Conception UML et diagrammes de classes

Benoît DARTIES

Mail : benoit.darties@umontpellier.fr

Web : https://benoit.darties.fr

Année Universitaire 2020 - 2021

UML : présentation, application

Conception d'un programme informatique

Le concept de modélisation :

- Conception d'un modèle : mathématique, géométrique, 3D, mécaniste,...
- Tout développement (informatique) propre débute par la modélisation

En informatique, un ***modèle*** a pour objectifs de :

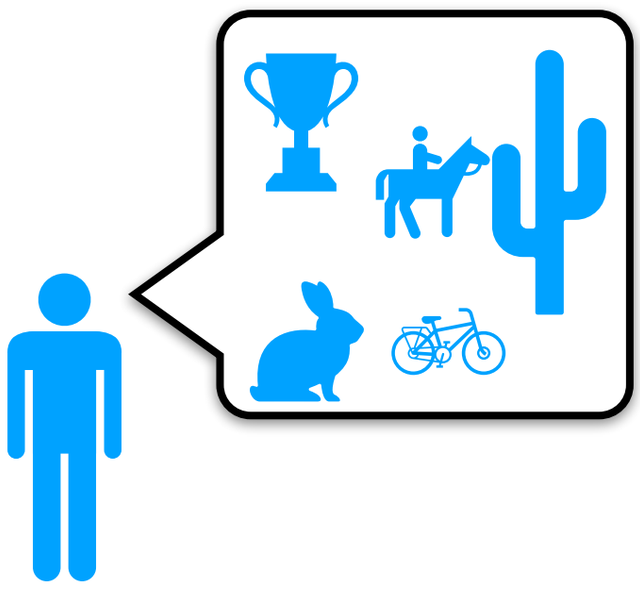
- Aider à la conception de l'architecture globale d'un système
- Présenter la structure et les dynamiques d'éléments logiciels
- Organiser des informations dans le programme
- Inutile pour des petites applications, essentiel pour des gros projets

- ***Modéliser, c'est décrire ce que l'on veut faire dans un langage commun.***
- ***C'est proposer une représentation abstraite d'une entité réelle***

Parler le même langage

Modélisation et informatique :

- Pas toujours évident de parler le même langage de modélisation
- ***UML : Universal Modeling Langage***
- Langage de modélisation référence
- Production d'un schéma formel depuis un énoncé en langage naturel
- A partir de ce schémas : production du code informatique associé



Modélisation UML

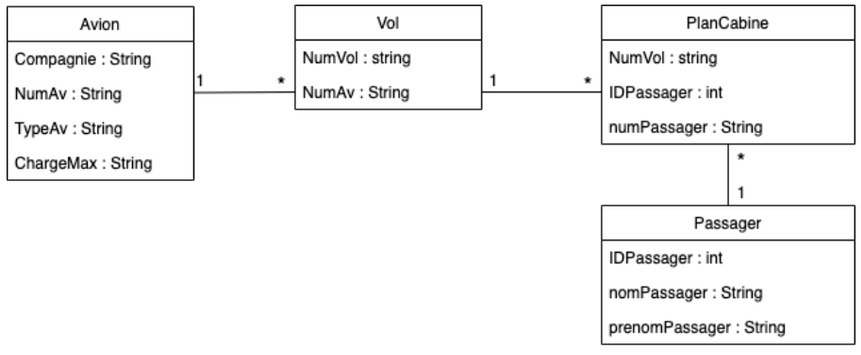


Diagramme UML

Modélisation à partir d'un énoncé

Le Hellfest est un festival, qui s'il n'avait pas été annulé pour cause de Covid, aurait rassemblé cette année encore de nombreux artistes de la scène Métal pendant 3 jours à Clisson au milieu de dizaines de milliers de festivaliers. L'un des points critiques de die festival est l'approvisionnement en bières. Soucieux de voir son festival se développer et corriger son principal défaut qui est la qualité de la bière, le fondateur du festival, Ben Barbaud, entreprend une vase opération de remise à niveau de la distribution de bière dans son festival :
Les points de distribution sont découpés en deux grandes familles : les buvette et les bars. Les buvettes ne proposent qu'un nombre limité de bières, exclusivement en pression et n'offrent aucune place assise, au mieux deux ou trois tables hautes sur lesquelles s'accouder rapidement. Les bars eux proposent une infrastructure plus complète, avec tables et places assises, et surtout la possibilité de servir des bières pression mais aussi des bières en bouteille (pour des raisons de sécurité, ces bières seront versées dans un gobelet). Une biere possède un nom et un degré d'alcool. Elle peut être vendue aux Hellfest sous forme de bouteille, ou de fut 30L.
Chaque point de distribution possède sa propre carte de bières, afin de faire tourner les références. Une bière est fournie par une brasserie, qui peut bien entendu fournir une ou plusieurs références au Hell Fest; une bière n'est fournie que par une référence, mais elle peut être fournie sous forme de fut pour les bières pression, ou sous forme de bouteille pour les bières bouteille. Comme les points de distribution sont gérés au niveau du festival, chaque point pratique les mêmes tarifs que les autres, pour le même type de produit.

Voici de plus 6 exemples de questions auxquelles votre modèle doit répondre :
- Quelles sont les bières fournies par la 'brasserie des Garrigues' ?
- Combien de références bières sont disponibles sur le bar "La Cathédrale"
- Combien de futs de Duvet Triple reste-il en stock dans la buvette "La porte de l'enfer" ?
- Quel bar possède le plus de références de bières?
- quel est le prix d'achat d'une bouteille 33 cl de "Duvel Triple Hop" ?
- quel est le nombre moyen de places assises des bars qui proposent de la Barbar?

Enoncé

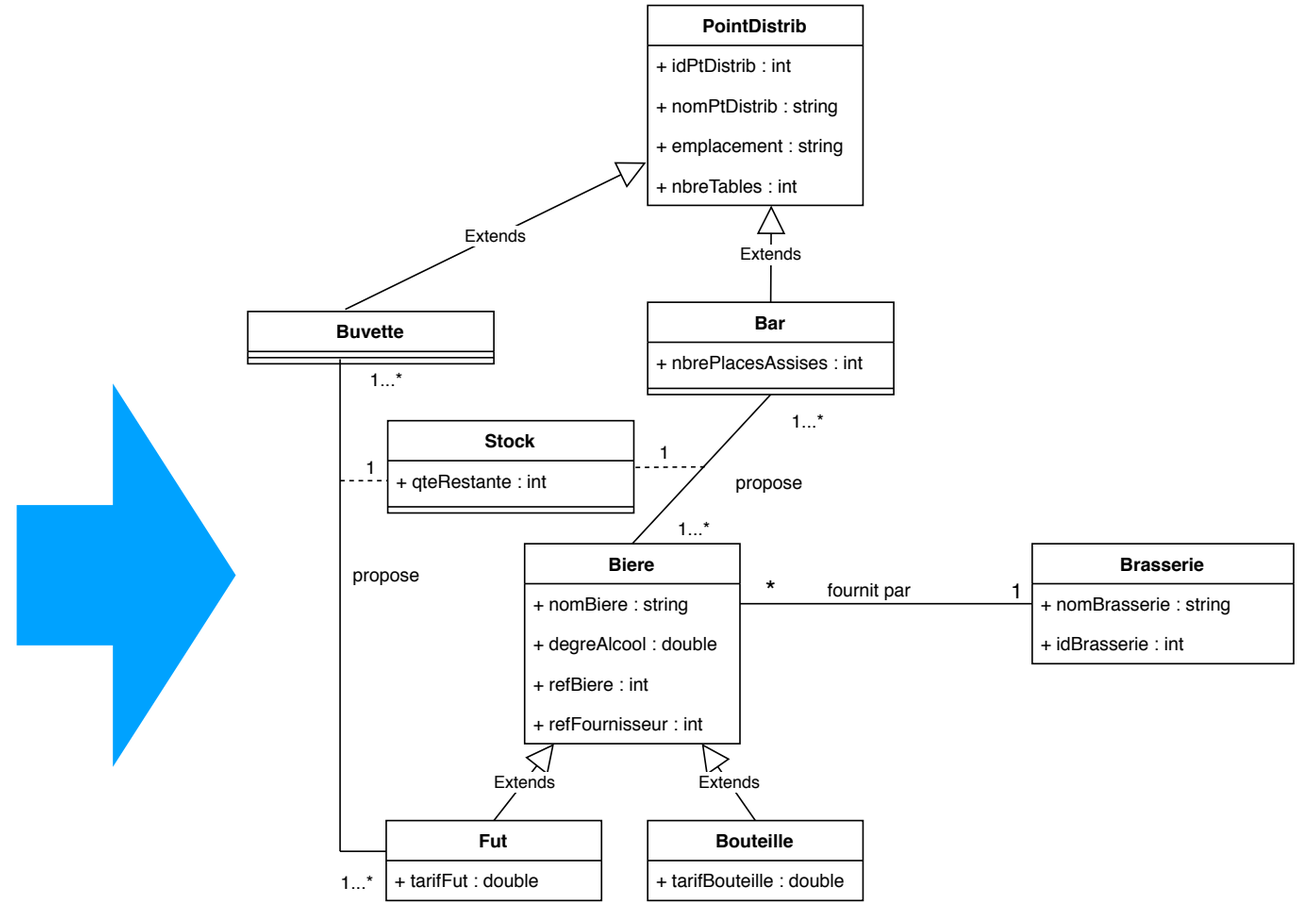


Diagramme UML

Modélisation et informatique

Enoncé en langage naturel

Diagramme UML (de classes)

