Conception Orientée Objets

Introduction à UML

charles.meunier@ube.fr

01

Présentation

charles.meunier@ube.fr

UML, qu'est-ce que c'est ?

UML (Unified Modeling Language) est un standard de modélisation graphique permettant de décrire à l'aide de diagrammes :

- des architectures logicielles,
- des comportements de systèmes
- des flux d'informations
- ...

dans le cadre de conceptions orientée objets.

Les diagrammes UML

UML propose **14 types de diagrammes** répartis en deux catégories :

DIAGRAMMES DE STRUCTURE

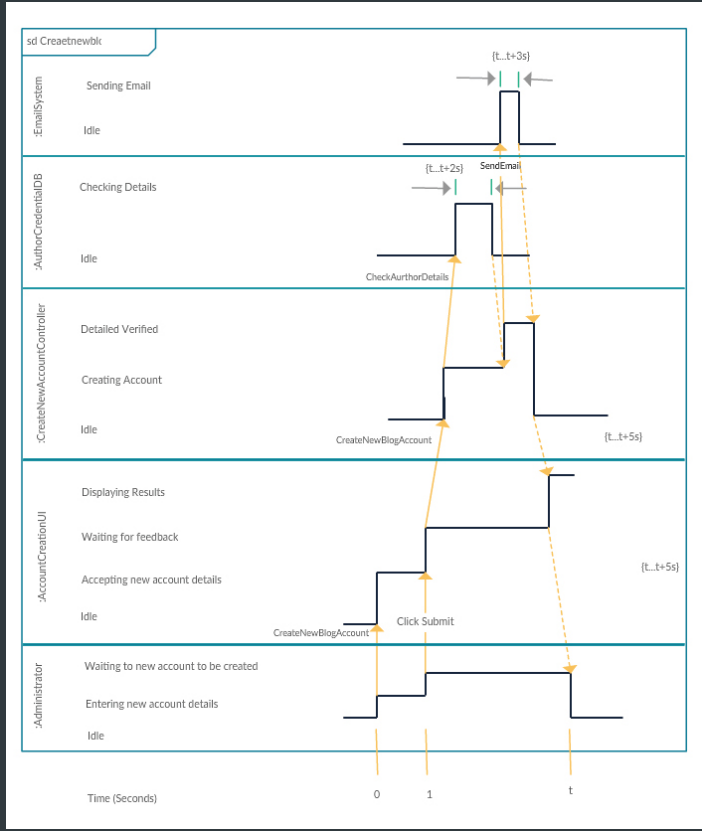
- **Diagramme de classes**
- Diagramme de composants
- Diagramme de déploiement
- Diagramme d'objets
- Diagramme de paquets
- Diagramme de profils
- Diagramme de structures composites

DIAGRAMMES COMPORTEMENTAUX

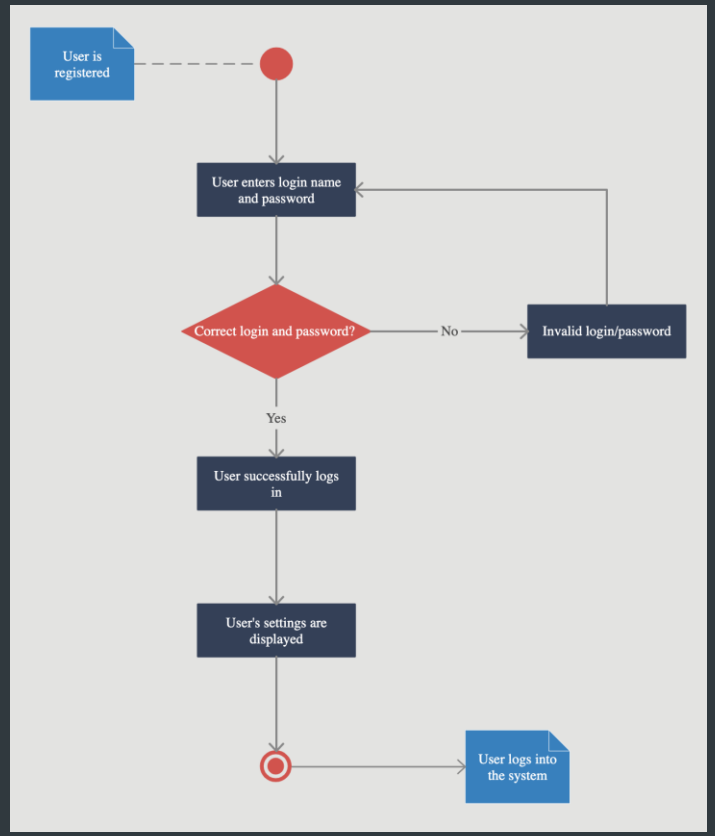
- **Diagramme des cas d'utilisation**
- Diagramme d'activité
- Diagramme de machine à états
- Diagramme de séquences
- Diagramme de communication
- Diagramme de synthèse des interactions
- Diagramme de synchronisation

