

UML

Pier Donini (Pier.Donini@heig-vd.ch)

Modélisation

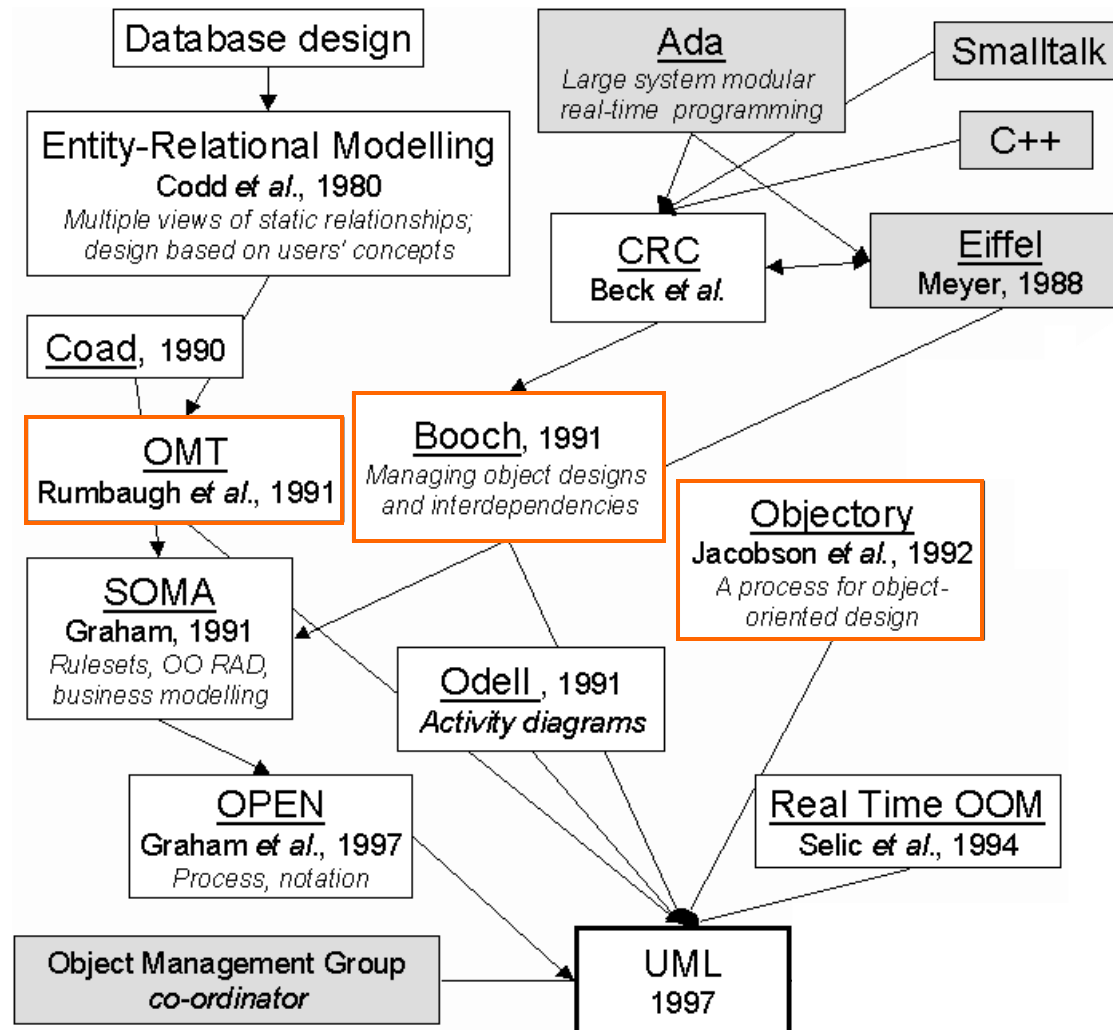
- La modélisation est un processus d'abstraction permettant de représenter un problème :
 - identifier les entités du monde réel et leurs traitements.

- Une composante essentielle du développement d'applications :
 - conception sans être contraint par des considérations d'implémentation ;
 - assurant que l'application représentera les objets nécessaires et remplira les fonctionnalités désirées ;
 - tout en satisfaisant aux exigences de documentation, sécurité, extensibilité, réutilisabilité, maintenance... ;
 - **avant** qu'une implémentation ne rende des modifications très difficiles.

UML

- Unified Modeling Language (langage de modélisation unifié).
- Permet de spécifier et de visualiser les composants d'une application.
- Basé sur une sémantique précise et une notation graphique expressive.
- Repose sur plusieurs types de diagrammes (classe, cas d'utilisation, etc).
- Est devenu **la** référence pour la modélisation objet.
- Né de la fusion de plusieurs méthodes (par Booch, Jacobson et Rumbaugh).
- Défini par le consortium OMG (*Object Management Group*, www.omg.org).
- UML n'est pas associé à une méthodologie particulière (pas de description du processus d'élaboration des modèles). Cf., génie logiciel...

Influences



Références UML

■ Manuels

- Le guide de l'utilisateur UML, *Booch, Rumbaugh, Jacobson*
- Modélisation objet avec UML, *Muller, Gaertner, Eyrolles*
- UML distilled, *Fowler, Scott*

■ Outils UML open source

- **Slyum** (heig-vd), github.com/HEIG-GAPS/slyum
- ArgoUML, argouml-tigris-org.github.io
- StarUML, staruml.io
- UMLet, umlet.com
- PlantUML (ML), plantuml.com
- Mermaid (ML), mermaid.js.org
- Autres ?