Base de données relationnelles

Schémas relationnel

Schéma relationnel normalisé

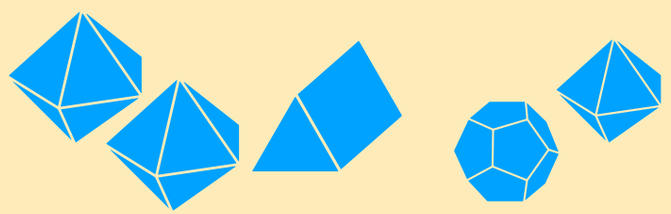
Base de données

programmation orientée objets

Diagramme de classes orienté programmation

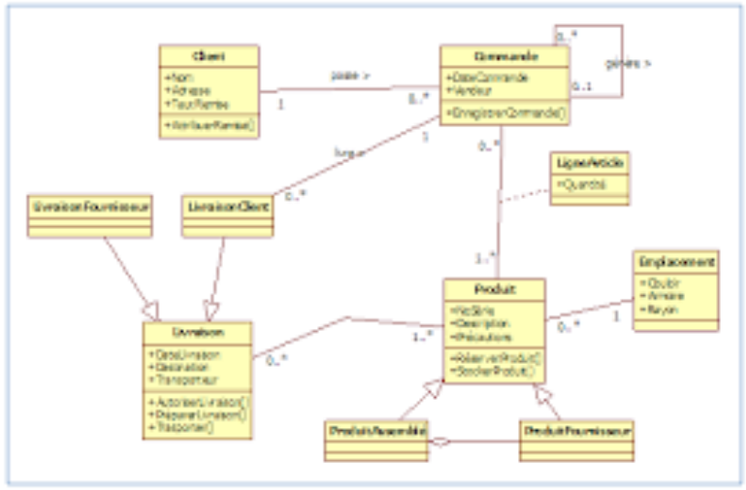
Algoritmes

Programme utilisant le paradigme des objets



Classes et objets

premiers diagrammes



Rappels de programmation

Types de données "primitifs" :

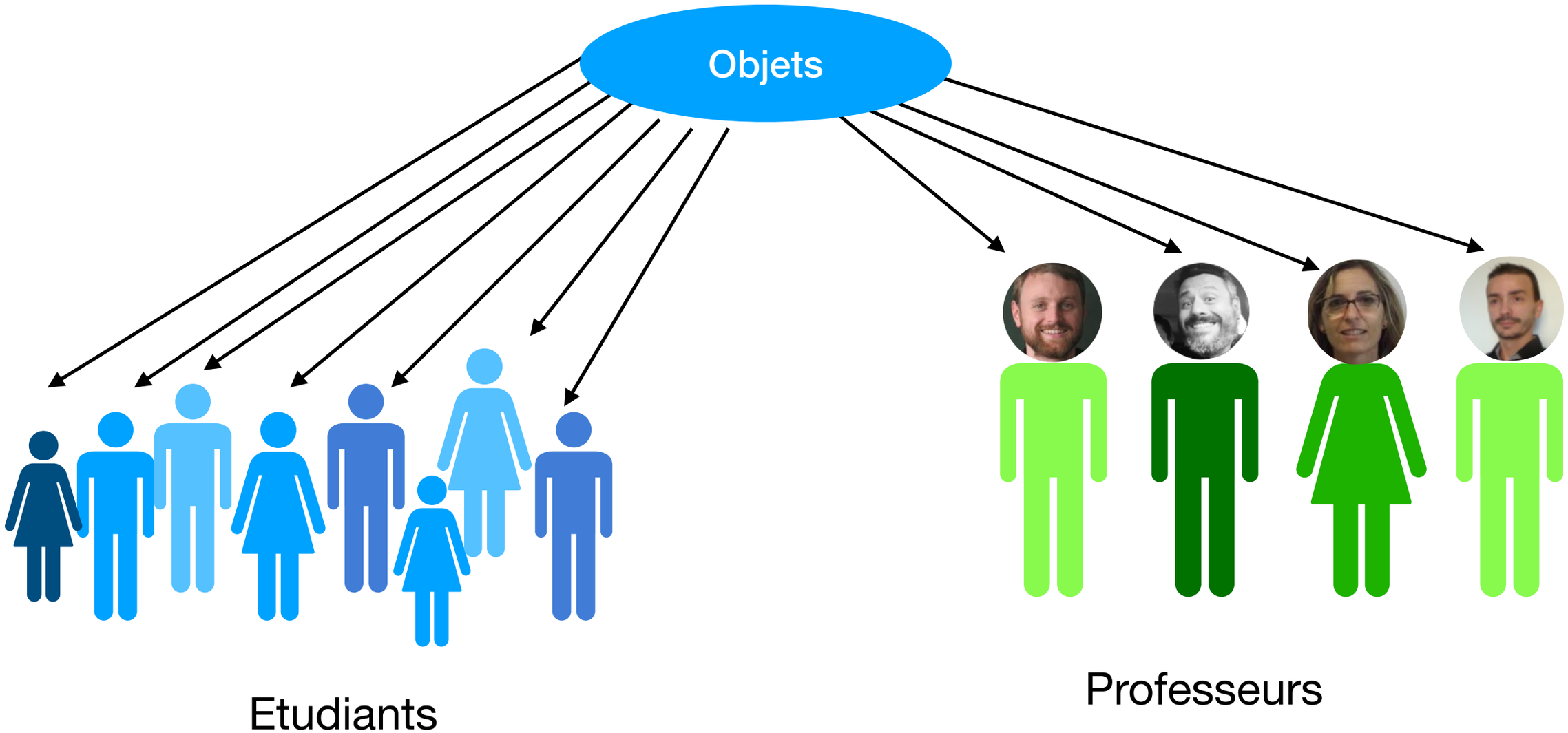
- En programmation on manipule des données qui ont un type
- Ce type peut être implicite (ex : en python, php) ou explicite (java / C / C++)

Type naturel	Représentation informatique	Ex : en java
Entiers	int	int a=5;
Chaines de caractères	String	String b = "hello";
Réels	float / double	float c=5.6;
booléens	bool	boolean d = true;
Dates	date	(compliqué, hors cours)
...	....	...

- Pour bien concevoir un programme, il est nécessaire de préciser le type de données que stockent les variables.

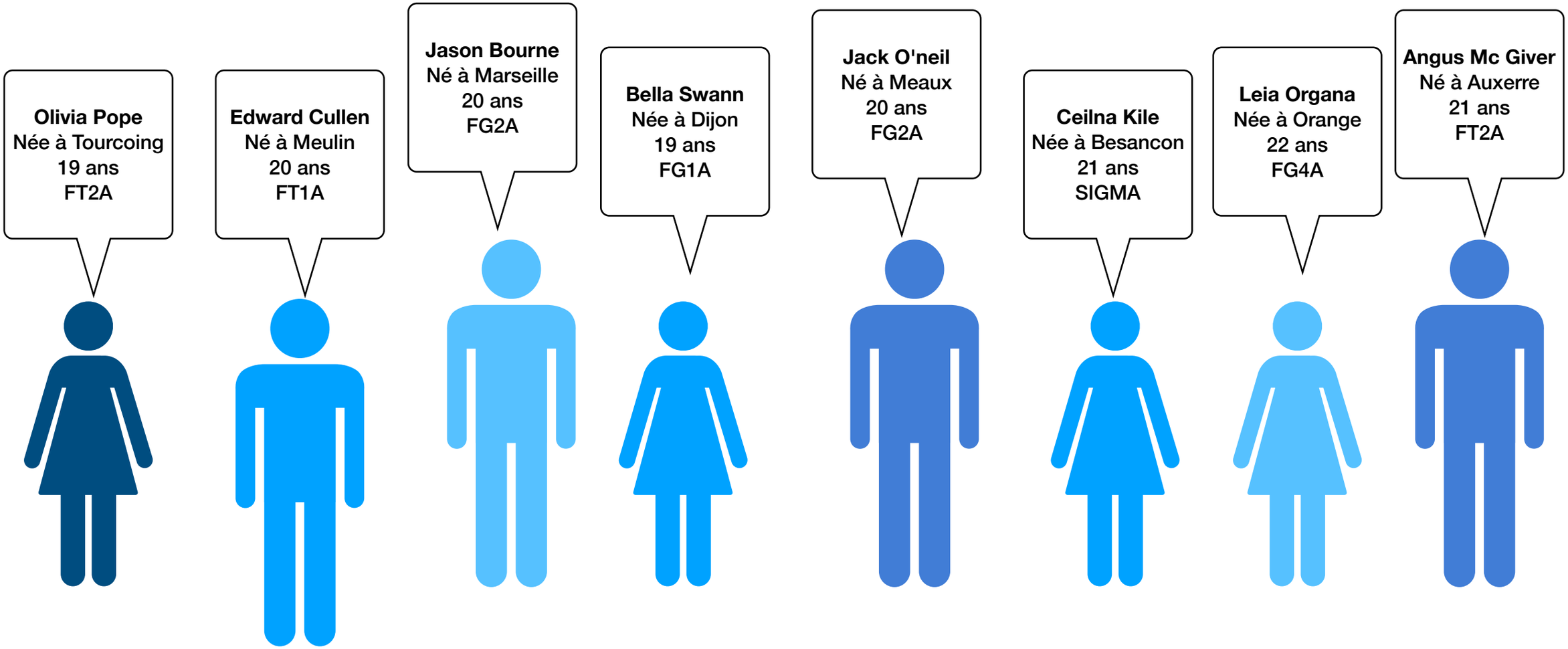
Un premier exemple

Vous souhaitez faire un logiciel (simplifié) de gestion d'école(s)



Un premier exemple

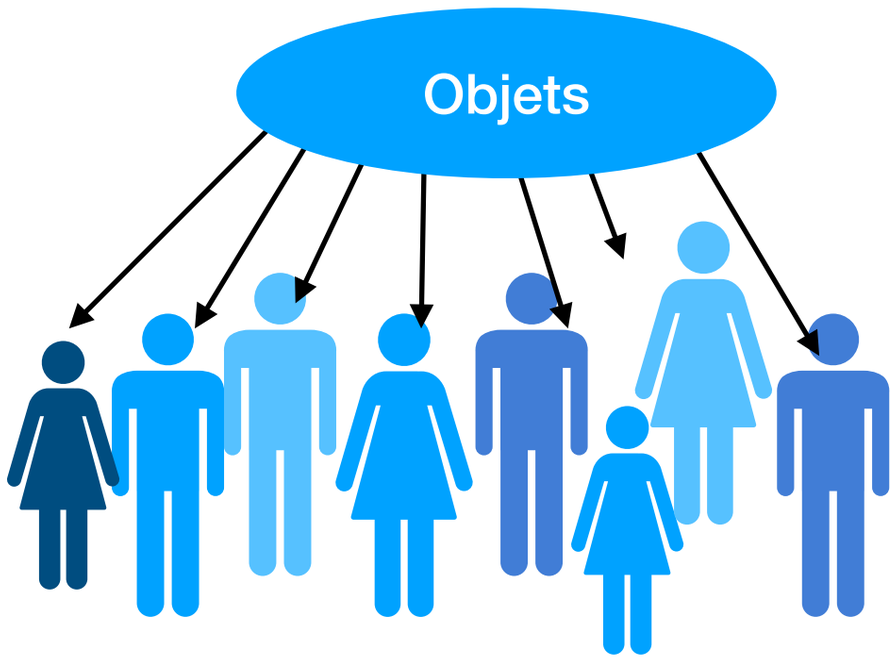
Chaque 'objet' Etudiant possède des attributs (ou propriétés) qui le caractérise : un nom, une ville de naissance, un âge, une promo ...