Quelques exemples de diagrammes

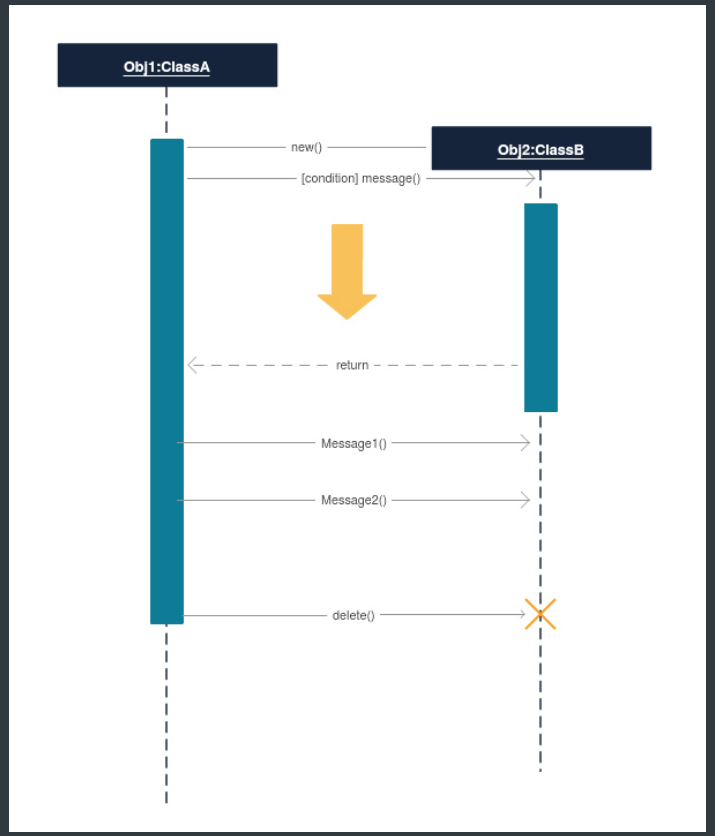
Synchronisation



Activités



Séquences



02

Cas d'utilisation

Diagramme des cas d'utilisation

Le diagramme des cas d'utilisation (Use Cases) est utilisé pour **décrire les interactions entre un utilisateur et un système**.

Il est constitué d'un système, d'acteurs, de cas d'utilisation et de relations entre les acteurs eux-mêmes ou les cas d'utilisation.

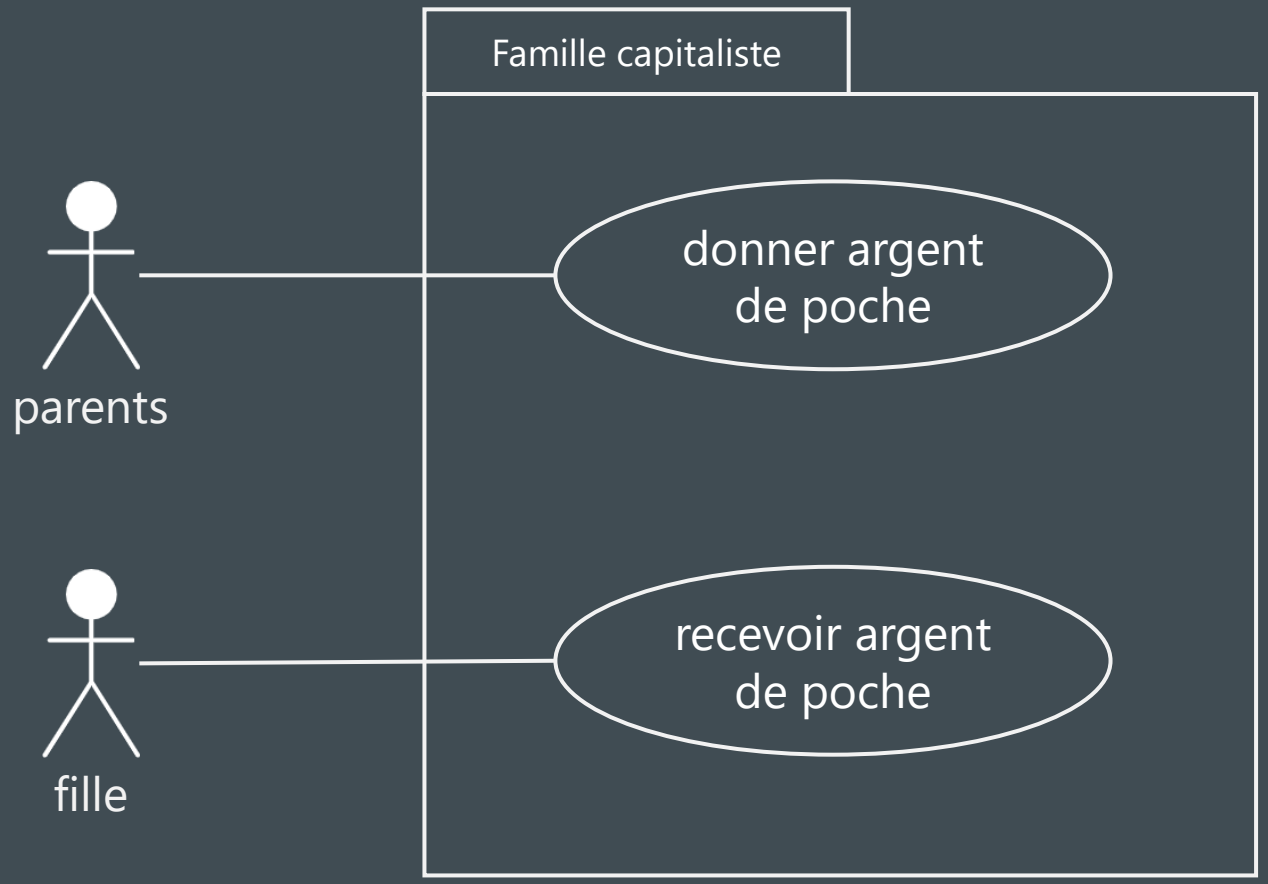
Acteur

Cas d'utilisation

Relations

Exemple simple

"Les parents donnent de l'argent de poche à leur fille"

