DIAGRAMME DE CLASSES

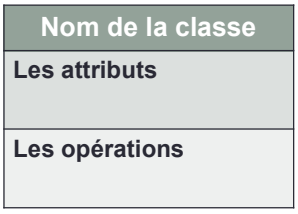
Chapitre 3

Diagrammes de classes

- Structure logique d'un système :
  - Classes
  - Relations entre ces classes
- Une classe décrit un ensemble d'objets
- Une association décrit un ensemble de liens
- Les objets sont des instances des classes et les liens sont instances des relations.

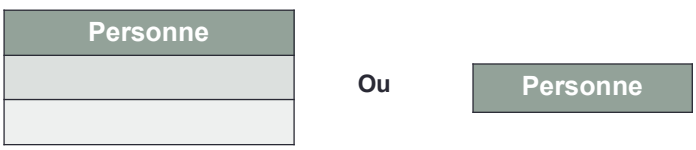
Classes

- Les classes sont représentés par des rectangles compartimentés contenant:
  - Le nom de la classe
  - Les attributs de la classe.
  - Les opérations de la classe.



Classes

- Une classe peut aussi, pour alléger un diagramme de classe et ne pas mettre tout le détail, être représentée comme suit :

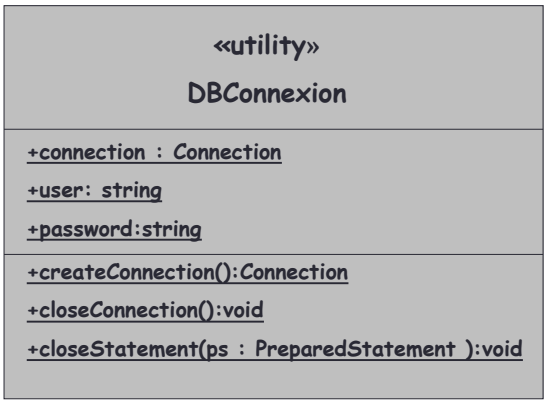


Classes

- Le nom d'une classe peut être précédé d'un stéréotype.
- Un stéréotype est une chaîne de caractère qui donne une signification particulière à une classe.
  - « interface »: une description des opérations visibles.
  - « enum »: un ensemble de valeurs prédéfinis
  - « utilitaire »: une classe effectuant des tâches communes récurrentes comme le tri.
  - « signal » : une occurrence remarquable qui déclenche une transaction dans un automate.

Classes

- Exemple :



Attributs d'une classe

- La syntaxe pour la description d'un attribut est la suivante:

  Attribut:Type=Valeur_Initiale

- Exemple: **Age : int=30**
- La syntaxe pour la description d'une opération est la suivante:

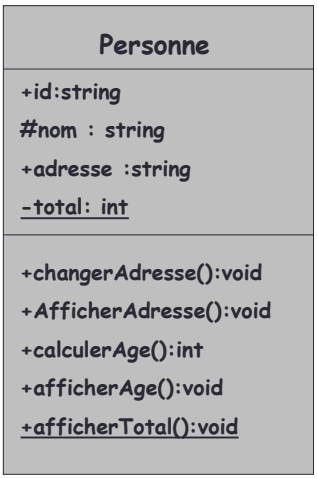
  Opération(arg:Type=ValeurDefaut,….):TypeRetour

- Exemple: **AfficherAge(Nom:string):void**

Visibilité des attributs et des opérations

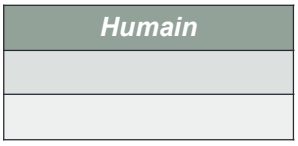
- UML définit 3 niveaux de visibilité pour les attributs et les opérations:
  - ***Public*** qui rend l'élément visible à tous les clients de la classe (symbole **+**).
  - ***Protégé*** qui rend l'élément visible aux sous-classes de la classe (symbole **#**).
  - ***Privé*** qui rend l'élément visible à la classe seule (symbole **-**).
- Les attributs ou opérations statiques sont soulignés.