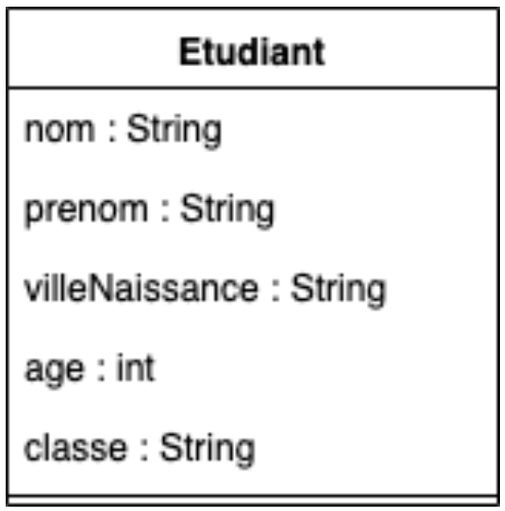
Etudiants

Un premier exemple

- **Les objets ont une existence**, des valeurs propres dans le programme
- **La classe est la représentation abstraite** des objets de même type
  - "moule à objets" : contient la liste des propriétés que les objets possèdent



Etudiants (objets)



Etudiants (classe)

Objectif : proposer des classes pertinentes, qui modélisent efficacement les objets qu'on va vouloir utiliser par la suite

Classe versus Objet

Une classe est un modèle

- Elle n'a pas de réelle existence : elle ne prend pas de mémoire
- C'est le modèle qui permet de créer des objets de même type

Un objet est la vraie entité manipulée

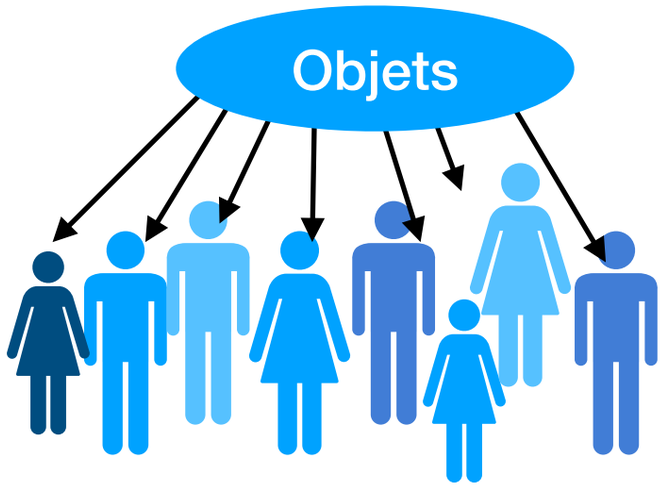
- Existe vraiment : prend de la place en mémoire.
- A des attributs qui peuvent contenir des valeurs.
- Chaque objet a ses propres valeurs.

Tout l'art de la programmation orientée objet, c'est:

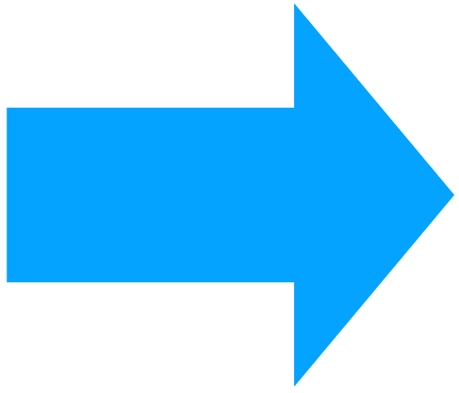
- ***bien savoir définir des classes (partie UML),***
- puis créer des objets à partir de ces classes
- Et les faire interagir, à la manière d'un scénario

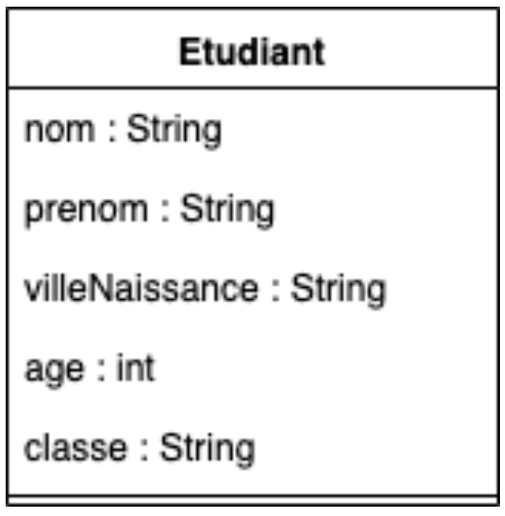
Une modélisation n'est jamais unique

Challenge : trouver la meilleure (?) modélisation



Etudiants (objets)

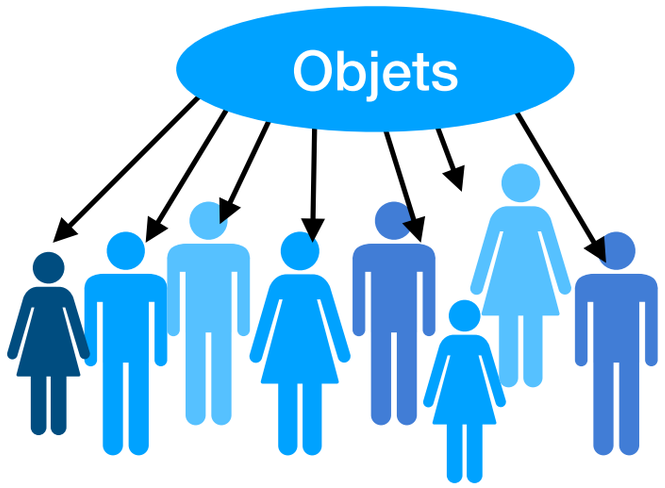




Etudiants (classe)

Une modélisation n'est jamais unique

Challenge : trouver la meilleure (?) modélisation



Etudiants (objets)

Etudiant
nom : String
prenom : String
villeNaissance : String
age : int
classe : String

Etudiants (classe)

Etudiant
nom prenom : String
villeNaissance : String
dateNaisssance : Date
classe : String

Etudiants (classe)

Etudiant
numEtudiant : int
nom : String
prenom : String
sexe : {M,F}
villeNaissance : String
dateNaisssance : Date
adresseResidence : String
CPResidence : int
villeResidence : String
taille : int
poids : int
classe : String

Etudiants (classe)

Doit répondre aux fonction-nalités attendues du projet

Un premier exemple

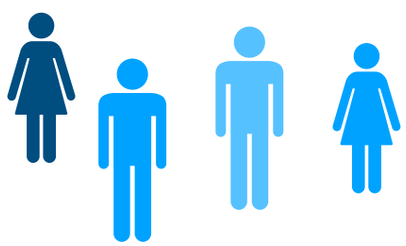
Les classes que l'on peut considérer seraient les suivantes :

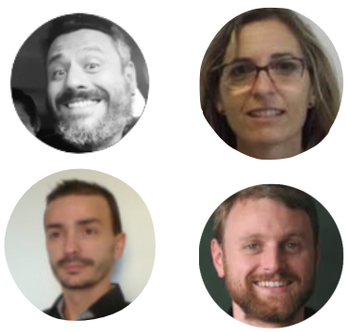
Ecole
nom : String
adresse : String
CP : int
Classe : String

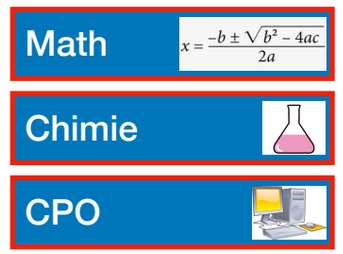
Etudiant
nom : String
prenom : String
dateNaissance : Date
Classe : String

Enseignant
nom : String
prenom : String
dateNaissance : Date
grade : String
salaireAnnuel : int

