

UML

≡ Créeur Robin Gillard (el_pablo_)

▼ Diagramme de classe

- Nom
 - simple
 - précis
 - Efficace
 - Camel case
- Attribut
 - Simple
 - Court
 - Minuscule
 - Camelcase
 - on l'écrit
 - +attribut : str
 - + → encapsulation (+ , - , #)
 - attribut → nom
 - str → type
 - Type de données qui peuvent sortir :
 - String : str
 - Entier : int
 - Réel : float
 - Booléen : bool
 - Liste : list
 - Dictionnaire : dict
 - On peut aussi avoir un attribut ou une classe

- Méthodes
 - Claire
 - Courte
 - Minuscule
 - On l'écrit :
 - `+nomMéthode(str) : str`
 - `+ → encapsulation (+, -, #)`
 - `nomMéthode → nom`
 - Le truc en parenthèses c'est le paramètre d'entrée (type = idem que ceux de sortie)
 - `str = type de donnée sortie`
 - Type de données qui peuvent sortir :
 - `String : str`
 - `Entier : int`
 - `Réel : float`
 - `Booléen : bool`
 - `Liste : list`
 - `Dictionnaire : dict`
 - On peut aussi avoir un attribut ou une classe
 - `Void : ne retourne rien`

Différents types d'encapsulation :

- `+ = public`
- `- = privé`
- `# = protected`

Classe abstraite → classe qu'on n'instancie jamais

Elle possède des attributs ou méthode abstraite

- Réduire complexité
- Évite la duplication de code
- Aide à renforcer la sécurité d'une application ou d'un programme car seuls les détails importants sont fournis

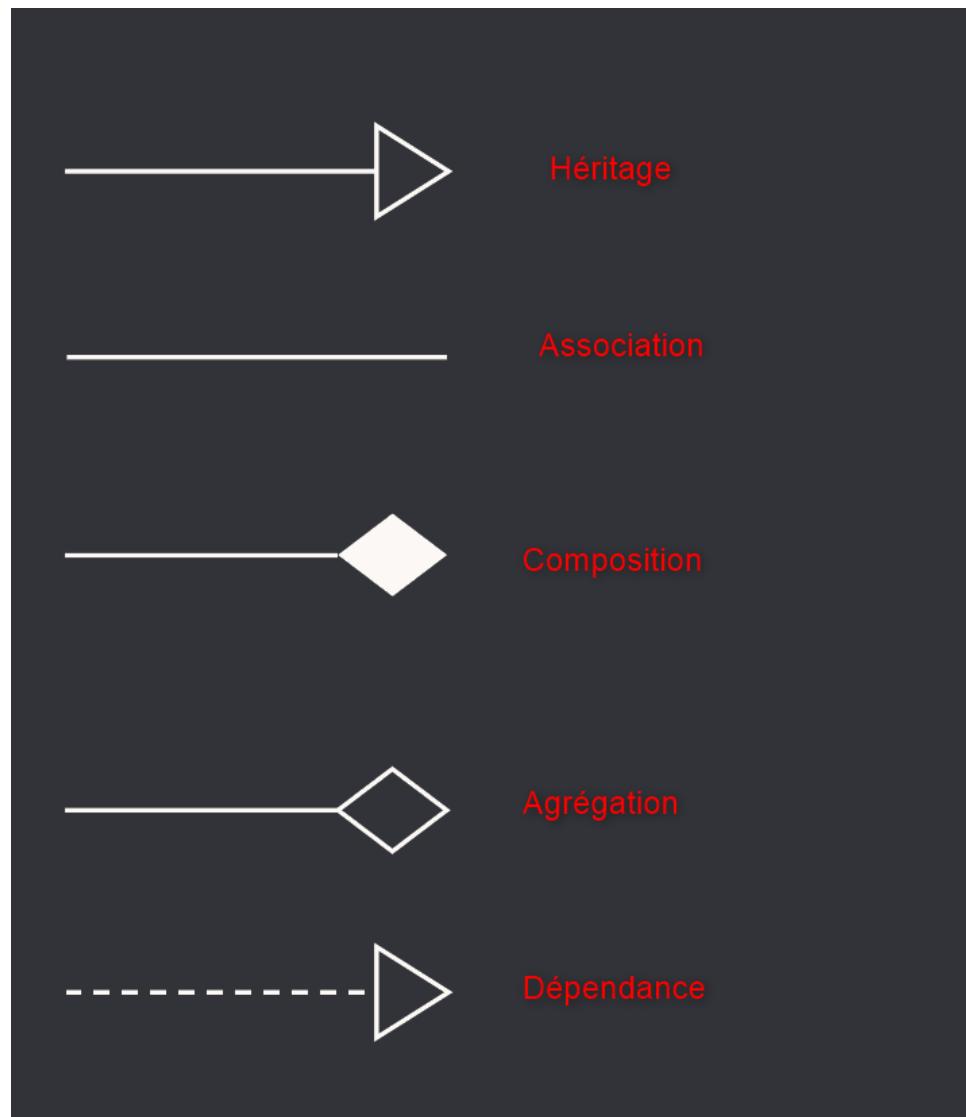
Une méthode abstraite est une méthode qui est déclarée dans la classe abstraite, mais elle doit être implémentée par les classes dérivées. Les méthodes abstraites n'ont pas de corps ; elles se contentent de définir la signature de la méthode

