conception et prog objet avancés

ACDA 3.3– CPOO

Florent Madelaine

IUT de Sénart Fontainebleau
Département Informatique

Année 2021-22
Cours 1

UML

- normalisation en cours depuis 1997
- adopté depuis cette date par un comité de standardisation international (Object Management Group) regroupant de nombreux industriels
- UML 2 norme ISO depuis 2005

UML pour nous

Utilisation Pragmatique

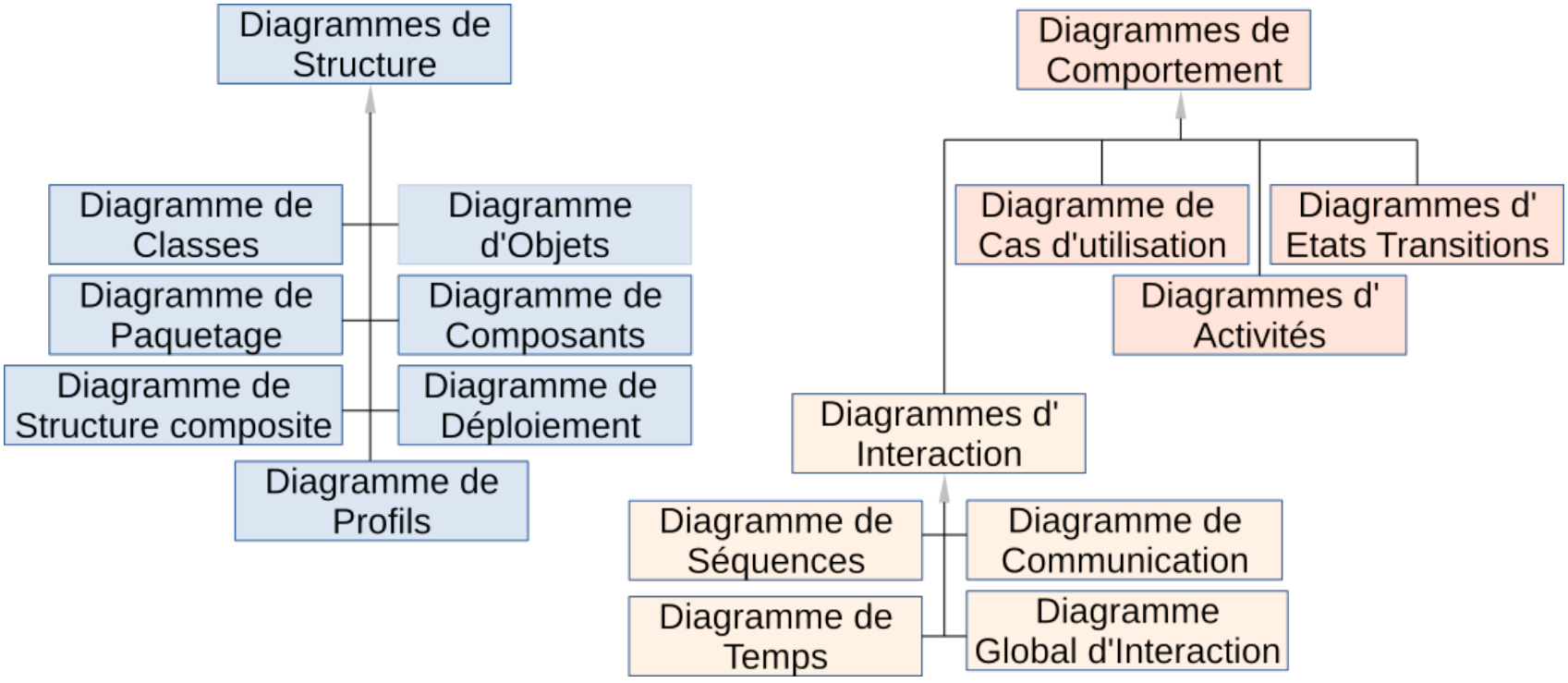
- Utilisation de certains diagrammes pour aider à la conception de logiciels.
- Utilisation correcte de la notation UML dans la mesure du possible mais *in fine* notre but est de réfléchir et de communiquer.

  (il vaut mieux parler le globish que de ne pas parler du tout par peur d'écorcher la langue de Shakespeare).

Utilisation plus poussée possible

- Si on veut vérifier (automatiquement et avec des preuves) des propriétés d'un modèle il faut d'abord expliquer ce dernier.
- Exemple : SysML (recherche de Régine).

Différents types de diagrammes



Alternatives

Il existe de nombreuses alternatives. En particulier le *modèle C4* est assez convaincant mais nous n'allons pas en parler dans ce cours, car il est plutôt adapté à des logiciels complexes dont on veut expliquer l'architecture.

On retrouve des diagrammes UML dans la dernière couche du modèle C4. Typiquement il est recommandé de fabriquer ces diagrammes automatiquement à partir du code (le logiciel de l'IUT starUML a un greffon qui permet de le faire).

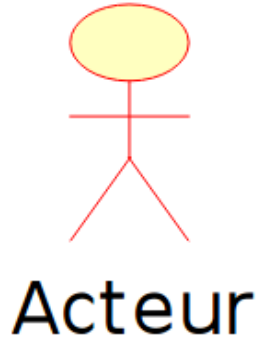
`https://en.wikipedia.org/wiki/C4_model`
`https://c4model.com/`

Utilité des cas d'utilisation

- Identifier les interactions du système avec son environnement : **humains** et **autres systèmes**.
- Identifier les besoins : fonctionnalités du système.
- Identifier les dépendences entre les fonctionnalités.

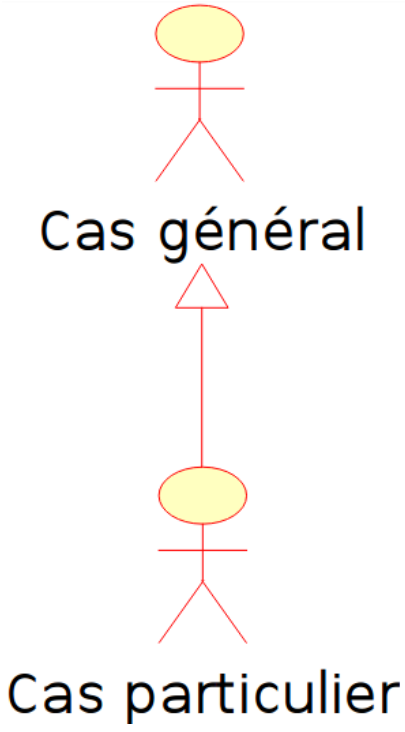
Les acteurs

- Tout sauf le système : personnes ou **autres systèmes**.
- Définit les rôles des interactions : une même **personne** peut se comporter suivant différents rôles.
- Les acteurs seront liés aux cas d'utilisation les concernant.
- Les acteurs peuvent être liés par **héritage/généralisation**.



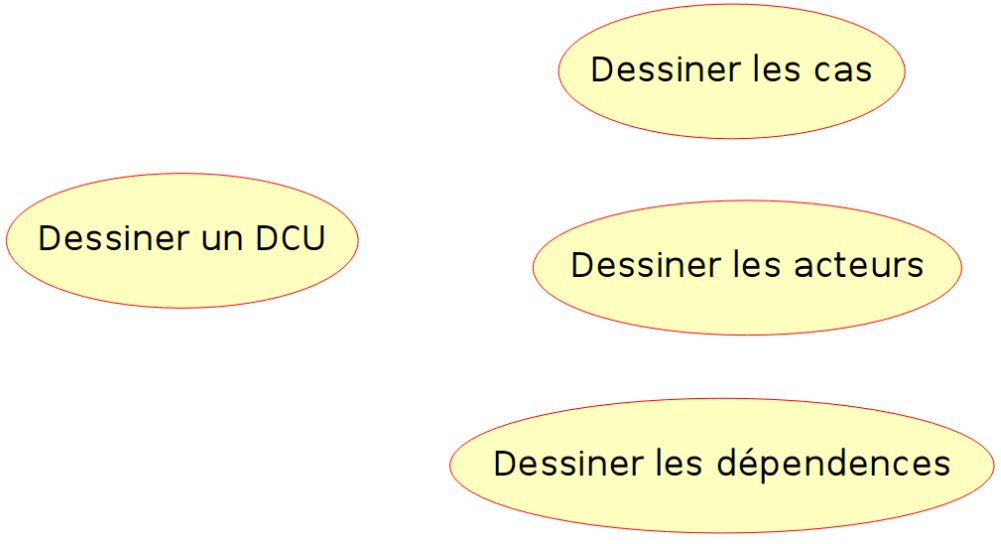
Les acteurs

- Tout sauf le système : personnes ou **autres systèmes**.
- Définit les rôles des interactions : une même **personne** peut se comporter suivant différents rôles.
- Les acteurs seront liés aux cas d'utilisation les concernant.
- Les acteurs peuvent être liés par **héritage/généralisation**.



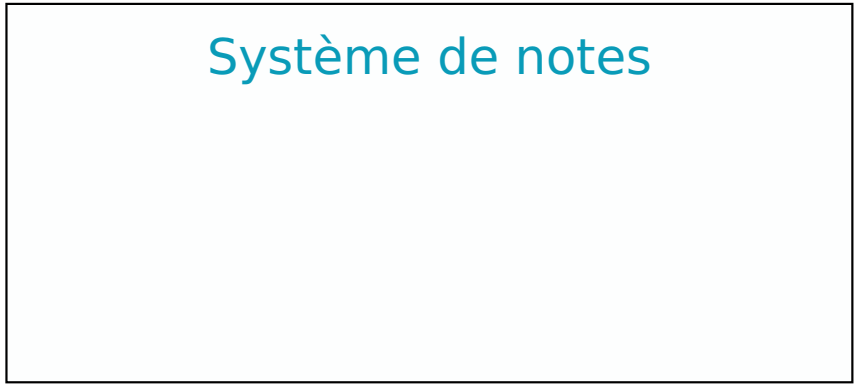
Les cas d'utilisation

- Recensent les besoins (phase d'analyse des besoins) ou les fonctionnalités (documentation).
- Souvent un verbe à l'infinitif avec un complément : Faire quelque chose.
- Leur atomicité dépend du niveau de détail voulu :



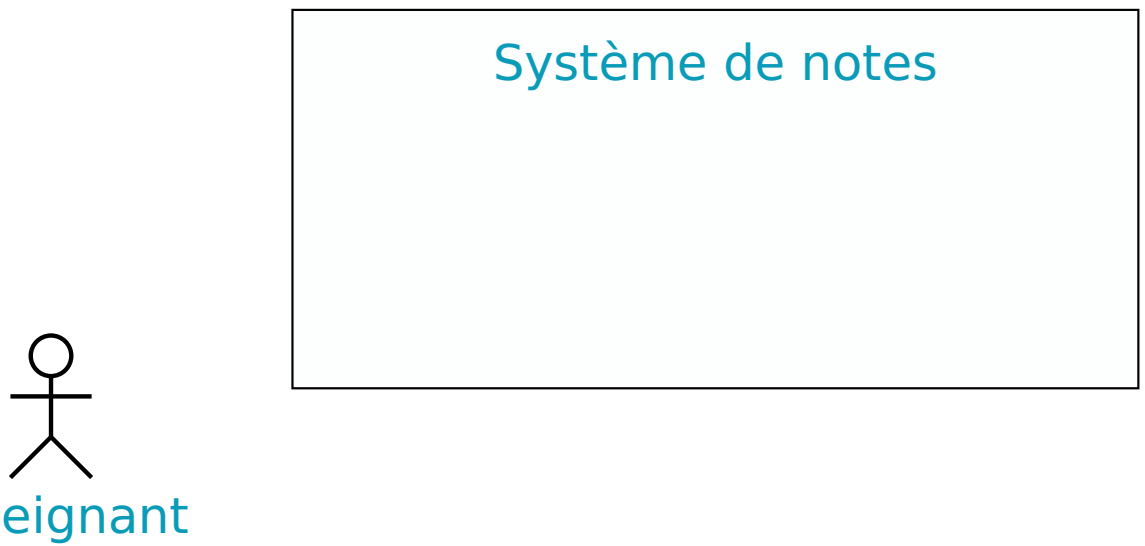
Exemple

On veut un système de notes où les enseignants pourront entrer leurs notes et les étudiants les consulter ; les enseignants peuvent également consulter les notes des étudiants. À la fin du semestre, les notes sont transmises au serveur central de notes qui s'occupera de l'édition des diplômes.



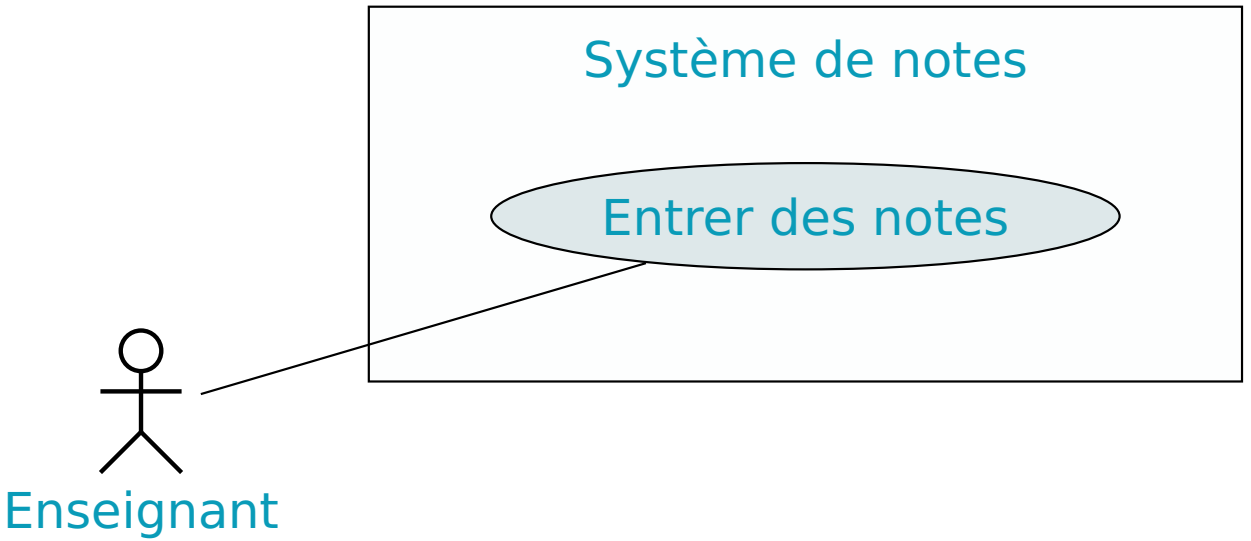
Exemple

On veut un système de notes où les enseignants pourront entrer leurs notes et les étudiants les consulter ; les enseignants peuvent également consulter les notes des étudiants. À la fin du semestre, les notes sont transmises au serveur central de notes qui s'occupera de l'édition des diplômes.



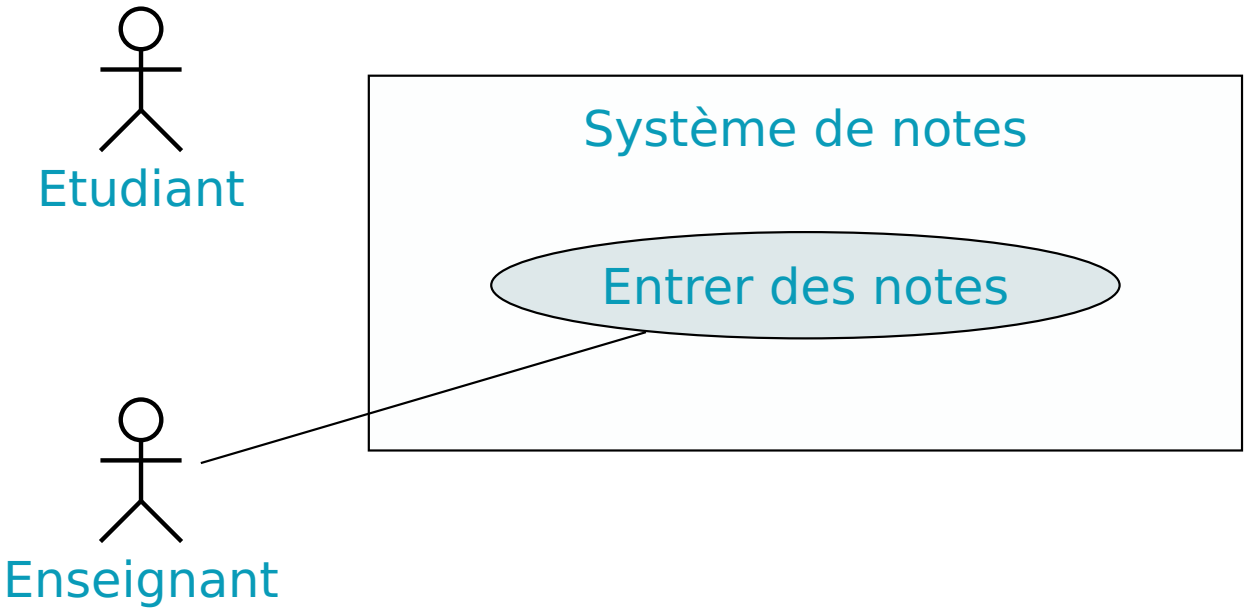
Exemple

On veut un système de notes où les enseignants pourront entrer leurs notes et les étudiants les consulter ; les enseignants peuvent également consulter les notes des étudiants. À la fin du semestre, les notes sont transmises au serveur central de notes qui s'occupera de l'édition des diplômes.



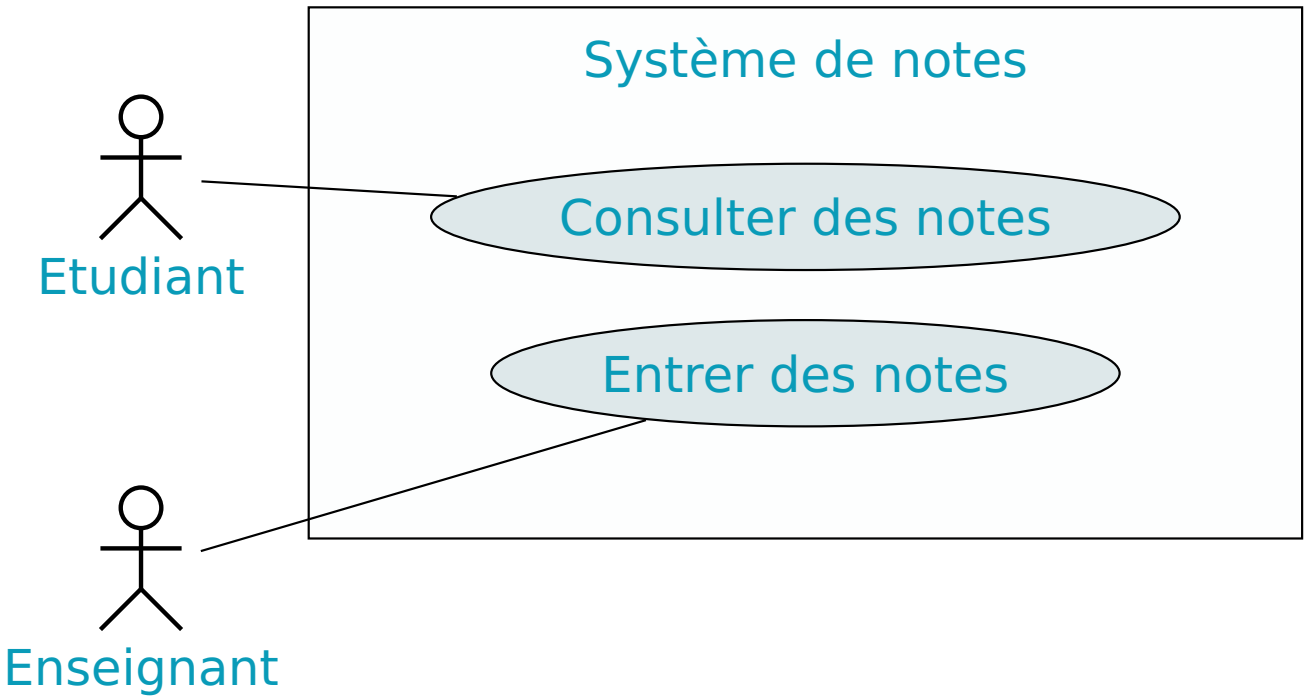
Exemple

On veut un système de notes où les enseignants pourront entrer leurs notes et les étudiants les consulter ; les enseignants peuvent également consulter les notes des étudiants. À la fin du semestre, les notes sont transmises au serveur central de notes qui s'occupera de l'édition des diplômes.