Références UML (2)

- Outils UML gratuits
 - BOUML, www.bouml.fr
 - Poséidon CE, www.gentleware.com/ce.html [404 ? → Google]
 - IntelliJ IDEA CE, www.jetbrains.com/idea
 - JDeveloper (orienté Java), www.oracle.com/application-development/technologies/jdeveloper.html
 - Visual paradigm, visual-paradigm.com/download/community.jsp
 - Draw.io, drawio.com
 - Yed, yworks.com/products/yed
- Outils UML payants
 - IntelliJ IDEA, jetbrains.com/idea
 - Visual paradigm, visual-paradigm.com
 - StarUML, staruml.io
 - Lucidchart, lucidchart.com

Diagrammes UML

- Diagrammes de **cas d'utilisation** (use case)
 - Partition des fonctionnalités du système selon les besoins *acteurs* (*quoi*, pas *comment*). Scénarii non techniques...
- Diagrammes de **classes**
 - Cœur du langage.
 - Description statique des classes et des liens entre elles.
- Diagrammes d'**objets**
 - Description d'instanciations particulières du diagramme de classes.
- Diagrammes de **séquences**
 - Description dynamique des opérations (quels objets, quels messages).
 - Organisés de manière temporelle.
- Diagrammes de collaborations, d'états, d'activités, de composants, de déploiement...

Diagrammes de cas d'utilisation

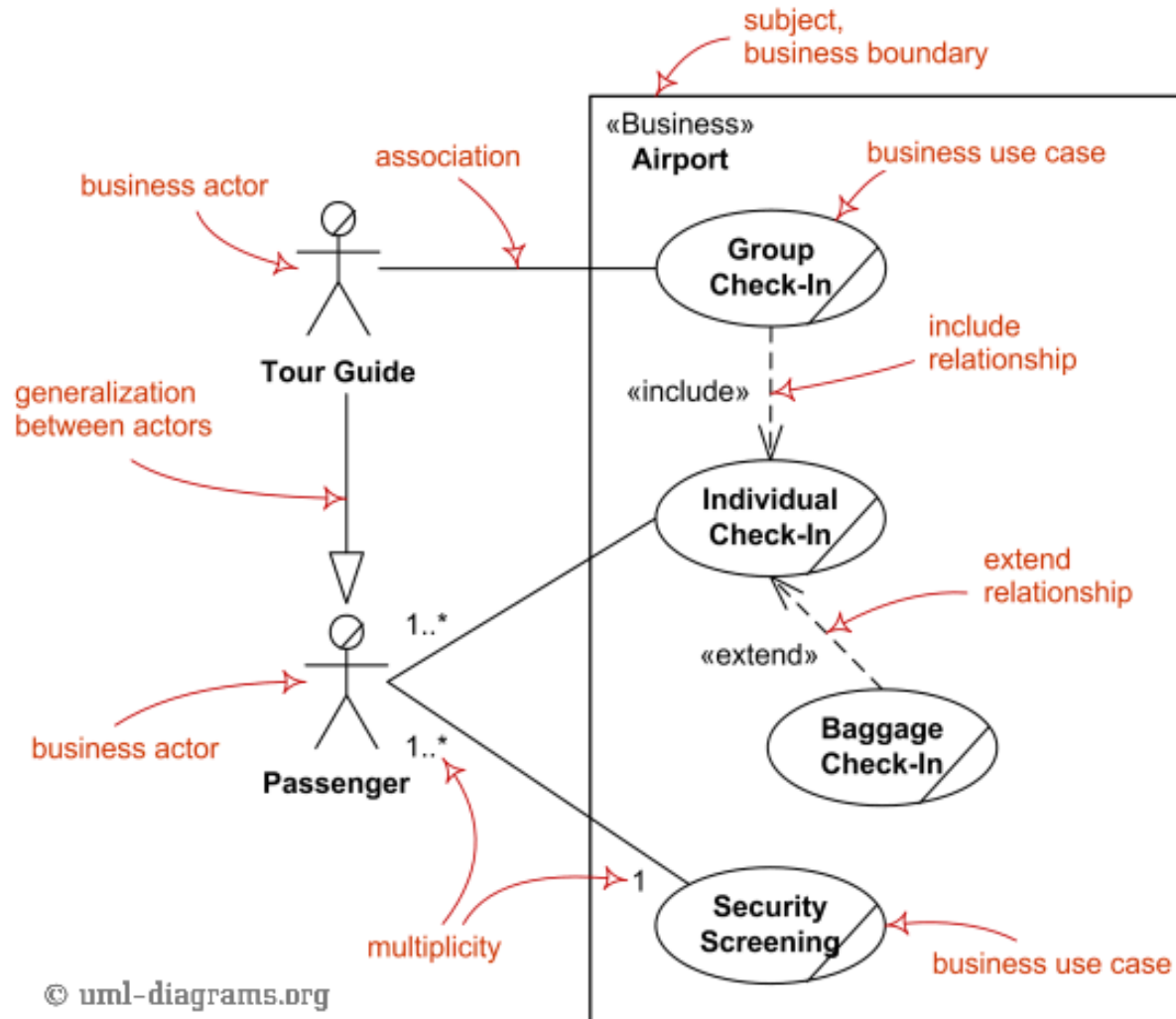
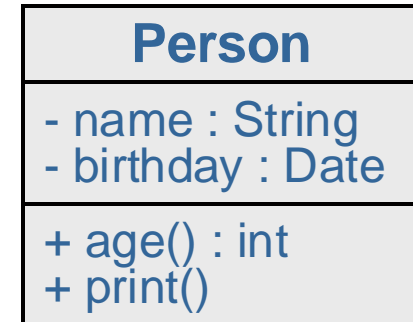


Diagramme de classes

- Modélisation statique de la réalité.
- **Classes**
 - Représentent un ensemble d'entités du monde réel perçues dans un contexte donné.
- **Propriétés** des classes
 - Attributs.
 - Opérations (méthodes).
- **Liens** entre les classes
 - Héritages.
 - Associations.

