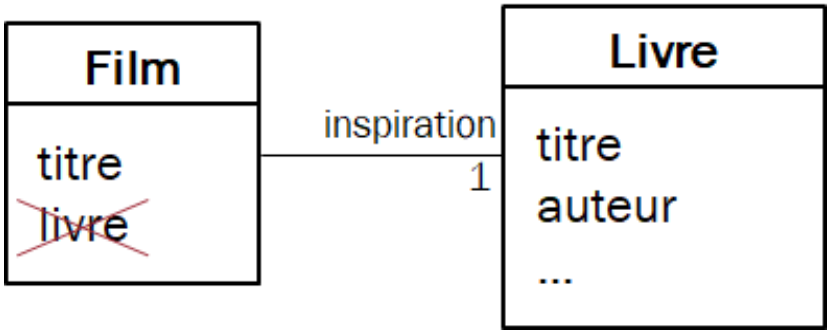
Modélisation = art (subjectif), pas science !

➔ Le contexte est très important.

(Étape 1) Exemple de départ
un film est inspiré d'un livre.

Modélisation = art (subjectif), pas science !

(Étape 2) On veut retenir l'auteur du livre en question.
Quelle solution est meilleure ?



(Étape 3) Et si certains films sont inspirés de plusieurs livres ?

Modélisation = art (subjectif), pas science !

« *Un bon modèle n'est pas un modèle où on ne peut plus rien ajouter mais un modèle où on ne peut plus rien enlever* »
(A. Saint-Exupéry)

Importance du choix :

- Qu'est-ce qui doit être précisé dans le modèle ?
- Que laisse-t-on à l'imagination / la décision du programmeur ?
- Quel niveau de détail (granularité) adopter dans le diagramme ?

➔ **importance de <u>bien lire l'énoncé</u> !!!**

Danger : suivre sa propre pensée plutôt que l'énoncé !!!
Dans un cas réel, vous auriez l'occasion de discuter avec le client pour clarifier les choses ou lui proposer des alternatives que vous pensez préférables… dans le cas des exercices du cours, ce n'est pas possible. Il faut donc suivre les idées <u>de l'énoncé</u> et <u>pas les vôtres</u> !

Exemple : Abonnements

- Un film a un titre, une durée, une année de sortie et une liste d'acteurs qui jouent dedans. Un acteur a un nom, un prénom et une date de naissance.
- Une série a un nom, une année de début et éventuellement une année de fin et est composée de saisons dans un certain ordre.
- Chaque saison a un numéro, une année de début et éventuellement une année de fin et est composée d'épisodes (qui sont en fait des films) dans un certain ordre.

Exemple : Abonnements

- Un client a un nom, un prénom et une adresse mail. Il peut prendre plusieurs abonnements pour lesquels on va retenir la date de début et la date de fin ainsi que leur prix et les films et/ou séries qu'ils couvrent
- Chaque abonnement peut être pris par plusieurs personnes mais on ne veut pas spécialement savoir retrouver les personnes à partir de l'abonnement.
- Lors de l'inscription, un client peut faire une sélection soit de 1 à 3 films soit de 1 à 3 acteurs qu'il aime.

Source

- Syllabus PPOO IG1