# UML

# Introduction

- UML (*Unified Modeling Language*) est un langage unifié pour la modélisation objet.
  - UML est un **langage** (**de modélisation** objet) et propose une notation et une sémantique associée à cette notation (i.e. des modèles), mais pas de processus (i.e. de démarche proposant un enchaînement d'étapes et d'activités qui mènent à la résolution d'un problème posé).
  - UML **unifie** des méthodes objet, et plus particulièrement les méthodes Booch'93 de G. Booch, OMT-2 (Object Modeling Technique) de J. Rumbaugh et OOSE (Object-Oriented Software Engineering) d'I. Jacobson.
  - UML a une approche entièrement (i.e. couvrant tout le cycle de développement : analyse, conception et réalisation) **objet** (et non fonctionnelle) : le système est décomposé en objets collaborant (plutôt qu'en tâches décomposées en fonctions plus simples à réaliser).

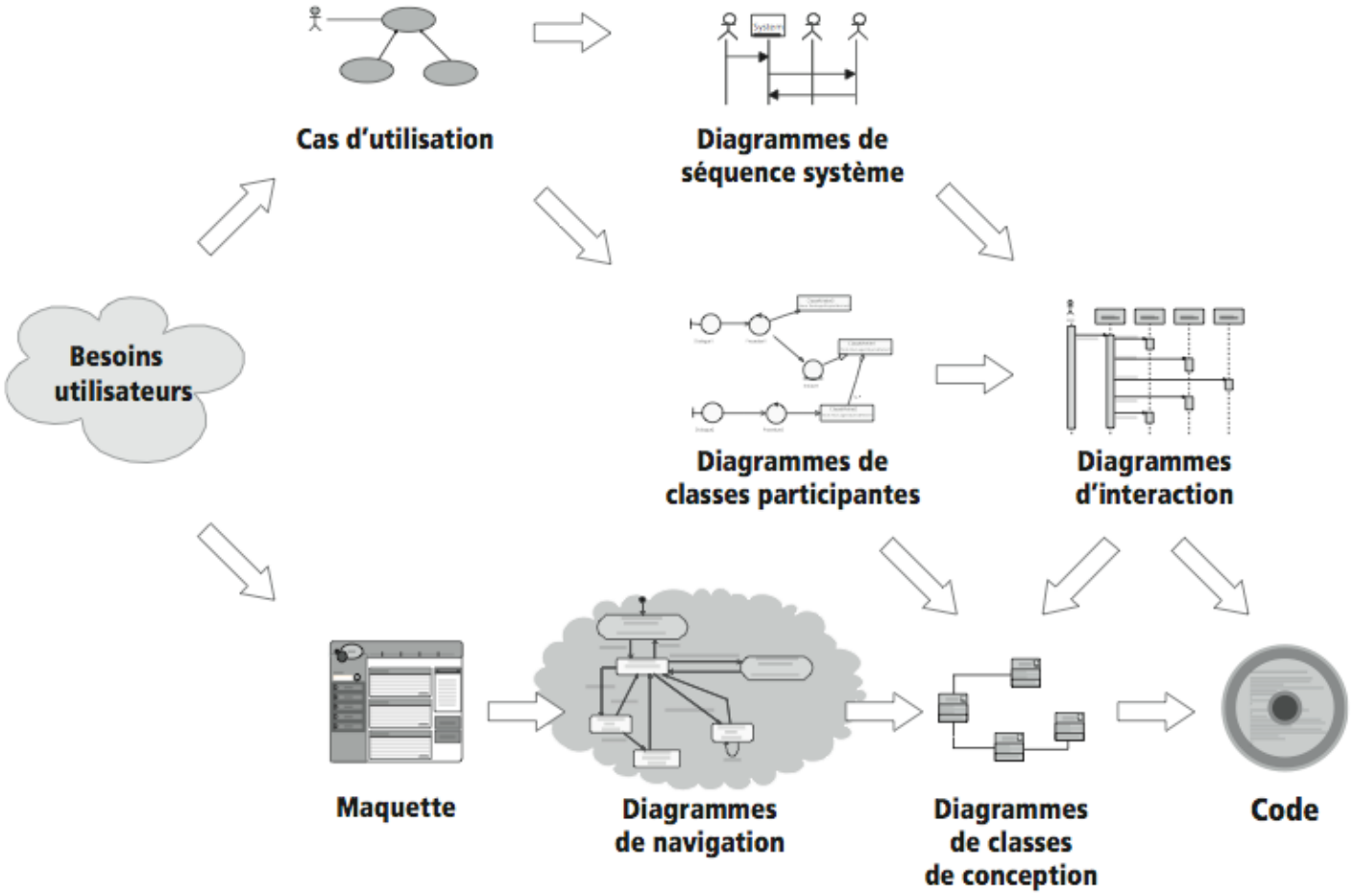
# Différents diagrammes

- Diagrammes capturant l ' aspect statique du système
  - Diagramme des cas d ' utilisation
  - Diagramme de classe et de paquetage
  - Diagramme d ' objet
  - Diagramme de composants
  - Diagramme de déploiement

# Différents diagrammes (2)

- Diagrammes capturant l ' aspect dynamique du système
  - Diagramme de collaboration
  - Diagramme de séquence
  - Diagramme d ' activité
  - Diagramme d ' état-transition

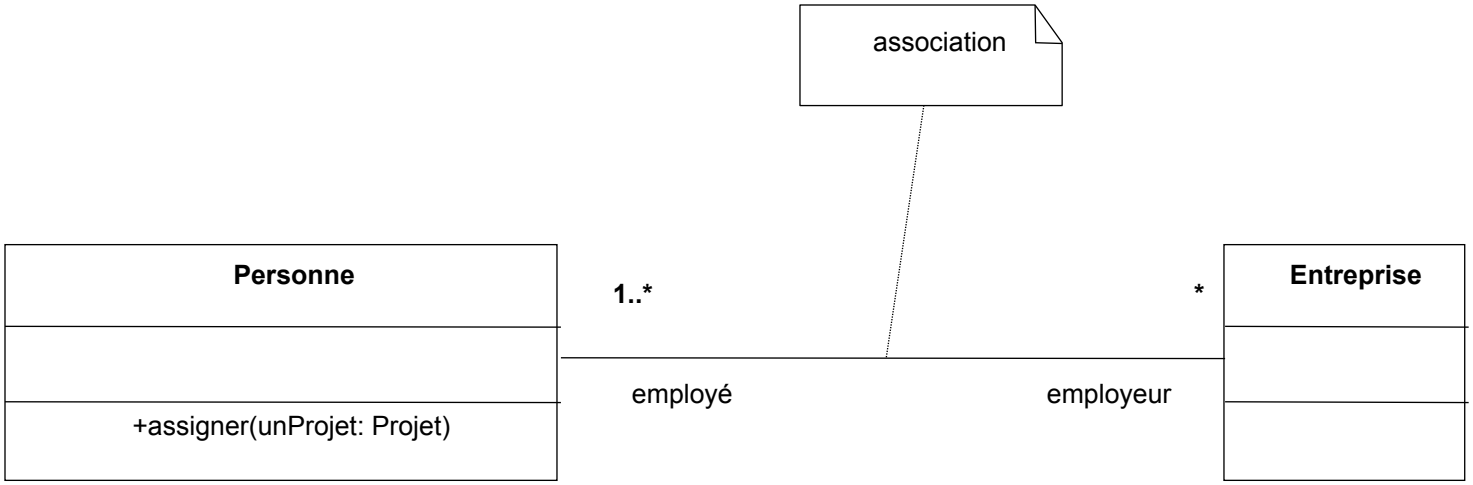
# Différents diagrammes (3)

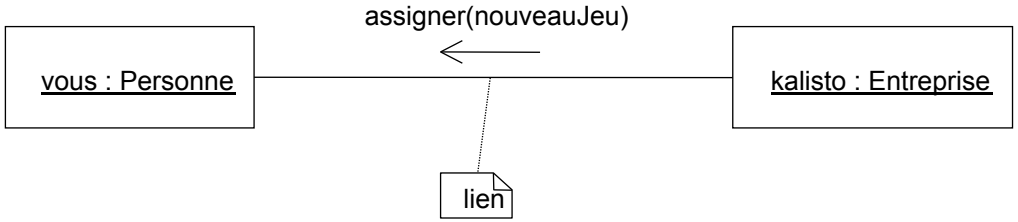


# Modèle de classes

- Saisit la structure statique d'un système en montrant les objets dans le système, les relations entre les objets, les attributs et les opérations qui caractérisent chaque classe d'objets
- Deux diagrammes
  - Diagrammes de classes
  - Diagrammes d’objets