Les classes abstraites sont utilisées comme une base pour d'autres classes. Elles permettent de définir un modèle (ou contrat) que toutes les sous-classes doivent suivre

Interface → contrat d'actions mais seulement abstraite (méthode abstraite),
Pas d'attribut

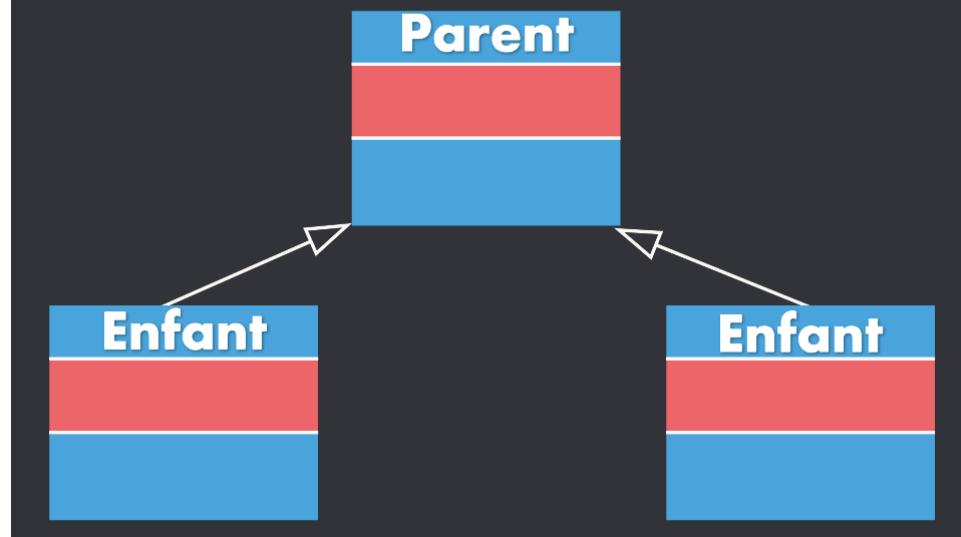
Lorsqu'une classe implémente une interface, elle indique ainsi qu'elle s'engage à fournir une implémentation(un corps) pour chacune des méthodes abstraites

Interface : je sais seulement le nom des méthodes que j'aurai besoin pour que le boulot soit fait, toi tu dois le mettre en application (lui donner un corps)



Héritage

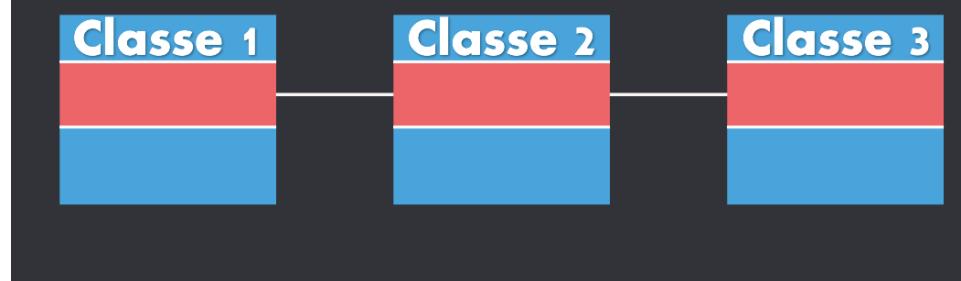
- De l'enfant au parent
- Phrase clé : est un