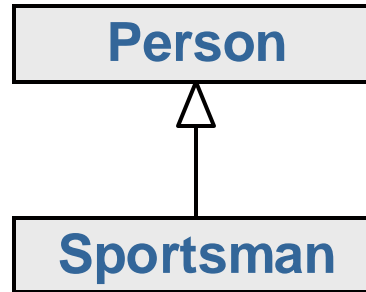
Classes

- Une **classe** est décrite par :
 - son **nom** ;
 - ses **attributs** et leur visibilité ;
 - ses **méthodes** et leur visibilité.
 - Les **propriétés de classe** (attributs et méthodes non liées à un objet, mais à la classe elle-même) sont soulignées.
 - Attributs et méthodes ne sont pas nécessairement représentés.
- Visibilités :
 - + publique
 - - privée
 - # protégée
 - ~ (ou rien) paquetage



Classes (2)

- **Héritage** : dénoté par une flèche en trait plein.



- Rappel
 - Les **propriétés** (attributs et méthodes) de la super classe sont **héritées** dans la sous-classe.
 - La sous-classe peut définir de **nouvelles propriétés**.
 - Les **méthodes** héritées peuvent être **redéfinies** (\Rightarrow liaison dynamique).

Classes (3)

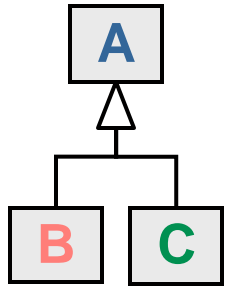
■ Classe abstraite

- Représentation d'entités abstraites (n'existant pas en tant que telles) : non instanciable directement, elle nécessite la définition de sous-classes.
- Exemple : une classe abstraite `Animal` et ses sous-classes `Chat`, `Chien`...
- Permet de définir des propriétés communes à une hiérarchie.
- Peut posséder des méthodes concrètes ou abstraites.
- Les classes abstraites sont représentées en *italique*.

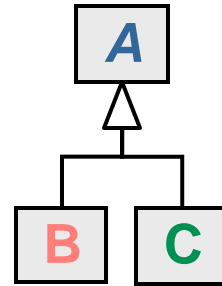
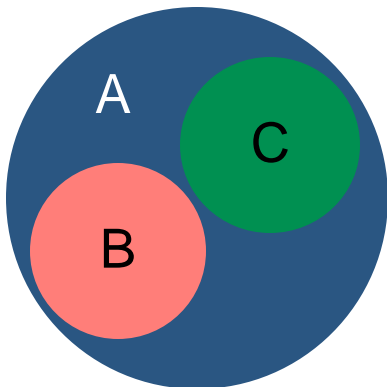
■ Méthode abstraite

- Seule sa signature est définie, pas son implémentation.
- Une classe déclarant une méthode abstraite est forcément abstraite et l'implémentation de la méthode doit être fournie dans les sous-classes.
- Les méthodes abstraites sont représentées en *italique*.
- Exemple : la classe abstraite `Animal` définit une méthode abstraite `son()` qui sera **redéfinie** pour faire aboyer pour un `Chien` et miauler un `Chat`.

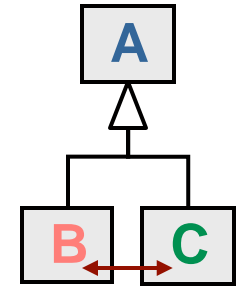
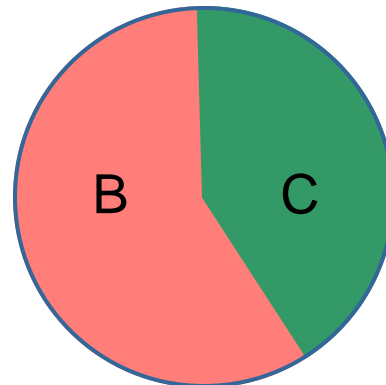
Héritage : populations



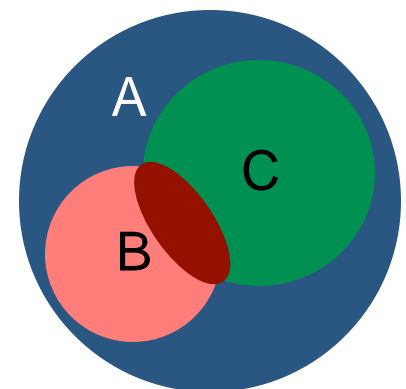
Disjonction
 $B \cap C = \emptyset$
 $B \cup C \neq A$