# Diagramme de classes/Diagramme d'objets



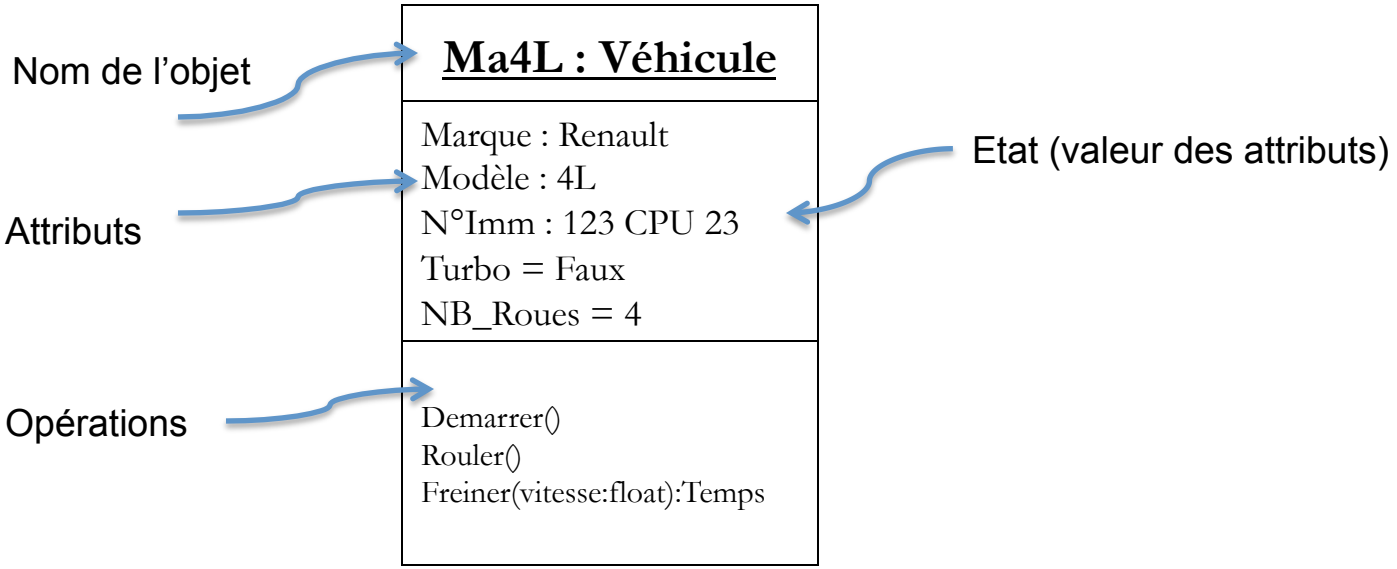



# Diagramme d’objets

- Représentation de l'état du logiciel (objets + relations)
- Diagramme évoluant avec l'exécution du logiciel
  - création et suppression d'objets
  - modification de l'état des objets (valeurs des attributs)
  - modification des relations entre objets

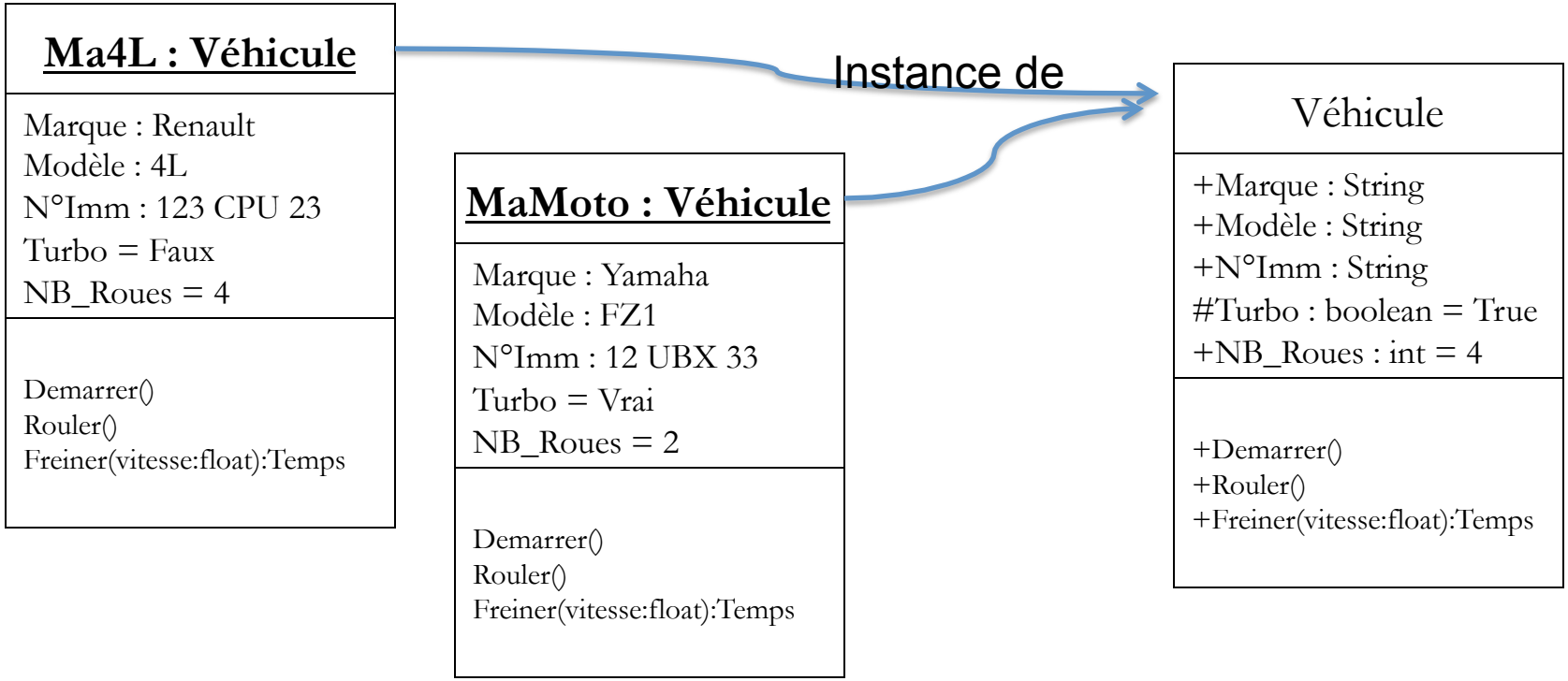
# Diagramme d'objets

- Objet
  - Entité concrète ou abstraite du domaine d'application
  - Décrit par état (attributs) + comportement (opérations)

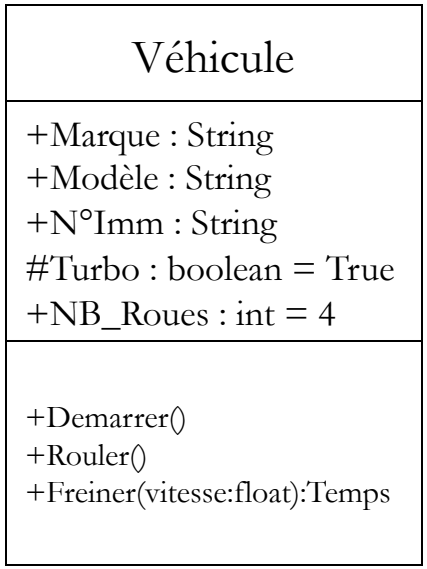


- Classe : Regroupement d'objets de même nature (mêmes attributs + mêmes opérations)

➔ Objet = instance d'une classe



# Description d'une classe



# Description d'une classe (2)

- Les attributs
  - Syntaxe complète :

    <visibilité> <nomAttribut>:<type>=<valeur par défaut>

  - Visibilité :
    - « + » pour une opération publique, c'est-à-dire accessible à toutes les classes (*a priori* associée à la classe propriétaire) ;
    - « # » pour les opérations protégées, autrement dit accessibles uniquement par les sous-classes de la classe propriétaire ;
    - « - » pour les opérations privées, inaccessibles à tout objet hors de la classe.

# Description d'une classe (3)

- Les opérations (ou méthodes)
  - Une opération est un travail qu'une classe doit mener à bien, un contrat qu'elle s'engage à tenir si une autre classe y fait appel.
  - Pour la programmation, il s'agit d'une méthode de la classe.
  - Syntaxe complète :

    <visibilité><nomOpération>(listeParamètres):typeRetour {propriété}

    propriété est un prédicat ou un invariant que doit satisfaire l'opération

# Relations entre classes

- Associations déclinées en
  - Association simple
  - Agrégation
  - Composition
- Généralisation/Spécialisation
- Réalisation