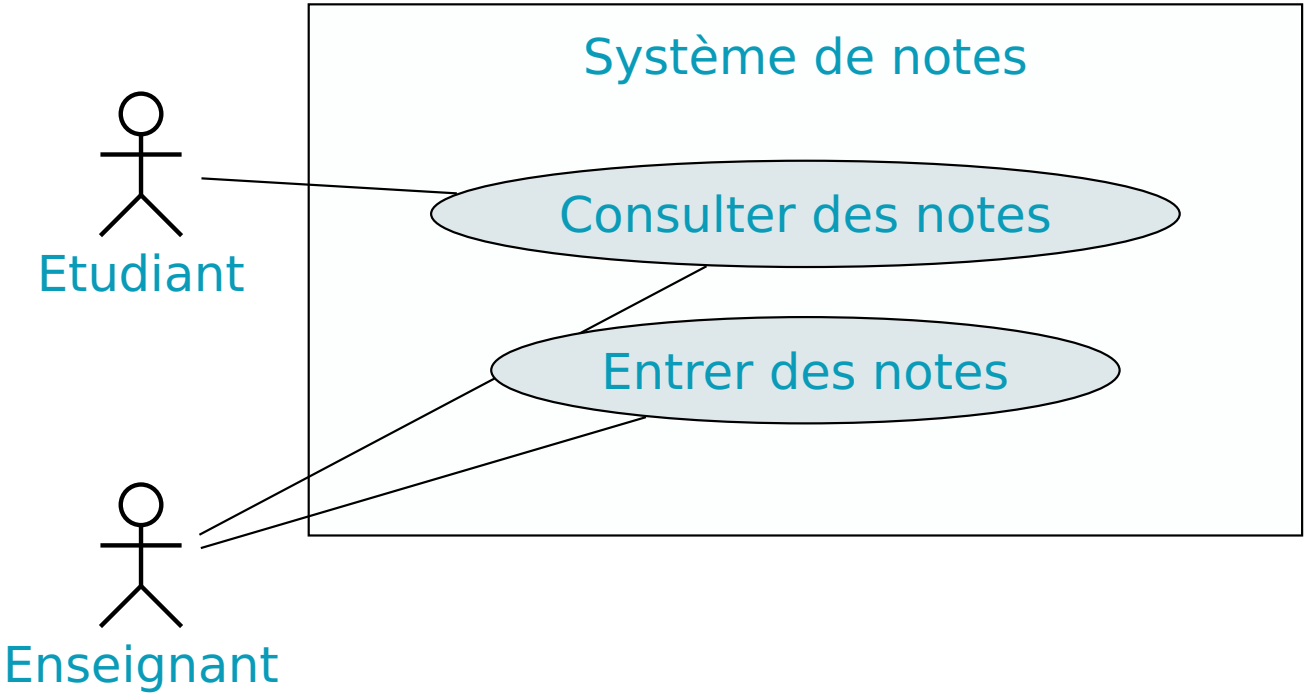
Exemple

On veut un système de notes où les enseignants pourront entrer leurs notes et les étudiants les consulter ; les enseignants peuvent également consulter les notes des étudiants. À la fin du semestre, les notes sont transmises au serveur central de notes qui s'occupera de l'édition des diplômes.



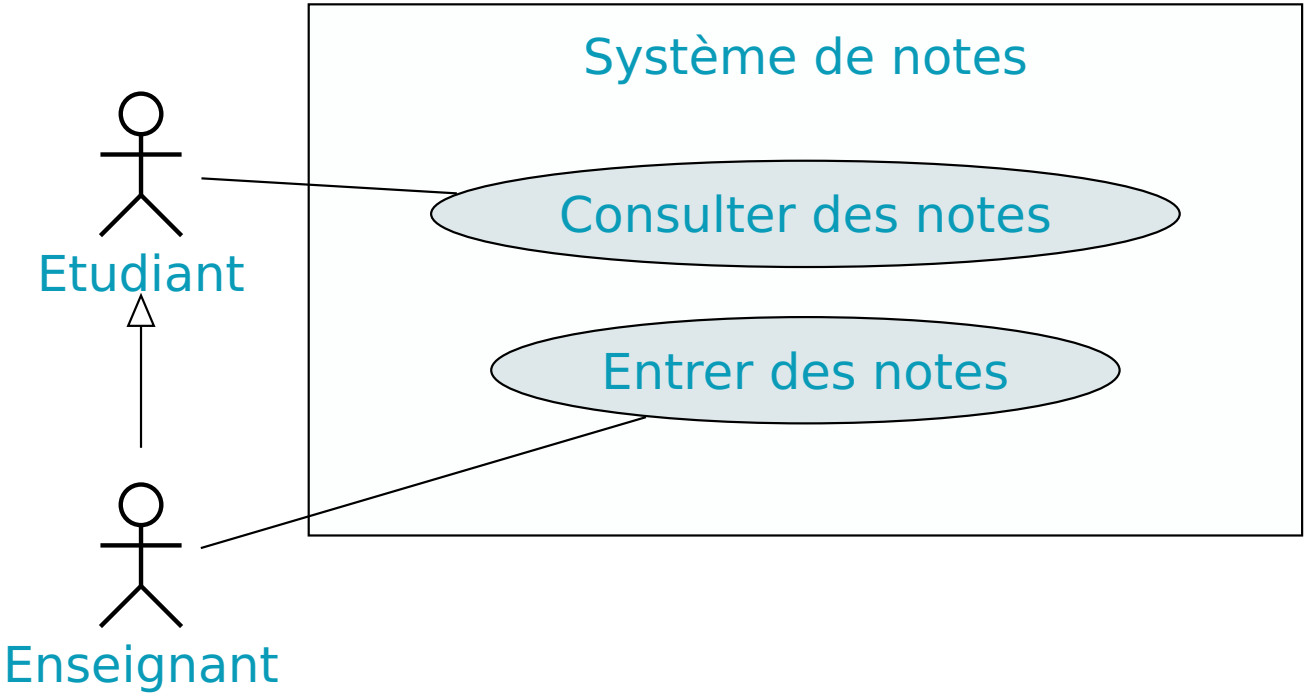
Exemple

On veut un système de notes où les enseignants pourront entrer leurs notes et les étudiants les consulter ; les enseignants peuvent **également** consulter les notes des étudiants. À la fin du semestre, les notes sont transmises au serveur central de notes qui s'occupera de l'édition des diplômes.



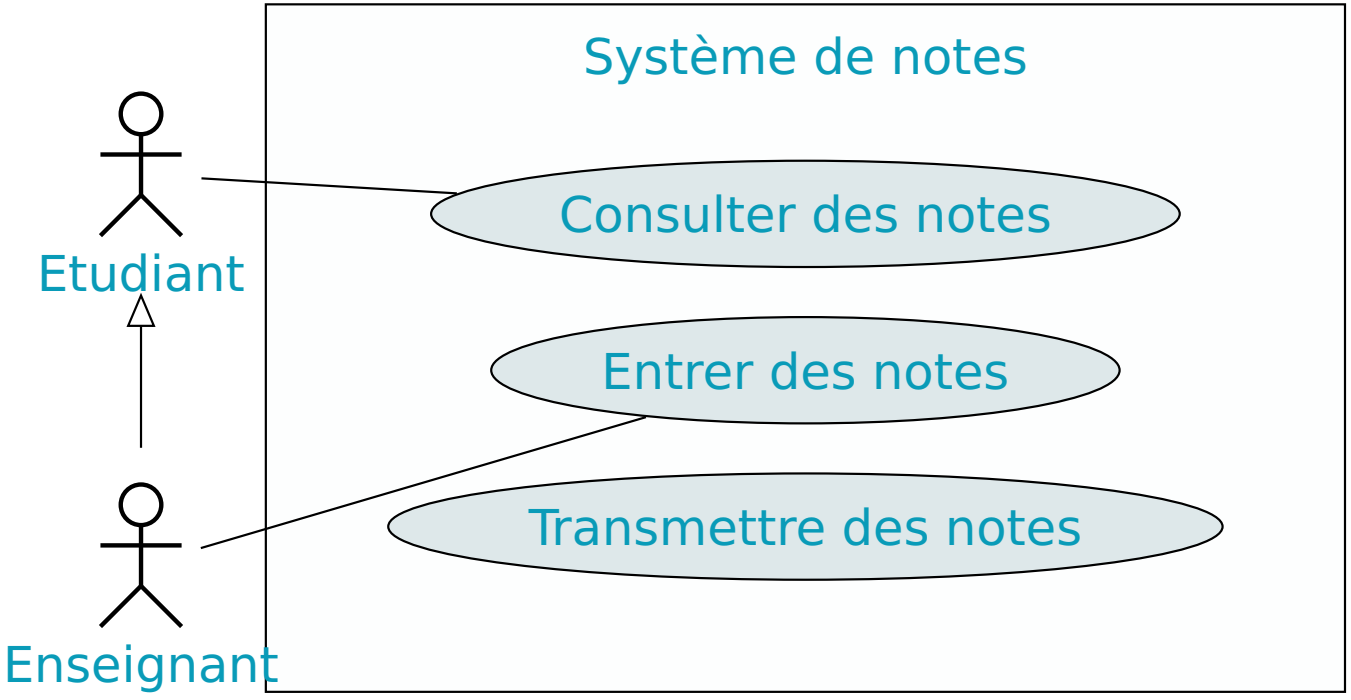
Exemple

On veut un système de notes où les enseignants pourront entrer leurs notes et les étudiants les consulter ; les enseignants peuvent **également** consulter les notes des étudiants. À la fin du semestre, les notes sont transmises au serveur central de notes qui s'occupera de l'édition des diplômes.



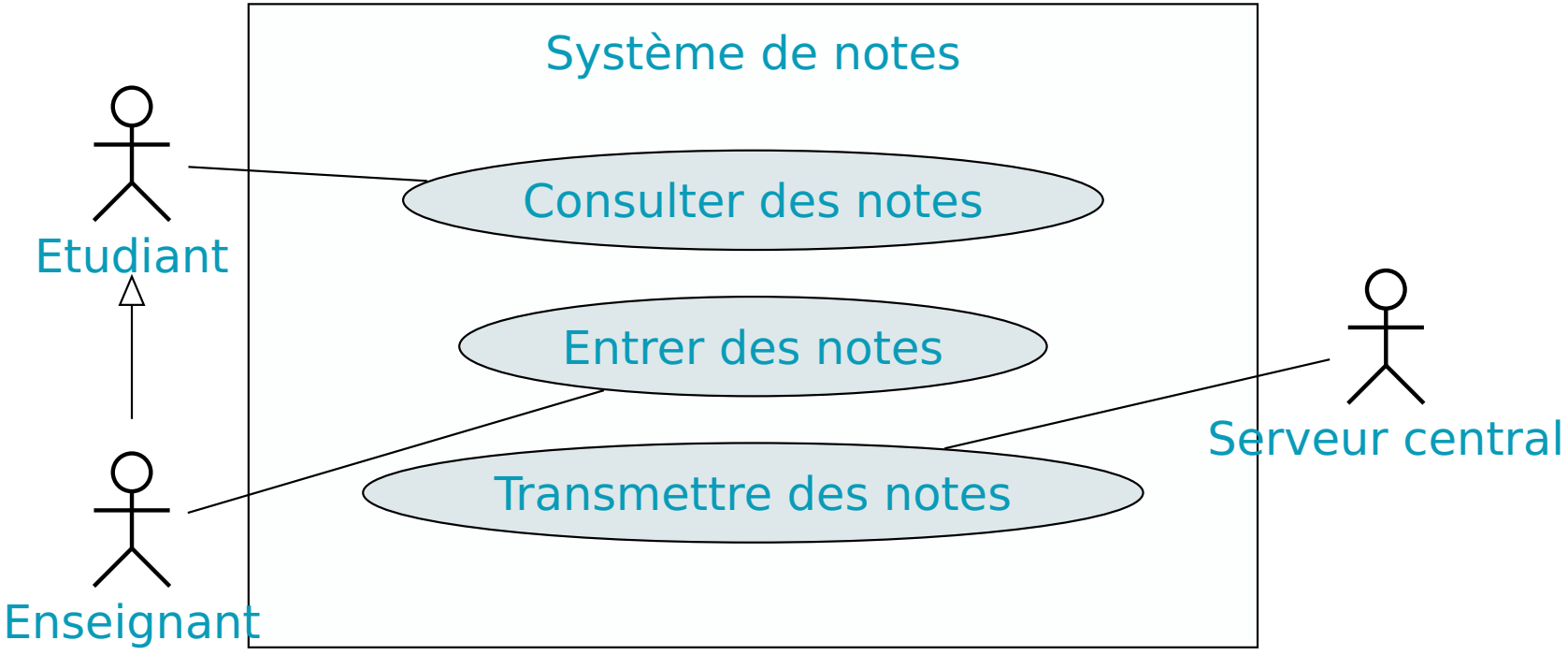
Exemple

On veut un système de notes où les enseignants pourront entrer leurs notes et les étudiants les consulter ; les enseignants peuvent également consulter les notes des étudiants. À la fin du semestre, les notes sont transmises au serveur central de notes qui s'occupera de l'édition des diplômes.



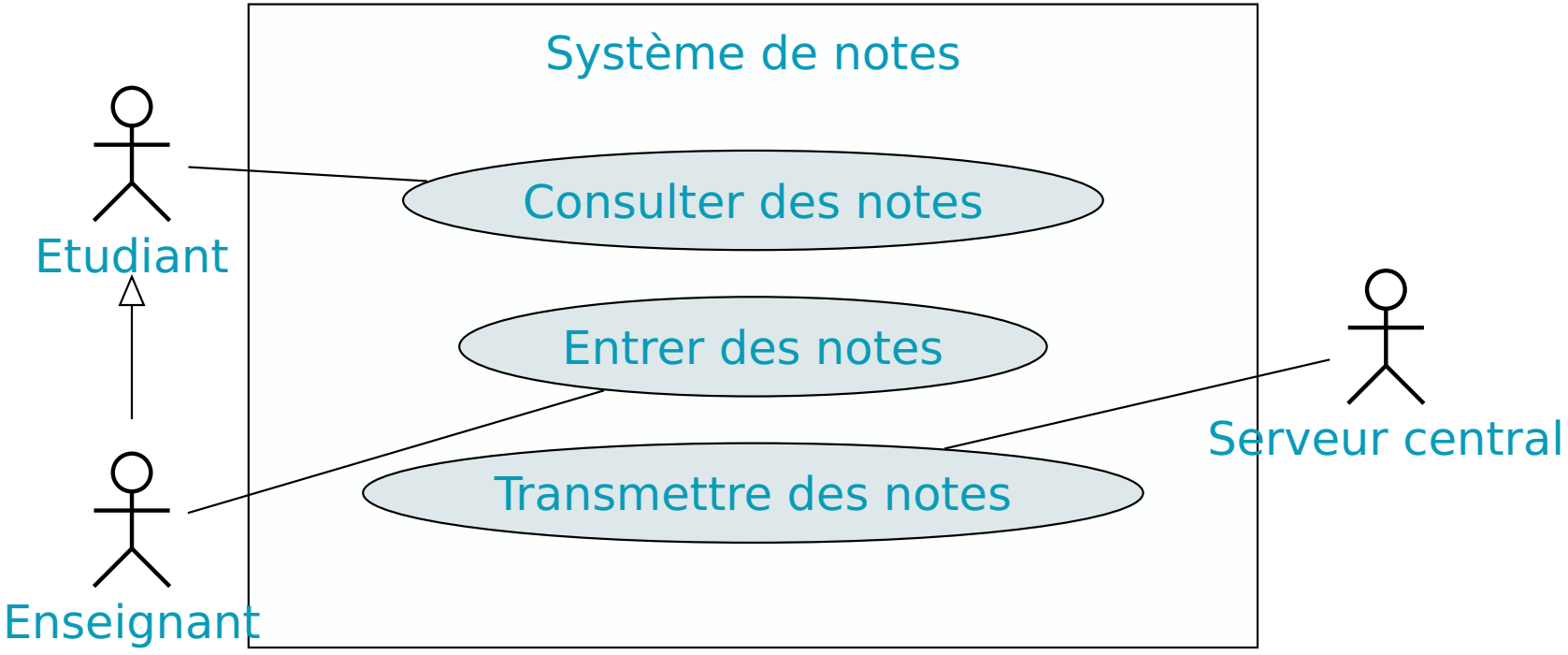
Exemple

On veut un système de notes où les enseignants pourront entrer leurs notes et les étudiants les consulter ; les enseignants peuvent également consulter les notes des étudiants. À la fin du semestre, les notes sont transmises au serveur central de notes qui s'occupera de l'édition des diplômes.



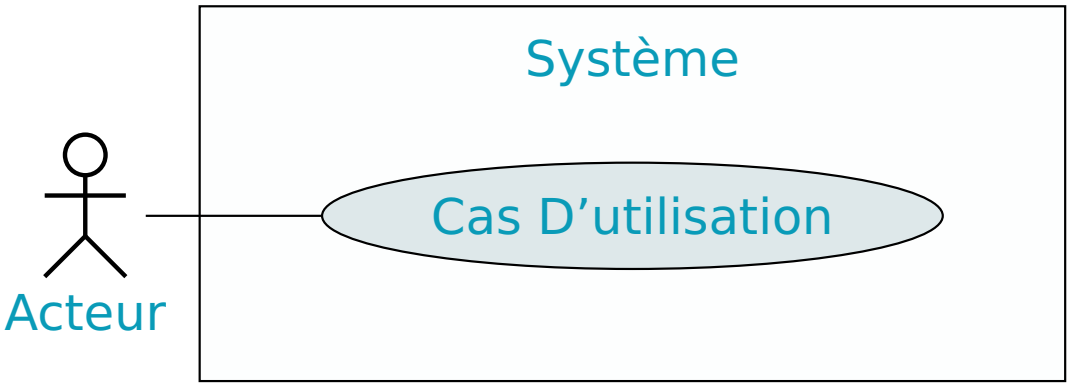
Exemple

On veut un système de notes où les enseignants pourront entrer leurs notes et les étudiants les consulter ; les enseignants peuvent également consulter les notes des étudiants. À la fin du semestre, les notes sont transmises au serveur central de notes qui s'occupera de **l'édition des diplômes**.



Relations acteur–cas d'utilisation

- Cas habituel : association.

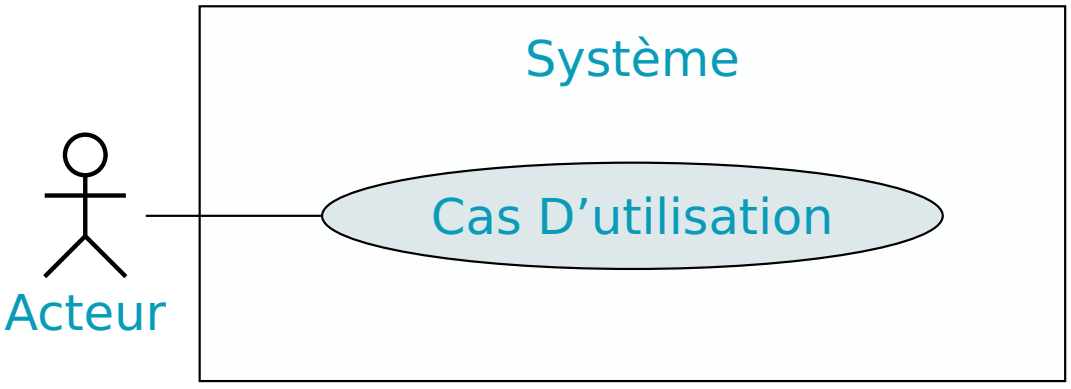


↪ interaction **bidirectionnelle**.

- Peut lier plusieurs acteurs à un même cas. Exemple :

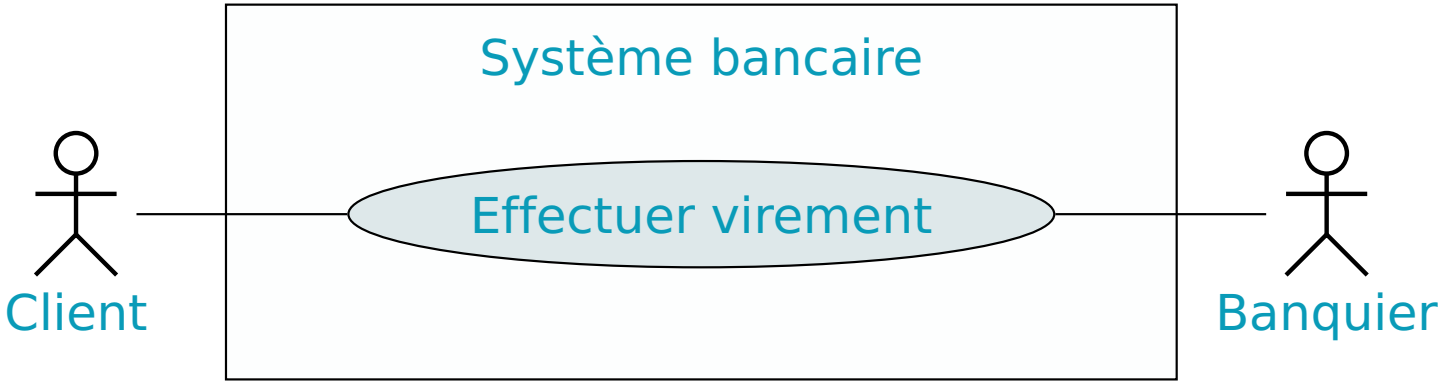
Relations acteur–cas d'utilisation

- Cas habituel : association.



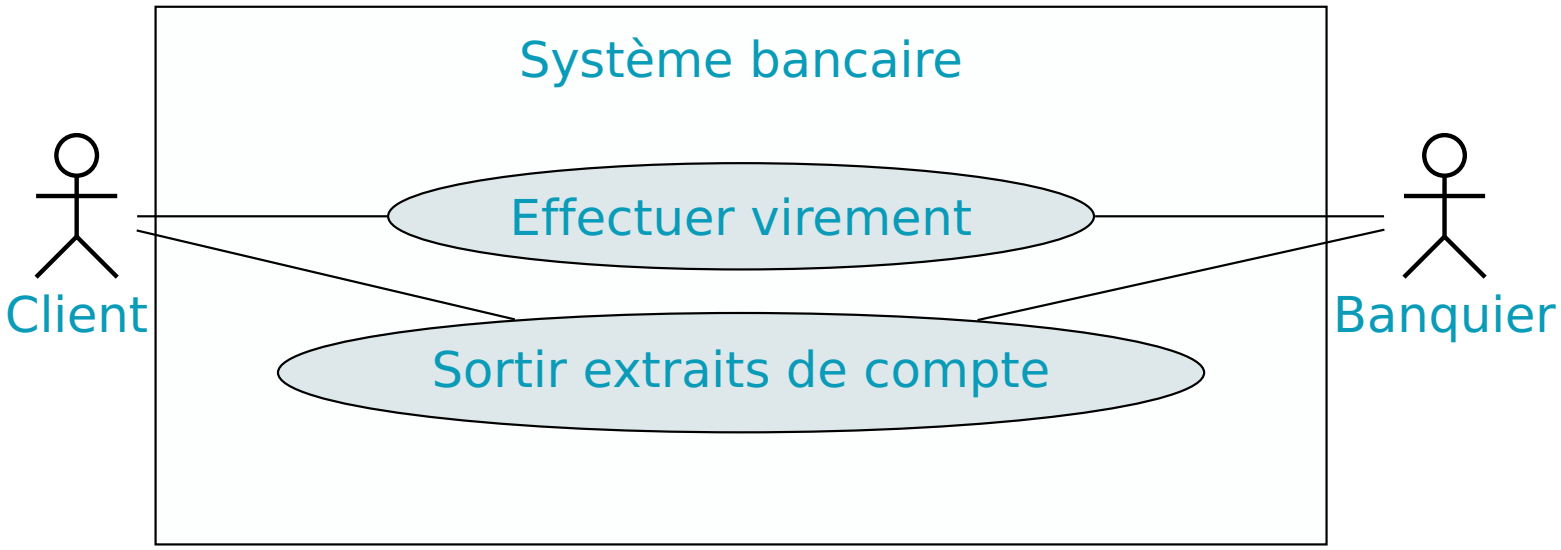
↪ interaction **bidirectionnelle**.