Partition
 $B \cup C = A$
 $B \cap C = \emptyset$



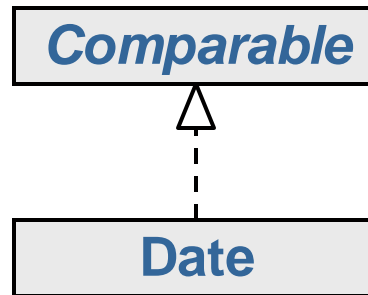
$B \cap C \neq \emptyset$
 Pas en OO
 Possible en BD



Classes (3)

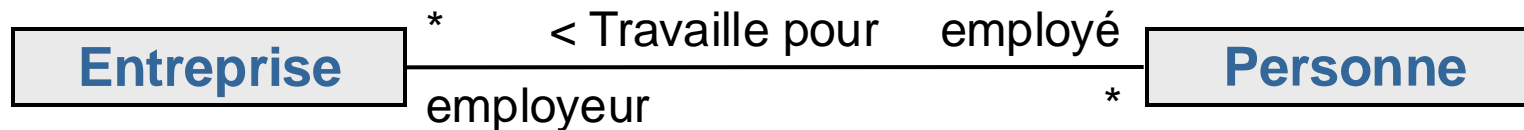
■ Interface

- Ensemble de méthodes abstraites que peuvent implémenter des classes (pour garantir un comportement).
 - Ne possède pas d'état (attributs).
 - Une interface est représentée en *italique*.
- L'implémentation d'une interface par une classe est représentée par une flèche en traitillés.



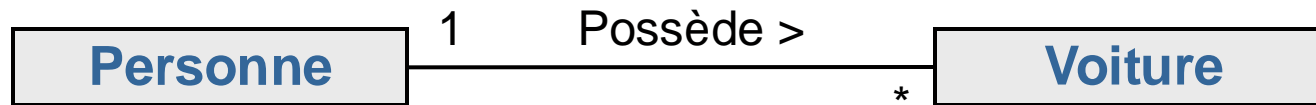
Associations

- Une **association** représente une relation qui existe entre plusieurs objets où chaque objet joue un *rôle* déterminé.
- Représentée par un lien reliant deux classes.
- Une association est caractérisée par :
 - un **nom** (généralement un verbe conjugué) éventuellement suivi d'une flèche précisant le sens de lecture ;
 - un **rôle** pour chacune des classes participantes – le rôle d'une classe précise comment cette classe est *vue* par l'autre classe ;
 - le nom ou les rôles d'une association doivent être spécifiés.
 - des **cardinalités** (ou multiplicités) indiquant le nombre d'objets liés par une association.

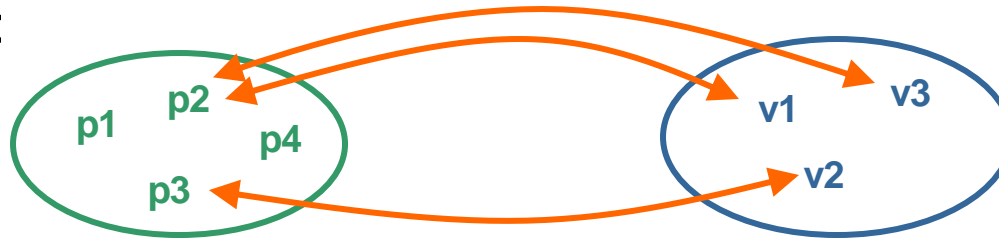


Cardinalités

- Les **cardinalités** précisent combien d'objets de chaque classe peuvent être liés à un objet de l'autre classe par l'association.
 - « Une personne peut posséder plusieurs voitures, mais une voiture n'a qu'un seul propriétaire »



- Populations :



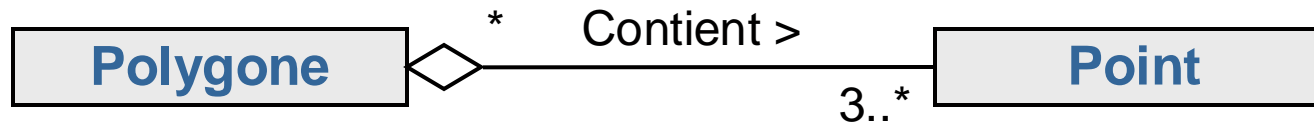
- Notation :

- | | | | |
|-------------|---------------------|--------|---------------------------|
| ● 1 | 1 et un seul | ● n | exactement n (n entier) |
| ● 0..1 | zéro ou 1 | ● n..m | de n à m (entiers), m > n |
| ● 0..* ou * | de zéro à plusieurs | ● n..* | n (entier) ou plus |
| ● 1..* | au moins 1 | | |

Agrégations

- **Agrégation** : association spécialisée indiquant qu'un *tout* « est composé de » *parties*.

- « Un polygone est composé de plusieurs points et un point peut appartenir à plus d'un polygone. »



- **Composition** : agrégation spécialisée décrivant une contenance structurelle.