# Associations

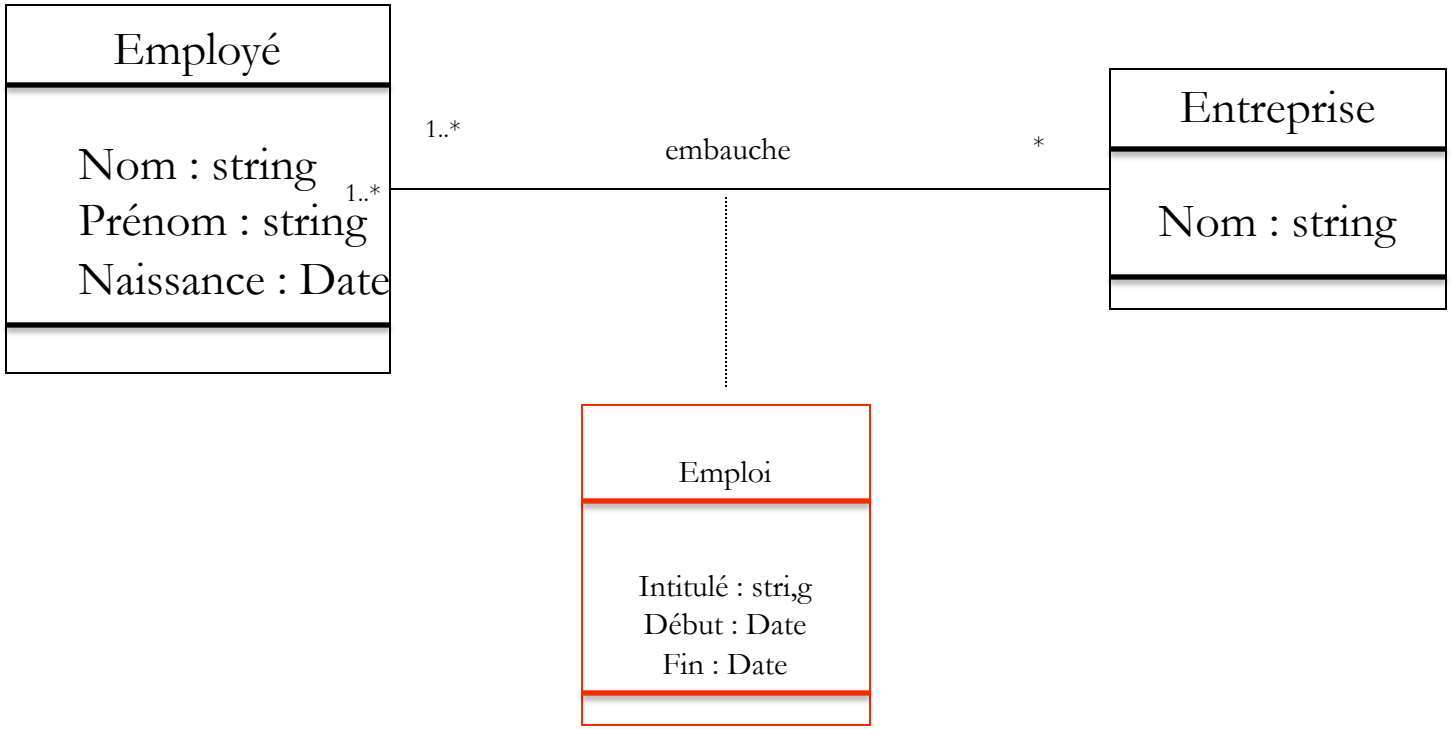
- Une association représente les liens qui existent entre les instances des classes associées. Plusieurs liens entre objet peuvent se projeter sur une même association
- L'association exprime une connexion sémantique bidirectionnelle entre classe

# Associations (2) : les cardinalités

- Un rôle est doté d'une multiplicité qui fournit une indication sur le nombre d'objets d'une même classe participant à l'association.
  - 1 : Obligatoire (un et un seul)
  - 0..1 : Optionnel (0 ou 1)
  - 0..* ou * : Quelconque
  - n..* : Au moins n
  - n..m : entre n et m
  - l,n,m : l , n, ou m
- les cardinalités se lisent en UML dans le sens inverse du sens utilisé dans MERISE

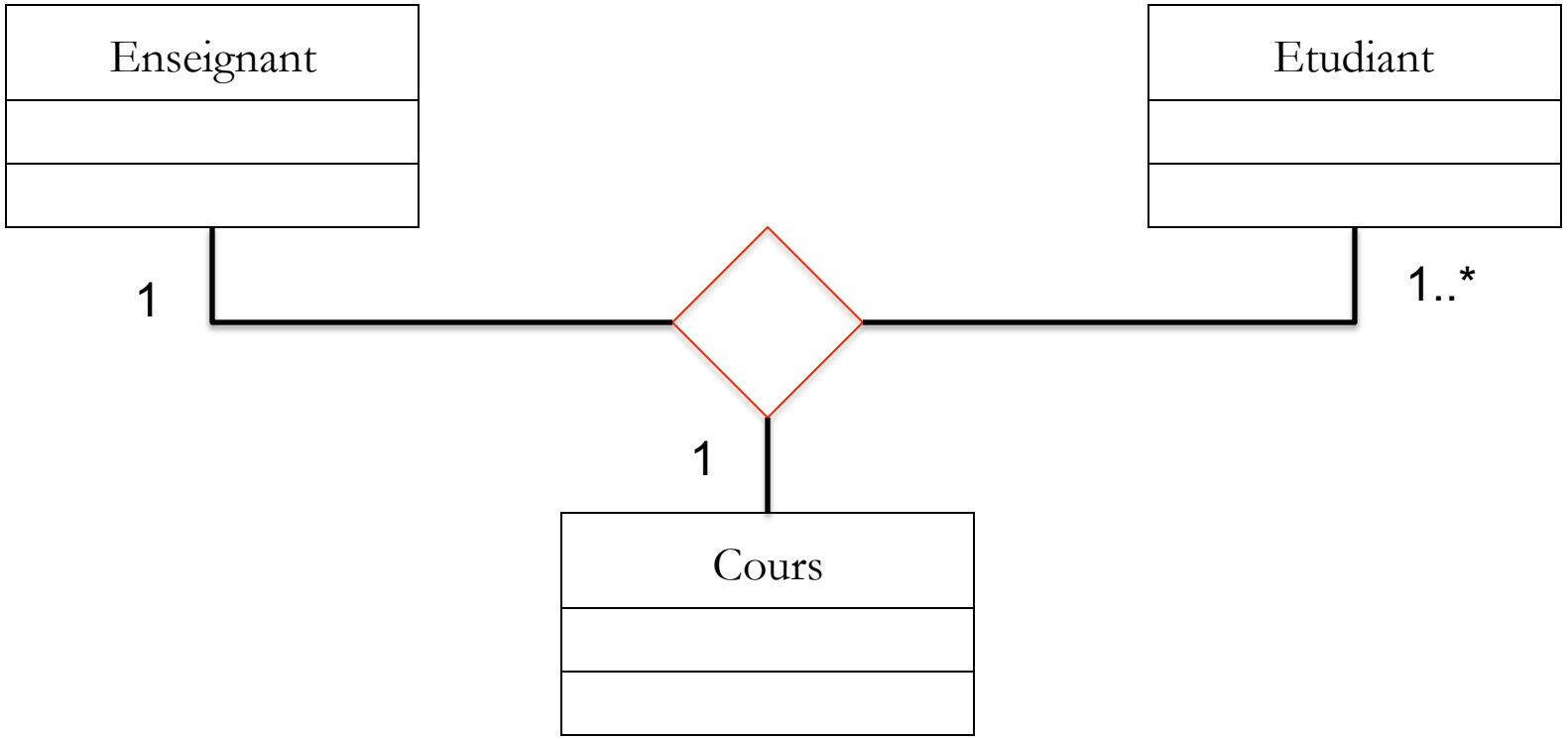
# Associations (3)

- Classe- association : Permet de paramétrer une association entre deux classes par une classe



# Association *n*-aire

- Association reliant plus de deux classes



# Un exemple complet

(d'après le cours de M. **Aouiche** )

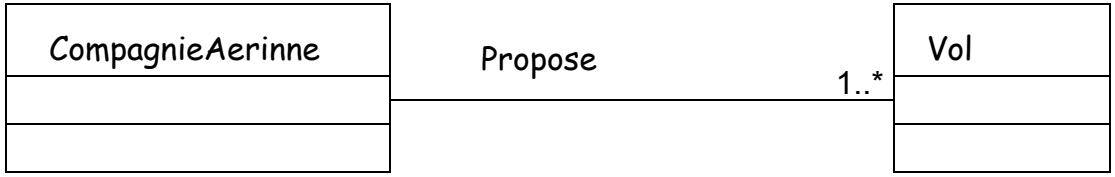
## Soit le cas ***''Réservation de vols dans une agence de voyage''***

1. Des compagnies aériennes proposent différents vols.
2. Un vol est ouvert à la réservation et fermé sur ordre de la compagnie.
3. Un client peut réserver un ou plusieurs vols, pour des passagers différents.
4. Une réservation concerne un seul vol, et un seul passager.
5. Une réservation peut être annulée ou confirmée.
6. Un vol a un aéroport de départ et un aéroport d'arrivée.
7. Un vol a un jour et une heure de départ et un jour et une heure d'arrivée.
8. Un vol peut comporter des escales dans des aéroports
9. Une escale a une heure d'arrivée et une heure de départ.
10. Chaque aéroport dessert  une ou plusieurs villes

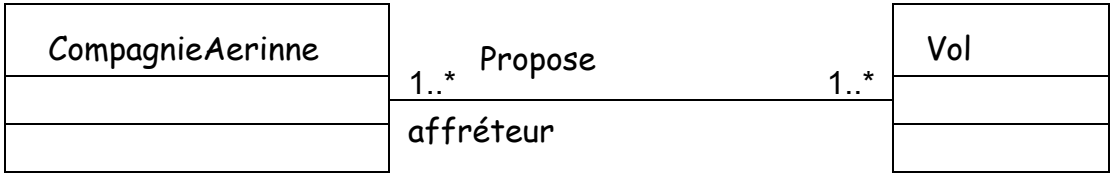
➢ Modélisation de la phrase :

1° Des compagnies aériennes proposent différents vols.