Visibilité des attributs et des opérations



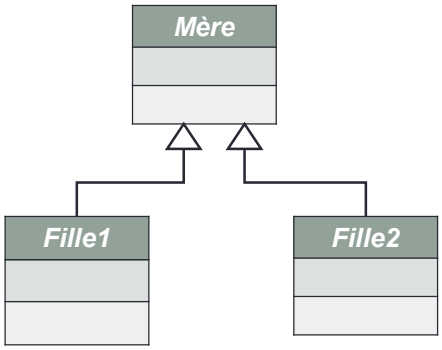
Classes abstraites

- Une classe abstraite est une classe qui n'est pas instanciable directement.
- Elle sert de spécification plus générale pour manipuler les objets instances d'une de leurs sous classes.
- Le nom d'une classe abstraite en UML est écrit en italique.



Héritage

- Les hiérarchies de classes permettent de gérer la complexité, en ordonnant les objets au sein d'arborescences de classes, d'abstraction croissante.
- Il permet de faire une:
  - Classification
  - Spécialisation
  - Généralisation



Héritage

- **Classification** :
  - Permet de ranger et d'organiser un ensemble d'objets en une hiérarchie basée sur l'héritage.
  - Une bonne classification est stable et extensible : ne classifiez pas les objets selon des critères instables (selon ce qui caractérise leur état) ou trop vagues (car cela génère trop de sous-classes).
  - Si Y hérite de X, cela signifie que "Y est une sorte de X" (analogies entre classification et théorie des ensembles).

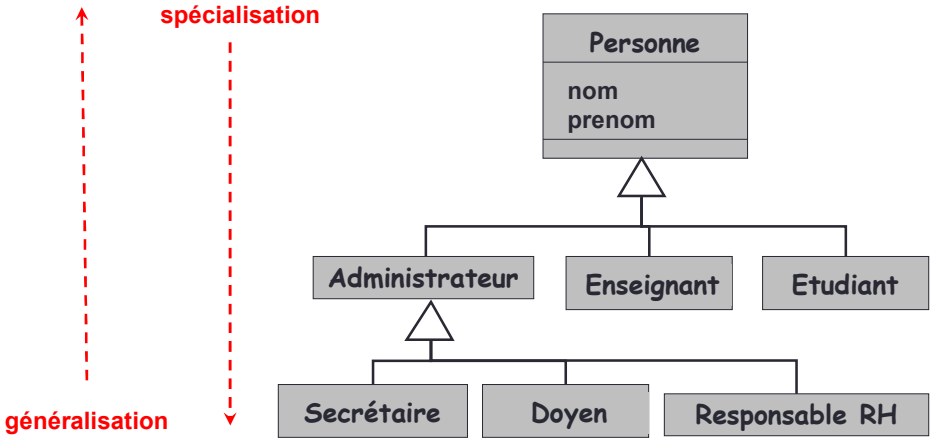
Héritage

- **Spécialisation**
  - Démarche descendante, qui consiste à capturer les particularités d'un ensemble d'objets, non discriminés par les classes déjà identifiées.
  - Consiste à étendre les propriétés d'une classe, sous forme de sous-classes, plus spécifiques (permet l'extension du modèle par réutilisation).

Héritage

- **Généralisation**
  - Démarche ascendante, qui consiste à capturer les particularités communes d'un ensemble d'objets, issus de classes différentes.
  - Consiste à factoriser les propriétés d'un ensemble de classes, sous forme d'une super-classe, plus abstraite (permet de gagner en généricité).

Héritage : Exemple



Interfaces

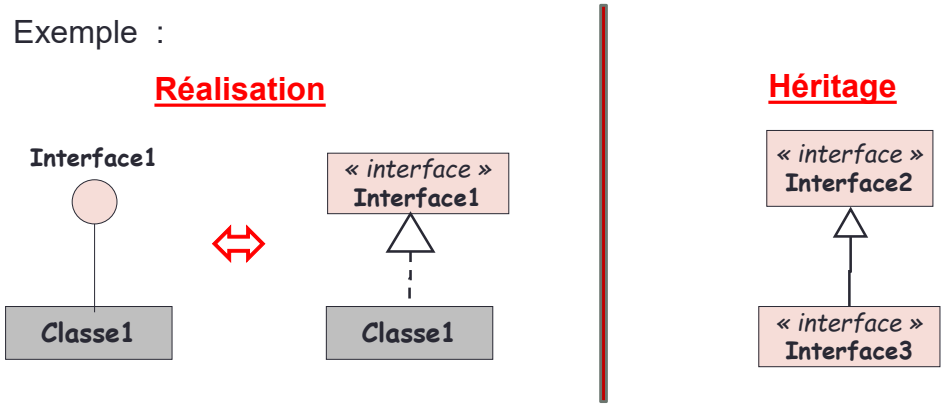
- Une interface est utilisée pour décrire le comportement visible d'une classe.
- UML représente une interface de deux manières possibles:
  - Un petit cercle
  - Une classe stéréotypé avec « interface ».