Cours
nom : String
nbCredits : int



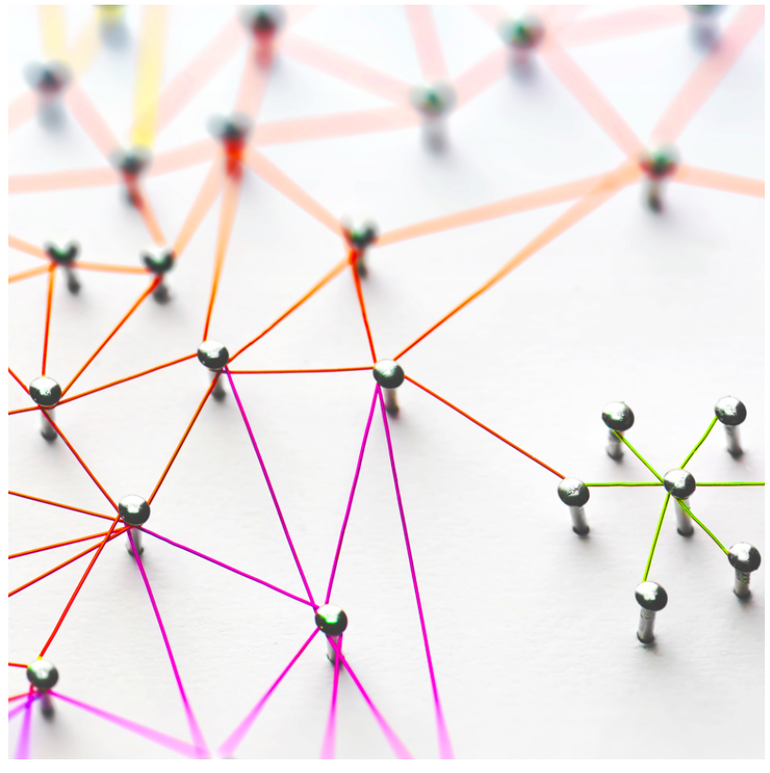




- Il y a des relations entre les objets de classes différentes : relation entre classes

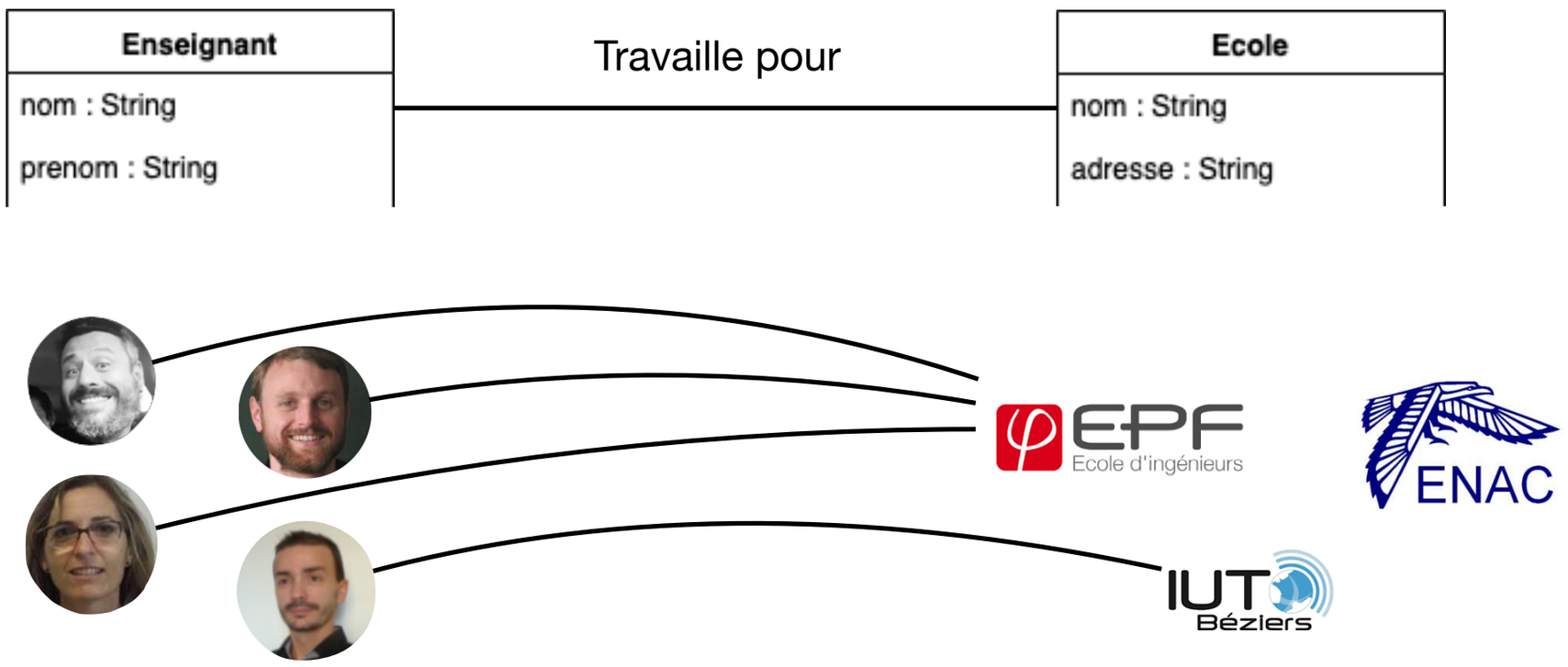
Relations entre classes

types d'associations



Association simple

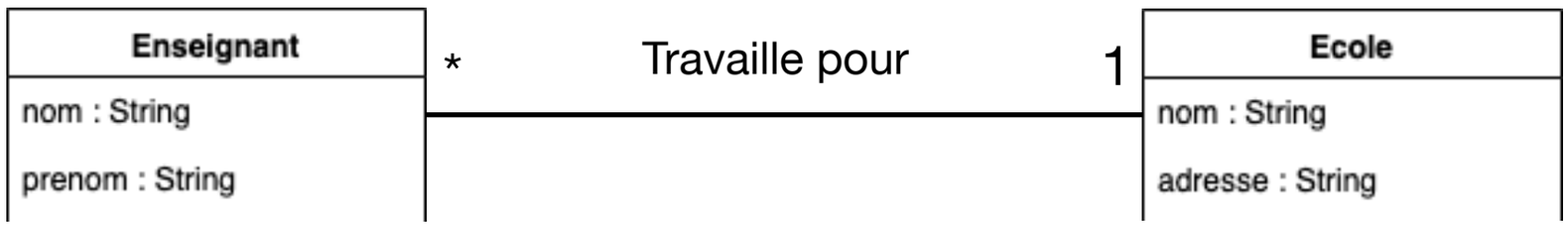
Associations simples (bidirectionnelles) :



- Importance de mettre des cardinalités sur les relations
  - Combien d'enseignants travaillent dans une même école?
  - Dans combien d'écoles travaille un même enseignant ?

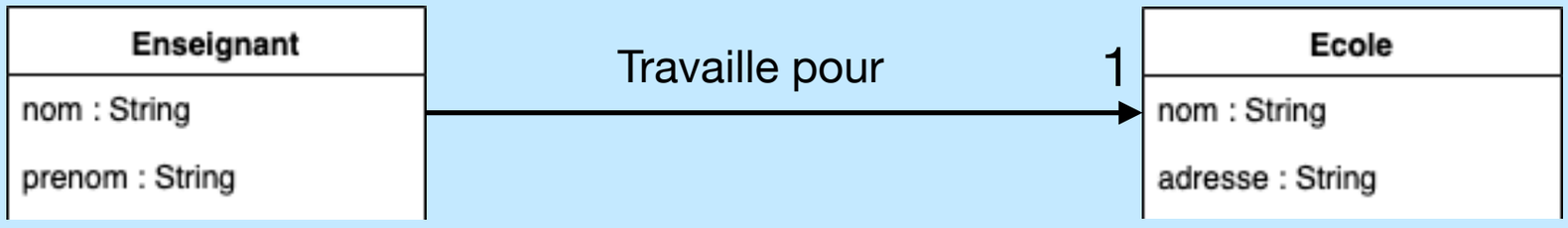
Association simple

Associations simples (bidirectionnelles) :

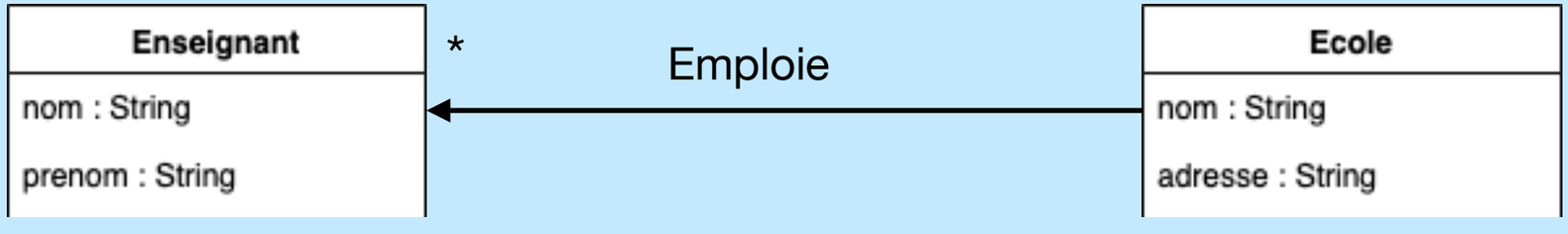


- **Importance des cardinalités : * et 1**

- Chaque enseignant travaille pour 1 école (exactement)