*CompagnieAerienne* et *Vols* sont 2 objets métiers : 2 classes

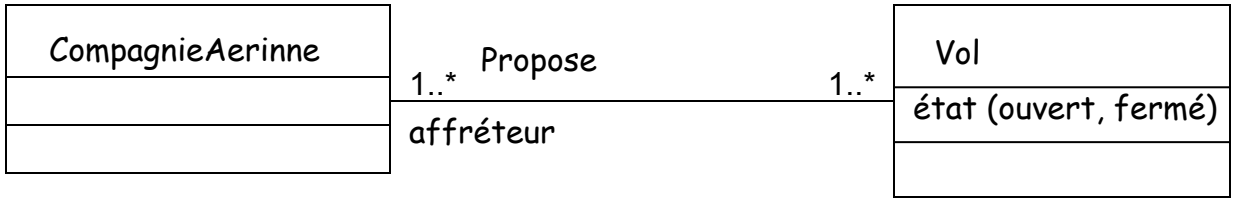



- Un vol est réalisé par une seule compagnie mais partagé par plusieurs affréteurs

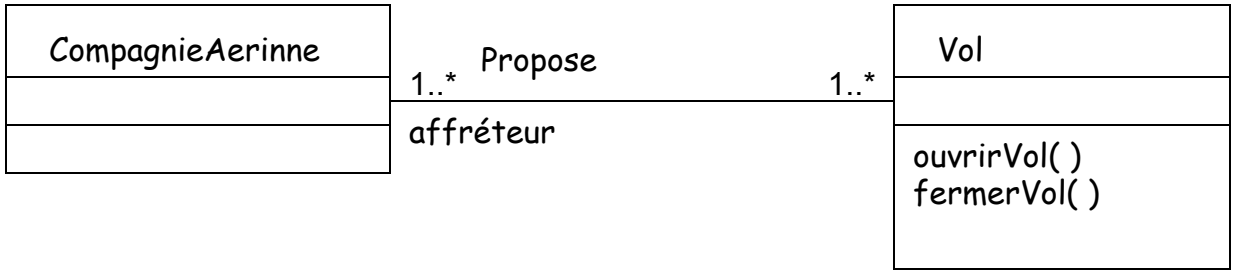


## Modélisation de la phrase :

2° Un vol est ouvert à la réservation et fermé sur ordre de la compagnie.




- Dans un diagramme de classes tout concept dynamique est modélisé en opération.
- Il faut représenter la 2° phrase par 2 opérations : *ouvrirVol( )* et *fermerVol( )*
- Dans quelle classe ? Responsabilité d' une classe

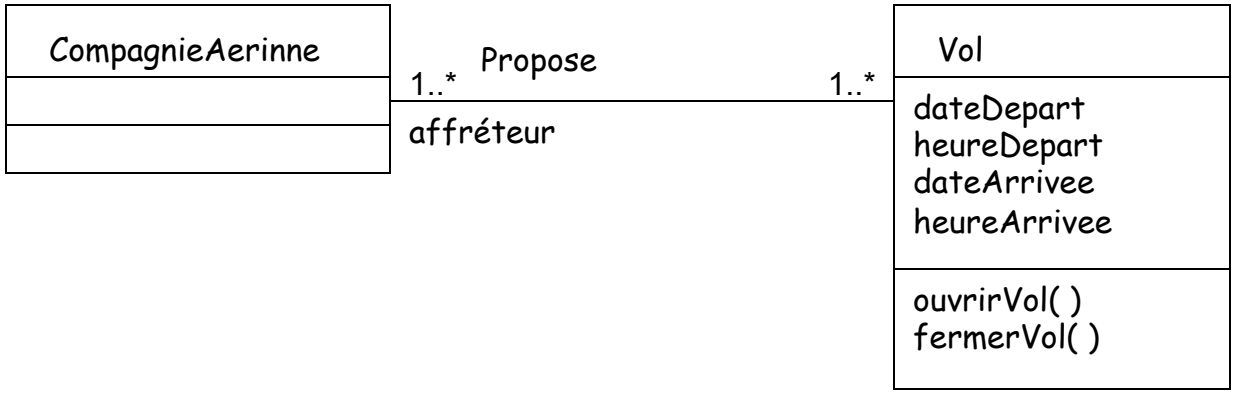



- Les opérations sont déclarées dans l' objet dans lequel elles doivent s' exécuter
- Les autres pourront déclencher ces opérations par envoi de messages
- Le classe CompagnieAerienne a une association avec la classe vol.

- Modélisation des phrases :

7° Un vol a un jour et une heure de départ et un jour et une heure d' arrivée.

- Les dates et les heures de départ et d' arrivée ne représentent que des valeurs : attributs.

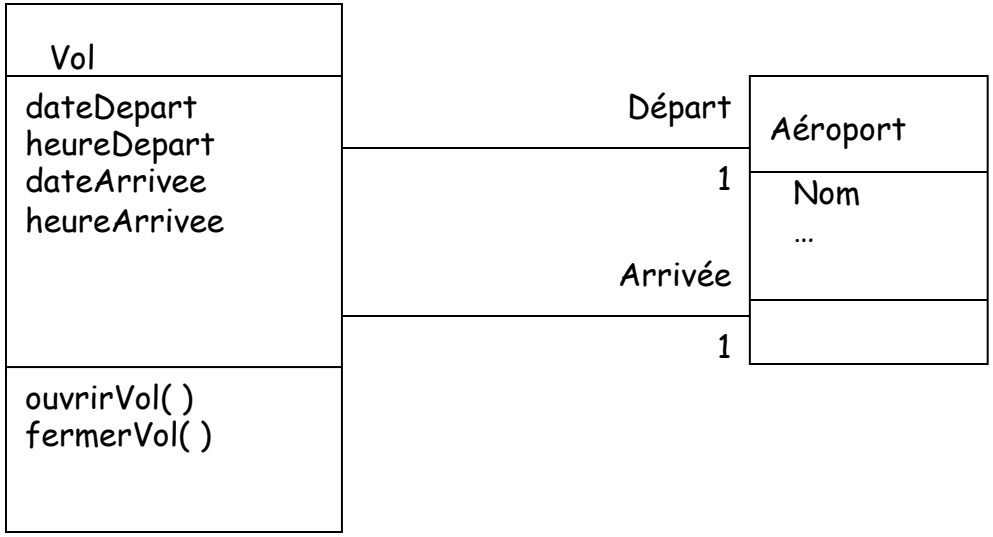



- Pour savoir si un élément doit être représenté en attribut ou en objet :
- S' il n' y a que sa valeur qui est intéressante : c' est plutôt un attribut.
- Si plusieurs questions peuvent concerner l' élément, alors il faut le représenter en objet.

- Modélisation des phrases :

  6° Un vol a un aéroport de départ et un aéroport d'arrivée.

  2. Soit avec 2 associations

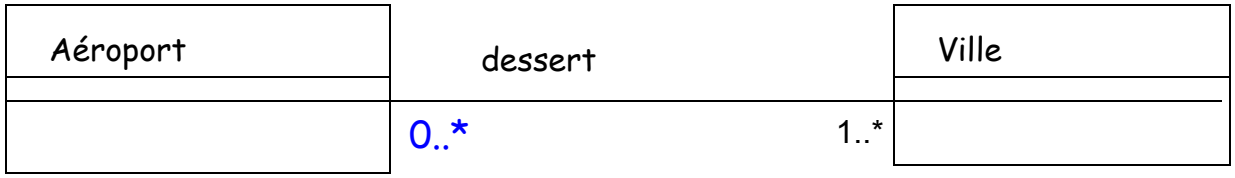



☞ Le rôle de chaque association précise son sens.

- Modélisation des phrases :

10° Chaque aéroport dessert une ou plusieurs villes

- On ne peut pas savoir la multiplicité de ‘’*Aéroport*’’



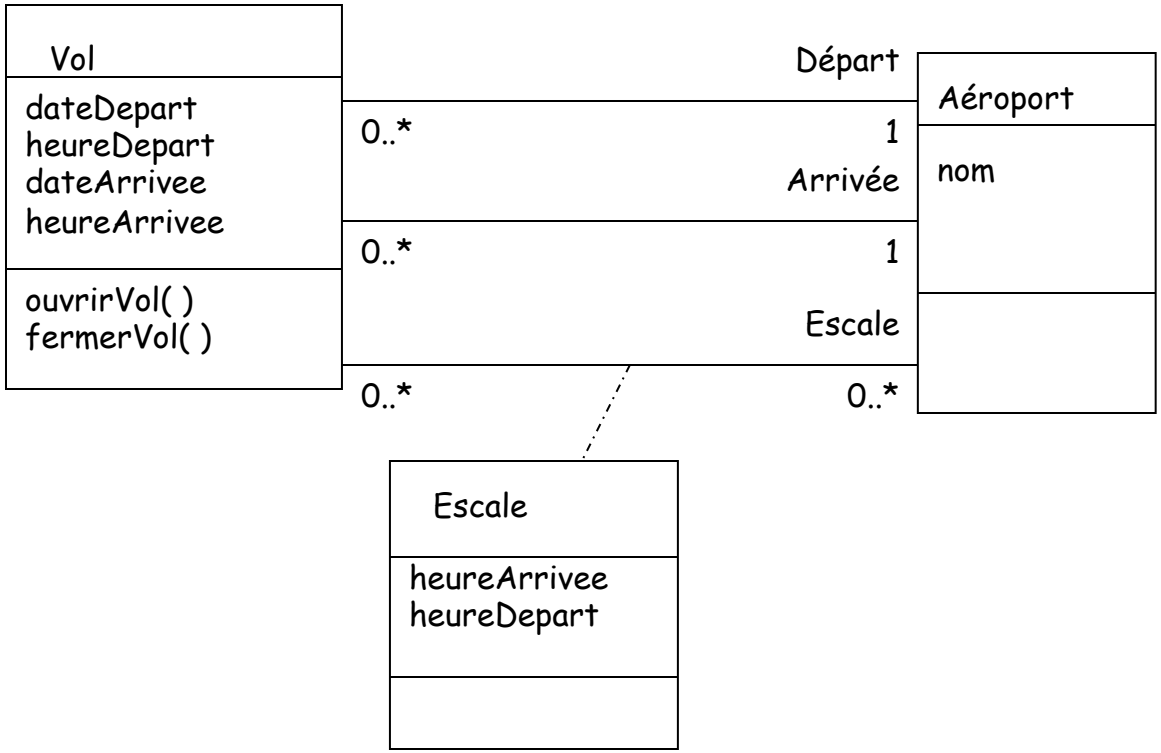

- Si on considère que desservir une ville signifie l’aéroport le plus proche, il n’ en y a qu’un :
  la multiplicité est de 1
- Si on considère que desservir une ville signifie les aéroports dans un rayon de 35 km :
  la multiplicité est de 0..*