Quand on a la phrase : est un → Héritage

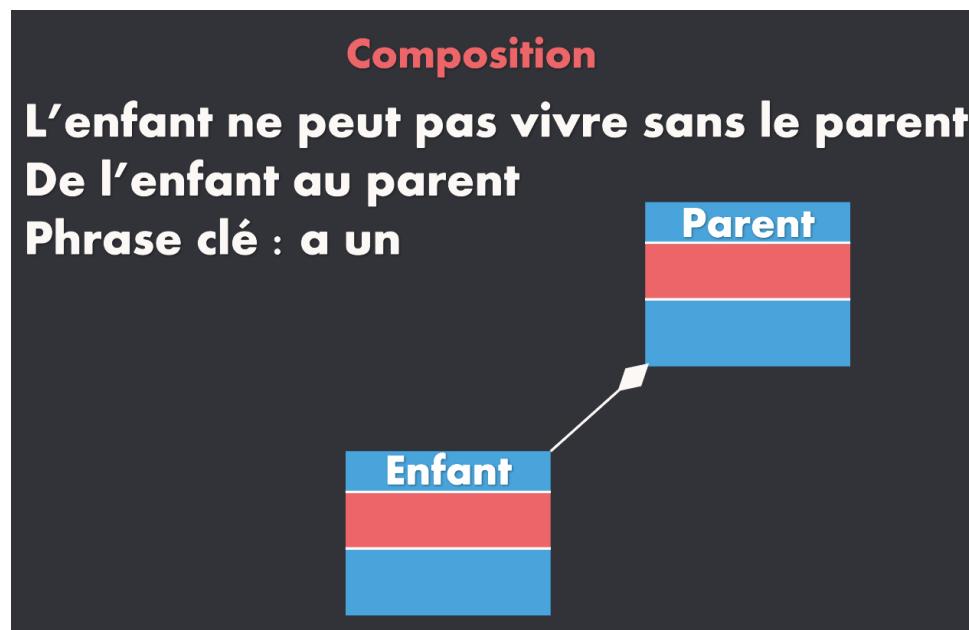
Association

- Interagissent entre elles
- Phrase clé : interact avec



Quand on appelle une autre classe dans un paramètre de méthode,
association

exemple : la classe Augus2 utilise la classe linux



Exemple :

- L'abeille et le dard
- L'opération dans le compte en banque

la phrase à avoir : Si le parent a un enfant, est-ce que mon enfant vit sans le parent ? :

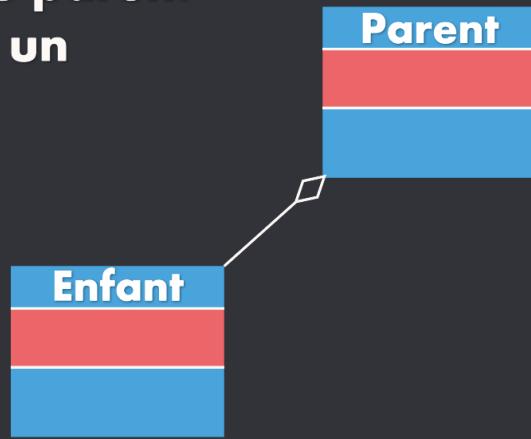
- Non → composition
- Oui → agrégation

Agrégation

L'enfant peut vivre sans le parent

De l'enfant au parent

Phrase clé : a un

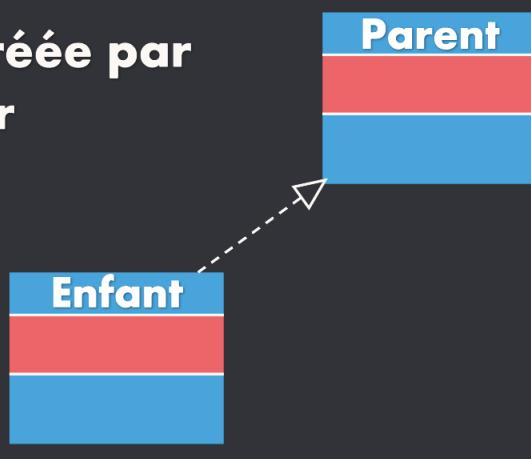


exemple : un passant et une rue, je peux être passant sans la rue (par exemple dans un couloir, dans une champ, etc) et la rue peut exister sans passant

Dépendance

L'enfant dépend du parent sans le composer

Phrase clé : créée par / modifiée par



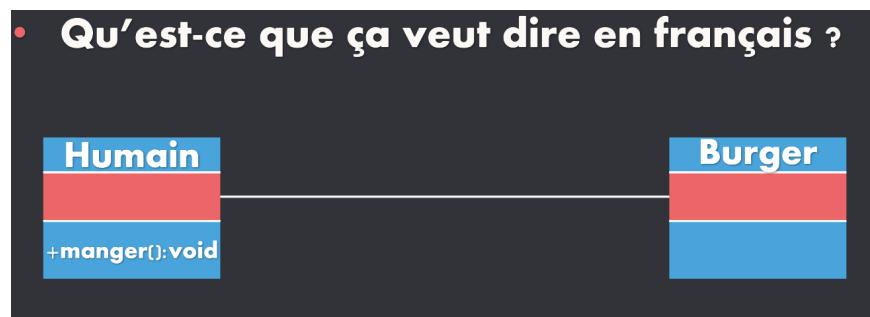
L'enfant est créée ou modifiée par le parent

exemple :