Réalisation

- Lorsqu'une classe hérite d'une interface nous parlons plutôt de réalisation que d'héritage.
- Certains langages comme Java appellent « implémentation » la relation de réalisation.
- Une classe qui réalise une interface est obligée de redéfinir toutes les méthodes de cette interface.
- **Remarque** : entre interfaces, nous parlons toujours d'héritage !

Réalisation

- La réalisation est notée comme une relation d'héritage mais avec une ligne en pointillée
- Exemple :



Associations

- Une association exprime une connexion sémantique entre deux classes.
- Le nom de l'association sert à comprendre sa sémantique
- Le petit triangle noir précise le sens de lecture

Navigabilité d'une association

- Une association peut être navigable :
  - Dans un seul sens : **unidirectionnelle**
  - Dans les deux sens : **bidirectionnelle**.
- Exemple d'association bidirectionnelle:

Navigabilité d'une association

- Lorsque l'association est bidirectionnelle, les flèches peuvent être omises.

- La réduction de la portée de l'association est souvent réalisée en phase d'implémentation

Multiplicité des associations

- La multiplicité ou la cardinalité d'une association précise le nombre d'instances qui participent à une relation.

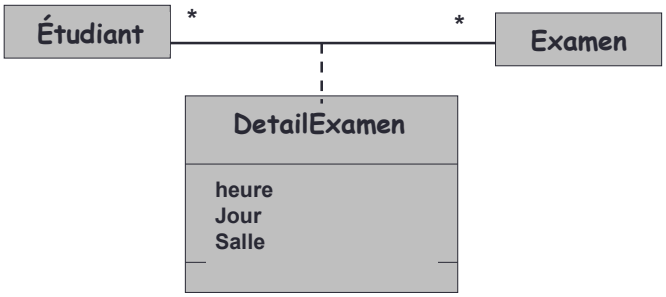
1	*Un et un seul*
0..1	*Zéro ou un*
M..N	*De M à N*
* Ou 0..*	*De 0 à plusieurs*
1..*	*D'un à plusieurs*

Rôles

- Un rôle spécifie la fonction d'une classe pour une association donnée (indispensable pour les associations réflexives).

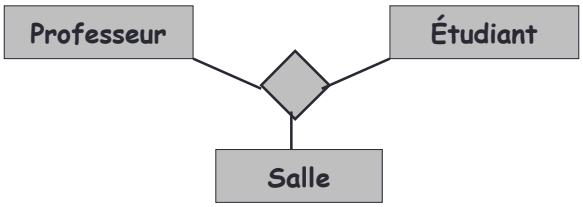
Classe d'association

- Il s'agit d'une classe qui réalise la navigation entre les instances d'autres classes.



Associations n-aire

- Une association peut relier **une**, **deux** ou **plusieurs** classes.
- Dans le cas où elle relie **n** classes elle est dite ***n-aire*** et dans le cas où elle relie **deux** classes, elle est dite ***binaire***.

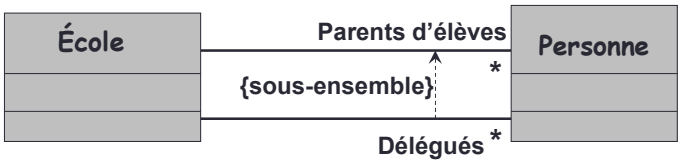


Contraintes sur les associations

- Contrainte {ordonnée}:
  - Peut être placée sur le rôle pour spécifier qu'une relation d'ordre décrit les objets placés dans la collection.
  - Le modèle ne spécifie pas comment les éléments sont ordonnés mais que l'ordre doit être maintenu durant l'ajout ou la suppression des objets par exemple.

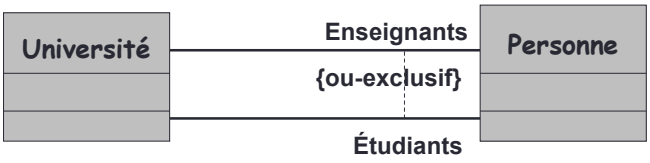
Contraintes sur les associations

- Contrainte {sous-ensemble}:
  - Indique qu'une collections est incluse dans une autre collection.