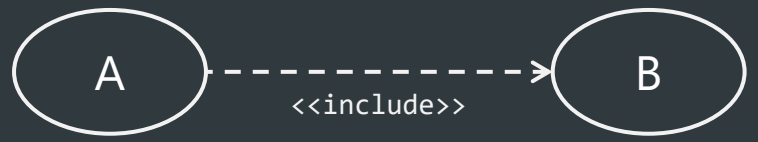


include / extends

Certains cas d'utilisation plus complexes peuvent être décomposés et reliés par des relations d'inclusion ou d'extension.

include

La relation *include* indique qu'un cas d'utilisation B **doit** être réaliser pour que A soit réalisable.

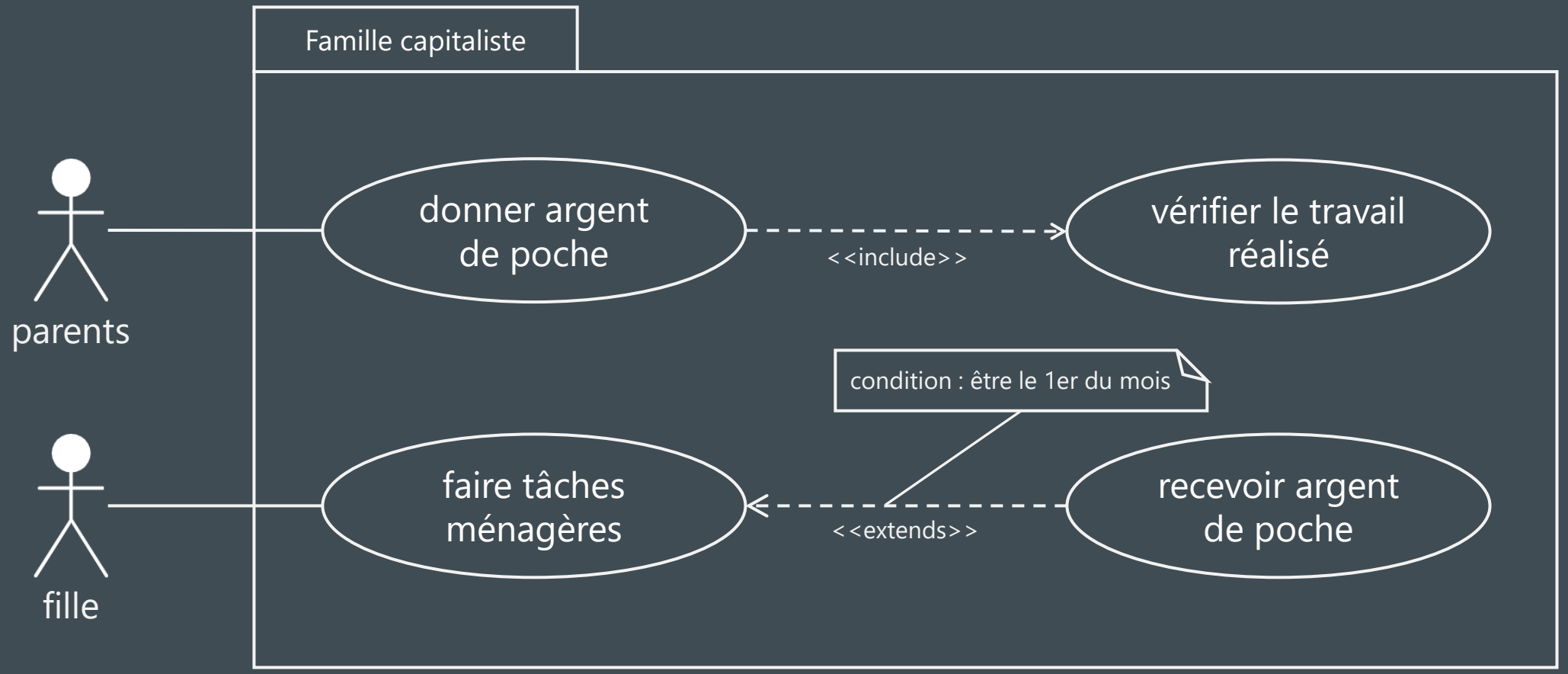


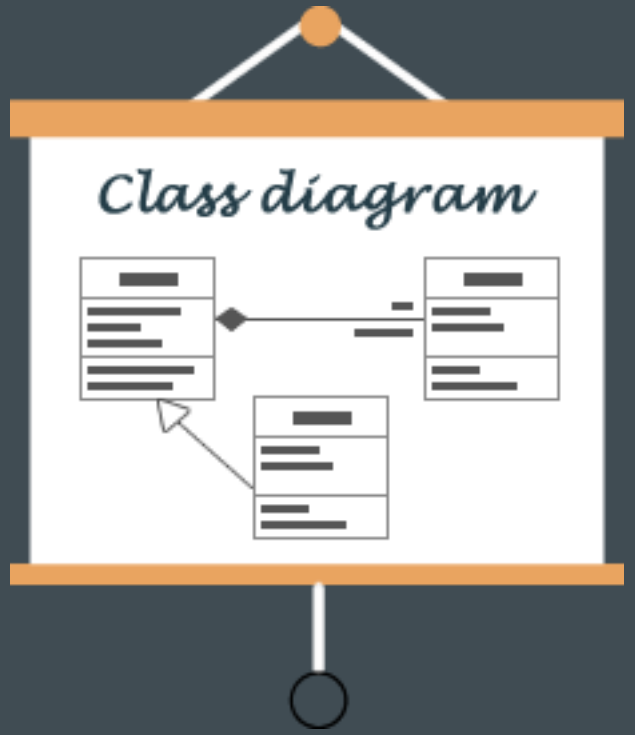
extends

La relation *extends* indique qu'un cas d'utilisation **peut** être réaliser à la suite d'un autre cas d'utilisation

Exemple plus détaillé

"Les parents donnent de l'argent de poche à leur fille tous les 1er du mois si elle participe aux tâches ménagères"





03

Classes

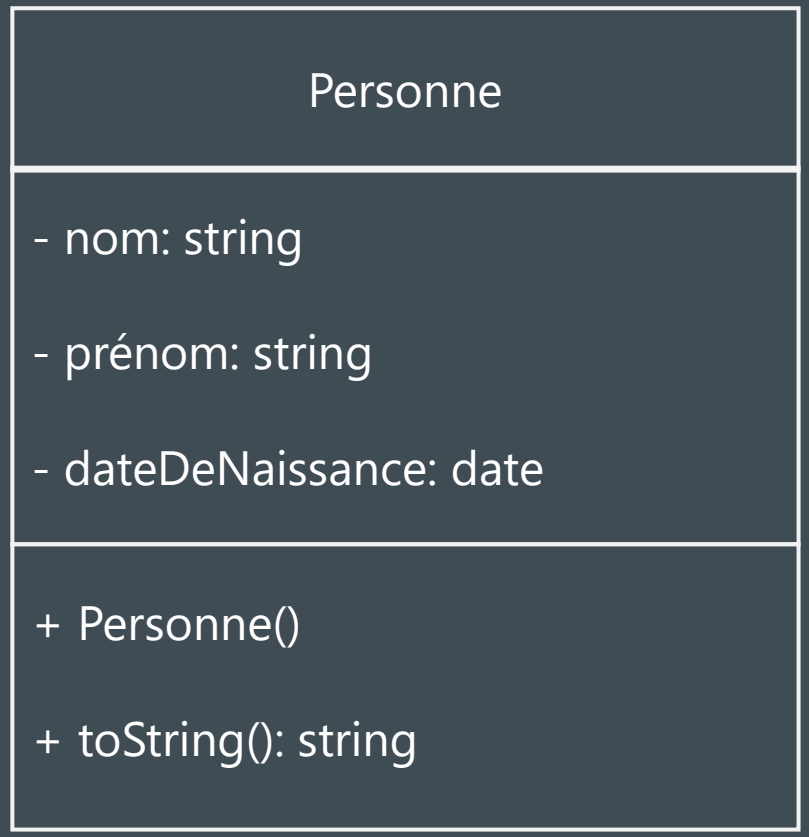
Diagramme des classes

Le diagramme des classes est un des éléments les plus importants dans la documentation d'une conception orientée objets.

Il constitue une sorte de plan de fabrication de l'application représentant chaque classe du système et ses relations.

Plan d'architecte	Diagramme des classes
Décrit les différentes pièces d'un bâtiments et leurs connexions	Décrit les classes d'un systèmes et leurs relations (héritage, association, ...)
Chaque pièce a des caractéristiques (surface, fenêtres, prises électriques, ...)	Chaque classe a des attributs
Les pièces sont reliées par des ouvertures	Les classes sont liées par des relations
Un plan sert de guide aux corps de métiers du bâtiment (maçons, électriciens, plombiers, ...)	Un diagramme des classes sert de guide aux équipes de développements (chef de projet, architecte, frontend, backend, ...)

Représentation d'une classe



En UML, une classe est représentée par un rectangle séparé en trois parties :