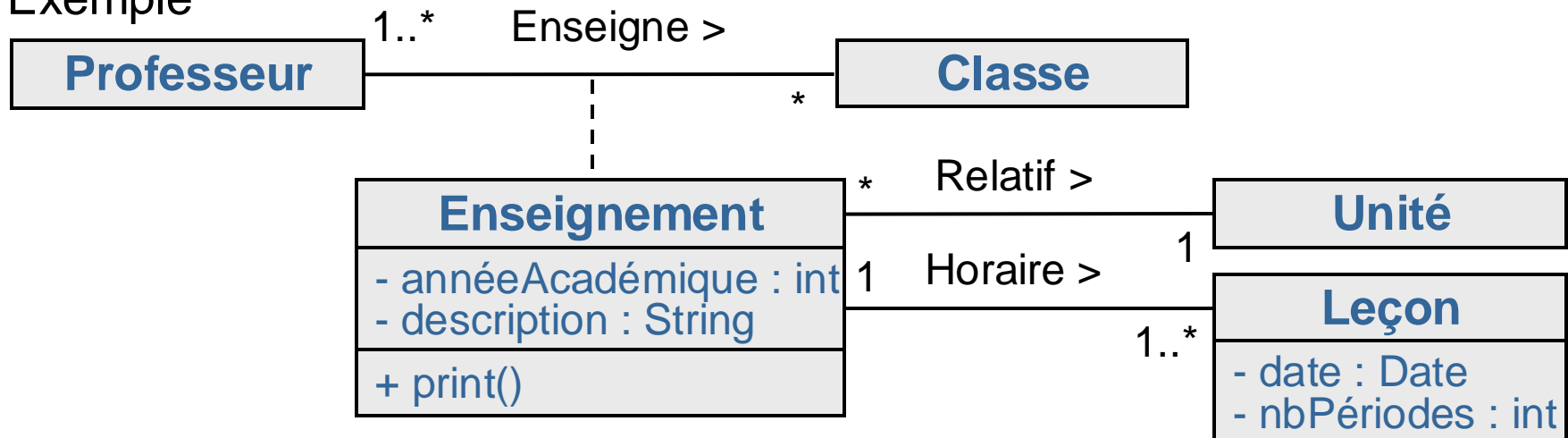
- La destruction d'un objet composite (le *tout*) implique la destruction de ses objets composants (les *parties*).
- Un objet composant ne peut faire partie que d'un seul objet composite. La cardinalité du côté du composite est au maximum 1 (1 ou 0..1).
- « Un ordinateur est composé de ses pièces. »



Classe d'association

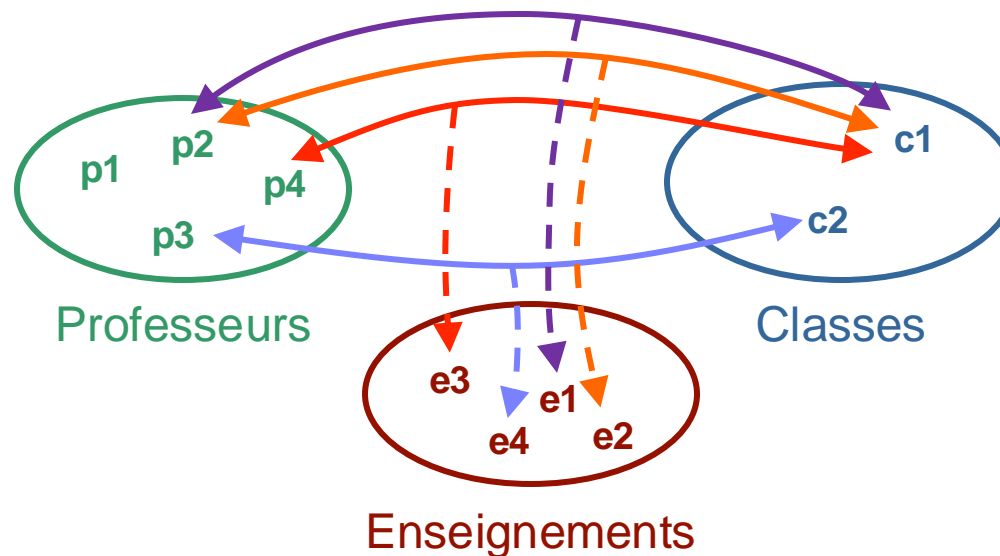
- Classe pour représenter les **propriétés propres à une association**.
 - Reliée à l'association par une ligne en traitillé.
 - Cette classe n'est pas nécessairement nommée (elle est identifiable par l'association concernée).
 - Elle peut être liée à d'autres classes.
 - Chaque occurrence de l'association est liée **à un et un seul** objet de la classe d'association, et réciproquement.

- Exemple




Classe d'association (2)

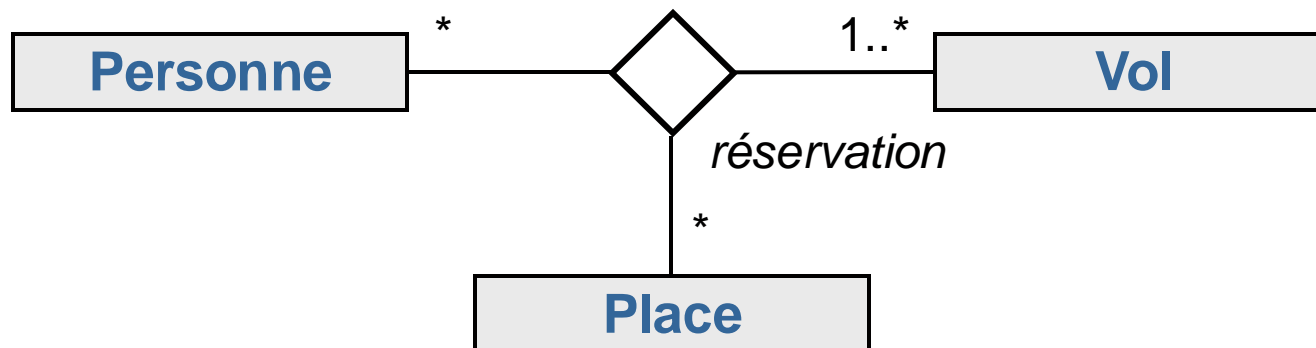
- Chaque occurrence de l'association **Enseigne**, représentée par un couple d'objets (**professeur**, **classe**), est liée à **un et un seul objet** enseignement, et réciproquement.
- Un même **professeur** et **classe** peuvent être liés par différentes occurrences de l'association **Enseigne** (plusieurs **enseignements** donnés par le même professeur à la même classe).