- Peut lier plusieurs acteurs à un même cas. Exemple :



Remarque

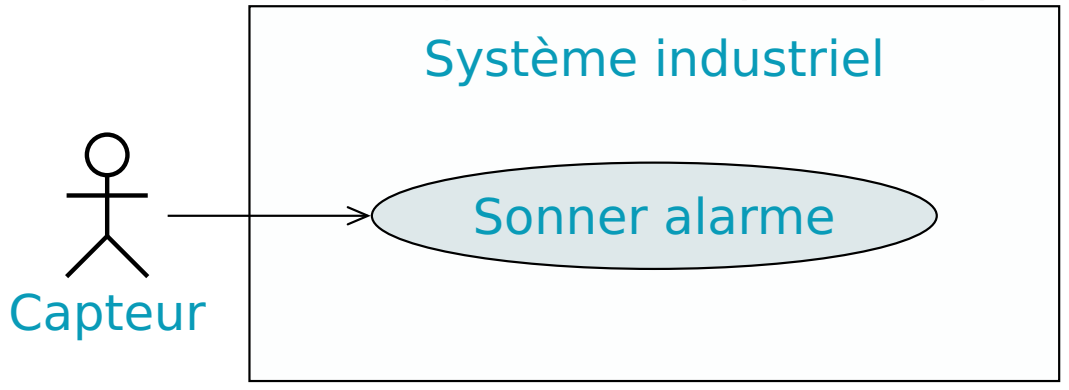
Un cas lié à plusieurs acteurs ne donne pas pour autant d'information sur la concomitance.



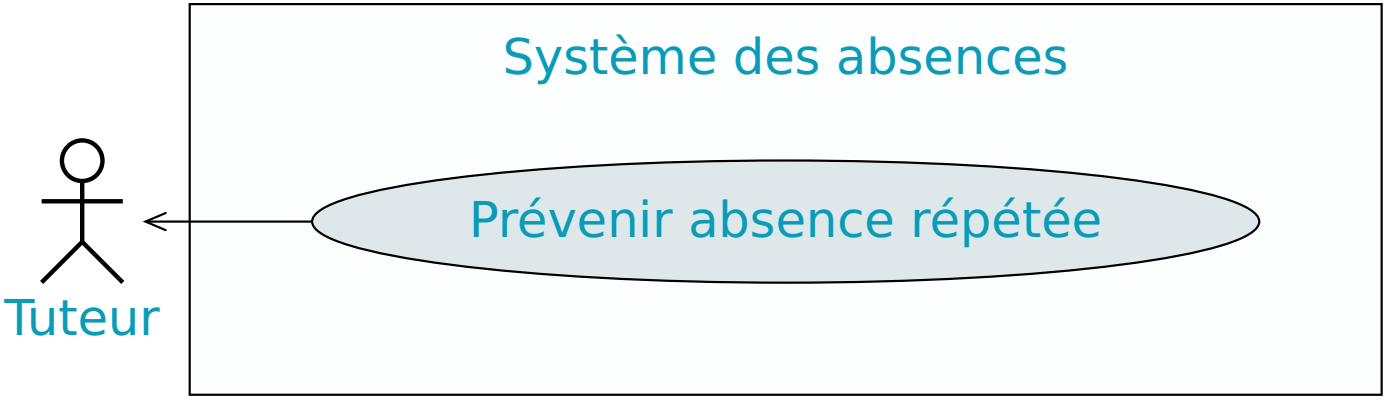
Relations acteur–cas d'utilisation — suite

- Association unidirectionnelle de l'acteur vers le système : stimulus, message entrant pour le système.

Exemple :

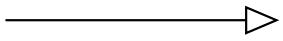


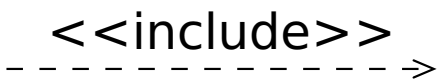
- Association unidirectionnelle du système vers l'acteur : message sortant. Exemple :



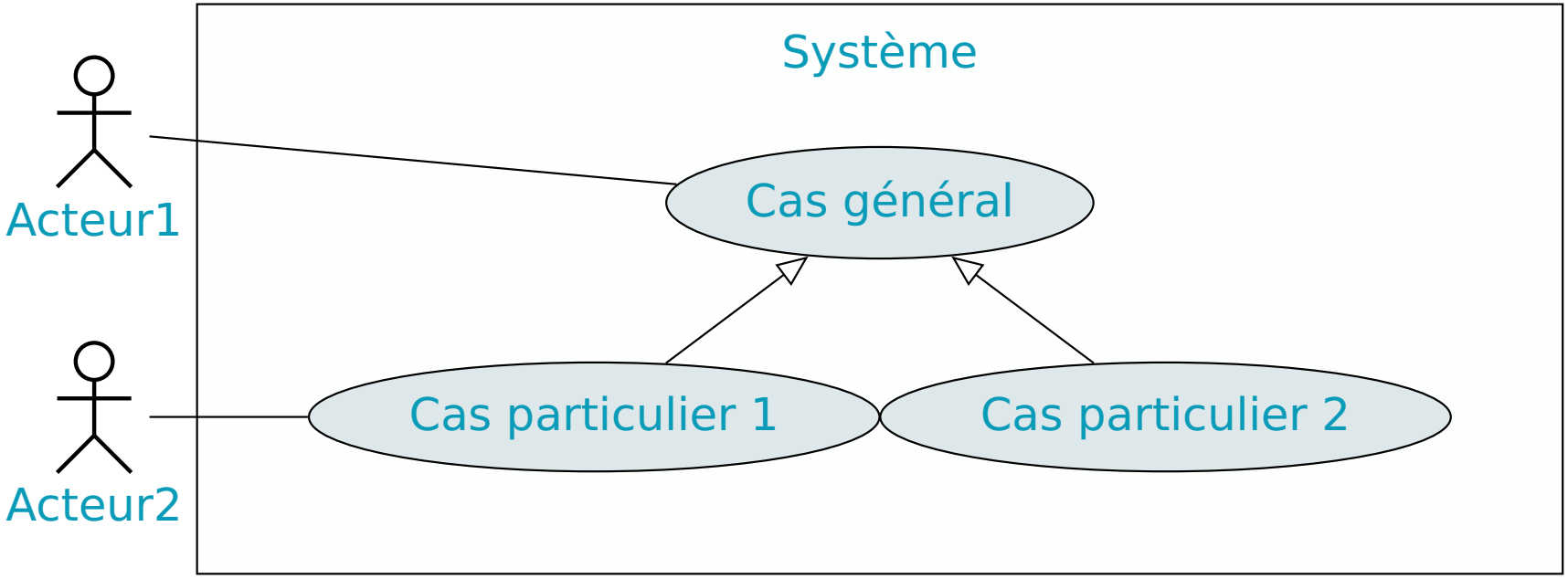
Relations entre cas d'utilisation

Trois types :

- Héritage/généralisation : 
- Extension : 

- Inclusion : 

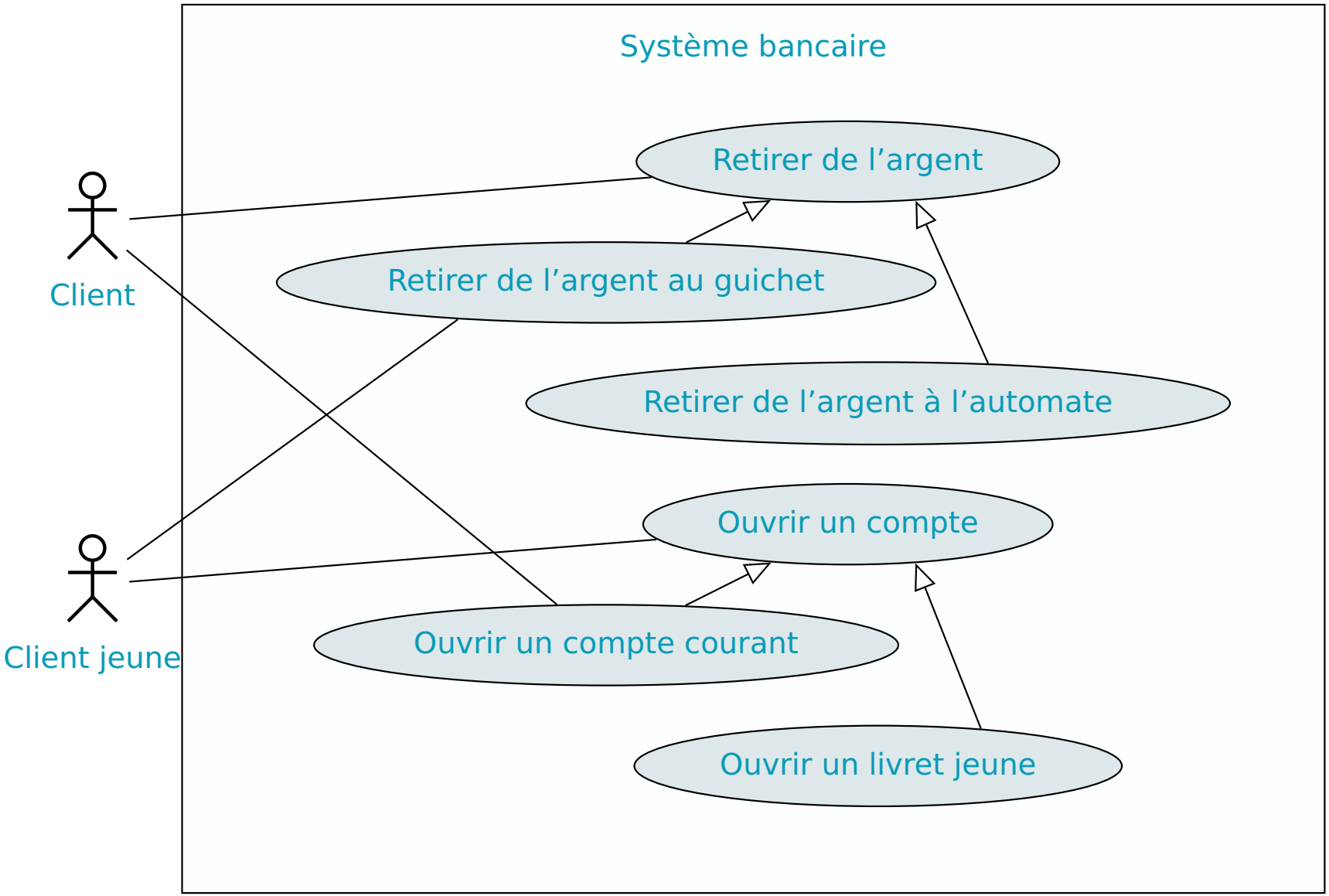
Héritage/Généralisation



Remarque

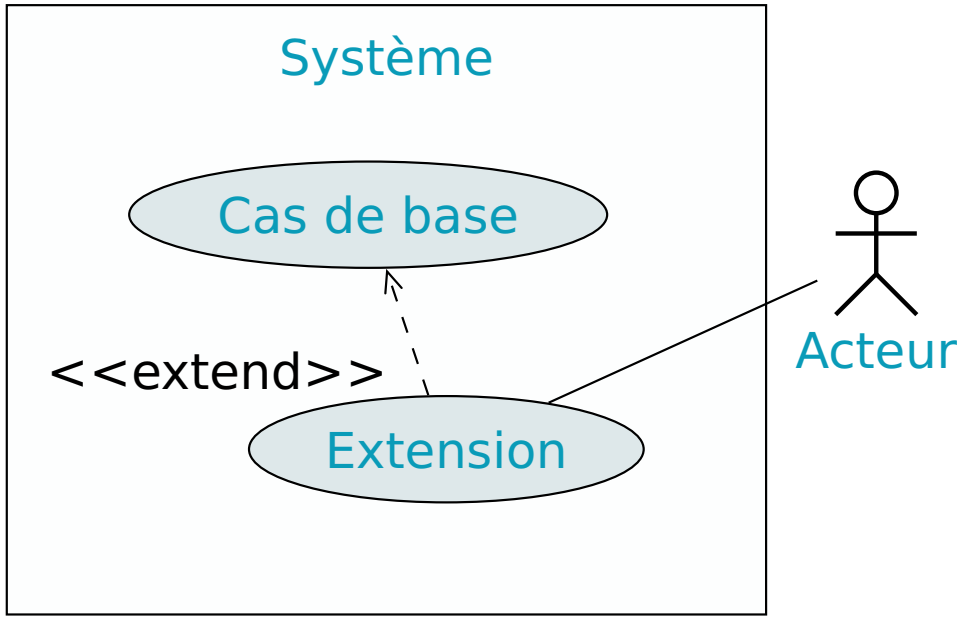
- Acteur1 peut faire le cas général, donc il peut faire les cas particuliers 1 et 2.
- Acteur2 ne peut pas faire le cas particulier 2.

Héritage/Généralisation — Exemple



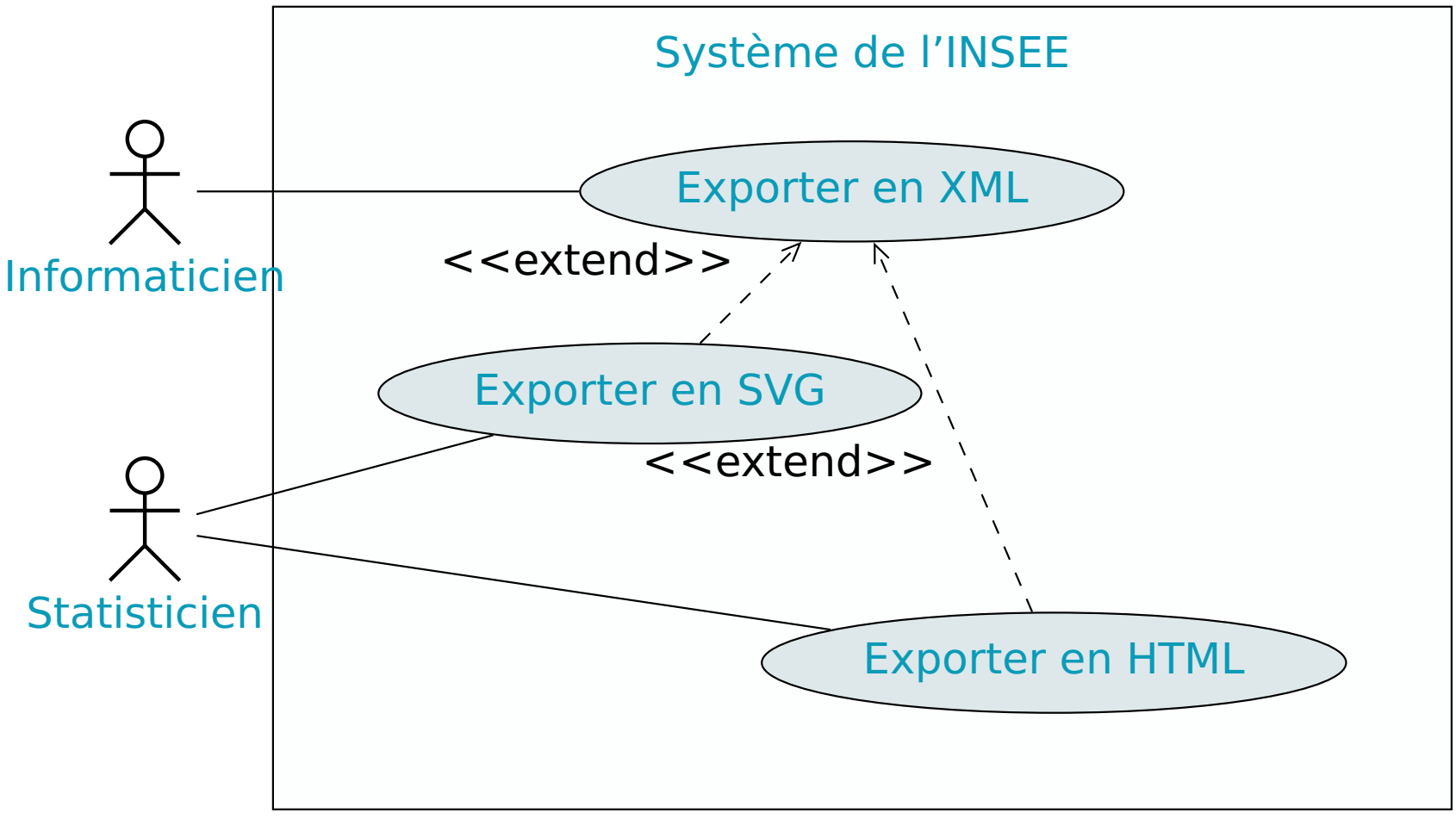
Extension

↪Un cas d'utilisation qui fait tout ce que fait un autre et plus.



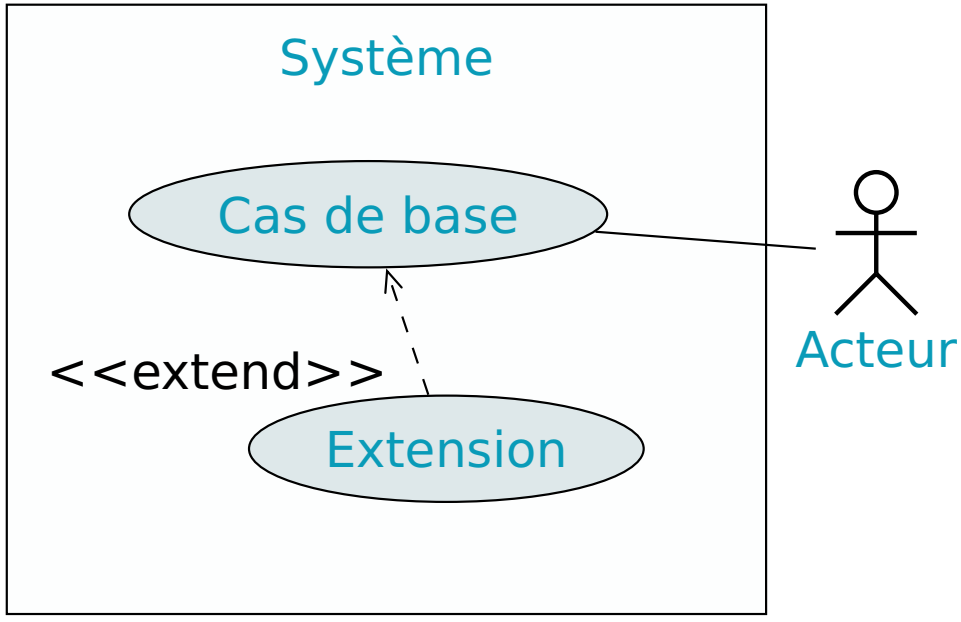
L'acteur est nécessairement impliqué dans le cas de base.

Extension — Exemple



Extension

↪Un cas d'utilisation qui fait tout ce que fait un autre et plus.



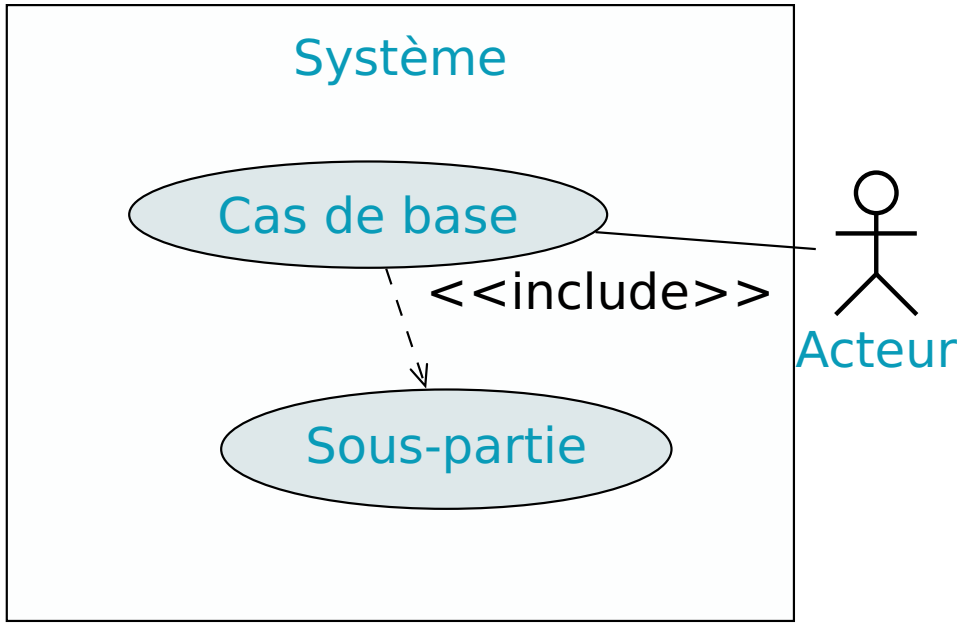
Attention

L'acteur a tout de même le droit de réaliser le cas étendu (attention tout le monde n'a pas la même lecture de ce diagramme).