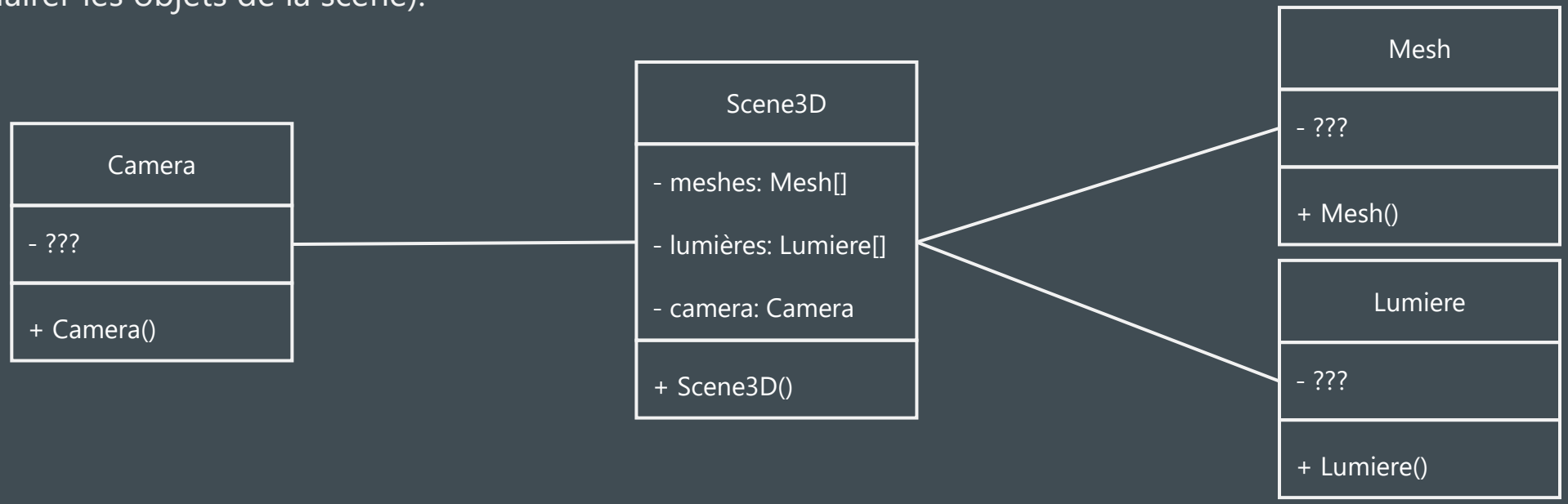
- Le nom de la classe (un nom commun)
- Les attributs (nom, type et mode d'accès)
- Les méthodes (nom, paramètres et valeur de retour)

Les modes d'accès sont représentés par un symbole avant le nom de l'attribut ou de la méthode :

- publique : **+**
- privé : **-**
- protected : **#**

Relations entre classes

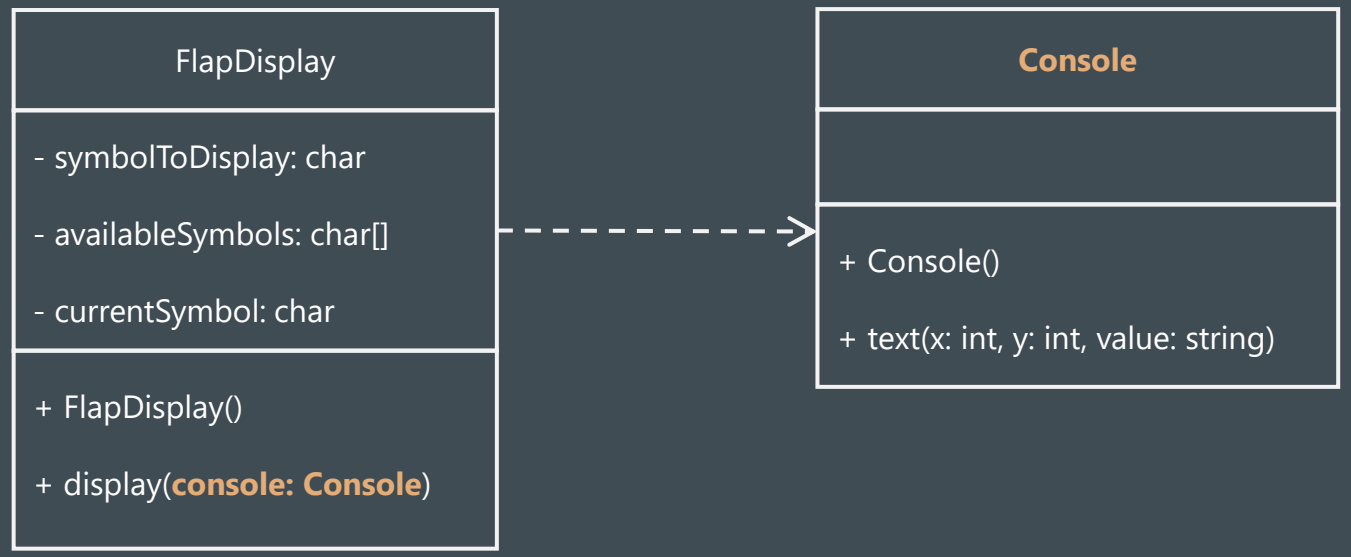
Sauf cas particulier, les classes d'une applications sont liées entre-elles. Par exemple, une scène 3D va être constituées de Mesh (les objets 3D à afficher) ainsi que d'une caméra (qui fournira le point de vue pour le rendu) et de lumières (pour éclairer les objets de la scène).



UML propose une représentation pour 6 types de relations (dépendance, association, agrégation, composition, héritage et implémentation)