Arité des associations

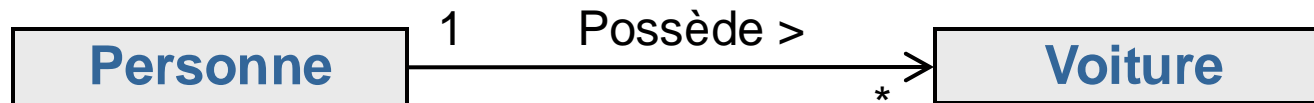
- Généralement lien binaire entre classes différentes (mais il peut exister plus d'une association entre deux classes données),
- Il peut exister des associations **réflexives**,

```
classDiagram
    class Personne
    Personne "*" -- "0..2" Personne : parent
    Personne "0..2" -- "*" Personne : enfant
```
- Ou **n-aires**.
 - Non triviales à définir (dans le doute préférer des associations binaires).
 - Une occurrence de l'association lie des objets de chacune des classes.
 - « Une place est réservable dans un vol pour une personne. »



Navigabilité

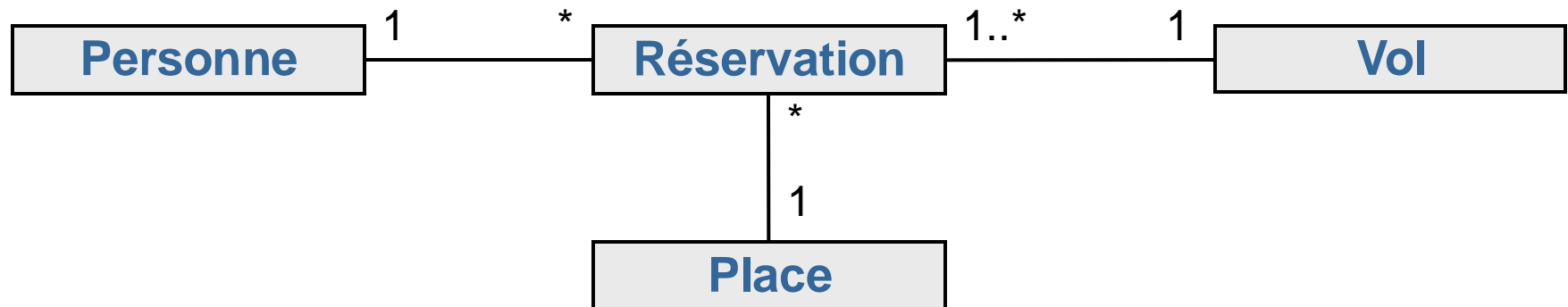
- Par défaut les associations peuvent être parcourues dans tous les sens.
- Afin de réduire le couplage entre classes, une propriété de navigabilité, dénotée par une flèche, peut être attachée à une des extrémités d'une association.



- Elle indique dans quel sens il est possible de traverser l'association lors de l'implémentation du diagramme de classe.
- Les objets de la classe *cible* participant à l'association doivent pouvoir être atteints directement depuis la classe *source*, mais non l'inverse.
- Ici, un objet **Personne** référence directement l'ensemble de ses objets **Voiture** qui, eux, ne référencent pas directement leur propriétaire.

Implémentation des associations

- Par des classes spécifiques :
 - pour implémenter des associations n-aires
(attention à l'inversion des cardinalités par rapport au schéma original) ;



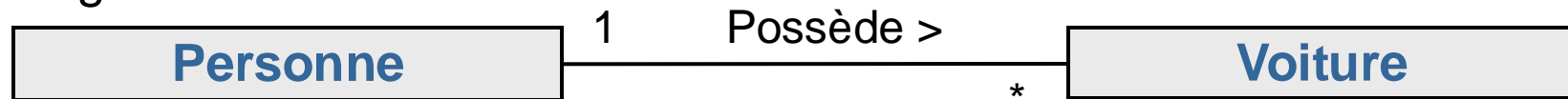
- pour implémenter des classes d'associations
(attention au déplacement des cardinalités sur la classe d'association).



Implémentation des associations (2)

- Dans les classes liées, selon les cardinalités et la navigabilité, définir :
 - des attributs références (pour les cardinalités 0..1 ou 1) ou
 - des collections (tableau, liste...) de références (pour les cardinalités > 1).
- Ne **jamais** indiquer dans un diagramme de classes les références ou les collections de références utilisées pour implémenter les associations.
- Une implémentation de l'association **Possède** (navigable dans les deux sens) :