Inclusion

↪ Une partie d'un autre cas d'utilisation qui

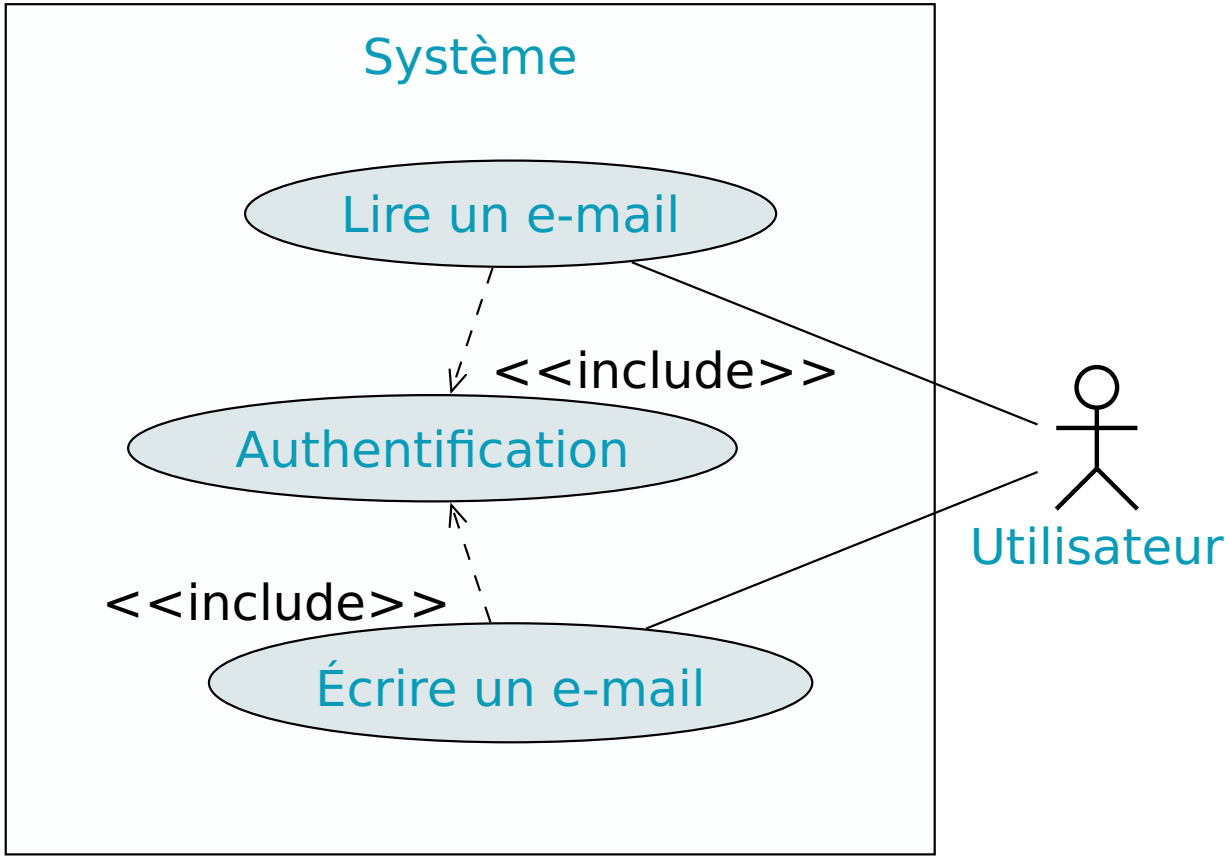
- n'est pas utile en soi (càd sans le cas contenant) ;
- peut être partagé entre divers cas contenants ;
- est nécessaire au bon fonctionnement du cas contenant.



Remarque

L'acteur est nécessairement impliqué dans la sous-partie.

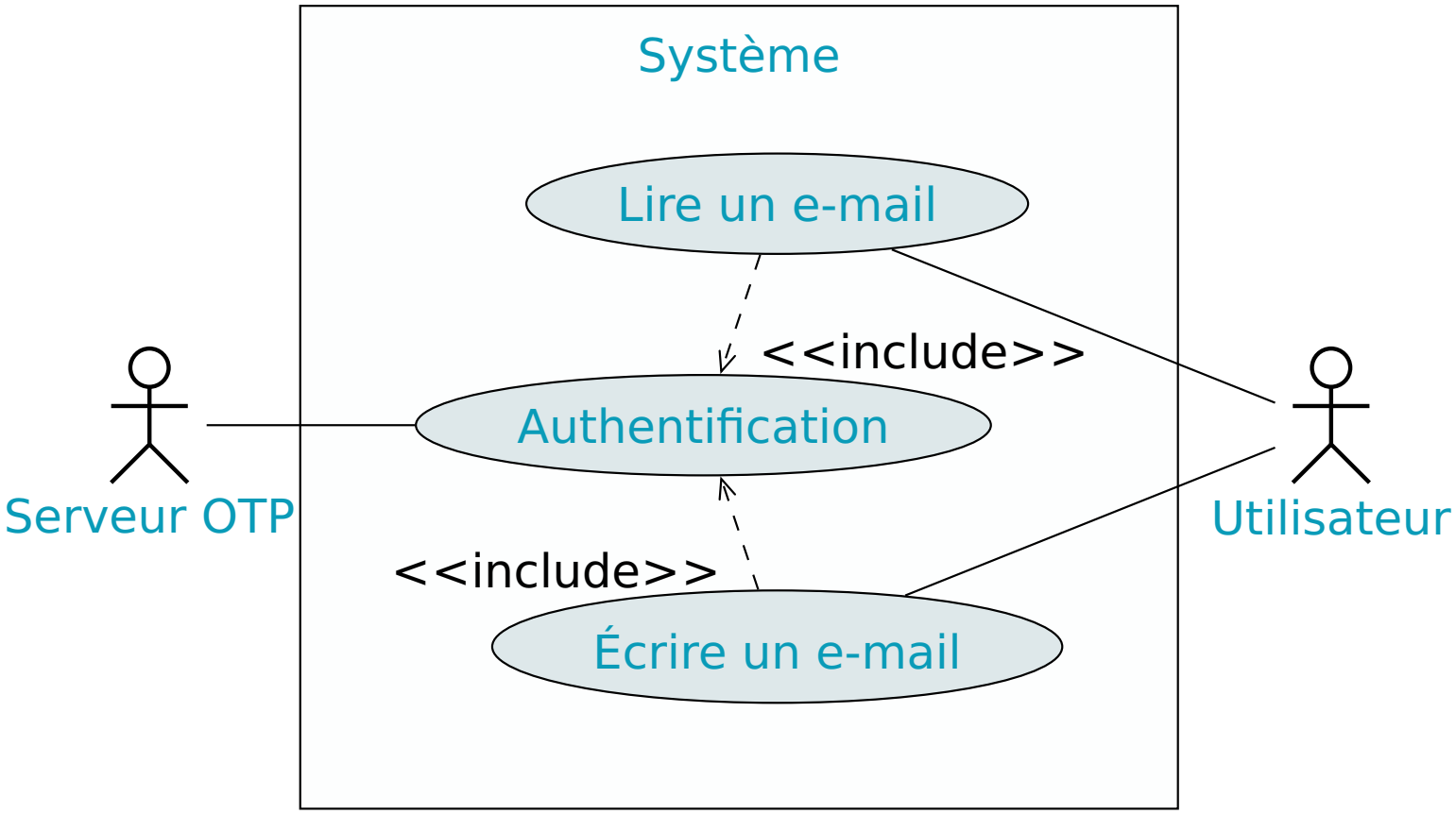
Inclusion — Exemple



Remarque

Le sous-cas peut avoir des associations propres.

Inclusion — Exemple



Remarque

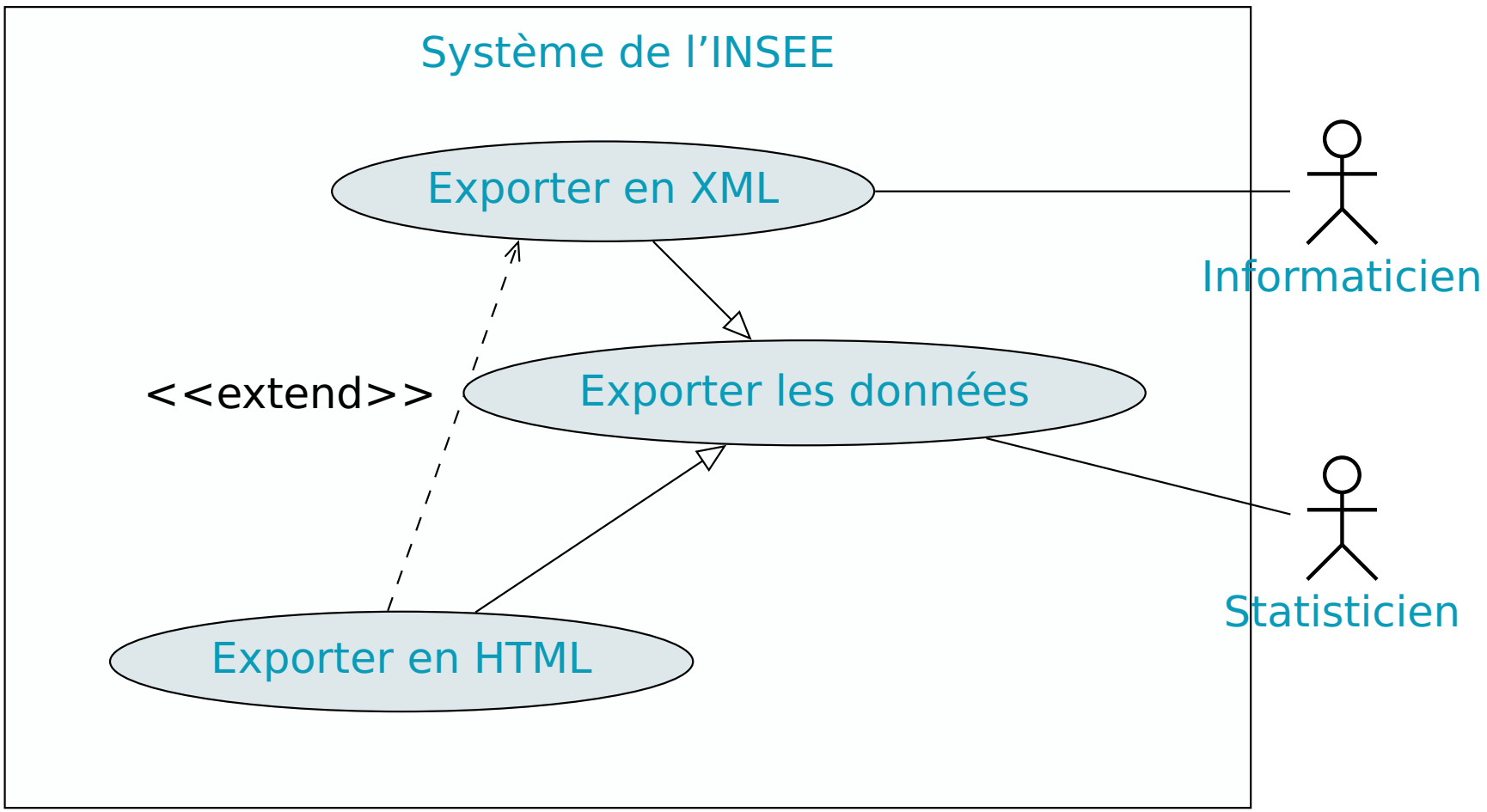
Le sous-cas peut avoir des associations propres.

Extension *vs* héritage

Pour éviter les confusions

Remarque

Une extension fait plus, un héritage fait différemment.



Extension *vs* Inclusion

Pour éviter les confusions

- Une extension et l'étendu sont des cas d'utilisation à part entière. Dans l'inclusion, l'inclus seul n'a pas de raison d'être.
- Un cas inclus est souvent partagé par plusieurs autres cas. Exemple canonique : S'authentifier.

Dans plusieurs cas, les deux seront acceptables. Il faut seulement être capable de le justifier.

Extension *vs* Inclusion

Pour éviter les confusions

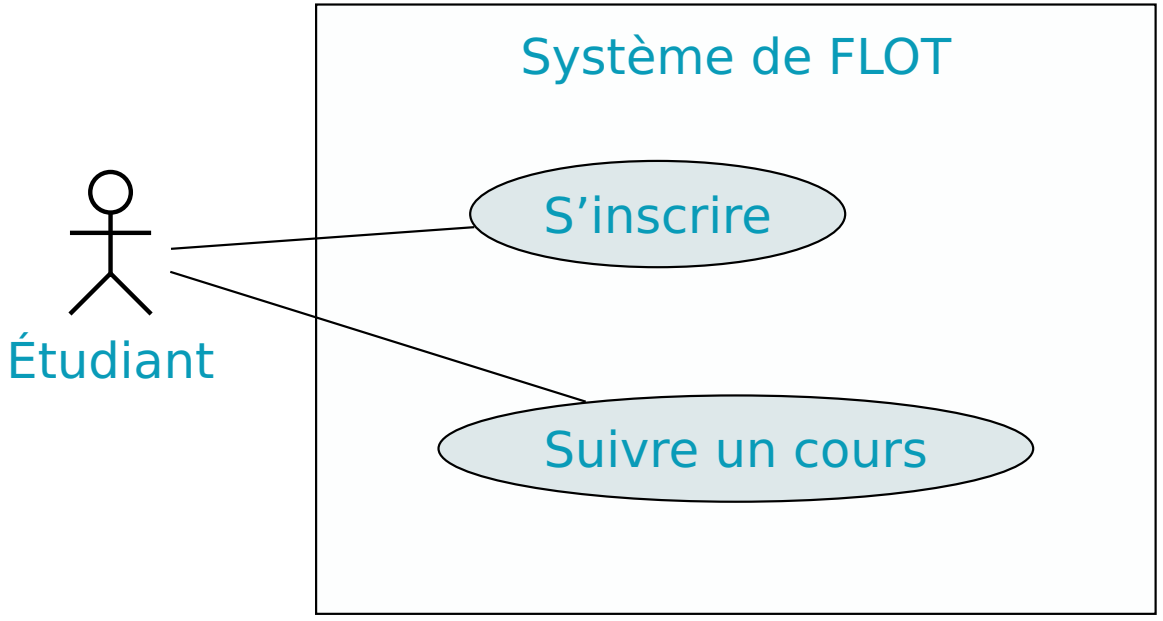
- Une extension et l'étendu sont des cas d'utilisation à part entière. Dans l'inclusion, l'inclus seul n'a pas de raison d'être.
- Un cas inclus est souvent partagé par plusieurs autres cas. Exemple canonique : S'authentifier.

Dans plusieurs cas, les deux seront acceptables. Il faut seulement être capable de le justifier.

Ce que ne dit pas un DCU

Pas d'information sur le temps

- Si les acteurs interagissent ou peuvent participer indépendamment à un cas commun.
- On ne fait pas apparaître l'enchaînement temporel de divers cas.



L’inscription à l’université

L’énoncé textuel

La scolarité de l’université s’occupe de la gestion des inscriptions. Cela consiste à inscrire les étudiants, modifier leurs inscriptions, voire les supprimer. Les inscriptions peuvent être particulières s’il s’agit d’une inscription en double cursus ou en alternance. Les inscriptions en alternance ont besoin de l’accord du CFA, et ce cas particulier est de la responsabilité de certains membres de la scolarité. L’inscription en double cursus peut être gérée par n’importe quel agent de la scolarité. Une fois l’inscription terminée, il faut imprimer la carte d’étudiant, à l’aide d’une imprimante spéciale. Que ce soit pour inscrire, modifier ou supprimer une inscription, la scolarité doit pouvoir vérifier le paiement des droits d’inscription.

Les différentes étapes de la construction du DCU

- Identifier les acteurs (et ce qui est dans le système).
- Identifier les héritages entre acteurs.
- Identifier les cas d'utilisation.
- Associer les cas d'utilisations et les acteurs.
- Trouver les dépendances (inclusion/extension) entre les cas d'utilisation.
- Supprimer les associations superflues.
- Factoriser par héritage.

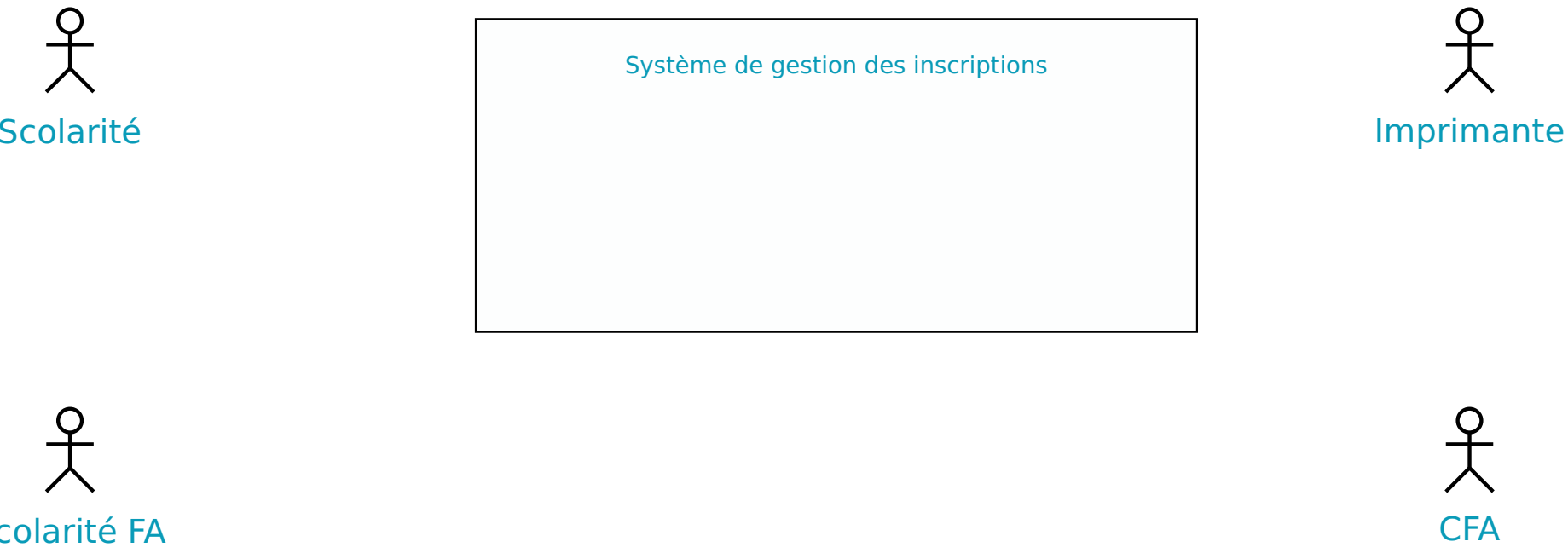
L’inscription à l’université

L’énoncé textuel

La scolarité de l’université s’occupe de la gestion des inscriptions. Cela consiste à inscrire les étudiants, modifier leurs inscriptions, voire les supprimer. Les inscriptions peuvent être particulières s’il s’agit d’une inscription en double cursus ou en alternance. Les inscriptions en alternance ont besoin de l’accord du CFA, et ce cas particulier est de la responsabilité de certains membres de la scolarité. L’inscription en double cursus peut être gérée par n’importe quel agent de la scolarité. Une fois l’inscription terminée, il faut imprimer la carte d’étudiant, à l’aide d’une imprimante spéciale. Que ce soit pour inscrire, modifier ou supprimer une inscription, la scolarité doit pouvoir vérifier le paiement des droits d’inscription.

L'inscription à l'université

Diagramme de cas d'utilisation



Les différentes étapes de la construction du DCU

- Identifier les acteurs (et ce qui est dans le système).
- Identifier les héritages entre acteurs.
- Identifier les cas d'utilisation.
- Associer les cas d'utilisations et les acteurs.
- Trouver les dépendances (inclusion/extension) entre les cas d'utilisation.
- Supprimer les associations superflues.
- Factoriser par héritage.

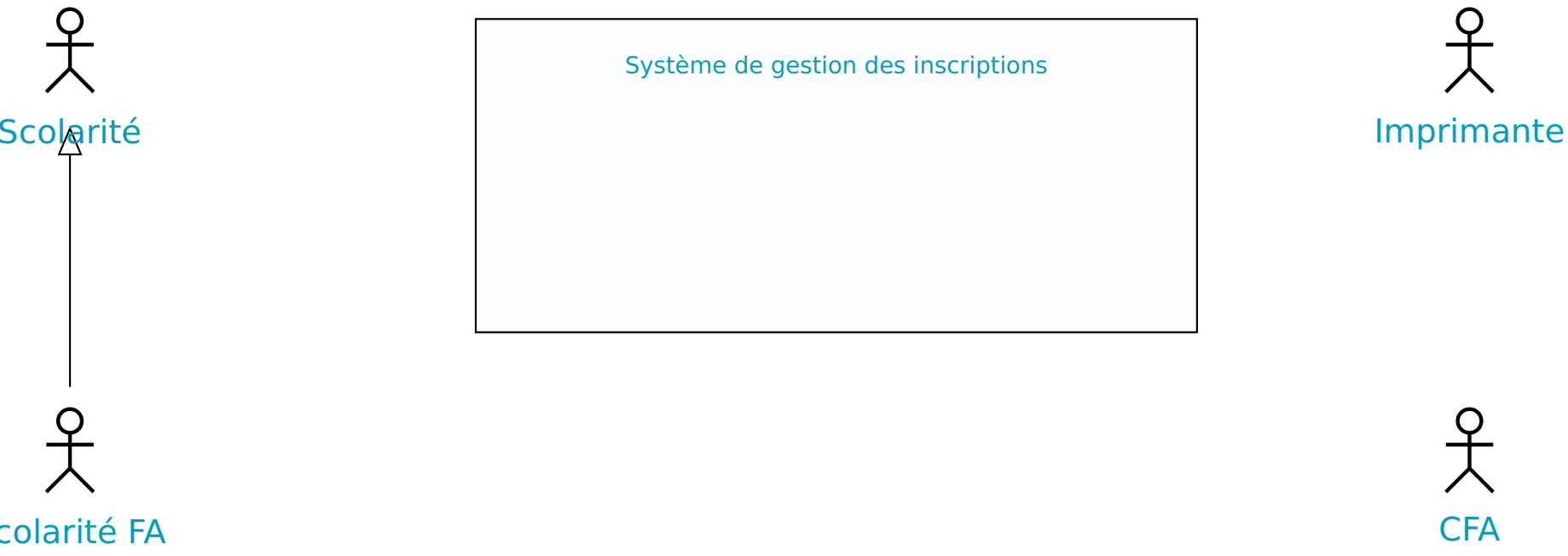
L’inscription à l’université

L’énoncé textuel

La scolarité de l’université s’occupe de la gestion des inscriptions. Cela consiste à inscrire les étudiants, modifier leurs inscriptions, voire les supprimer. Les inscriptions peuvent être particulières s’il s’agit d’une inscription en double cursus ou en alternance. Les inscriptions en alternance ont besoin de l’accord du CFA, et ce cas particulier est de la responsabilité de certains membres de la scolarité. L’inscription en double cursus peut être gérée par n’importe quel agent de la scolarité. Une fois l’inscription terminée, il faut imprimer la carte d’étudiant, à l’aide d’une imprimante spéciale. Que ce soit pour inscrire, modifier ou supprimer une inscription, la scolarité doit pouvoir vérifier le paiement des droits d’inscription.