- Modélisation des phrases :

8° Un vol peut comporter des escales dans des aéroports

9° Une escale a une heure d'arrivée et une heure de départ.

- *''Escale''* a peu d'informations propres. Elle n'est qu'une partie de *''Vol''* .
- On peut la représenter comme une spécialisation de *''Aéroport''* . Mais elle n'est pas totalement un aéroport.
- La meilleure solution serait de la modéliser comme une classe-association entre et *'Vols''* et *''Aéroport''*.

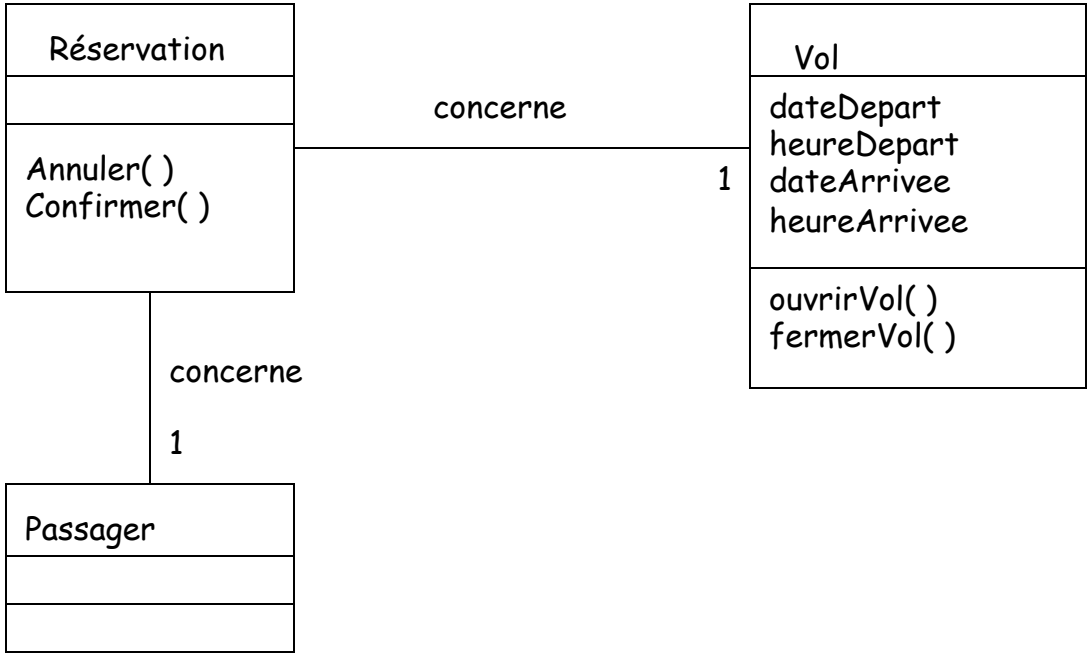


- Modélisation des phrases :

4° Une réservation concerne un seul vol, et un seul passager.

5° Une réservation peut être annulée ou confirmée.

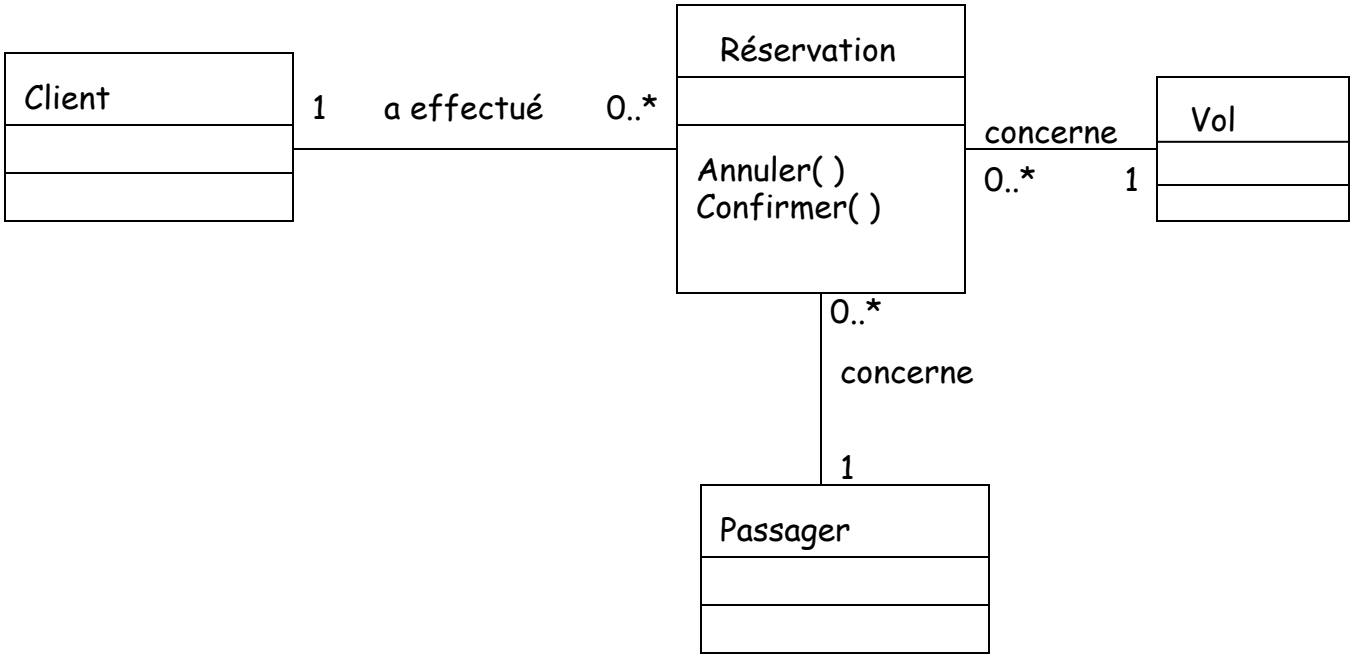
- La réservation et le passager sont 2 concepts métier : 2 classes d'objets
- Un réservation concerne un seul vol et un seul passager: donc 2 associations entre '' *Vol*'' et '' *Réservation*'' et entre '' *Réservation*'' et '' *Passager*''.
- La 5° phrase se traduit par l'ajout de 2 opérations *annuler( )* et *confirmer( )* dans '' *Reservation*''.