- Chaque école emploie un nombre indéterminé (plusieurs) enseignants



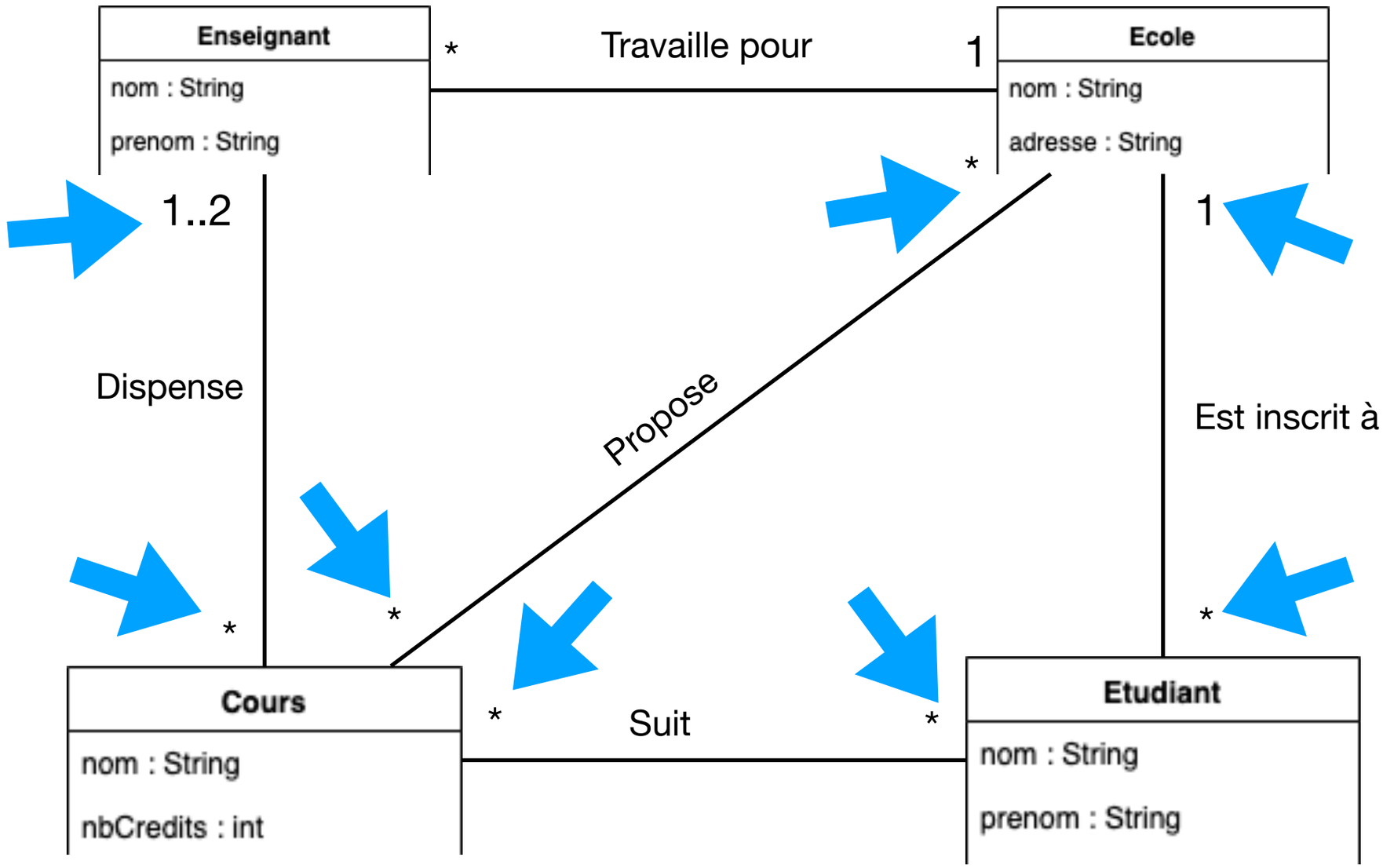
Cardinalités des relations

On peut utiliser les cardinalices suivantes sur les relations

Cardinalite	Explication
1	Un seul élément
N	Exactement N éléments (N>0)
*	De zéro à plusieurs
N..M	De N à M éléments (M > N >= 0)
N .. *	De N à plusieurs (N > 0)

- la cardinalité indique le nombre d'objets d'une classe pouvant être associés à **un seul objet** d'une autre classe

Poser les cardinalités sur un schéma



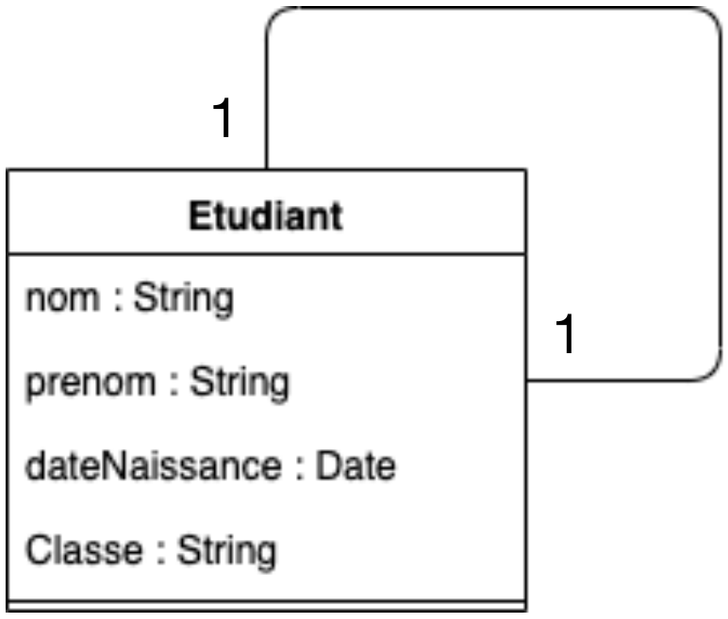
Relations réflexives

Des objets peuvent être en relation avec :

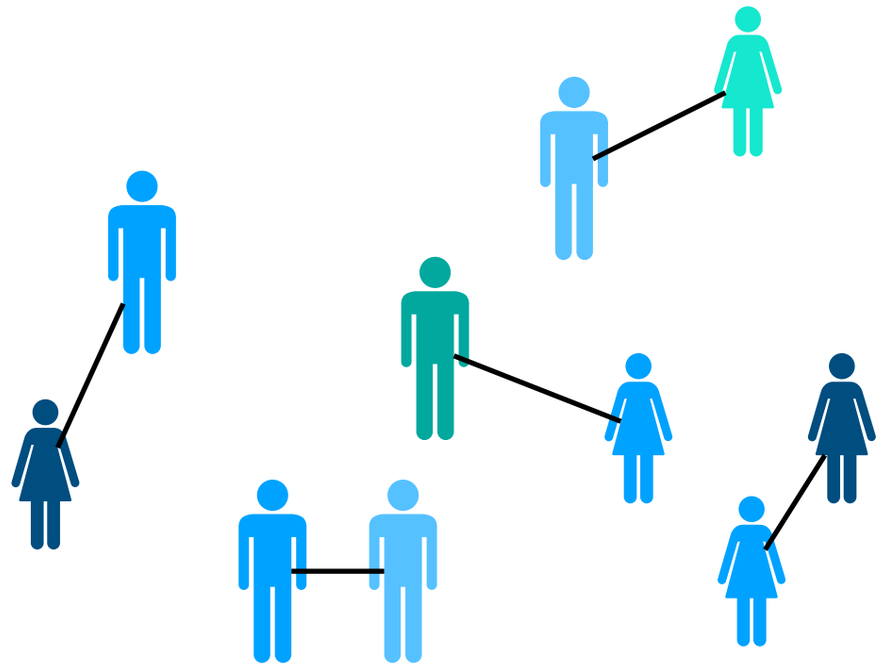
- Des objets d'autres classes : relation simple
- Mais aussi des objets issus de la même classe : **relation réflexive**

Exemple : représenter les relations entre binômes

- Chaque étudiant A est le binôme d'un autre B : il est donc impliqué dans la relation de binôme de B. **Association réflexive 1 à 1**



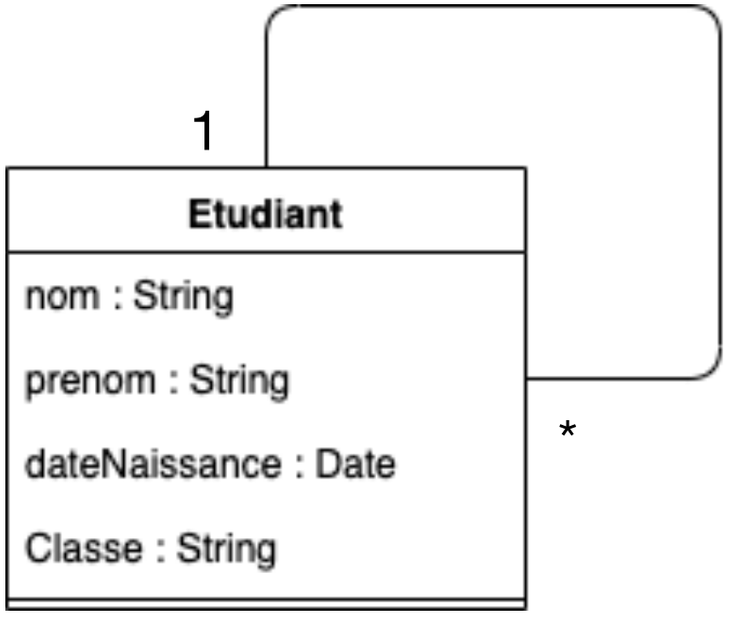
Etre binôme de



Relations réflexives

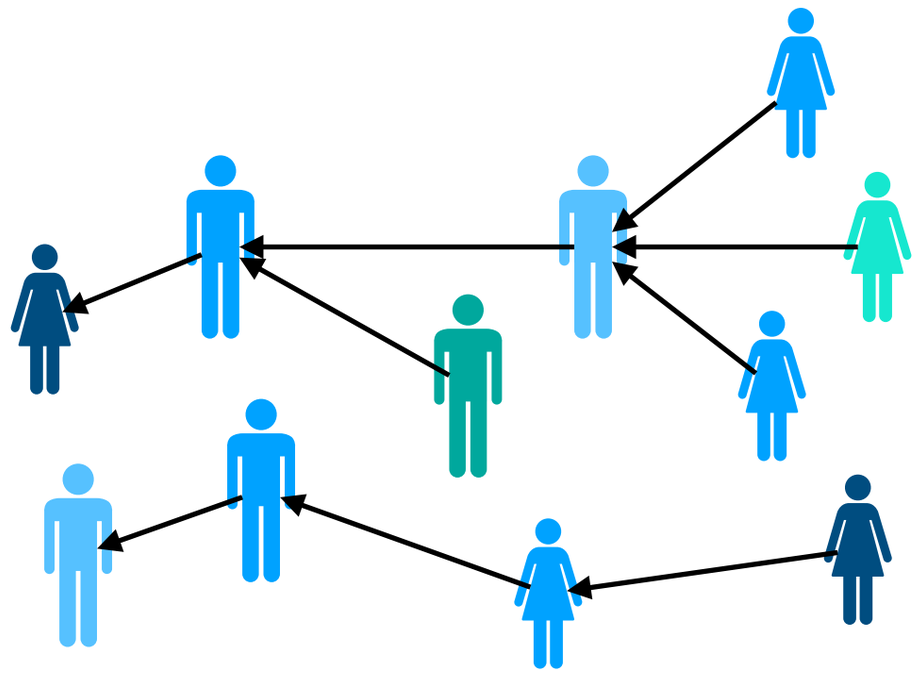
Relations réflexives et cardinalités

- On peut avoir des cardinalités différentes sur chaque extrémité
- Exemple : association réflexive **1 à plusieurs**