- Diagramme de classes

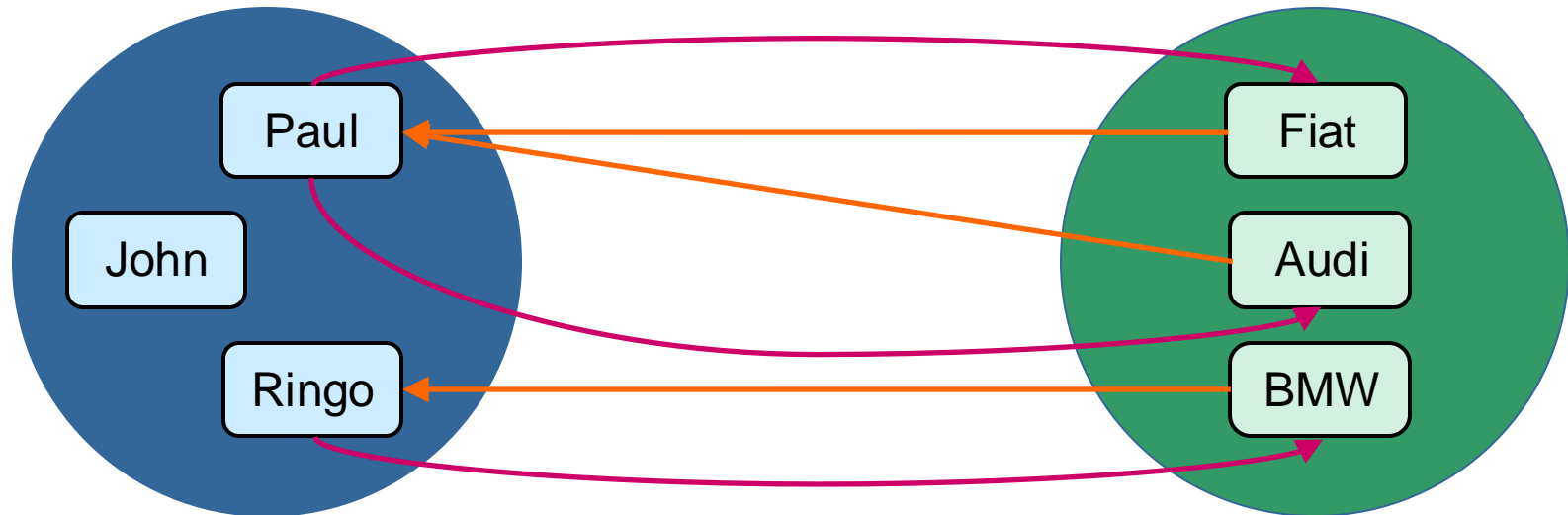


- Implémentation



Implémentation des associations (3)

- Chaque occurrence de **Possède**, entre une personne et une voiture, induit deux références : *Personne* → *Voiture* et *Voiture* → *Personne*



- Attention à maintenir la cohérence des références lors de l'implémentation d'associations navigables dans les deux sens.
 - Paul* possède une *Fiat* ⇔ la *Fiat* a pour propriétaire *Paul*
 - Ringo* vend sa *BMW* à *Paul* ➔ 1) *Paul* possède la *BMW*, 2) la *BMW* a pour propriétaire *Paul*, 3) *Ringo* ne possède plus la *BMW*