L'inscription à l'université

Diagramme de cas d'utilisation



Les différentes étapes de la construction du DCU

- Identifier les acteurs (et ce qui est dans le système).
- Identifier les héritages entre acteurs.
- Identifier les cas d'utilisation.
- Associer les cas d'utilisations et les acteurs.
- Trouver les dépendances (inclusion/extension) entre les cas d'utilisation.
- Supprimer les associations superflues.
- Factoriser par héritage.

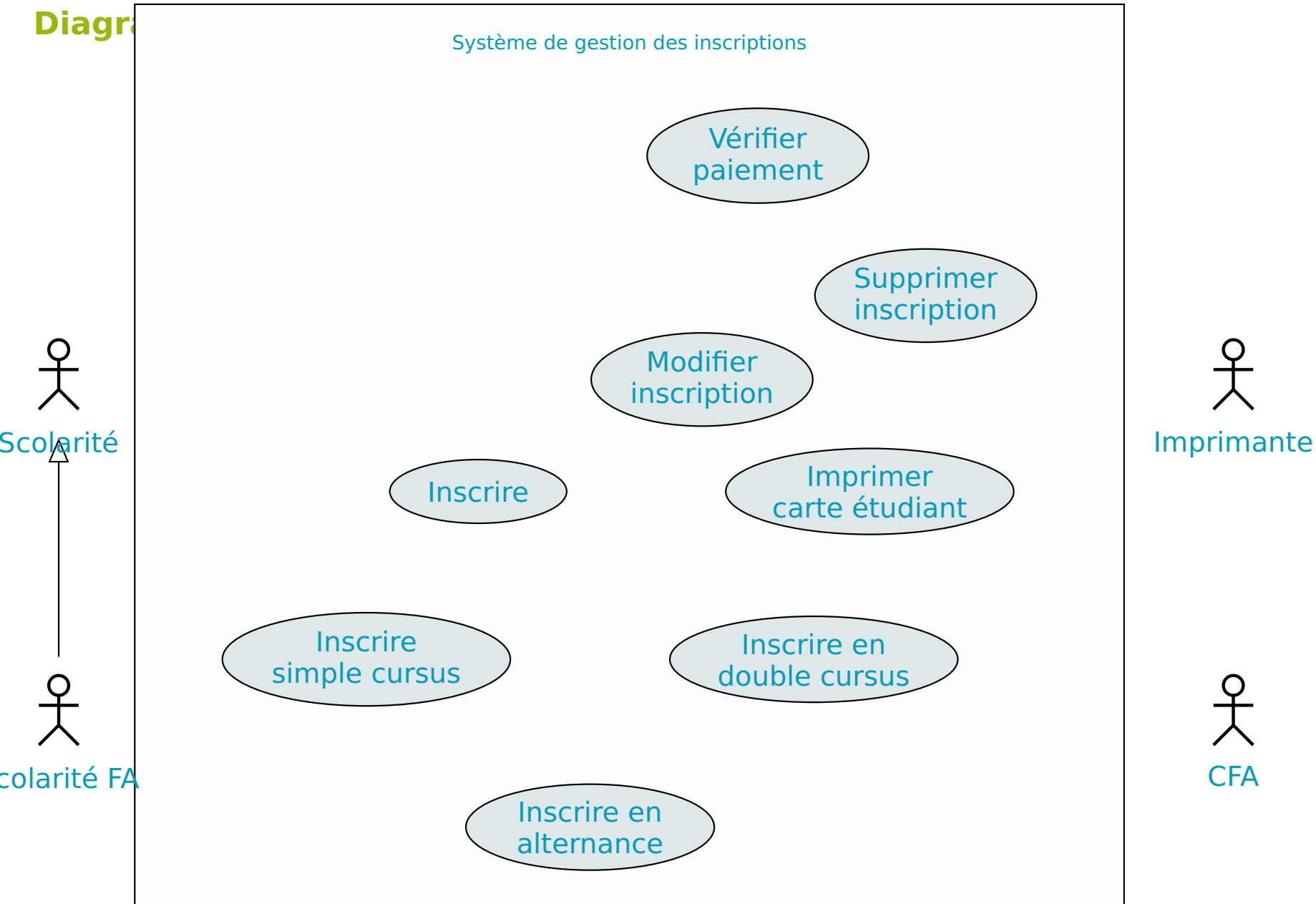
L’inscription à l’université

L’énoncé textuel

La scolarité de l’université s’occupe de la gestion des inscriptions. Cela consiste à inscrire les étudiants, modifier leurs inscriptions, voire les supprimer. Les inscriptions peuvent être particulières s’il s’agit d’une inscription en double cursus ou en alternance. Les inscriptions en alternance ont besoin de l’accord du CFA, et ce cas particulier est de la responsabilité de certains membres de la scolarité. L’inscription en double cursus peut être gérée par n’importe quel agent de la scolarité. Une fois l’inscription terminée, il faut imprimer la carte d’étudiant, à l’aide d’une imprimante spéciale. Que ce soit pour inscrire, modifier ou supprimer une inscription, la scolarité doit pouvoir vérifier le paiement des droits d’inscription.

L'inscription à l'université

Diagr



Les différentes étapes de la construction du DCU

- Identifier les acteurs (et ce qui est dans le système).
- Identifier les héritages entre acteurs.
- Identifier les cas d'utilisation.
- Associer les cas d'utilisations et les acteurs.
- Trouver les dépendances (inclusion/extension) entre les cas d'utilisation.
- Supprimer les associations superflues.
- Factoriser par héritage.

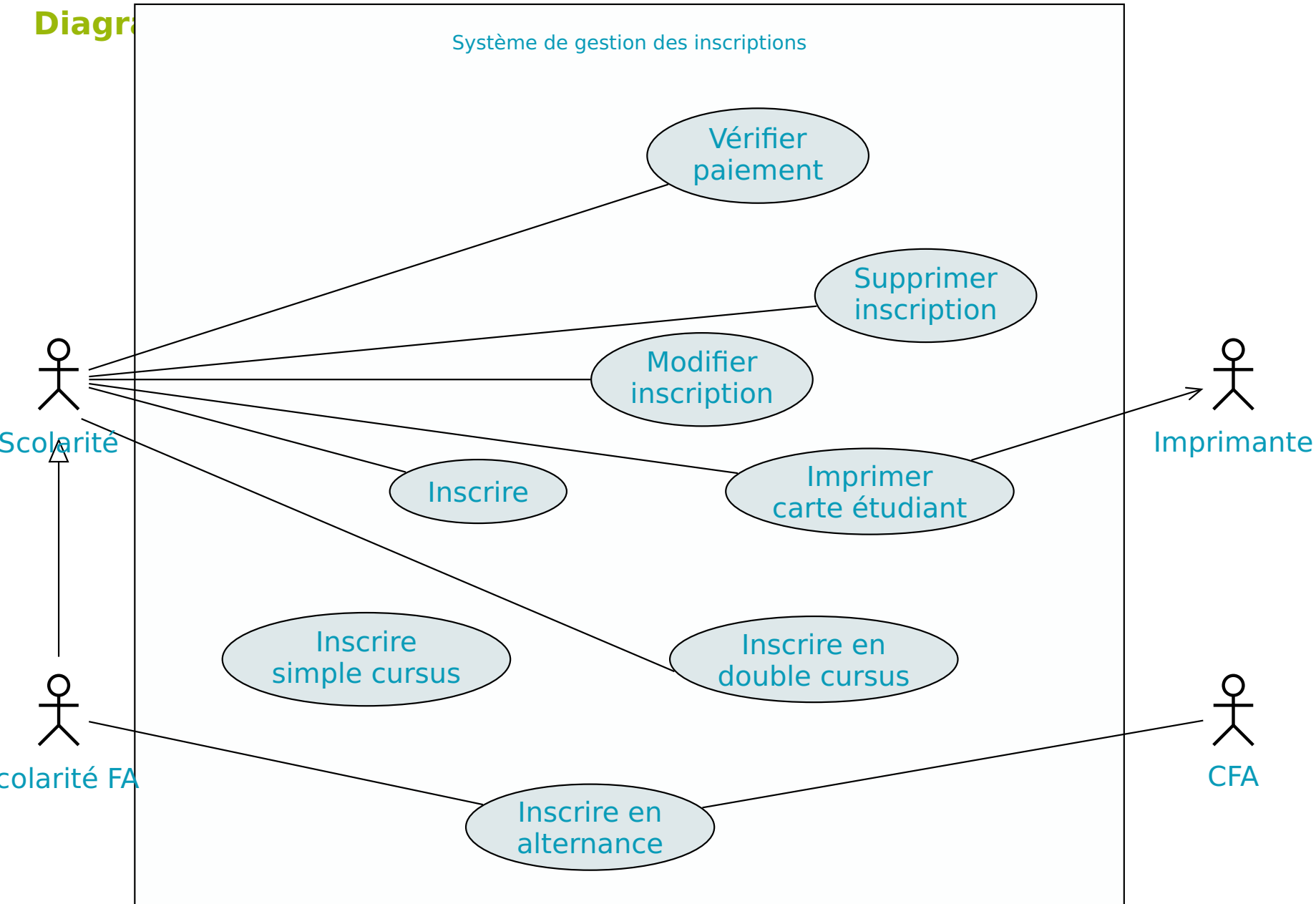
L’inscription à l’université

L’énoncé textuel

La scolarité de l’université s’occupe de la gestion des inscriptions. Cela consiste à inscrire les étudiants, modifier leurs inscriptions, voire les supprimer. Les inscriptions peuvent être particulières s’il s’agit d’une inscription en double cursus ou en alternance. Les inscriptions en alternance ont besoin de l’accord du CFA, et ce cas particulier est de la responsabilité de certains membres de la scolarité. L’inscription en double cursus peut être gérée par n’importe quel agent de la scolarité. Une fois l’inscription terminée, il faut imprimer la carte d’étudiant, à l’aide d’une imprimante spéciale. Que ce soit pour inscrire, modifier ou supprimer une inscription, la scolarité doit pouvoir vérifier le paiement des droits d’inscription.

L'inscription à l'université

Diagr

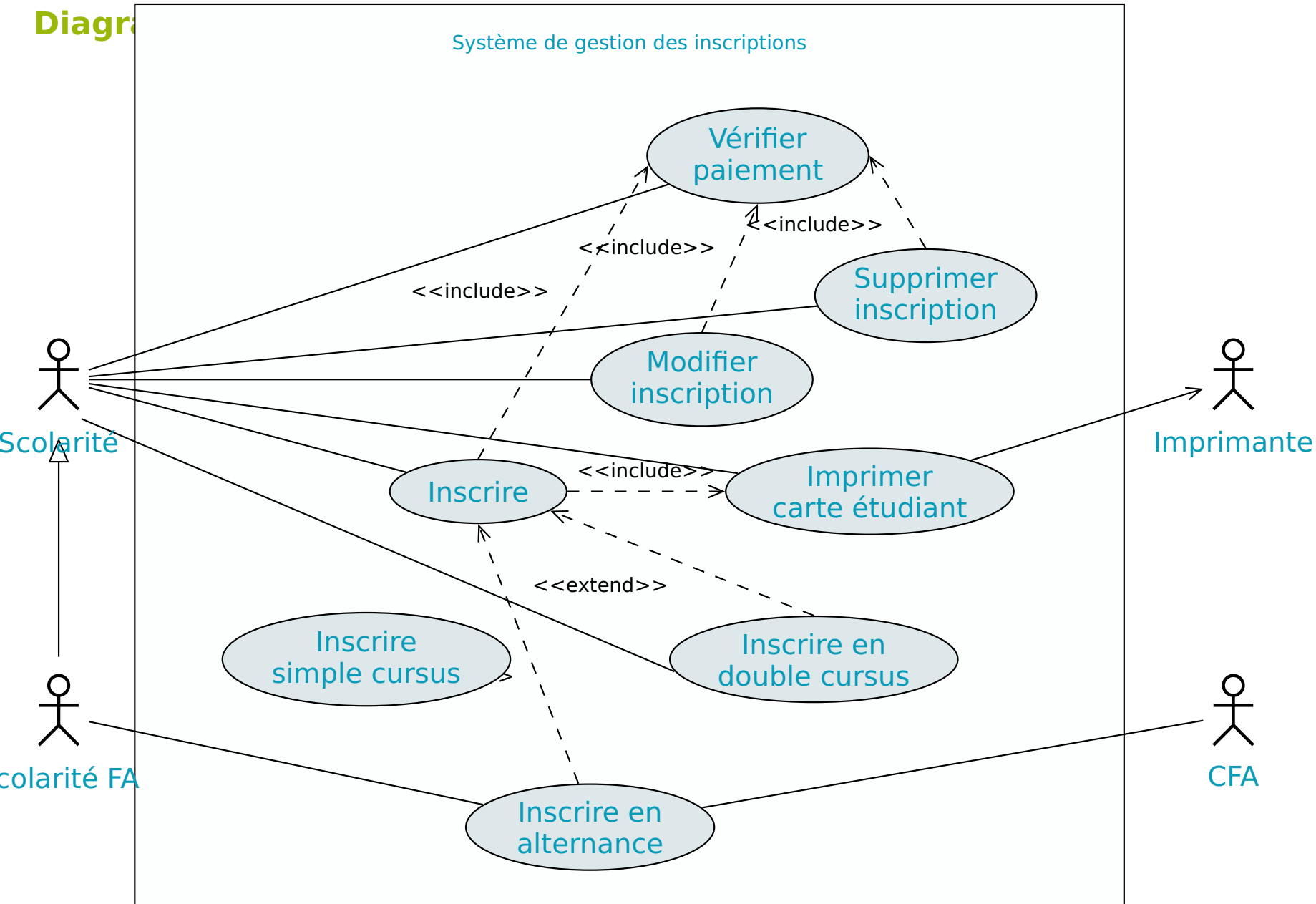


Les différentes étapes de la construction du DCU

- Identifier les acteurs (et ce qui est dans le système).
- Identifier les héritages entre acteurs.
- Identifier les cas d'utilisation.
- Associer les cas d'utilisations et les acteurs.
- Trouver les dépendances (inclusion/extension) entre les cas d'utilisation.
- Supprimer les associations superflues.
- Factoriser par héritage.

L'inscription à l'université

Diagr

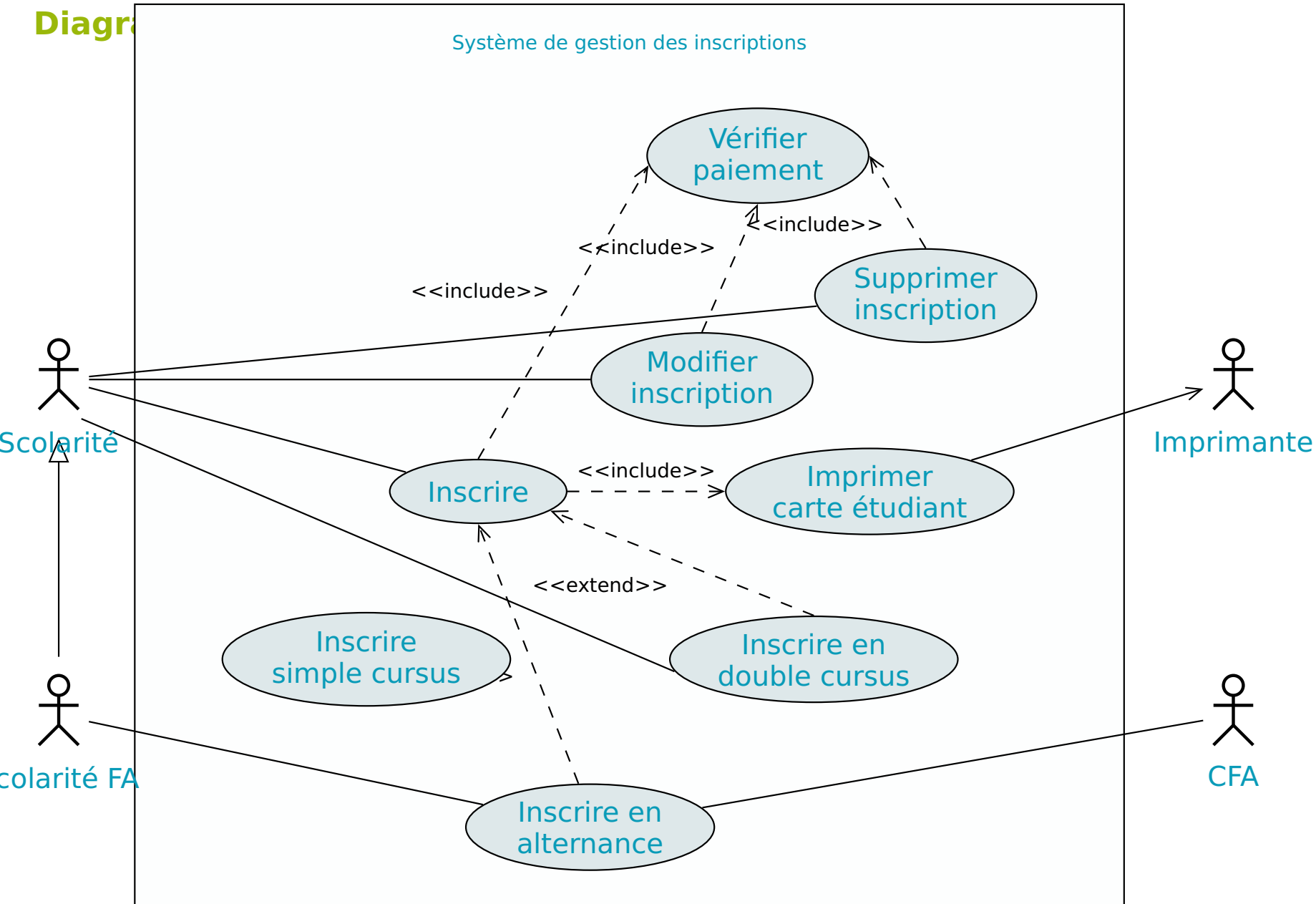


Les différentes étapes de la construction du DCU

- Identifier les acteurs (et ce qui est dans le système).
- Identifier les héritages entre acteurs.
- Identifier les cas d'utilisation.
- Associer les cas d'utilisations et les acteurs.
- Trouver les dépendances (inclusion/extension) entre les cas d'utilisation.
- Supprimer les associations superflues.
- Factoriser par héritage.

L'inscription à l'université

Diagr

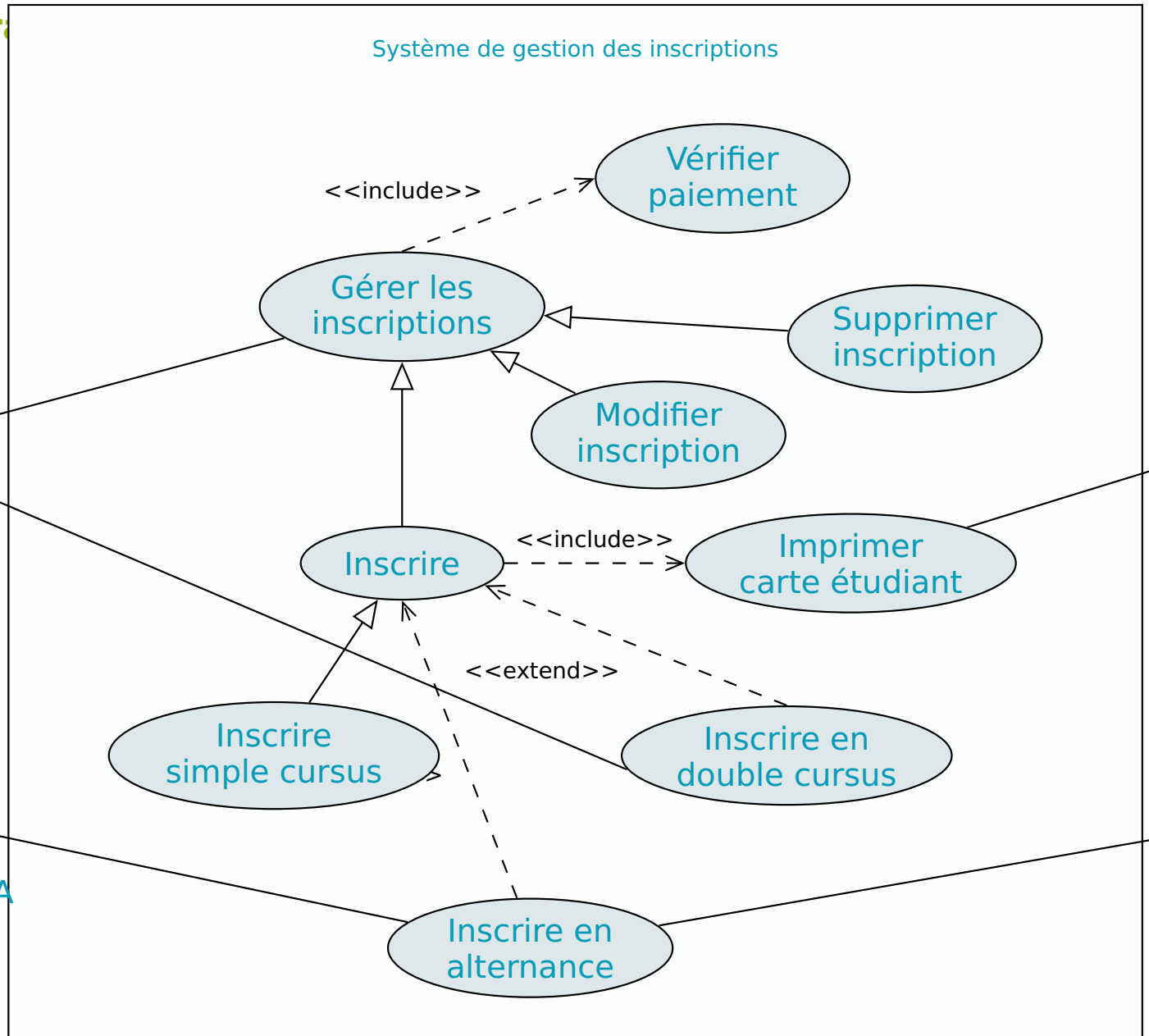


Les différentes étapes de la construction du DCU

- Identifier les acteurs (et ce qui est dans le système).
- Identifier les héritages entre acteurs.
- Identifier les cas d'utilisation.
- Associer les cas d'utilisations et les acteurs.
- Trouver les dépendances (inclusion/extension) entre les cas d'utilisation.
- Supprimer les associations superflues.
- Factoriser par héritage.