Avoir comme
parrain étudiant

Être parrain étudiant de



Relations réflexives

Relations réflexives et cardinalités

- On peut avoir des cardinalités différentes sur chaque extrémité
- Exemple : association réflexive **plusieurs à plusieurs**

Avoir comme pays voisin

France : {Espagne, Andorre, Belgique, Luxembourg, Allemagne, Suisse, Italie }

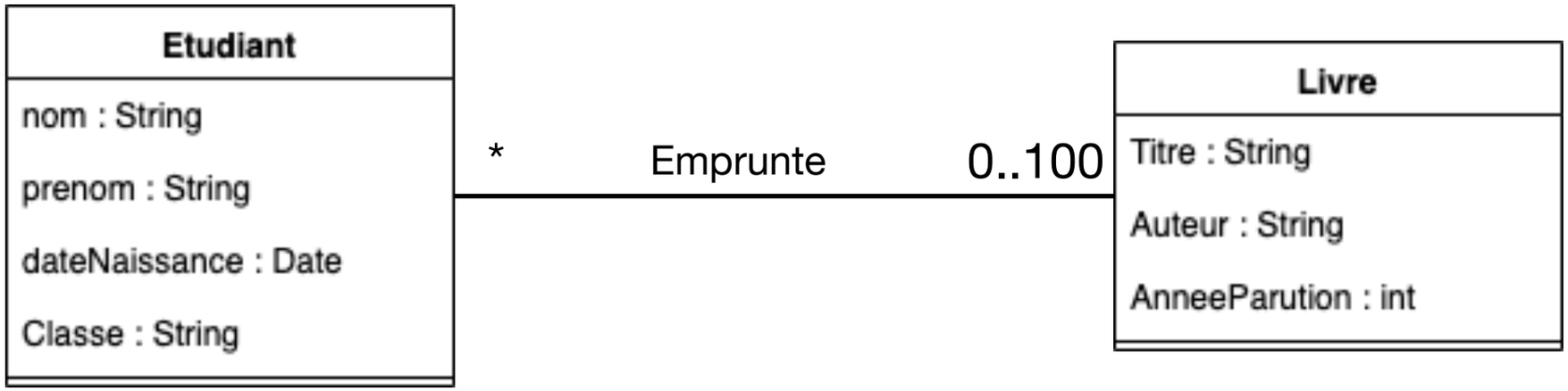
Espagne : {Portugal, **France**}

Suisse : {**France**, Allemagne, Italie, Autriche }

Luxembourg : {Belgique, Allemagne, **France** }

Limite des associations simples

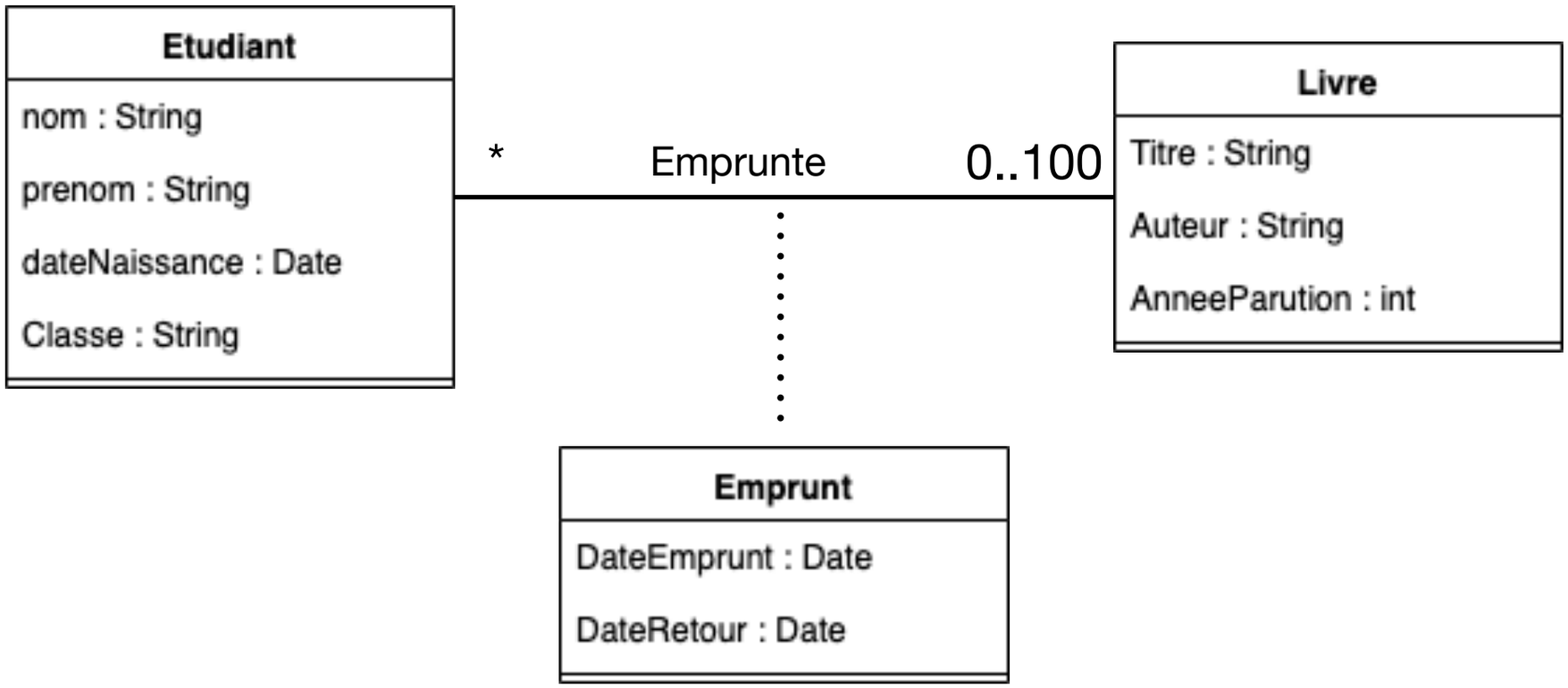
On souhaite faire un logiciel de gestion des emprunts de livres en gérant l'historique des emprunts de livres



- Où placer les attributs "***dateEmprunt***" et "***dateRetour***" ?
  - Dans Etudiant : un étudiant ne pourra emprunter qu'un livre à la fois
  - Dans Livre : chaque livre n'aura qu'une date d'emprunt. Impossible de sauvegarder l'historique des emprunts précédents
- Ces attributs sont propres à un emprunt : propres à l'association

Classes d'associations

On souhaite faire un logiciel de gestion des emprunts de livres en gérant l'historique des emprunts de livres

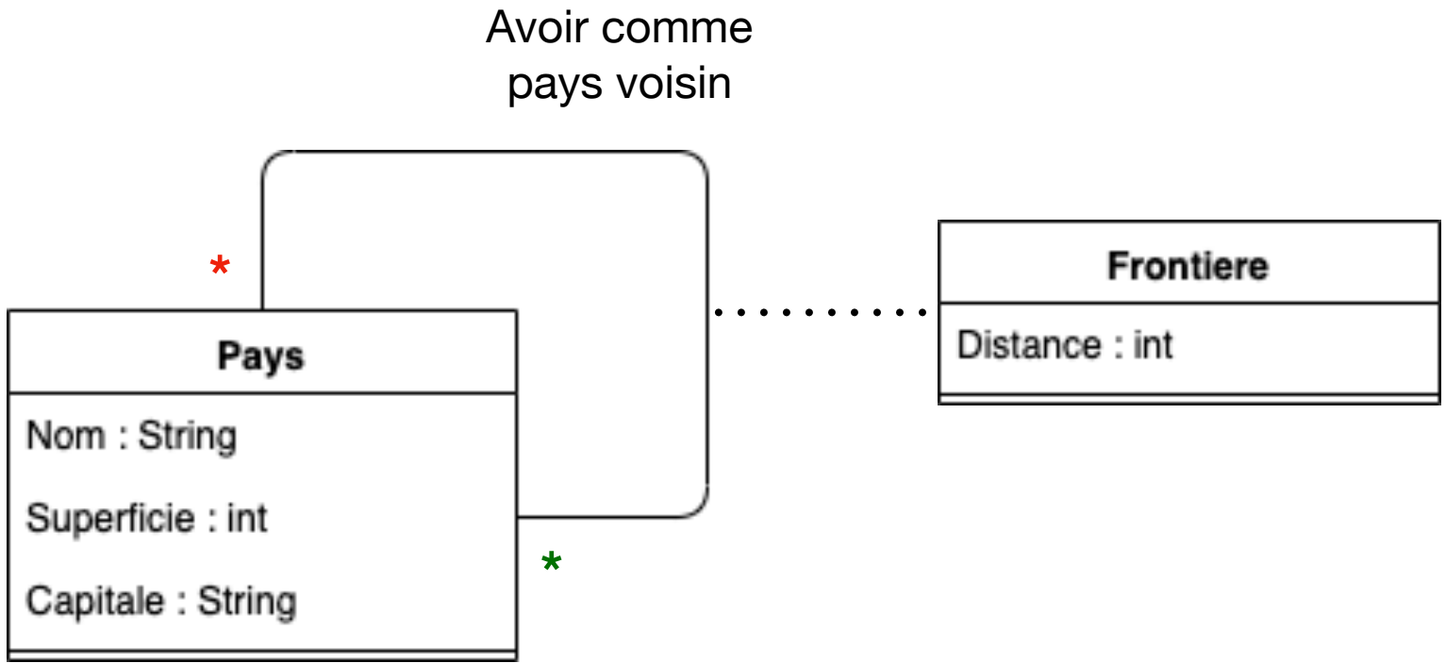


- On crée **une classe d'association**, reliée à l'association par des tirets

Classes d'associations sur relations réflexives

On peut avoir des classes d'association sur des relations réflexives

- Complétion de l'exemple précédent : on veut stocker la distance de la frontière entre deux pays frontaliers