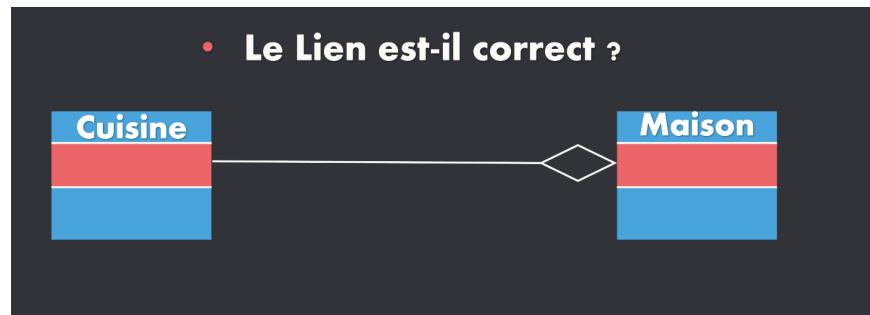
- Le burger est créée par le cuisinier
- Mon repas dépend de ma plaque de cuisson, car pour être chaud il doit être chauffer

En gros la dépendance c'est souvent pour une méthode, pour un changement de méthode.

Dans le cas de mon repas, pour qu'il soit chaud (état modifié de froid à chaud), il est modifiée par la plaque de cuisson et sa méthode chauffer (explication bidon mais normalement ça explique correctement)

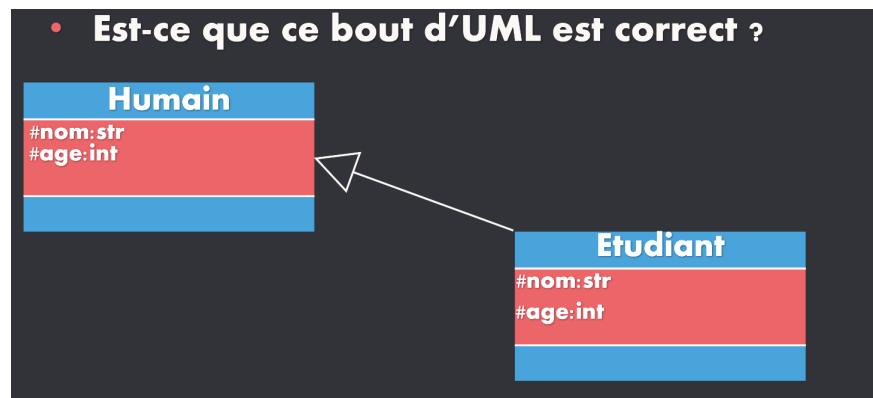


L'association le principe est bon, mais à aucun moment on a le paramètre dans la méthode manger, donc c'est bon dans le principe, mais dans le fond non



Si c'est la pièce, non car la cuisine n'existe pas sans la maison

Si c'est le meuble, oui car le meuble cuisine peut exister sans la maison



Non, dans le cas où on ne redéfinit pas un attribut, il est inutile voir contre-productif de ré écrire ceux-ci

Exo :

- **Chaque client possède un ou plusieurs comptes. Les comptes sont des comptes chèques ou des comptes épargne. Un client est caractérisé par son nom, son adresse, téléphone**
- **Le client peut effectuer des opérations sur ses comptes. Une opération est caractérisée par un montant, un type (débit / crédit) et une date. Chaque compte possède un solde (qui est déduit des opérations effectuées sur ce compte et de l'ancien solde...). De plus les comptes chèque possèdent un découvert autorisé.**
- **Les comptes épargne donnent droit à des versements d'intérêts. Ces intérêts sont fonction du solde du compte épargne et d'un taux d'intérêt constant de 3,5%. À l'opposé, un compte chèque peut se voir débiter des agios si son solde est négatif. Le taux d'agios est constant (10%).**

Pour mettre quelque chose en abstrait, on mets le nom de la classe entre guillemets, reprenons l'exemple de Compte dans cet exercice, on le marquera "Compte"

▼ Réponse possible