Example

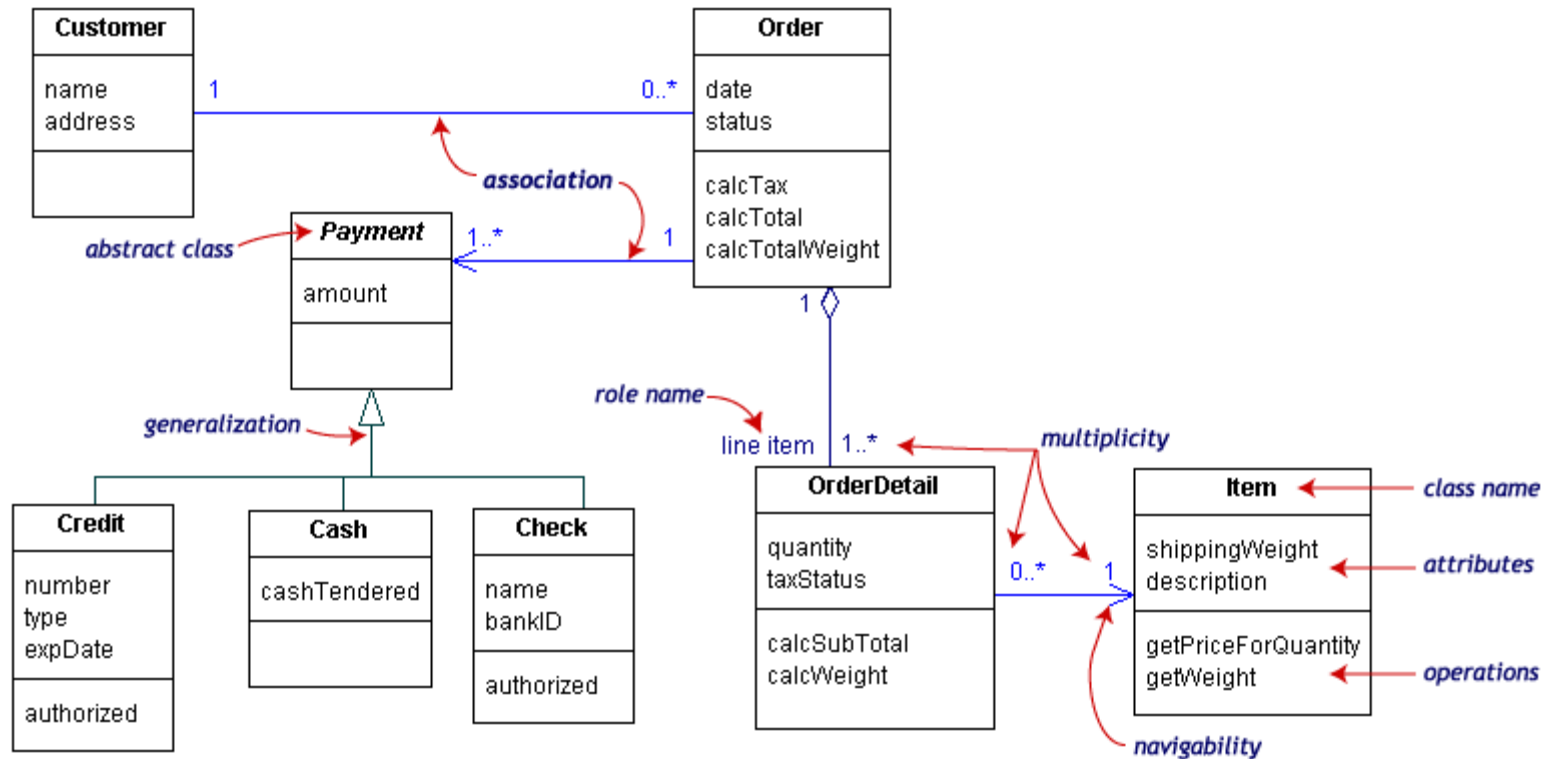


Diagramme d'objets

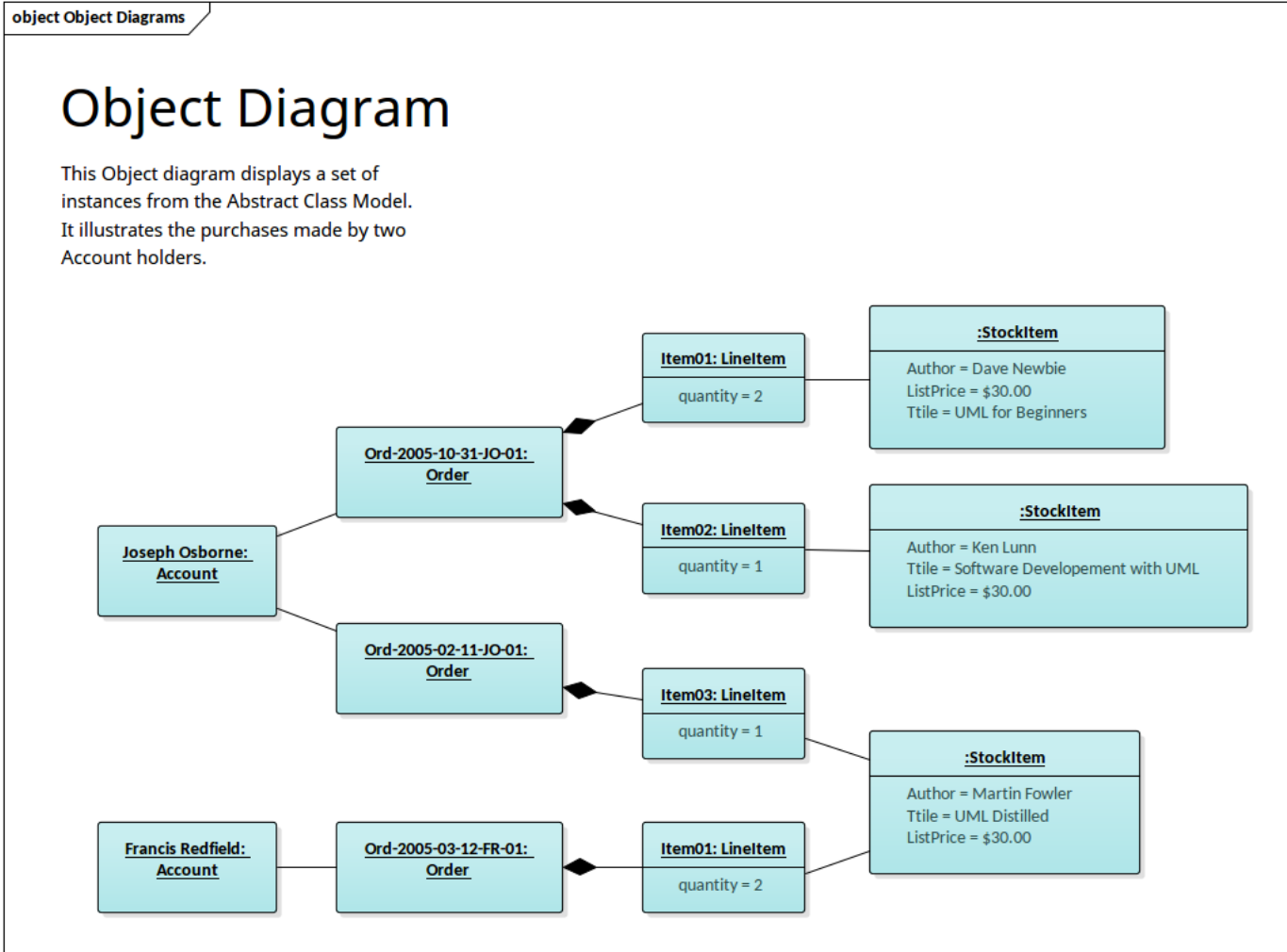


Diagramme de séquences