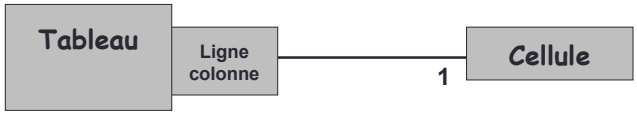
Contraintes sur les associations

- Contrainte {ou-exclusif}:
  - Indique que, pour un objet donné, une seule association parmi un groupe d'associations est valide.



Qualification

- La Qualification permet de sélectionner un sous-ensemble d'objets, parmi l'ensemble des objets qui participent à une association.
- La restriction de l'association est définie par une clé, qui permet de sélectionner les objets ciblés.

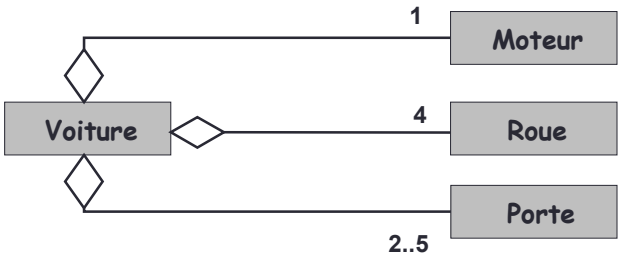


Agrégation

- L'agrégation est une association qui exprime une relation de type "ensemble / élément".
- Une agrégation peut notamment exprimer :
  - qu'une classe (un "élément") fait partie d'une autre ("l'agrégat"),
  - qu'un changement d'état d'une classe, entraîne un changement d'état d'une autre,
  - qu'une action sur une classe, entraîne une action sur une autre.

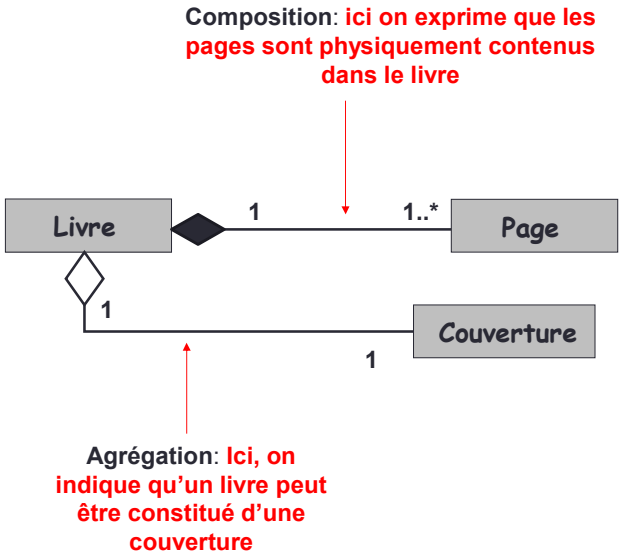
Agrégation

- Une instance d'élément agrégé peut exister sans agrégat (et inversement) : les cycles de vies de l'agrégat et de ses éléments agrégés peuvent être indépendants.



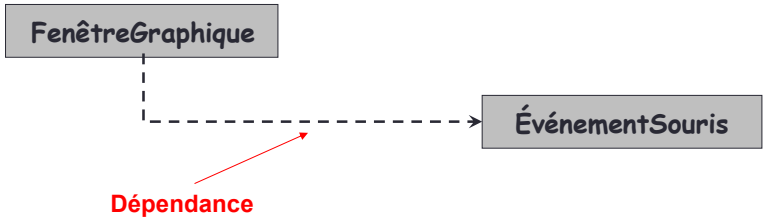
Composition

- La composition est une agrégation forte (agrégation par valeur).
- Les cycles de vies des éléments (les "composants") et de l'agrégat sont liés : si l'agrégat est détruit (ou copié), ses composants le sont aussi.
- A un même moment, une instance de composant ne peut être liée qu'à un seul agrégat.
- Les "objets composites" sont des instances de classes composées.



Relation de dépendance

- Est une relation d'utilisation unidirectionnelle et d'obsolescence (une modification de l'élément dont on dépend, peut nécessiter une mise à jour de l'élément dépendant).



Exercice

- Reprendre l'étude de cas définie dans le chapitre précédent
- Construire le diagramme de classes correspondant.
- Utiliser un logiciel de modélisation UML comme Visual Paradigm, Power AMC, MagicDraw,….