France : {Espagne, Andorre, Belgique, Luxembourg, Allemagne, Suisse, Italie }

Espagne : {Portugal, **France**}

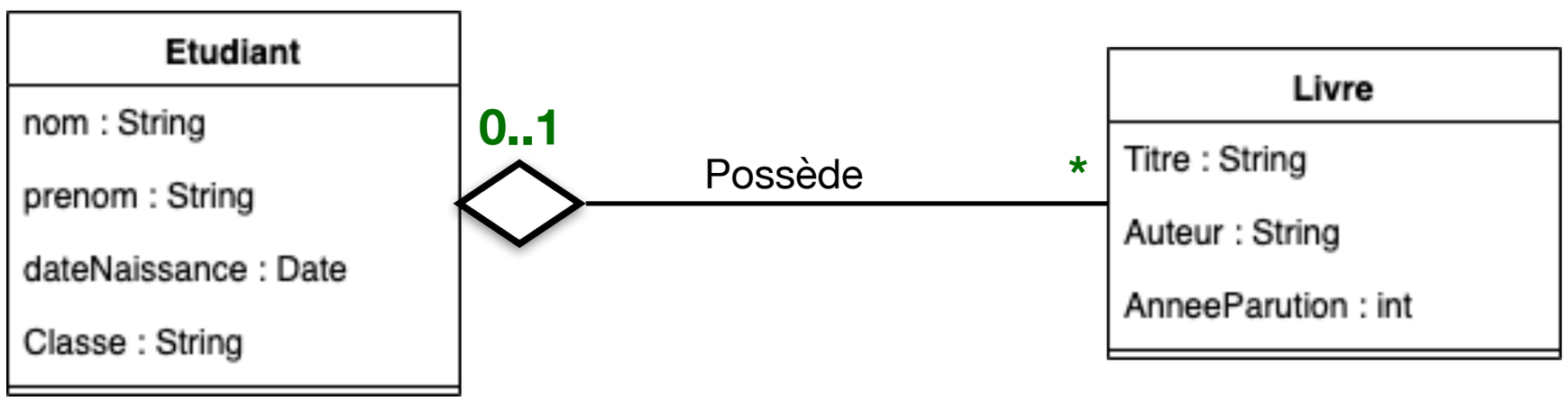
Suisse : {**France**, Allemagne, Italie, Autriche }

Luxembourg : {Belgique, Allemagne, **France** }

Les agrégations

Les agrégations sont des associations simples particulières

- Elles expriment une relation de **possession**
- Plus forte qu'une simple association
- Symbolisé par un losange blanc
- S'utilise forcément avec une cardinalité 0..1 ou 1 coté losange

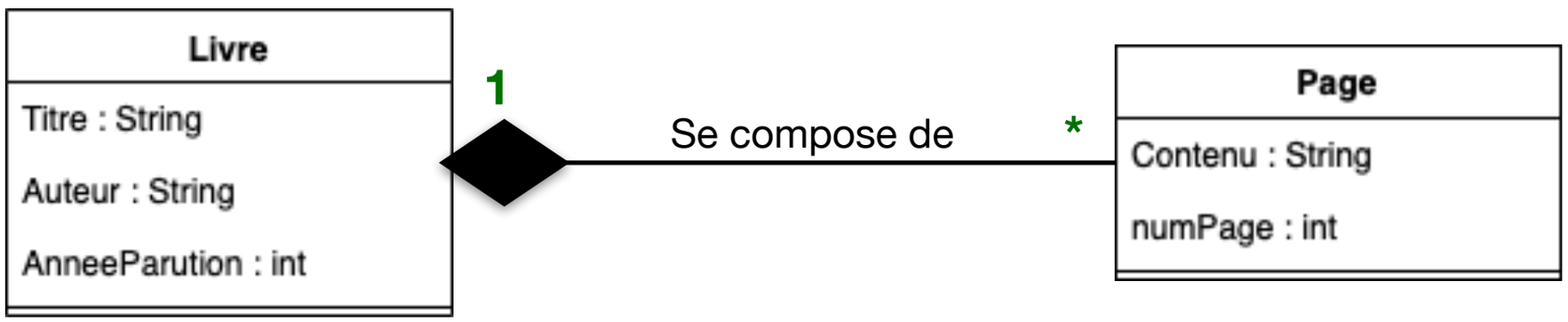


- L'agrégation est assimilée à une appartenance 'faible'

Les compositions

Les compositions sont des associations simples particulières

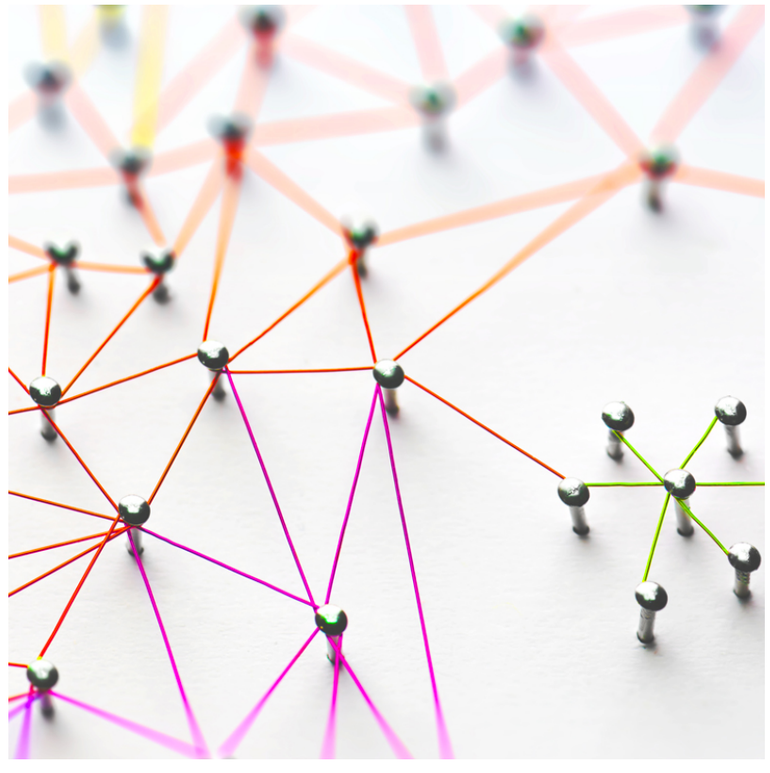
- Elles expriment une relation de **composition**
- Plus forte qu'une simple association et qu'une agrégation
- Symbolisé par un losange noir
- S'utilise forcément avec une cardinalité 1 coté losange



Les compostions sont des appartenances 'fortes'

- La destruction du composé implique la destruction des composants

Héritage de classes

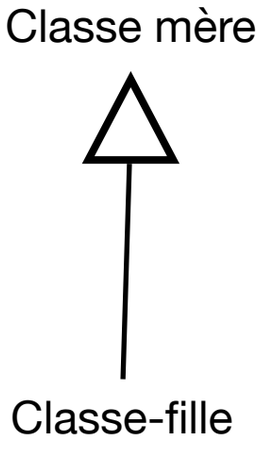


L'héritage

Les classes et sous-classes

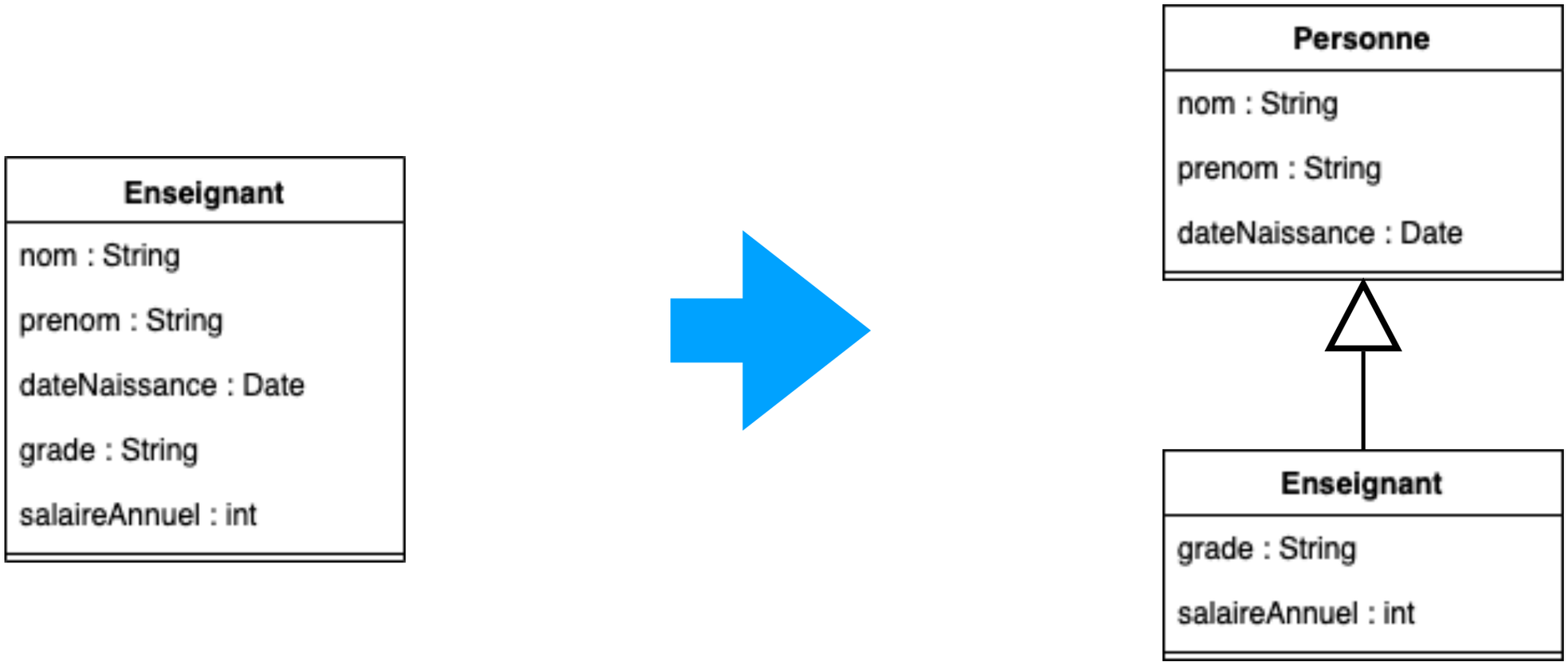
- Une sous-classe (classe fille) permet de spécialiser une classe mère
- Au sens sémantique : "est un sous-type de"
- La sous-classe contient tous les attributs de la classe mère
- Une classe mère contient les attributs communs à toutes ses sous-classes
- Pas de cardinalité ici, ca n'aurait pas de sens