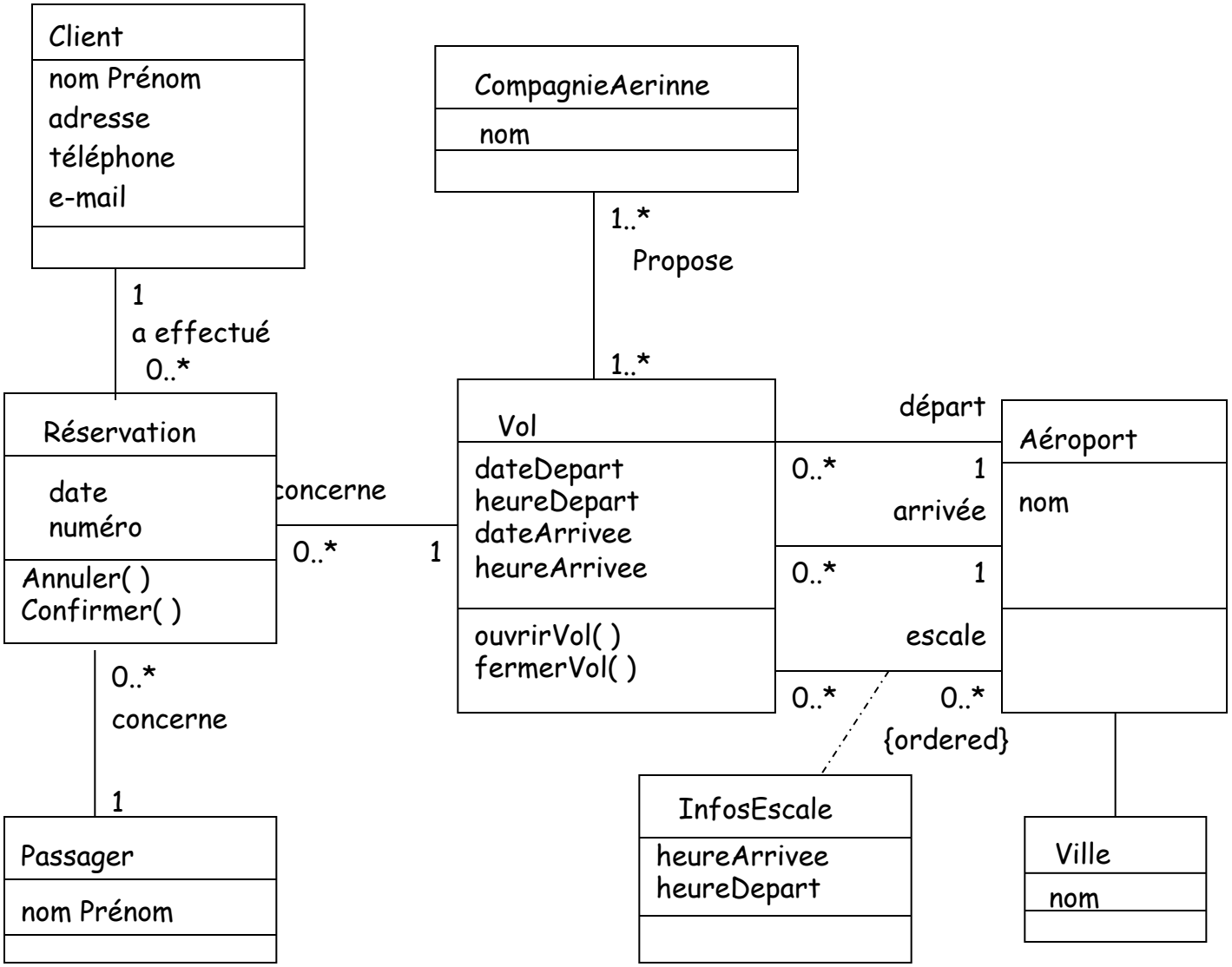
- Modélisation des phrases :

3° Un client peut réserver un ou plusieurs vols, pour des passagers différents.

- Il faut discerner un client d' un passager

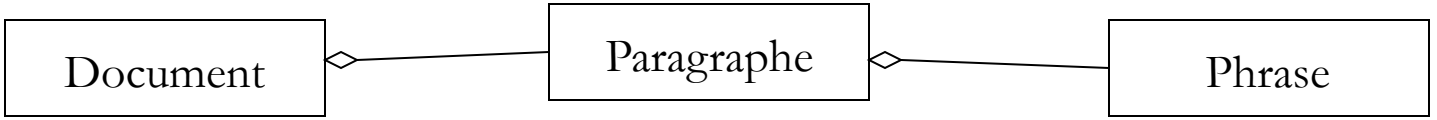


➢ Le diagramme des classe complet est :



# Agrégation

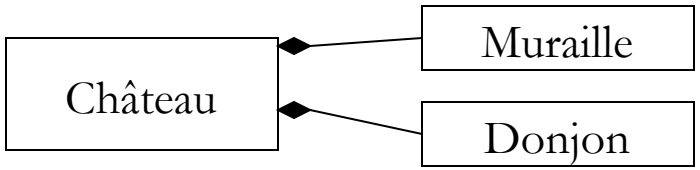
- L'*agrégation* est une relation « composé-composant » ou « partie de » dans laquelle les objets représentant les composants d'une chose sont associés à un objet représentant l'assemblage (ou l'agrégation) entier.
- L'agrégation est une forme spéciale d'association. Le symbole représentant l'agrégation est le losange. Il est situé à côté de la classe composée.

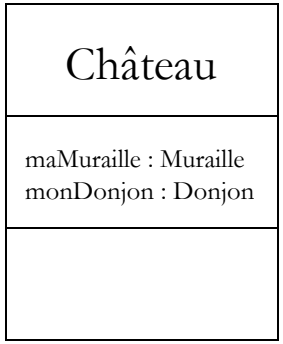


- Agrégation par référence
  - Le composite fait référence à ses composants
  - La création ou destruction du composite est indépendante de la création ou destruction de ses composants
  - Un objet peut faire partie de plusieurs composites à la fois
- Exemple
  - Une pièce est composée de murs
  - Un mur peut être commun à plusieurs pièces

# Composition

- Relation partie-de plus stricte : pas de partage
- Le composé encapsule ses composants
- Durée de vie du composant est égale à celle du composé (le composé n'a pas d'existence propre).



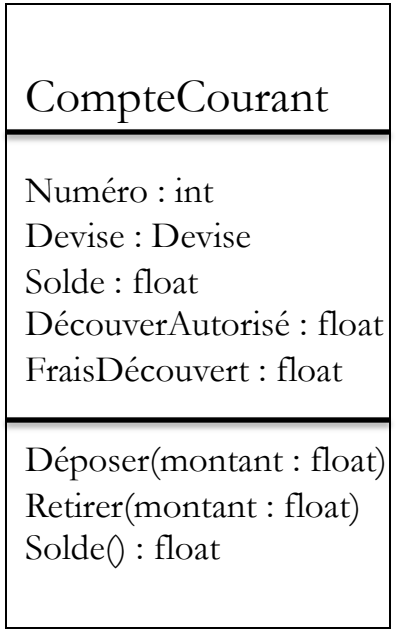



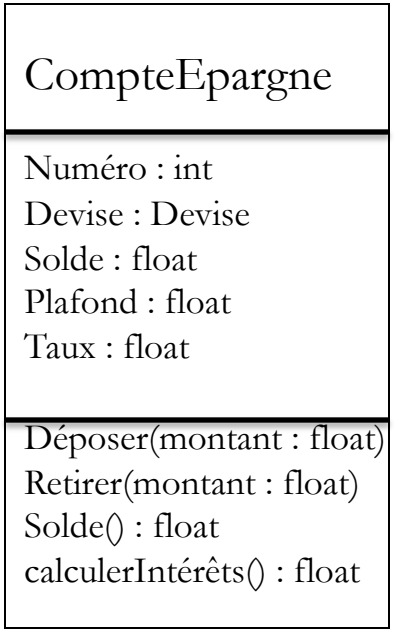
# Composition

- Agrégation par valeur
  - Le composite contient ses composants
  - La création ou destruction du composite entraine la création ou destruction de ses composants
  - Un objet ne fait partie que d'un composite à la fois
- Exemple
  - Un mur est composé de briques
  - Une brique n'appartient qu'à un mur

# Spécialisation/Généralisation

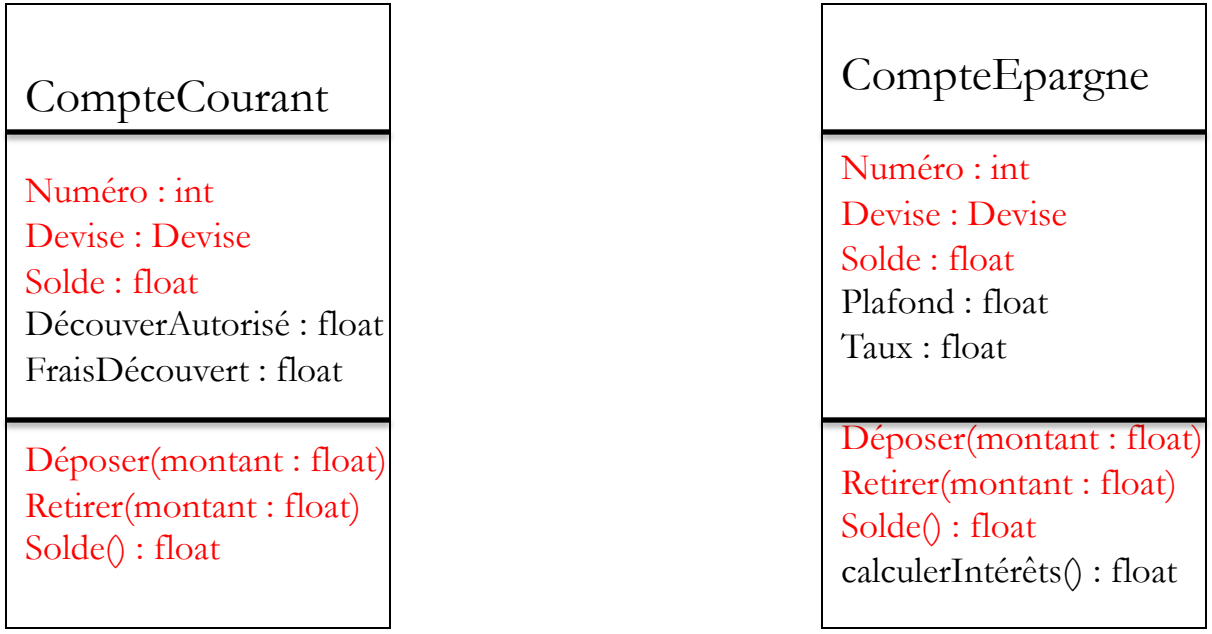
- Principe : Regrouper les classes partageant des attributs et des opérations et les organiser en arborescence
  - Spécialisation : raffinement d'une classe en une sous-classe
  - Généralisation : abstraction d'un ensemble de classes en super-classe