L'inscription à l'université

Diagr



Rappel Vocabulaire

Classe

essentiellement un type général (un moule pour fabriquer des objets) avec

1. une structure (des attributs)
2. un comportement (des méthodes – *stricto sinsu* on doit dire opération en UML)

Objet

Un objet est une instance d'une classe (est sorti du moule de la classe). En pratique, c'est essentiellement un morceau de la mémoire qui contient les valeurs des attributs.

Mémoire en java

Les objets sont stockés dans le tas (heap). En cours d'exécution, si une méthode a besoin d'accéder à un objet car c'est la valeur d'une variable, la pile (stack) va avoir un emplacement correspondant qui aura pour valeur la référence de cet objet dans le tas.

Deux types de diagrammes

On peut soit parler de manière générale de ce qu'on veut représenter c'est-à-dire des classes et de leurs relations
↪Diagramme de classe.

ou bien on parle d'exemples concrets avec des objets concrets et de leurs relations à un instant donné de l'exécution de notre programme.
↪Diagramme d'objets.

Visibilité

L’abstraction proposée par les classes permet entre autres en java d’exposer seulement une partie de ces attributs et méthodes à d’autres objet. Il existe un mécanisme de droit assez fin dans l’esprit du ugo des fichiers unix.

Notation UML

- – Privé : visible uniquement depuis l'intérieur de l'objet.
- \+ Public : visible depuis l'extérieur.
- # Protégé : visible depuis l'intérieur et depuis l'intérieur d'objets de classe en **héritant**.
- ~ Paquetage : visible seulement depuis les autres objets définis dans le paquetage.

Recommandation

Les attributs seront privés (à la rigueur protégés), les opérations seront publiques.

Éléments de classe

Propriété « statique »*(static)*

L'élément est relatif à la classe et non aux objets instance de cette classe.

- Concerne les attributs comme les opérations.
- Pour les attributs, utile pour spécifier des constantes.
- Pour les opérations, utile pour spécifier des opérations qui sont logiquement dans cette classe mais qui n'agissent pas sur des objets : sous-routines, comparaisons d'objets de cette classe. . .
- Un élément **static** apparaît souligné.

Remarque

Ne pas confondre avec les clefs primaires utilisées pour la modélisation de bases de données.

Autres propriétés des attributs

Dérivé Qui peut être calculé.

- Concerne uniquement les attributs.
- Indiqué par un symbole /
- Utile si l'attribut est important, utilisé souvent, calculable mais à grand coût...
- Exemple : `-/ âge` est calculable depuis `- dateNaissance`.

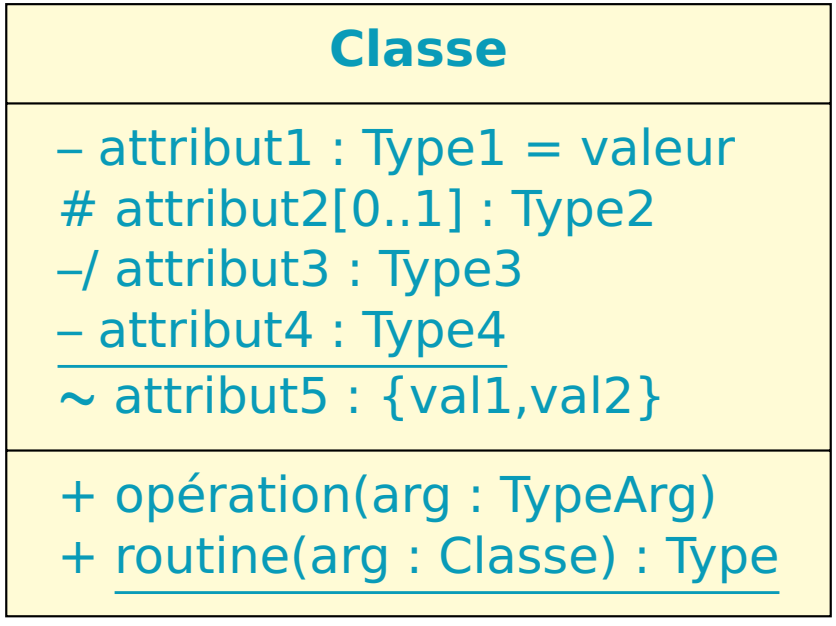
Cardinalité Les attributs peuvent avoir une multiplicité.

- Exemple : `- autresPrenoms[0..3]`.
- Plus d'infos sur les cardinalités lorsqu'on parlera des associations.

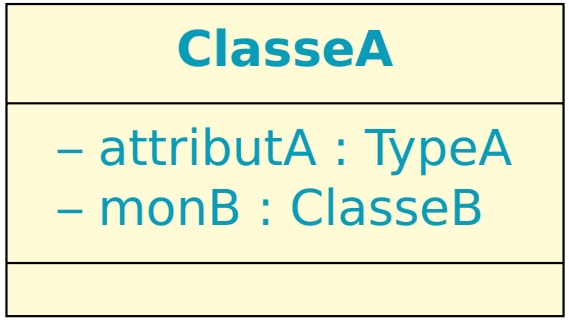
Valeur par défaut Indiquée par `= valeur`.

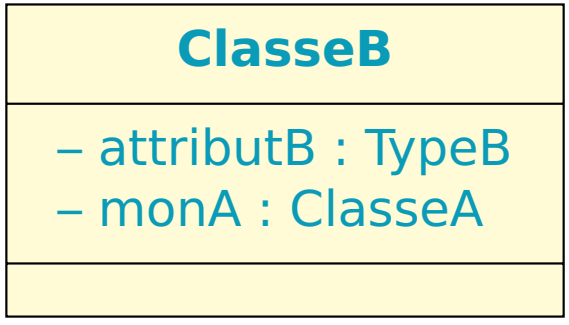
Type énumération Type (anonyme) qui a un nombre fini de valeurs possibles : `- jour: {'Lundi',...}`.

Représentation graphique des classes



Association simple

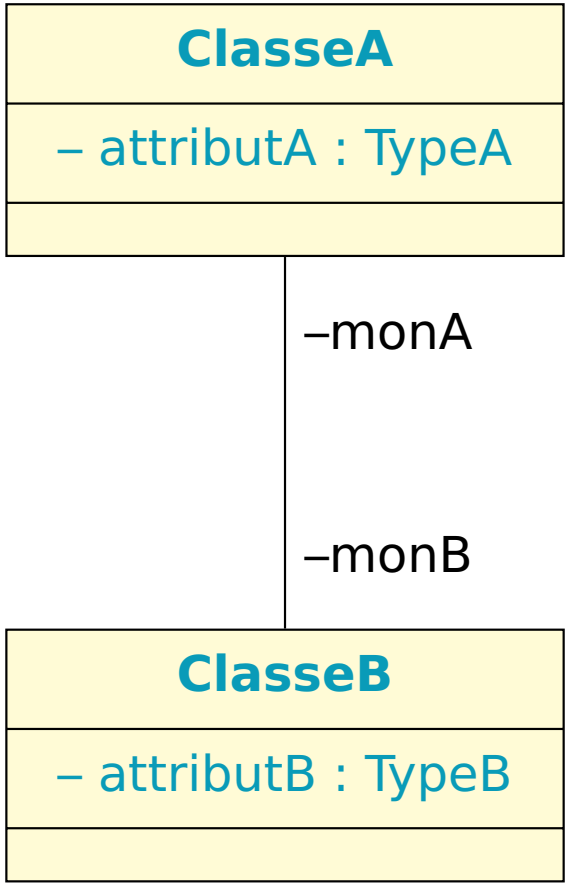




- Lorsqu'un attribut a pour type une classe **présente sur le diagramme**, on explicite cette relation par une association.
- On peut nommer l'association pour expliciter la nature de la relation.
- Une association fournit donc ces attributs implicites dont le nom est fourni par les rôles.

⇝ Les rôles ont une **visibilité**.

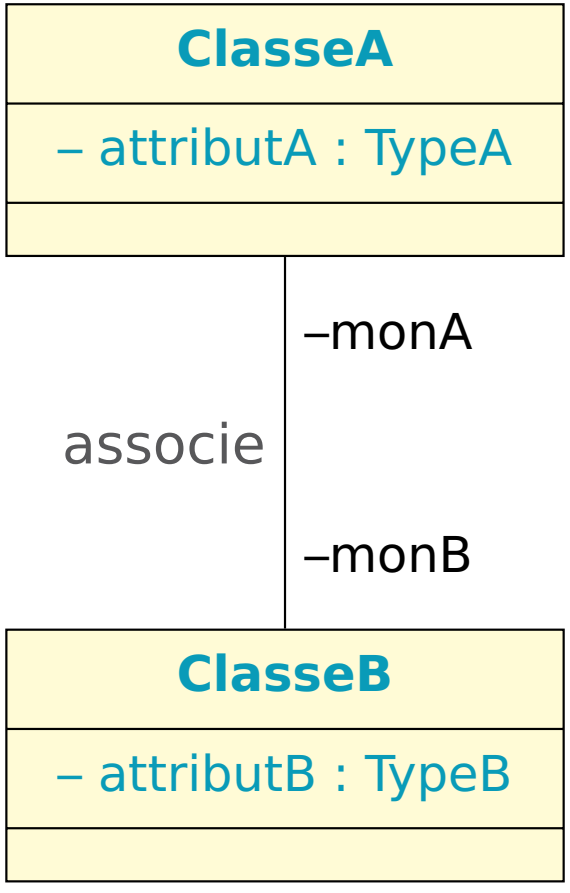
Association simple



- Lorsqu'un attribut a pour type une classe **présente sur le diagramme**, on explicite cette relation par une association.
- On peut nommer l'association pour expliciter la nature de la relation.
- Une association fournit donc ces attributs implicites dont le nom est fourni par les rôles.

⇝ Les rôles ont une **visibilité**.

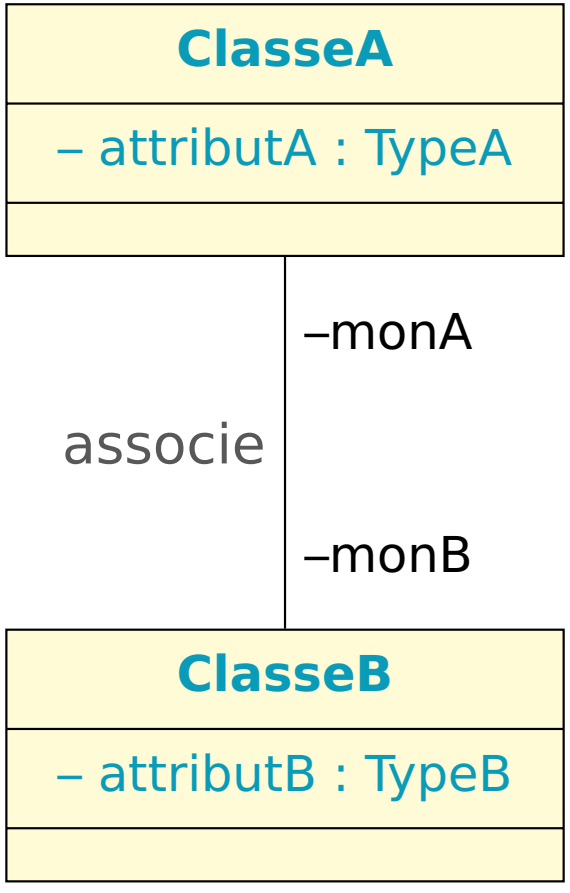
Association simple



- Lorsqu'un attribut a pour type une classe **présente sur le diagramme**, on explicite cette relation par une association.
- On peut nommer l'association pour expliciter la nature de la relation.
- Une association fournit donc ces attributs implicites dont le nom est fourni par les rôles.

⇝ Les rôles ont une **visibilité**.

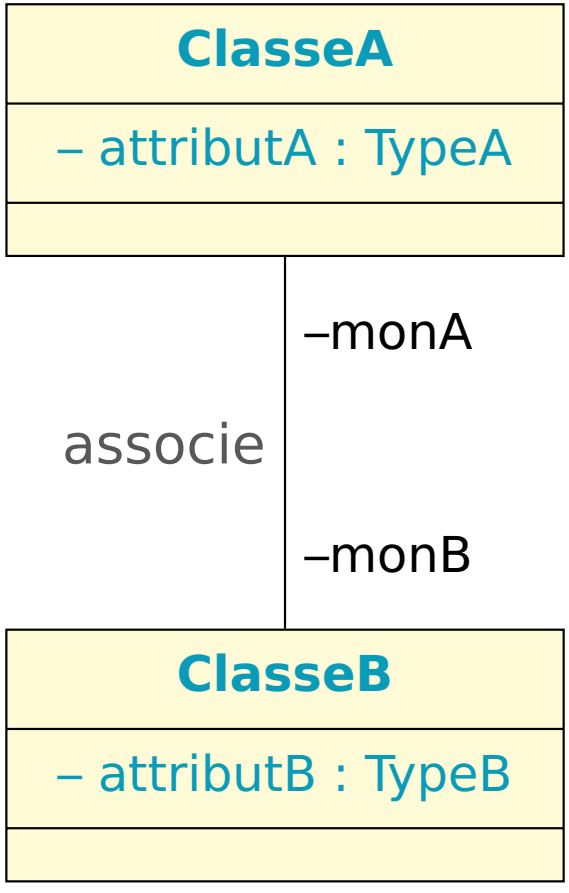
Association simple



- Lorsqu'un attribut a pour type une classe **présente sur le diagramme**, on explicite cette relation par une association.
- On peut nommer l'association pour expliciter la nature de la relation.
- Une association fournit donc ces attributs implicites dont le nom est fourni par les rôles.

⇝ Les rôles ont une **visibilité**.

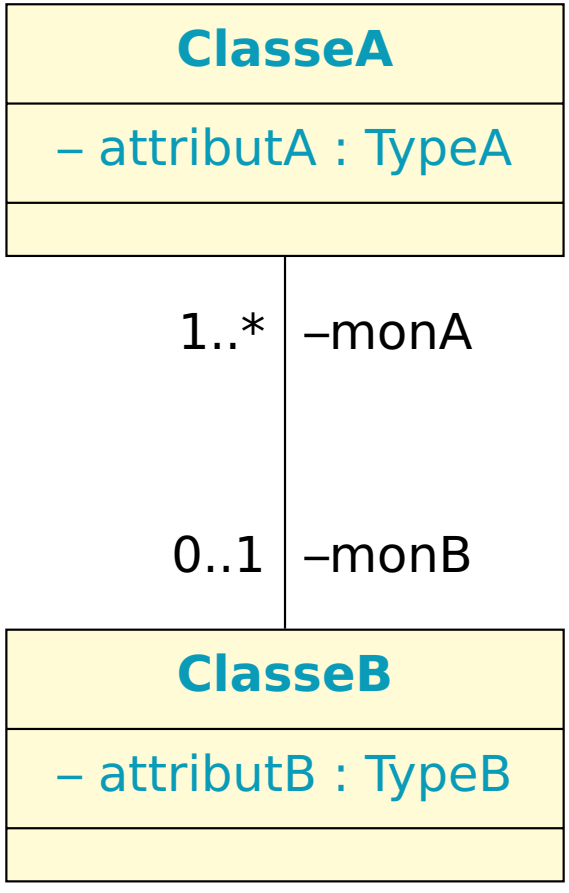
Association simple



- Lorsqu'un attribut a pour type une classe **présente sur le diagramme**, on explicite cette relation par une association.
- On peut nommer l'association pour expliciter la nature de la relation.
- Une association fournit donc ces attributs implicites dont le nom est fourni par les rôles.

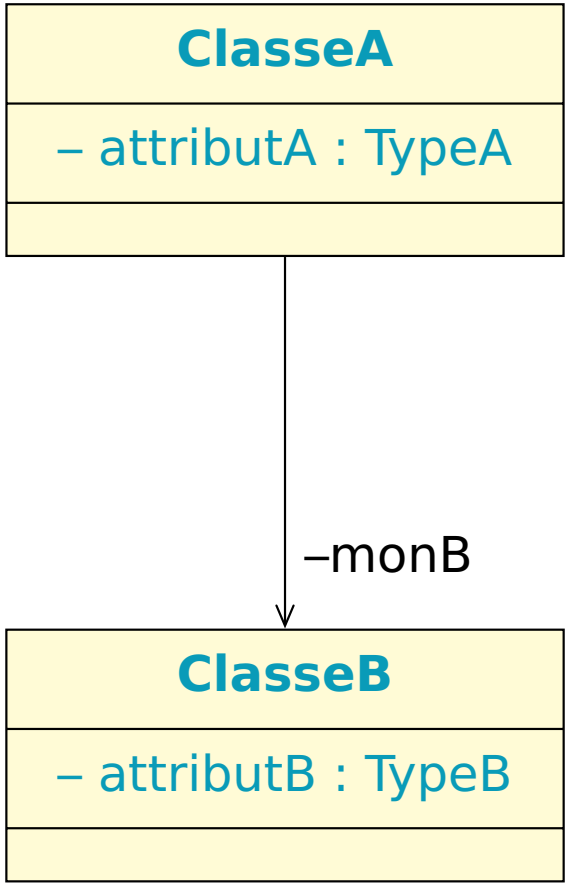
⇝ Les rôles ont une **visibilité**.

Cardinalités



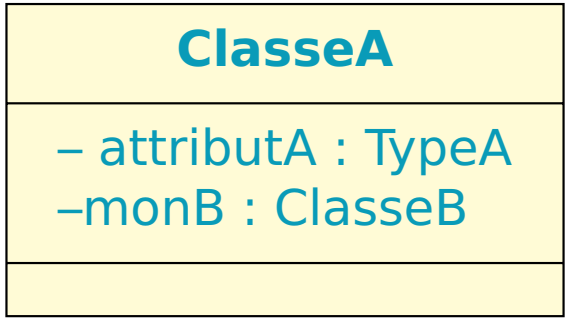
- Indique le nombre d'objets qui peuvent être ainsi associés.
- Syntaxe : intervalle *x..y*, point *x*, arbitrairement *.
- Les plus utilisés :
  - **1** Unique (par défaut).
  - **0..1** Optionnel.
  - **1..*** Au moins un.
  - * Autant que l'on veut (potentiellement 0).
- Les **rôles** s'entendent comme l'ensemble des objets associés.

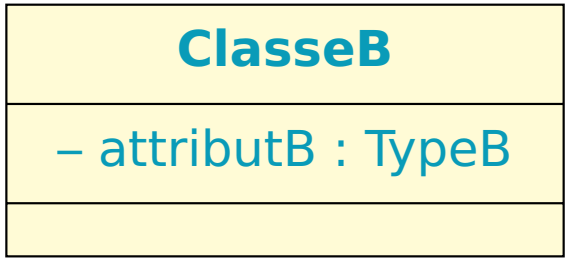
Navigabililité



- Indique que l'accès ne se fait pas dans les deux sens.
- On remarque qu'il n'y a donc plus de rôle **monA**.
- Par défaut : la navigabilité est bidirectionnelle.

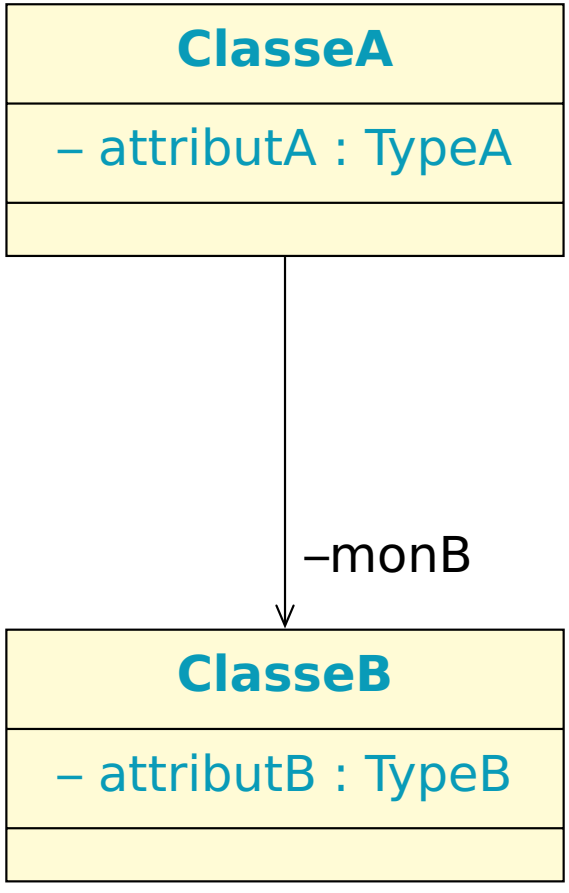
Navigabililité





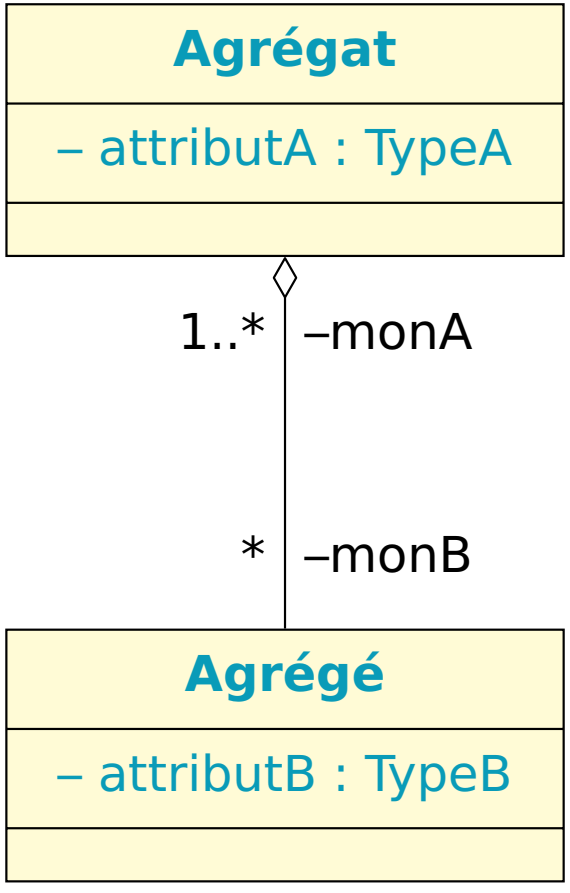
- Indique que l'accès ne se fait pas dans les deux sens.
- On remarque qu'il n'y a donc plus de rôle **monA**.
- Par défaut : la navigabilité est bidirectionnelle.