Modélisation :



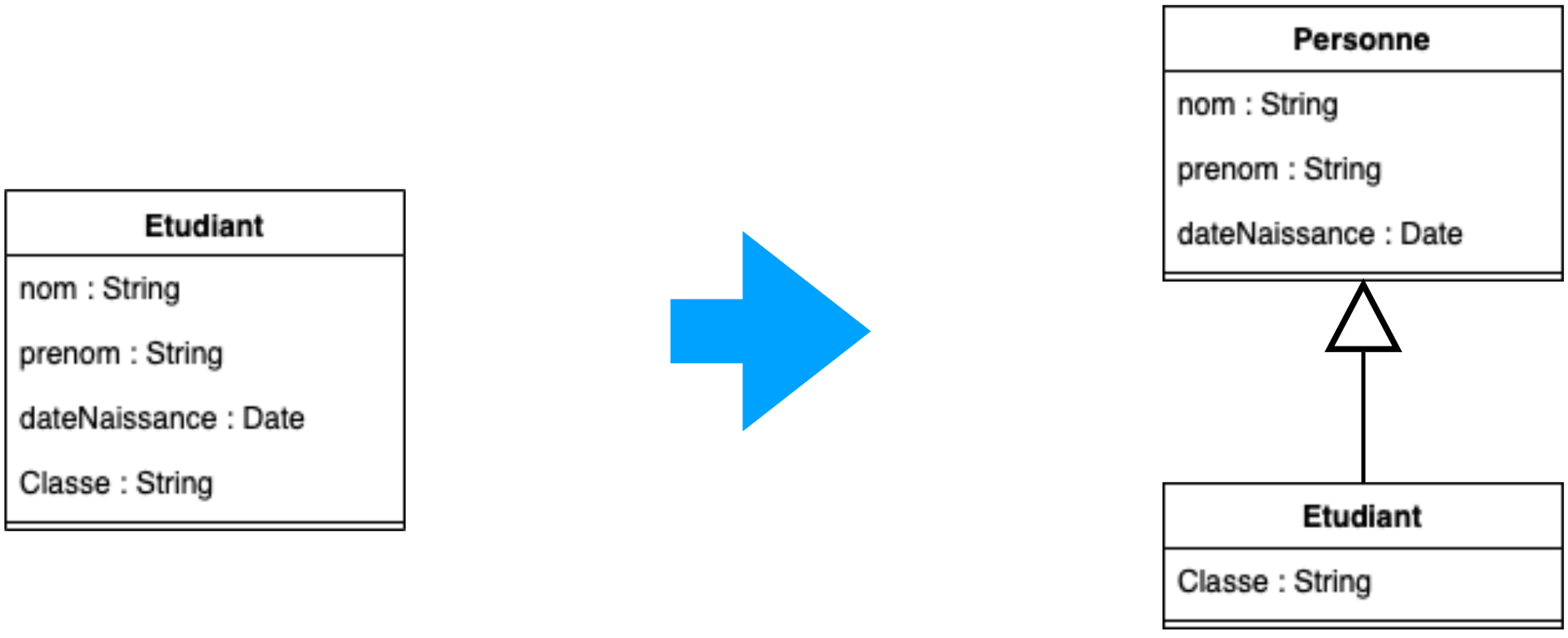
L'héritage

Un exemple simple :



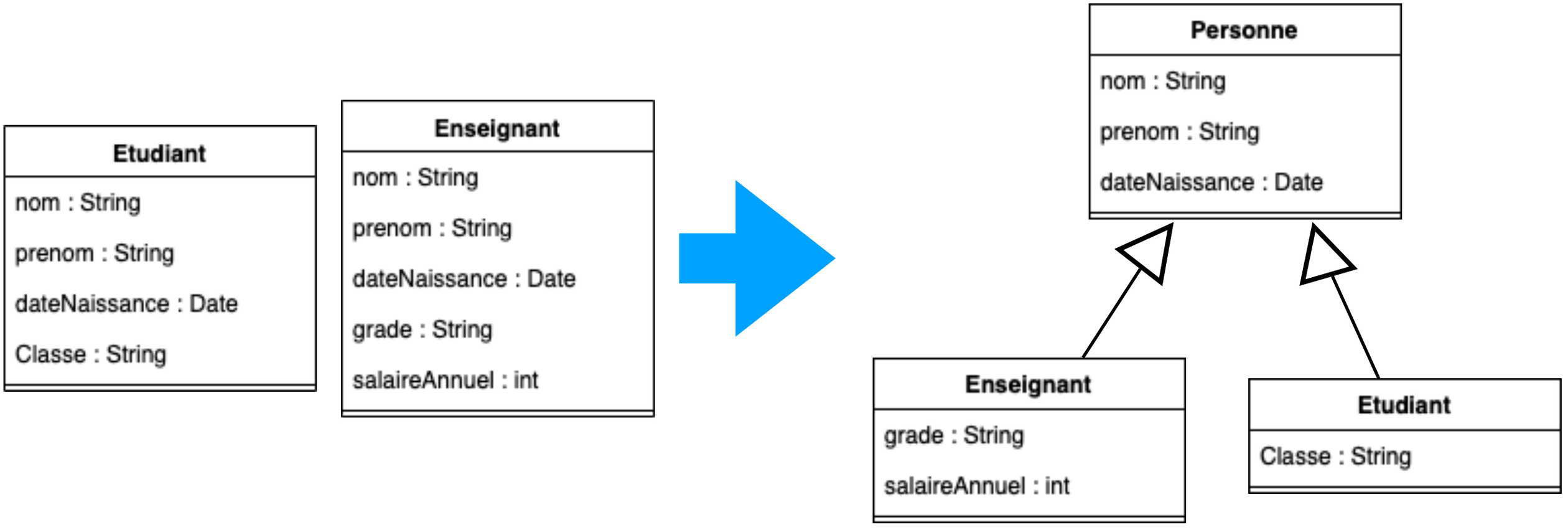
L'héritage

Un exemple simple :



L'héritage

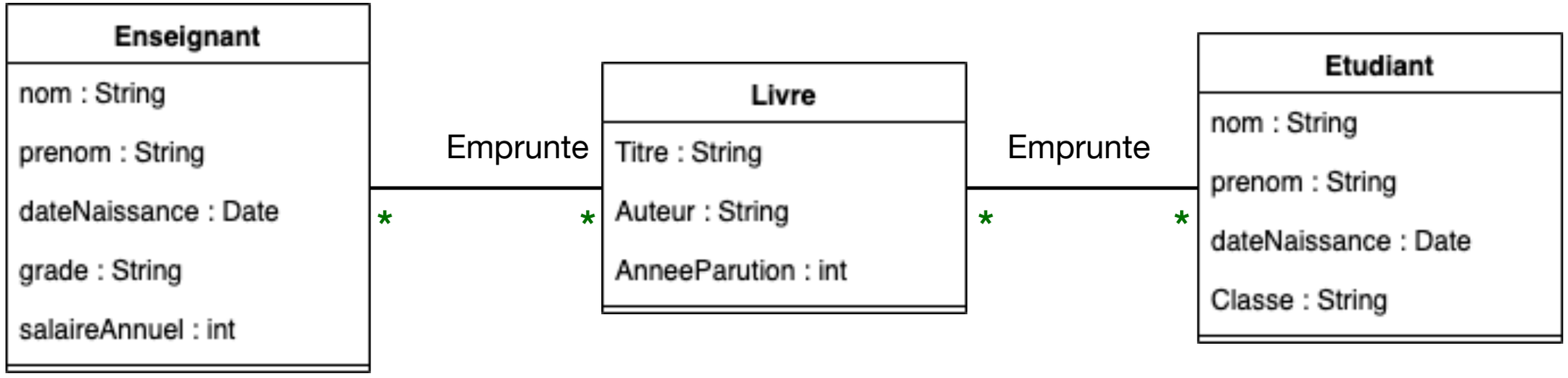
Un exemple simple :



L'héritage

Sans héritage :

- Redondance des attributs des classes proches
- Si relation avec une autre classe, il faut répéter la relation sur chaque classe



L'héritage

Avec héritage :

- Si relation avec une autre classe, on peut limiter la relation à la classe mère
- Gain de cohérence