Relations de dépendance

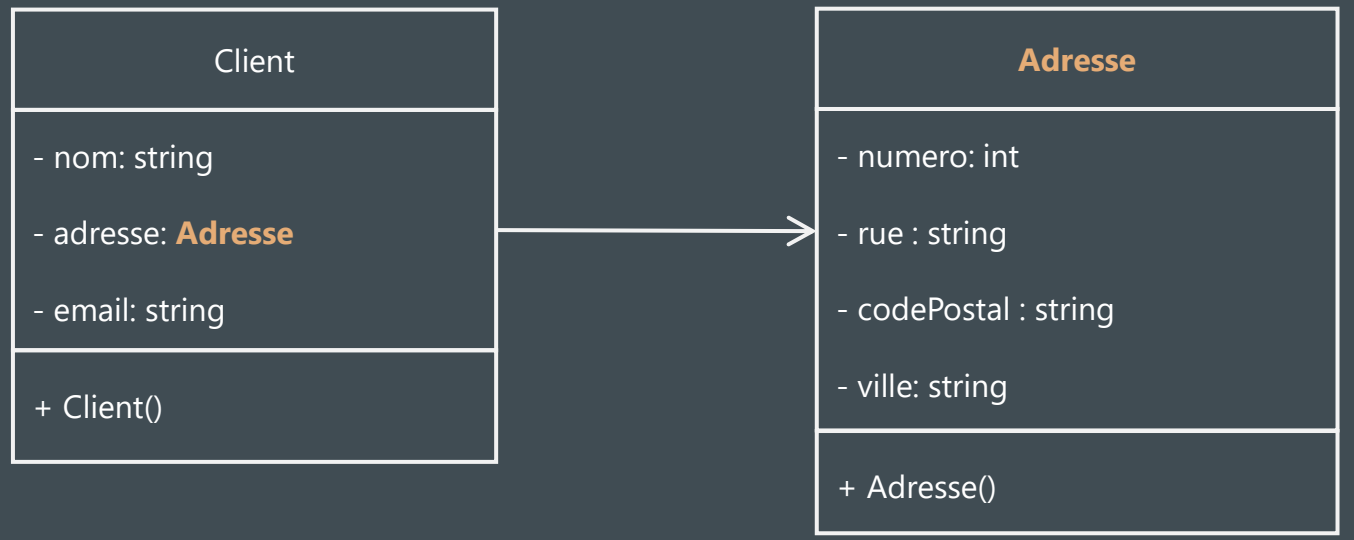
Lorsque qu'une classe A nécessite d'utiliser une classe B sans que B ne soit un attribut de A, on parle de relation de dépendance :



Ici, **`FlapDisplay`** a une relation de dépendance avec **`Console`** car il doit savoir à quoi correspond le paramètre **`console`** de sa méthode **`display`**. Mais **`console`** n'est pas un attribut de **`FlapDisplay`**.

Relations d'association

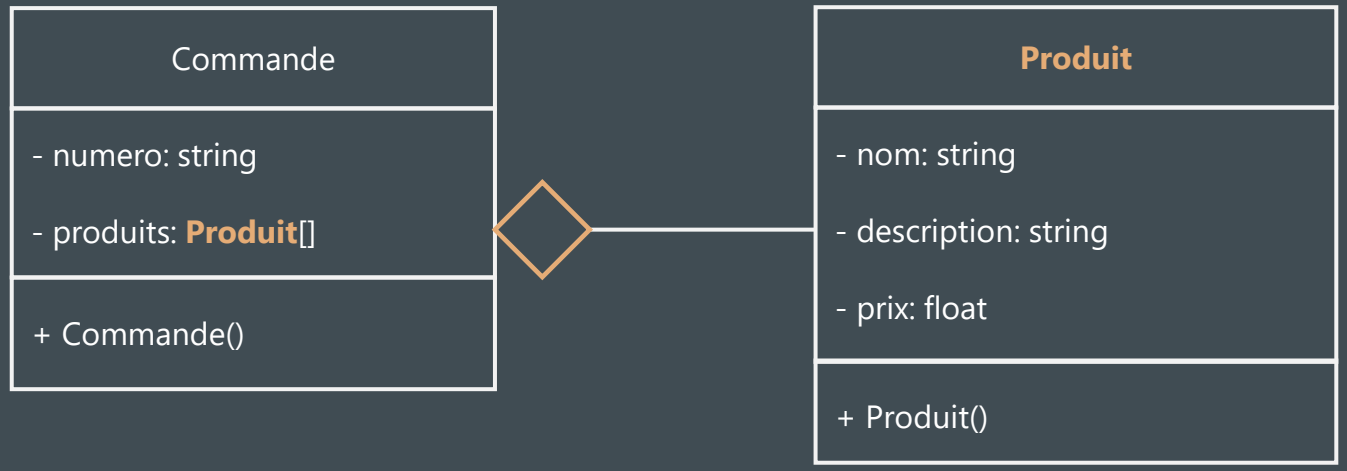
Lorsqu'une classe A possède un attribut de type classe B, on parle de relation d'association :



Ici, la classe **`Client`** possède un attribut de type **`Adresse`**. Il existe donc une relation d'association entre **`Client`** et **`Adresse`**. La relation d'association est plus forte que la relation de dépendance car la classe B devient constitutive de la classe A.