Navigabililité



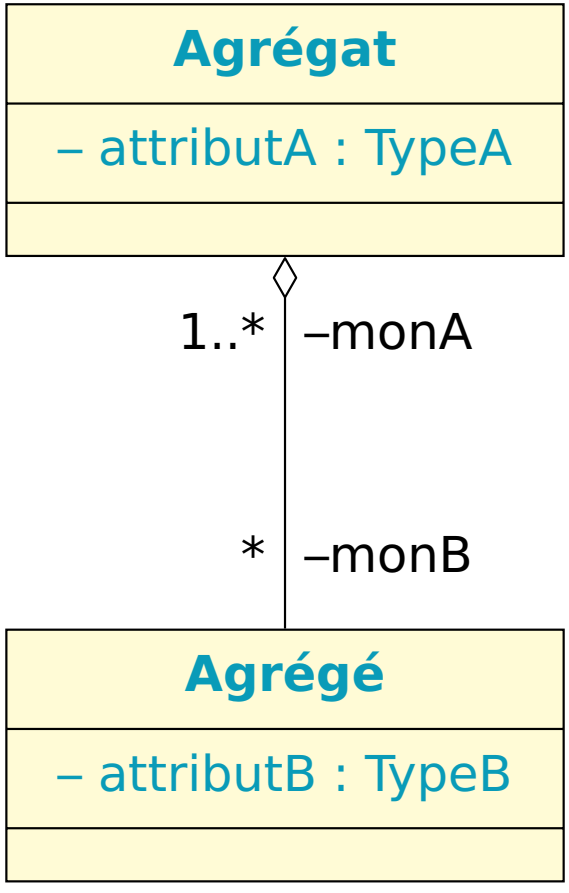
- Indique que l'accès ne se fait pas dans les deux sens.
- On remarque qu'il n'y a donc plus de rôle **monA**.
- Par défaut : la navigabilité est bidirectionnelle.

Agrégation



- Un raccourci pour la notion d'ensemble.
- L'agrégat est formée des objets agrégés.
- Un objet peut appartenir à plusieurs agrégats.
- Détruire un agrégat ne détruit pas les agrégés.

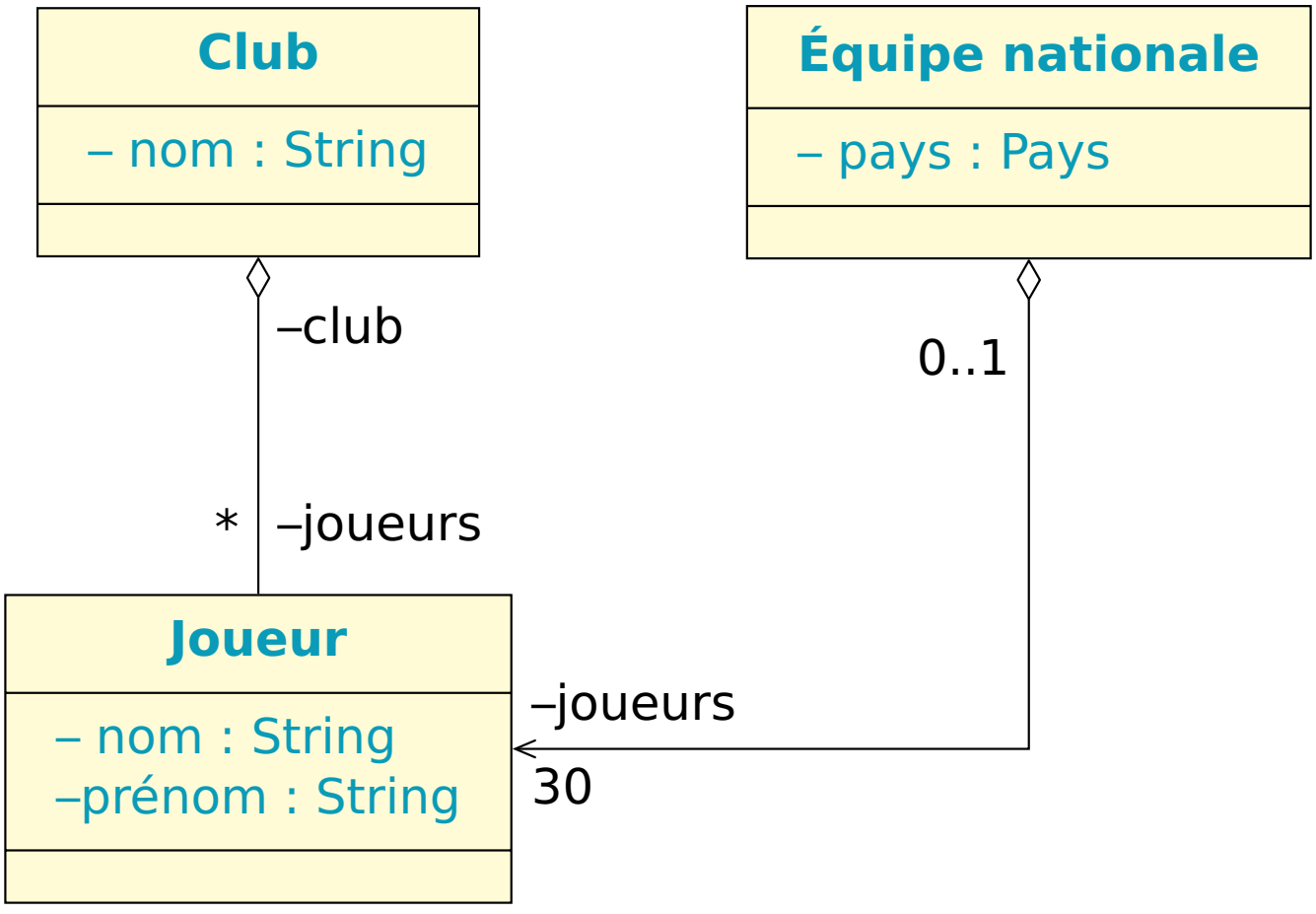
Agrégation



- Un raccourci pour la notion d'ensemble.
- L'agrégat est formée des objets agrégés.
- Un objet peut appartenir à plusieurs agrégats.
- Détruire un agrégat ne détruit pas les agrégés.

Agrégation — Exemple

Le diagramme de classes



Agrégation — Exemple

Le diagramme objets

stadeToulousain: **Club**
nom = « Stade Toulousain » joueurs = {dusautoir, tolofua, . . .}

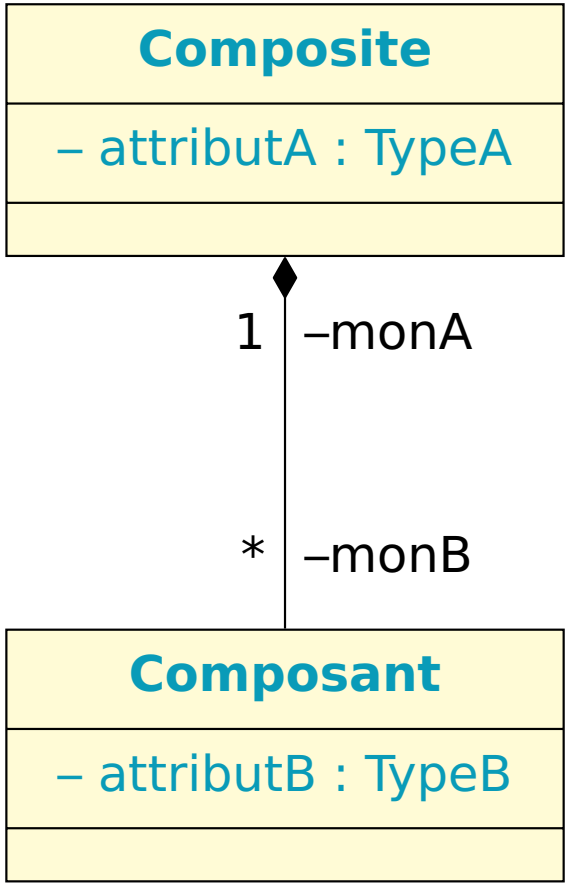
dusautoir: **Joueur**
nom = « Dusautoir » prénom = « Thierry » club = stadeToulousain

xvFrance: **Équipe nationale**
pays = france joueurs = {dusautoir, kayser, . . .}

Remarque

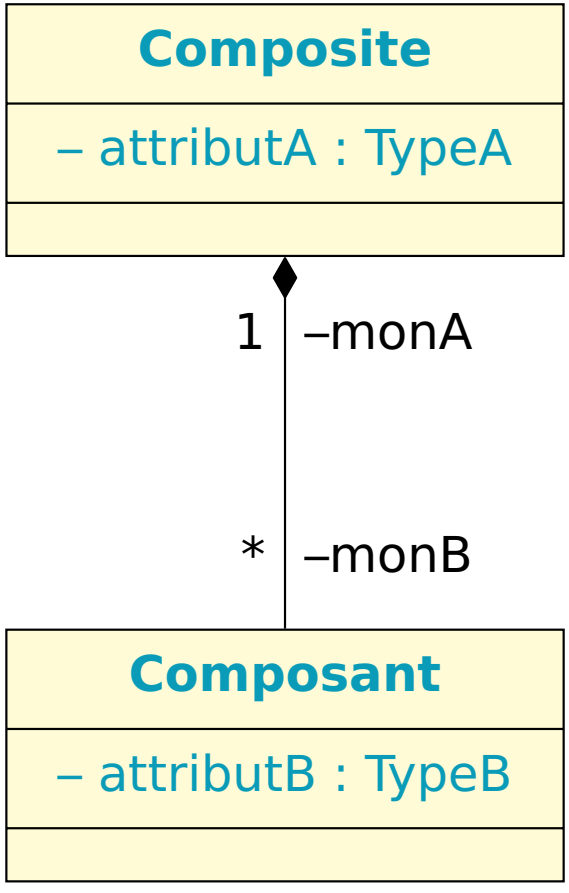
- Détruire une équipe ne détruit pas un joueur.
- Un joueur peut appartenir à plusieurs Clubs.

Composition



- Un raccourci pour la notion de partie.
- Le composite est formée des objets composants.
- Un **objet** ne peut appartenir qu'à un seul composite : cardinalité 1 ou 0..1 seulement.
- Détruire un composite signifie détruire ses composants

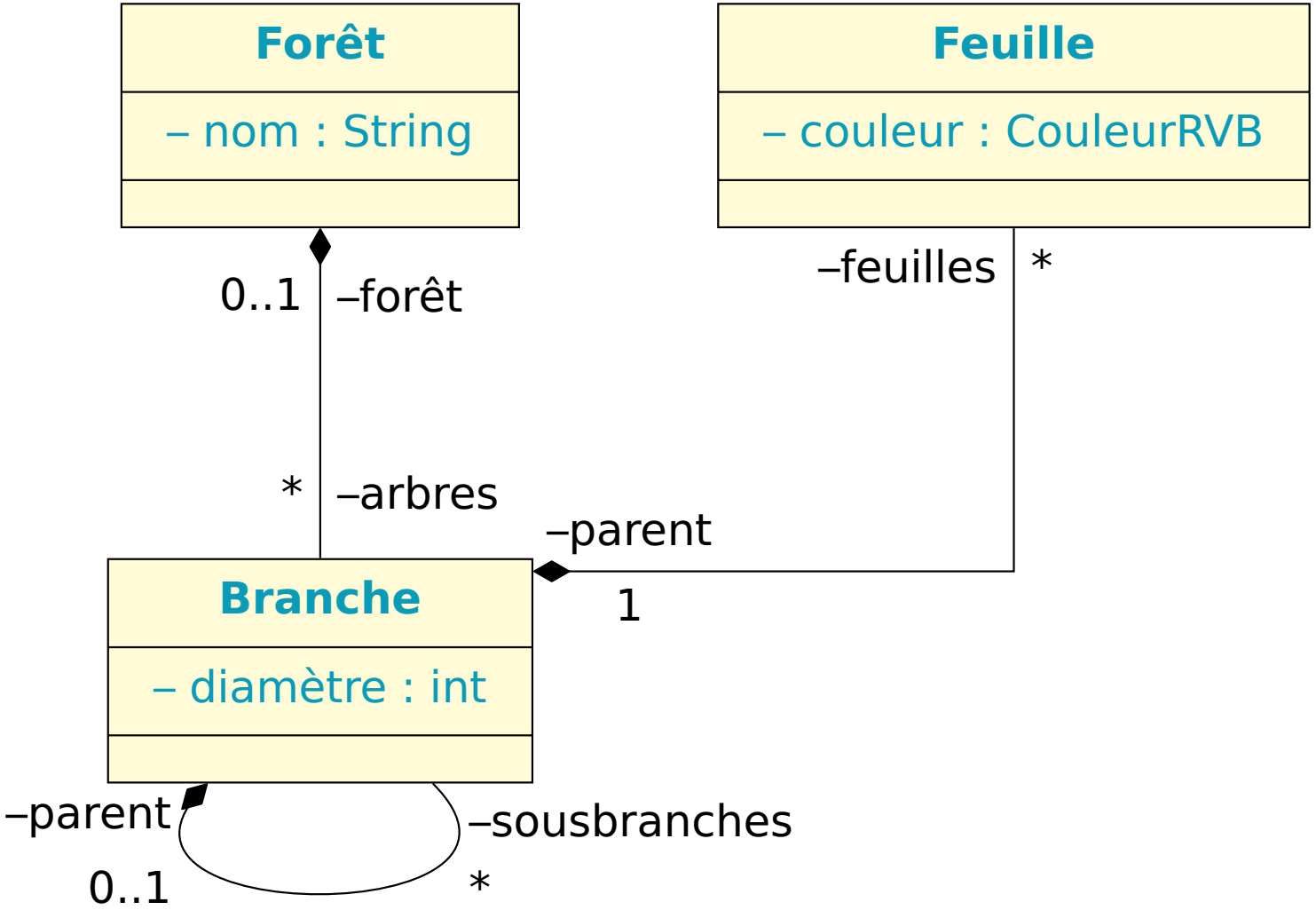
Composition



- Un raccourci pour la notion de partie.
- Le composite est formée des objets composants.
- Un **objet** ne peut appartenir qu'à un seul composite : cardinalité 1 ou 0..1 seulement.
- Détruire un composite signifie détruire ses composants

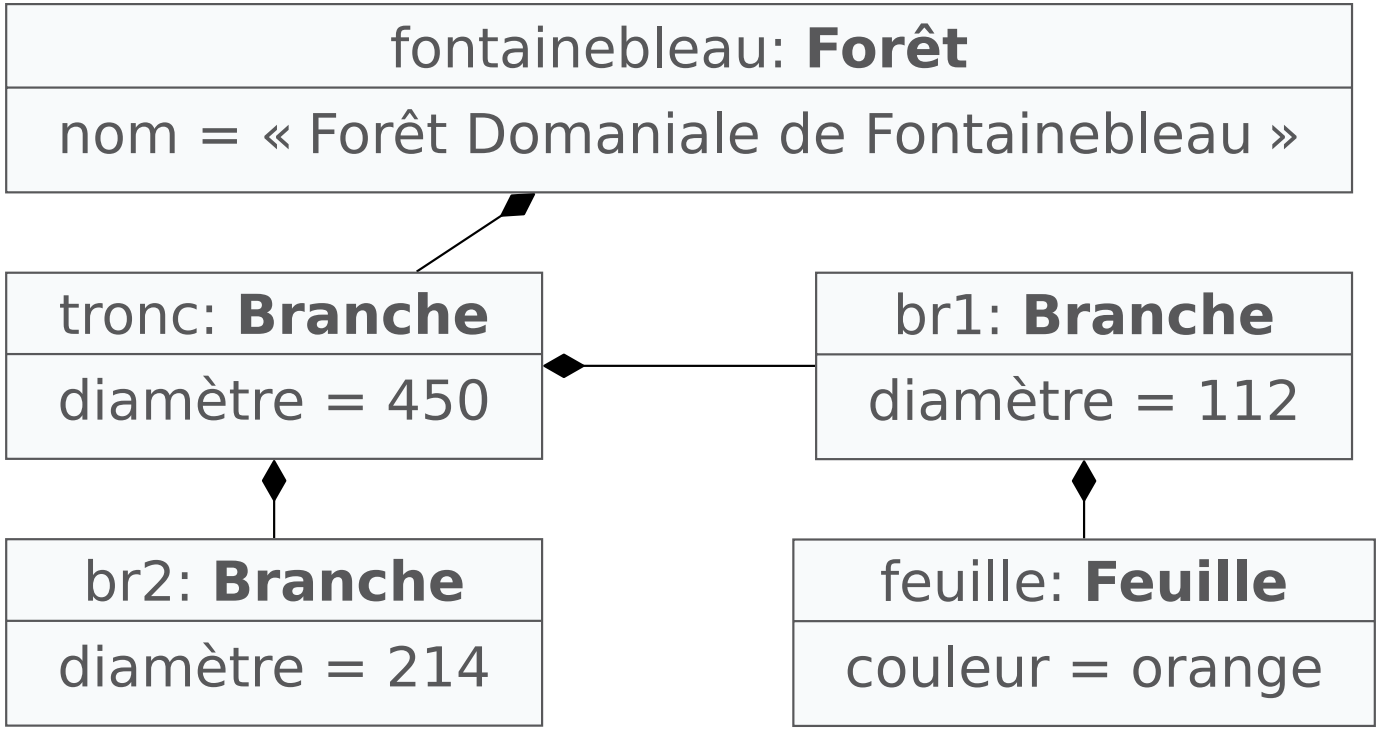
Composition — Exemple

Le diagramme de classes



Composition — Exemple

Le diagramme objets



Remarque

- Chaque branche a un seul parent.
- Si l'on abat le tronc, toutes les sous-branches meurent.
  Si l'on abat une forêt, tous les arbres sont coupés.

Mais si on code ?

Cardinalités, agrégations et compositions dans le code

- Des cardinalités * impliquent souvent que l'attribut est un ensemble : collection, liste, tableau. . .

↪ Il faut souvent avoir les accesseurs *ad-hoc* pour modifier ces ensembles :

  - il est utile de nommer le **get** adéquatement : getListe. . .
  - insérer, supprimer au lieu du **set** habituel.

⇝ C'est souvent le cas avec des agrégations ou compositions.

- Il n'y a pas vraiment moyen de contrôler la différence entre **agrégation** et **composition** dans le code : le code effectué à la destruction du composite doit se charger de détruire le composant (si on veut être propre, autrement il y a les *garbage-collectors*. . .).

Mais si on code ?

Cardinalités, agrégations et compositions dans le code

- Des cardinalités * impliquent souvent que l'attribut est un ensemble : collection, liste, tableau. . .

↪ Il faut souvent avoir les accesseurs *ad-hoc* pour modifier ces ensembles :

  - il est utile de nommer le **get** adéquatement : getListe. . .
  - insérer, supprimer au lieu du **set** habituel.

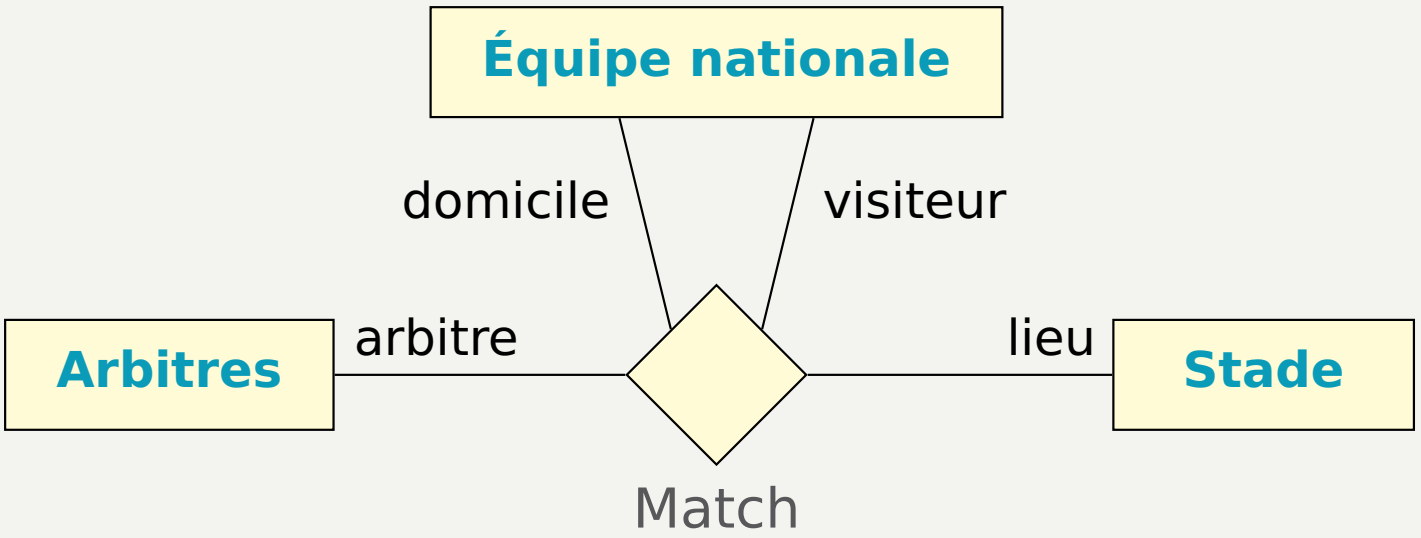
⇝ C'est souvent le cas avec des agrégations ou compositions.