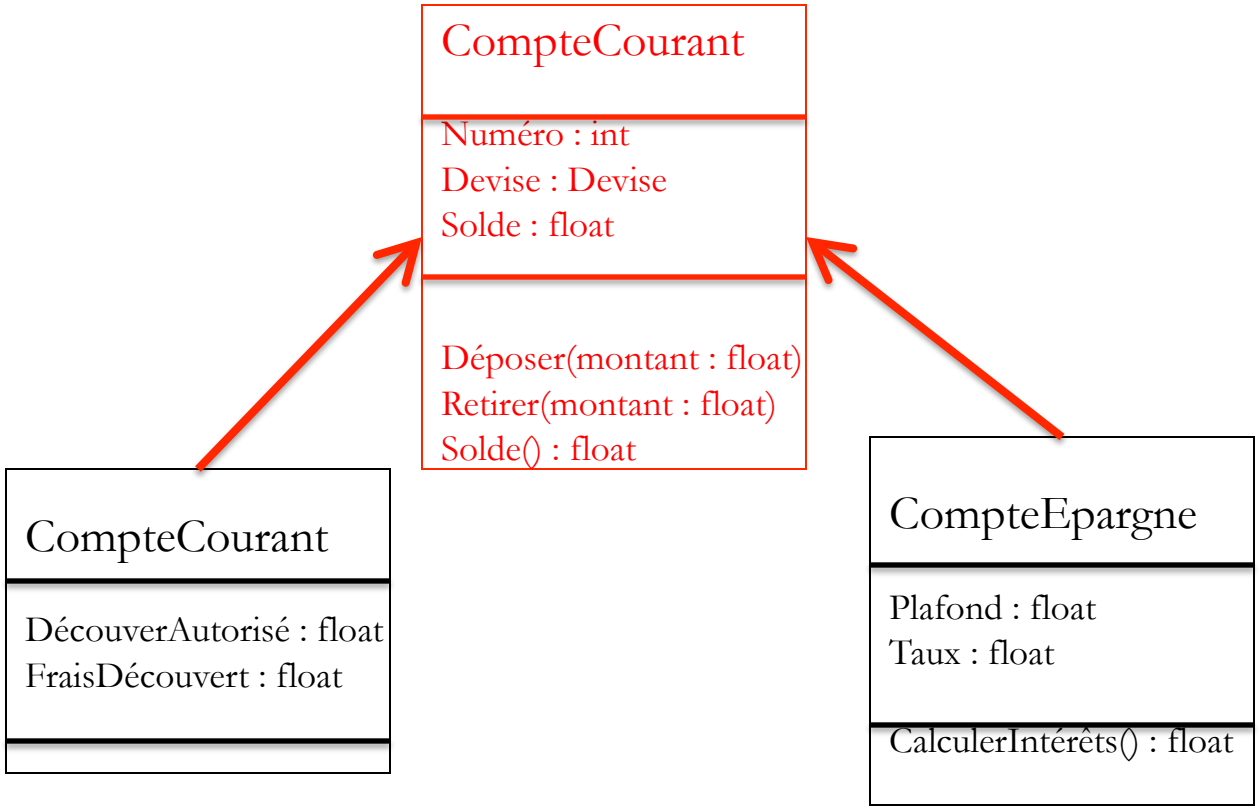
# Spécialisation/Généralisation

- Principe : Regrouper les classes partageant des attributs et des opérations et les organiser en arborescence
  - Spécialisation : raffinement d'une classe en une sous-classe
  - Généralisation : abstraction d'un ensemble de classes en super-classe



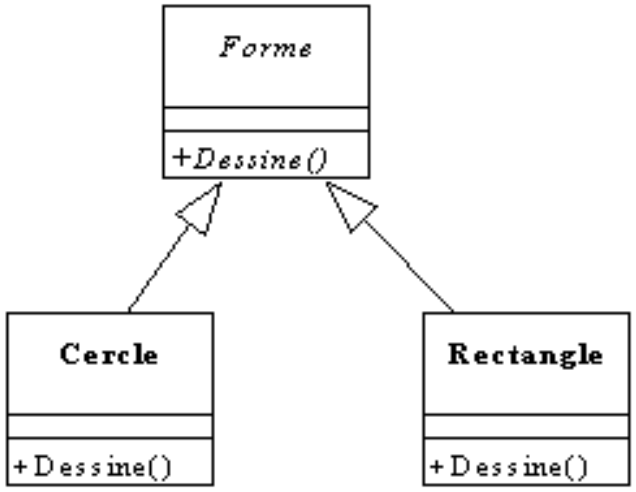
# Spécialisation/Généralisation

- Principe : Regrouper les classes partageant des attributs et des opérations et les organiser en arborescence
  - Spécialisation : raffinement d'une classe en une sous-classe
  - Généralisation : abstraction d'un ensemble de classes en super-classe



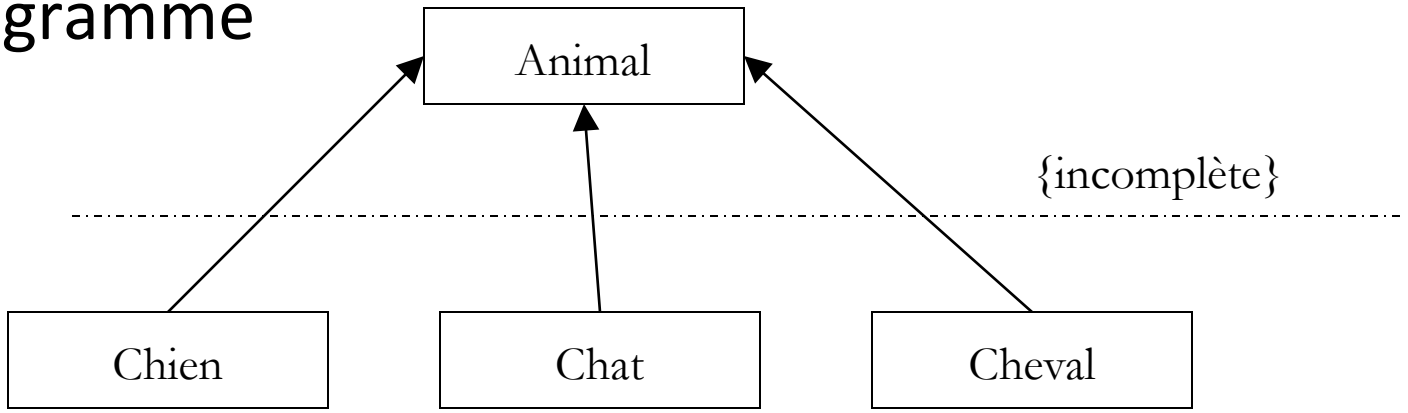
# Spécialisation/Généralisation

- Classe abstraite : classe sans instance car certaines opérations non définies
  - Opération non définie en italique
  - Nom de la classe en italique (ou stéréotype « abstract »)
- Exemple : On ne peut pas dessiner une forme sans savoir de quelle forme il s'agit.



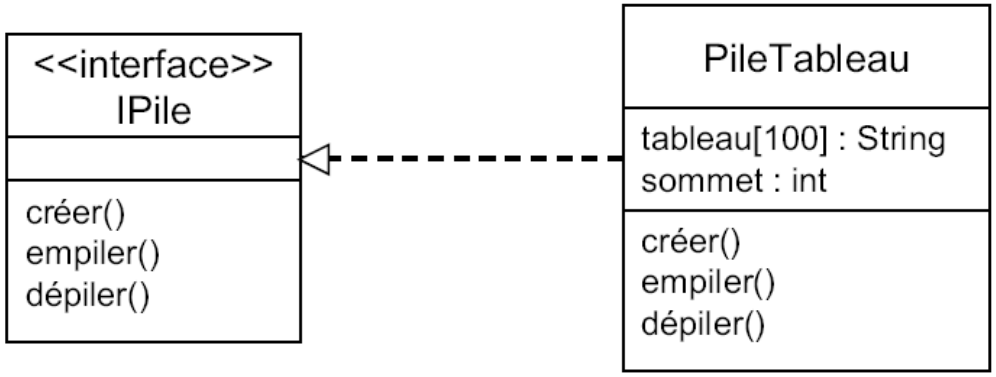
# Spécialisation/Généralisation (2)

- Généralisation complète ou incomplète
  - Une généralisation peut être précisée *complète* ou *incomplète* grâce à un commentaire associé au diagramme



# Réalisation

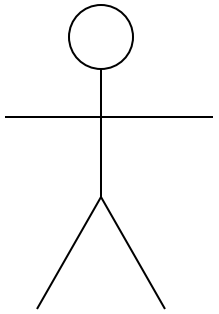
- Une réalisation est une association entre deux classificateurs dans laquelle un classificateur s'engage à exécuter le contrat défini par l'autre classificateur