Relations d'agrégation

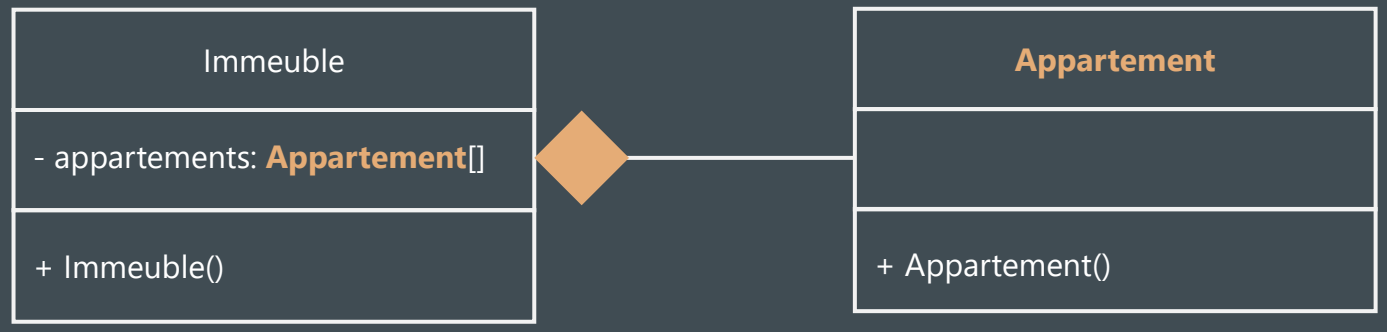
Lorsqu'une classe A possède un attribut de type classe B et que l'on souhaite mettre en avant une **relation hiérarchique**, on parle d'agrégation :



La classe `Commande` contient des instances de la classe `Produit`. Il existe donc à minima une relation d'association entre ces deux classes. Mais ici, on a voulu insister sur la relation hiérarchique entre les deux classes avec une agrégation représentée par le **losange vide** au début de la relation.

Relations de composition

Comme la relation d'agrégation, la relation de composition indique une **relation hiérarchique** entre deux classes. Mais avec un **lien beaucoup plus fort**.



Le diagramme ci-dessus représente la relation qui existe entre la classe `Immeuble` et la classe `Appartement`. La relation de composition (**losange plein**) renforce ici le caractère hiérarchique de la relation (les appartements de l'immeuble) en indiquant que c'est à l'immeuble qu'appartient **la responsabilité de créer et de détruire** les appartements.

En effet, lorsque l'on construit un immeuble les appartements sont créés en même temps. Et lorsque l'on détruit l'immeuble, il ne serait pas logique que les appartements continuent d'exister.

Association, agrégation ou composition ?

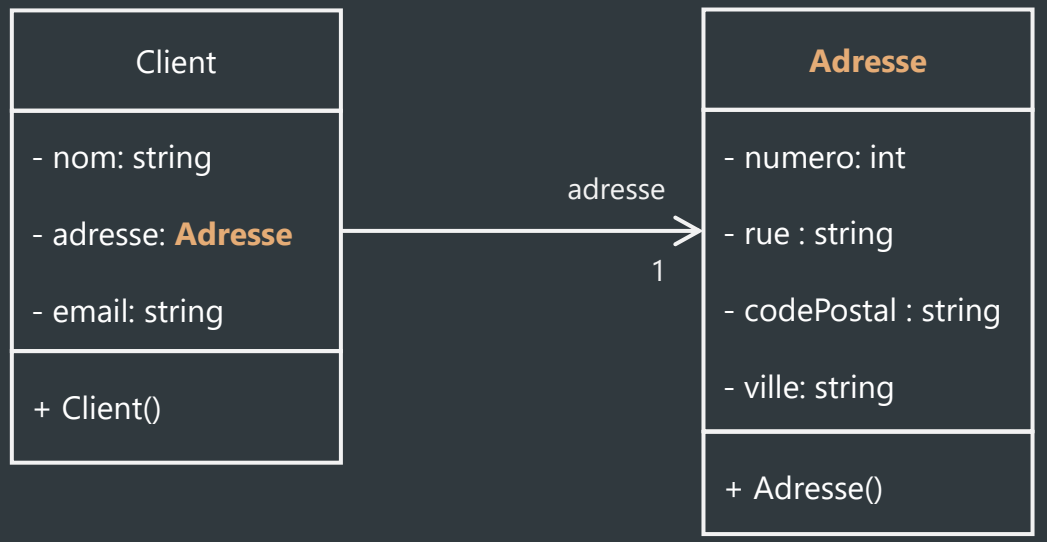
Les relations d'association, d'agrégation et de composition représentent, à la base, la même chose : le fait qu'une Classe A est un attribut de type Classe B.

L'agrégation et la composition viennent simplement fournir des informations complémentaires : existe-t-il un lien de hiérarchie ? Et si oui, est il fort au point de laisser à la classe A la responsabilité de créer et détruire les instances de la classe B ?

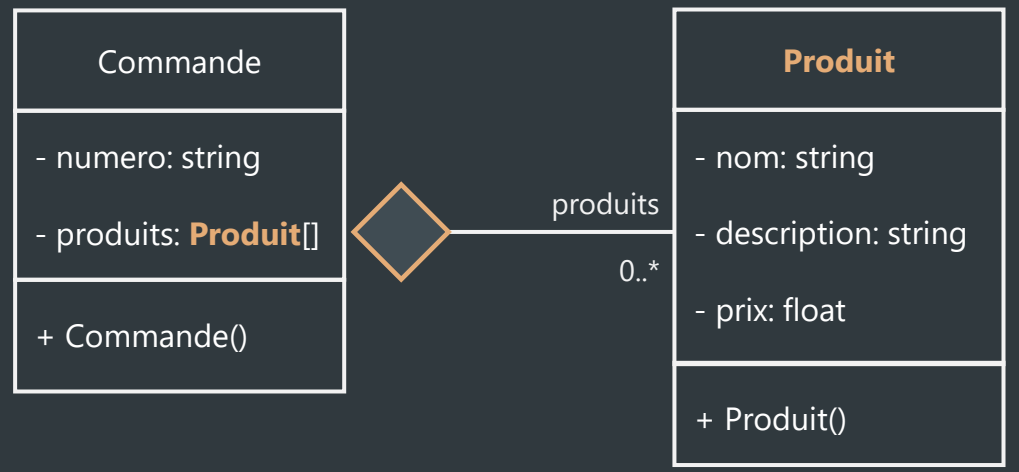
Il s'agit donc, avant tout, de sémantique pour aider les personnes qui liront le diagramme à mieux comprendre comment a été pensée l'application. Mais d'un point de vue programmation, il n'y a pas de différence, par exemple, entre une association et une agrégation.