- Il n'y a pas vraiment moyen de contrôler la différence entre **agrégation** et **composition** dans le code : le code effectué à la destruction du composite doit se charger de détruire le composant (si on veut être propre, autrement il y a les *garbage-collectors*. . .).

Associations *n*-aires

- Dans le cas de diagrammes de classe d'analyse, on peut envisager des relations *n*-aires.
- Au sens mathématique : n-uplet d'objets instances des classes liées : $(x_1, \ldots, x_n) \in X_1 \times \cdots \times X_n$.

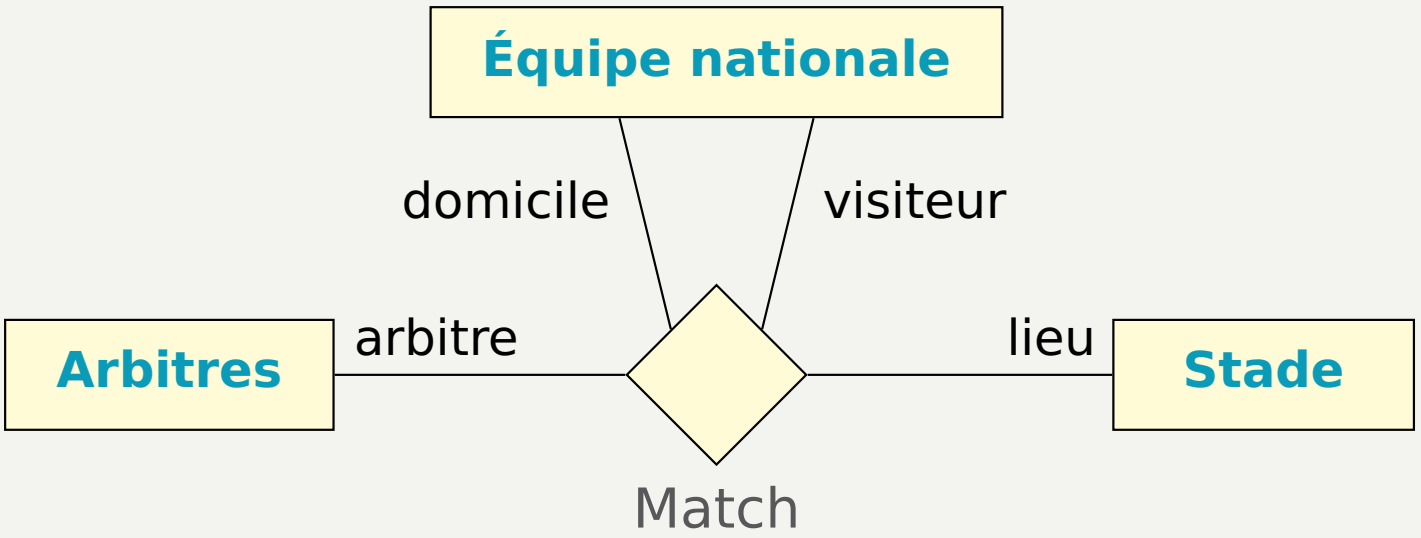
Exemple 1



Associations *n*-aires

- Dans le cas de diagrammes de classe d'analyse, on peut envisager des relations *n*-aires.
- Au sens mathématique : n-uplet d'objets instances des classes liées : $(x_1, \ldots, x_n) \in X_1 \times \cdots \times X_n$.

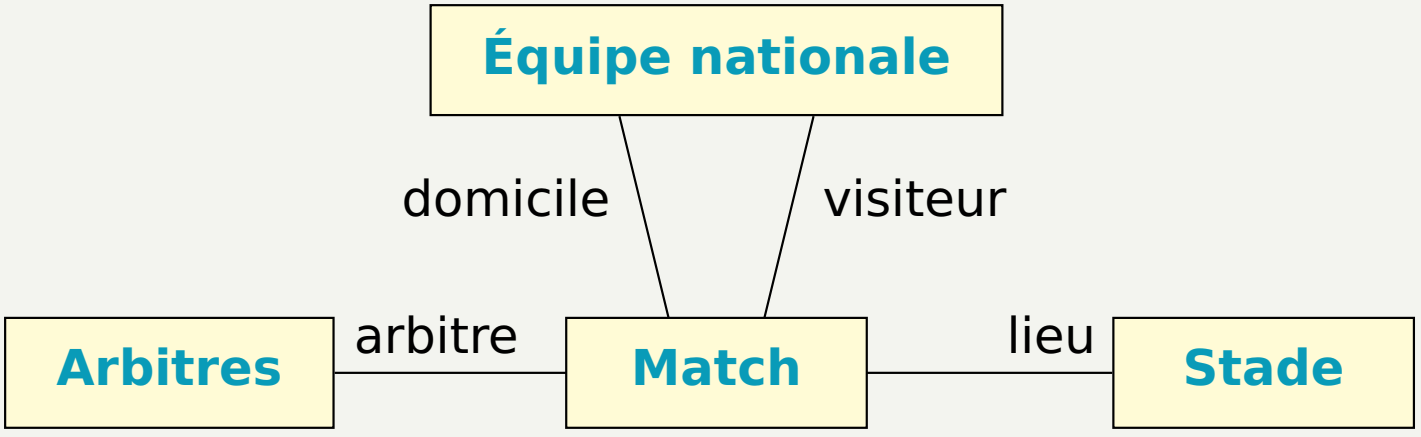
Exemple 1



Associations *n*-aires

- Dans le cas de diagrammes de classe d'analyse, on peut envisager des relations *n*-aires.
- Au sens mathématique : n-uplet d'objets instances des classes liées : $(x_1, \ldots, x_n) \in X_1 \times \cdots \times X_n$.

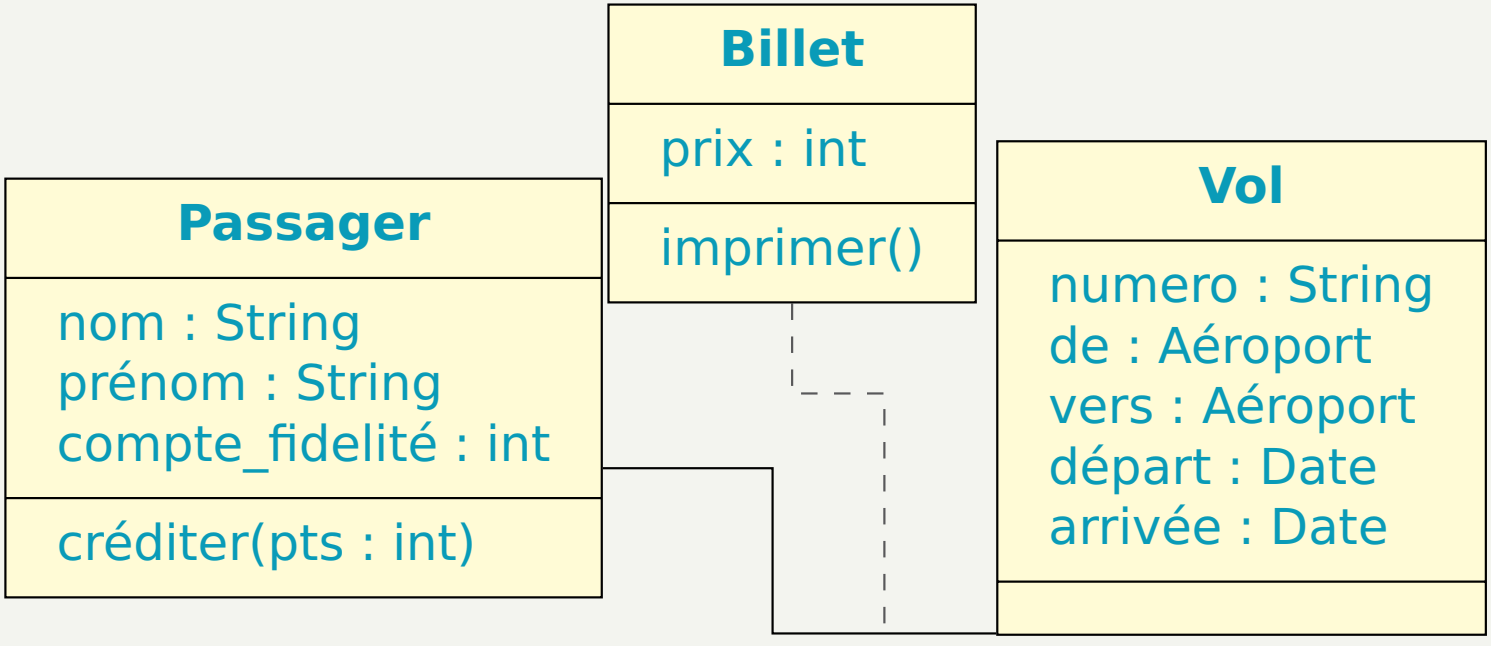
Exemple 1



Classes d'association

- Lorsque l'association entre deux classes dispose elle même d'attributs (voire d'opérations).
- Les instances de cette classe n'existent que lorsque l'association existe.
- Modélisent les informations relatives à l'interaction.

Exemple 2

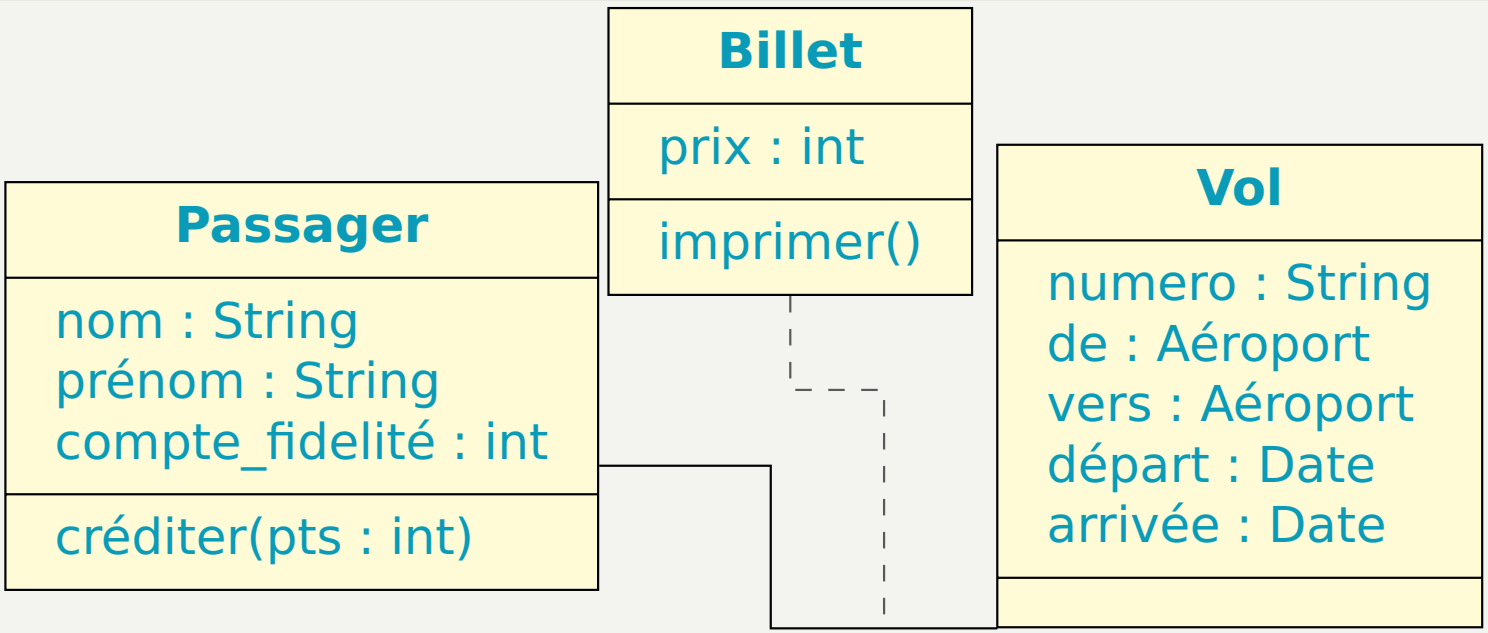


Mais si on code ?

Associations *n*-aires et classes d'associations dans le code

- Les classes d'association ou les associations *n*-aires n'ont pas de pendant dans les langages objets.
- On replace l'association *n*-aire par une classe sans autre attribut que ses associations.
- On remplace l'association disposant d'une classe par deux associations vers la classe d'association :

Exemple 3

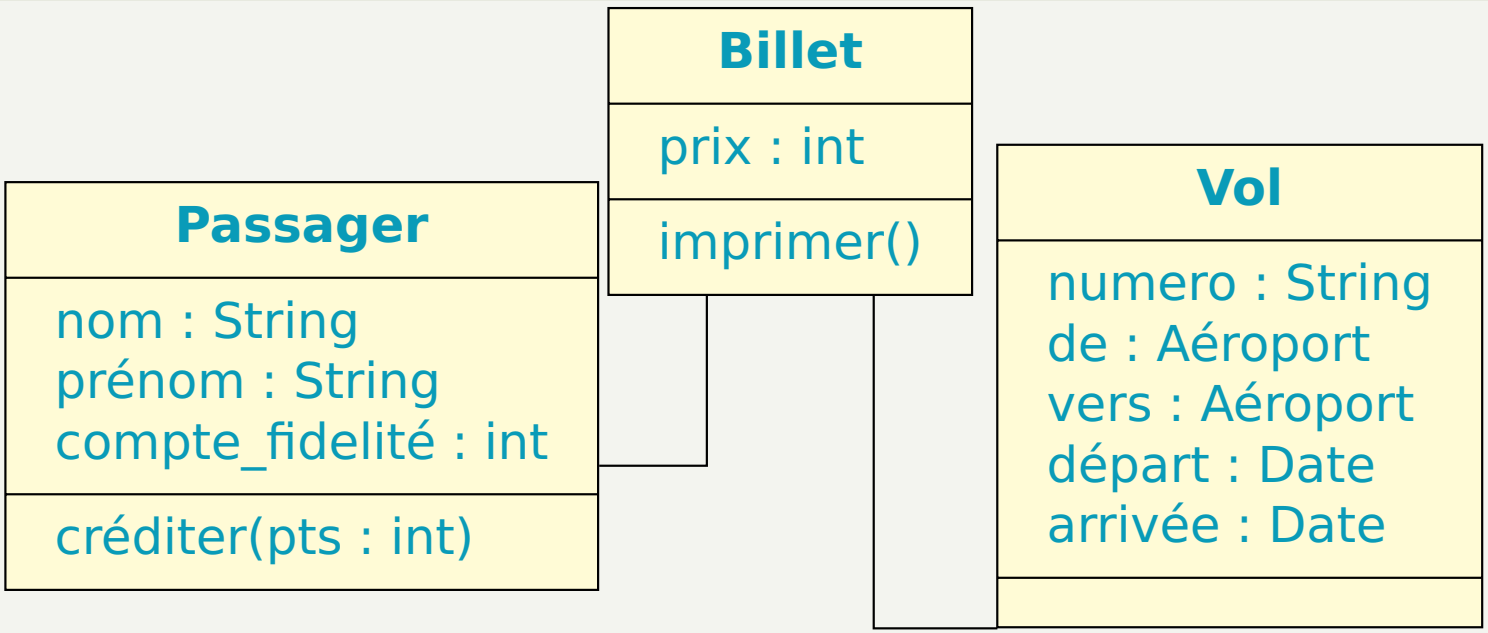


Mais si on code ?

Associations *n*-aires et classes d'associations dans le code

- Les classes d'association ou les associations *n*-aires n'ont pas de pendant dans les langages objets.
- On replace l'association *n*-aire par une classe sans autre attribut que ses associations.
- On remplace l'association disposant d'une classe par deux associations vers la classe d'association :

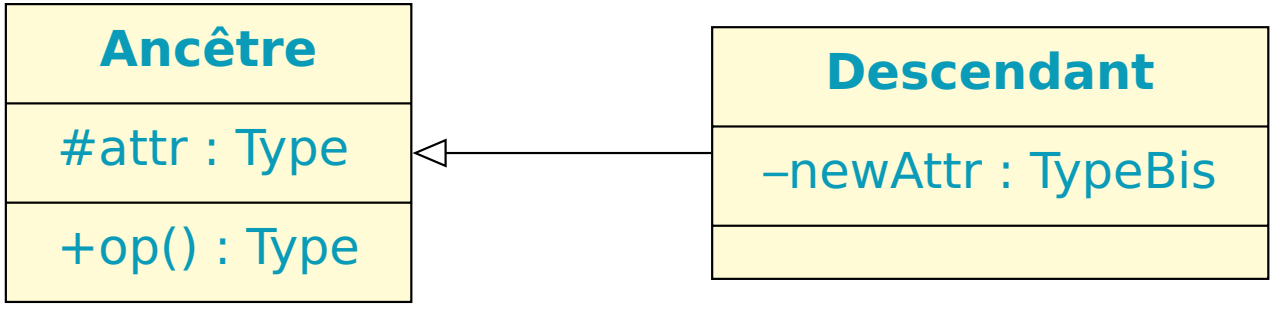
Exemple 3



Héritage

Pourquoi ?

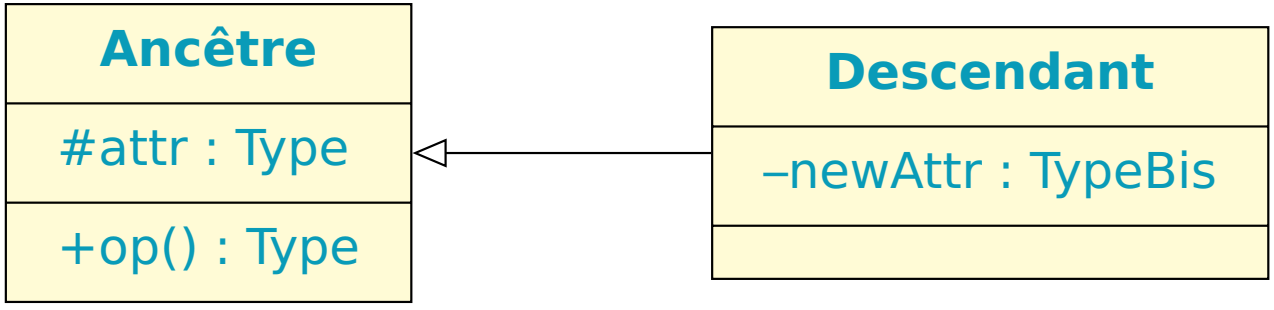
Permet de factoriser du code.

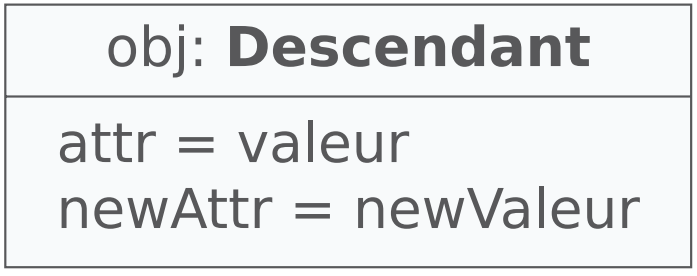


Héritage

Pourquoi ?

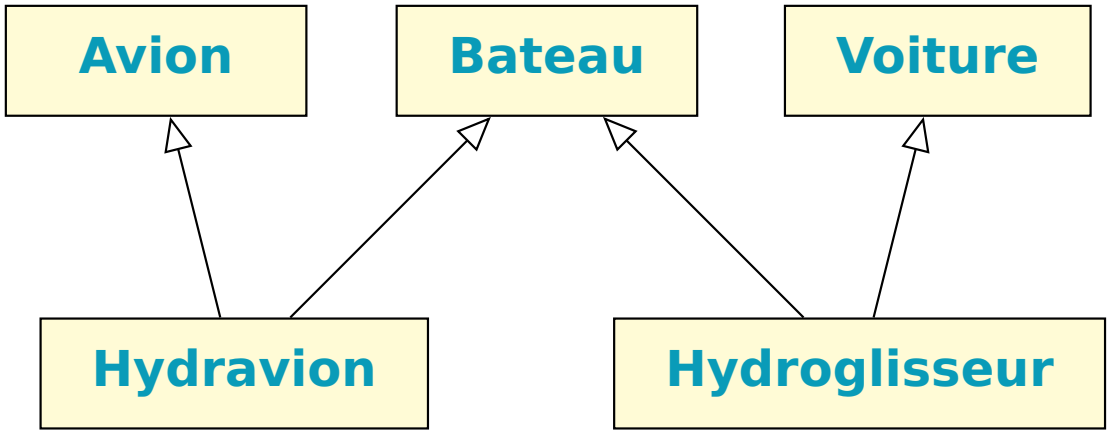
Permet de factoriser du code.





L'héritage multiple

- On hérite de tous les attributs même implicites ⇝associations.
- On hérite des opérations mais elle peuvent être redéfinies *(override)* ⇝**polymorphisme** : l'opération de l'ancêtre prend plusieurs forme dans les objets hérités.
- On peut hériter d'une classe héritant. . .
- En théorie, on peut hériter de plusieurs classes à la fois :



Remarque

En pratique, ce n'est pas toujours possible (par exemple pas en java).