Nom et cardinalité

Les relations d'association, d'agrégation et de composition doivent être nommées et quantifiées à l'aide d'une cardinalité.



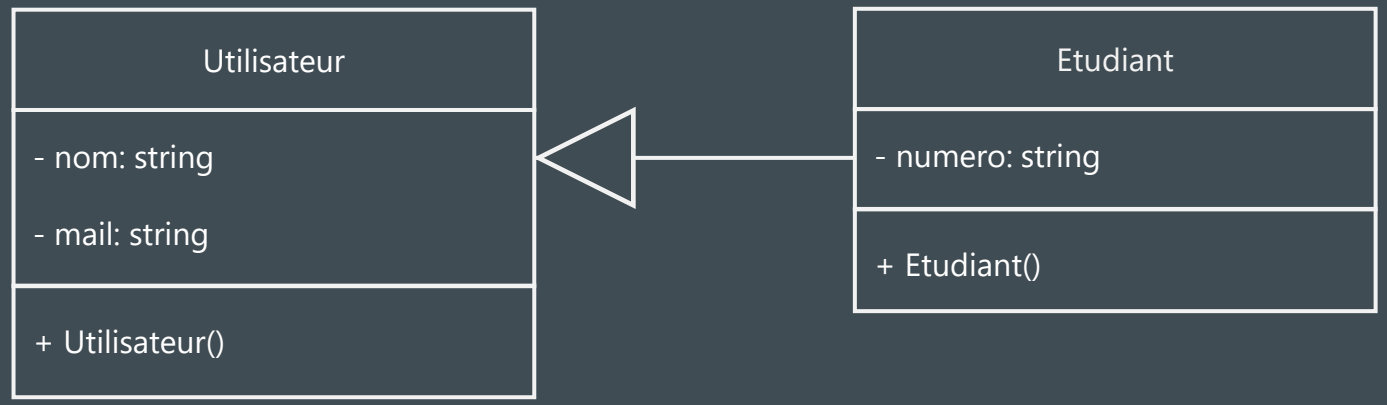
Un client possède **1** adresse



Une commande est composée de **0** ou **plusieurs (*)** produits

Relations d'héritage

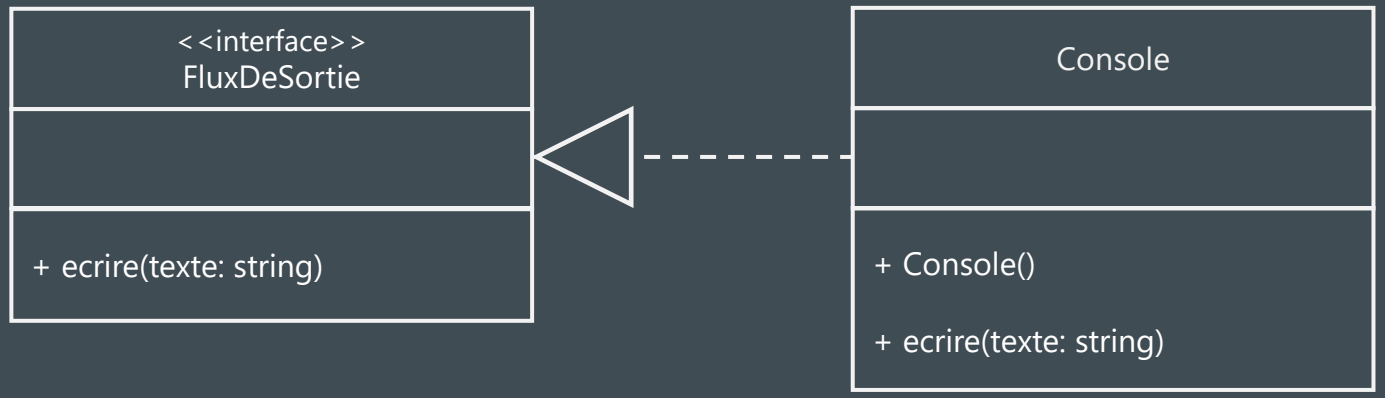
La relation de spécialisation/généralisation, ou encore héritage, est également représentée dans le diagramme des classes :



La classe **`Etudiant`** hérite de la classe **`Utilisateur`**. La relation est représentée par une flèche fermée vide.

Relations d'implémentation

Dans le cas d'une interface, on ne parle pas d'héritage mais d'implémentation. On dit qu'une **classe implémente une interface**, c'est-à-dire qu'elle respecte le contrat imposé par l'interface en redéfinissant toutes ses méthodes abstraites.



Ici, la classe **`Console`** implémente l'interface **`FluxDeSortie`** et respecte le contrat en proposant une méthode **`ecrire`** qui prend en paramètre une chaine de caractères.