# Cas d’utilisation
# (Use Case)

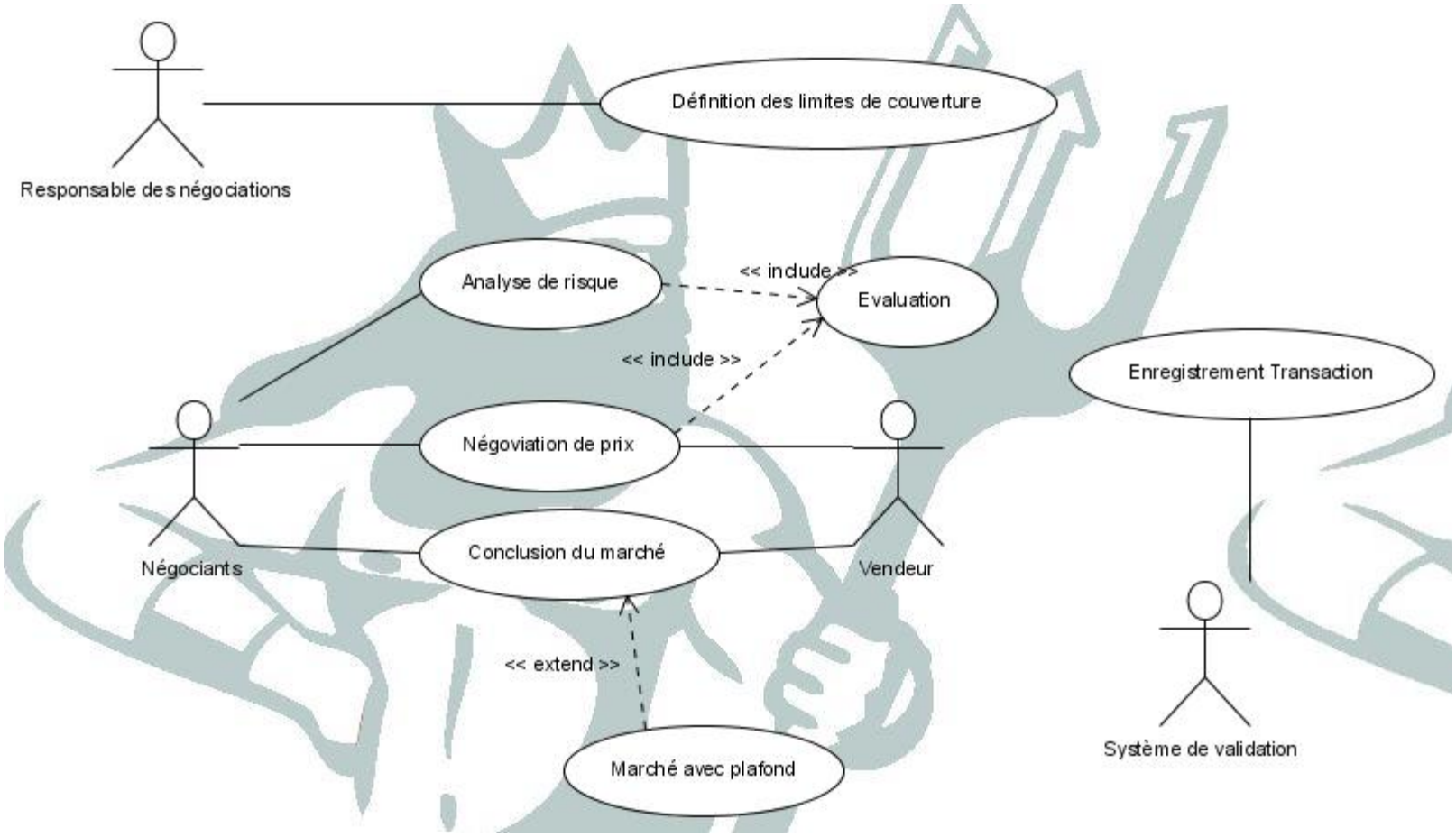
- Les cas d ' utilisation permettent de définir un système en fonction des besoins exprimés par les utilisateurs.
  - Recenser et clarifier les besoins des clients
    - Définition des acteurs (Rôle).
    - Description de leurs actions vis à vis du système
    - Délimitation de la frontière du système
    - Définition des actions/réactions du système

- Un **acteur** (*actor*) est un ensemble cohérent de rôles joués par des entités externes (utilisateur, dispositif matériel ou autre système) qui interagissent avec le système.
  - Plusieurs utilisateurs peuvent tenir le même rôle.
    - Pour une salle des marché, il y a plusieurs personnes qui joue le rôle de négociant
  - Un utilisateur peut avoir plusieurs rôles.
    - Un négociant peut acheter pour son propre compte.
- Représentation graphique :

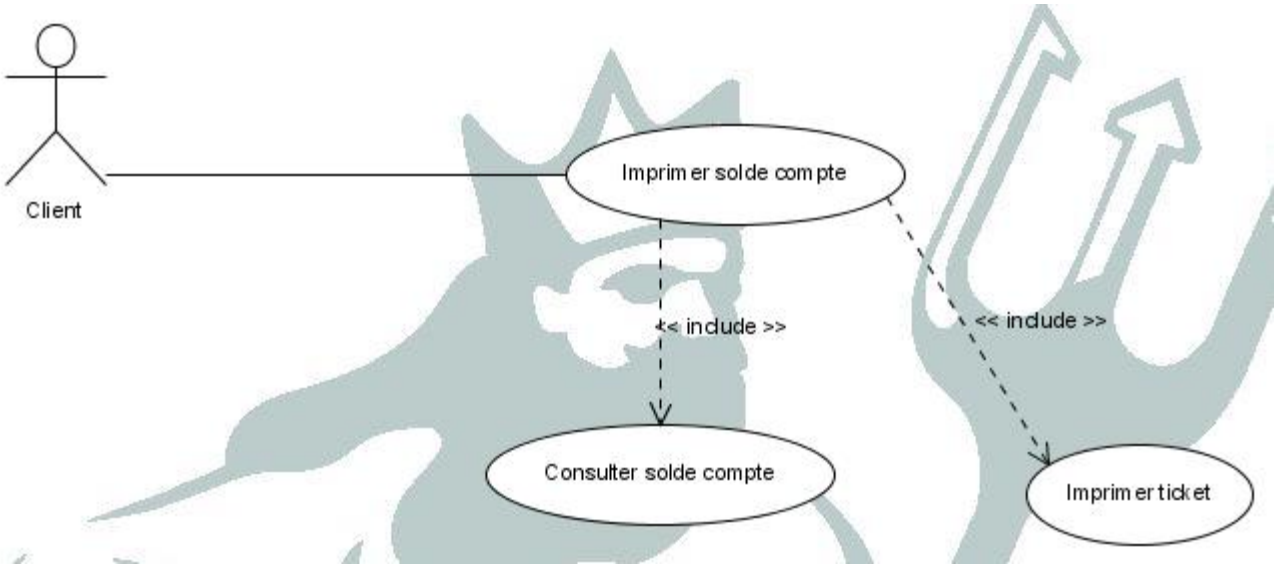


- L ’acteur et le système:
  - L'acteur consulte ou modifie l'état du système.
    - Quelles informations  communiquées au système.
  - Le système répond à une action de l ’acteur.
    - Quelles sont les actions de l ’acteur
    - Quelles informations sont communiquées par le système.

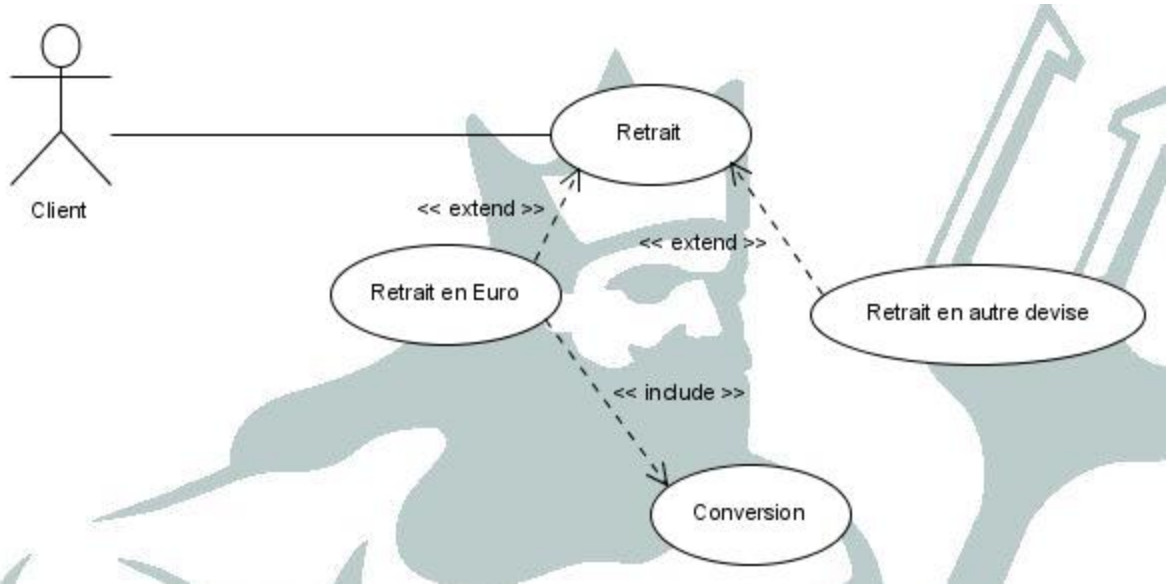
# Cas d'utilisation : exemple



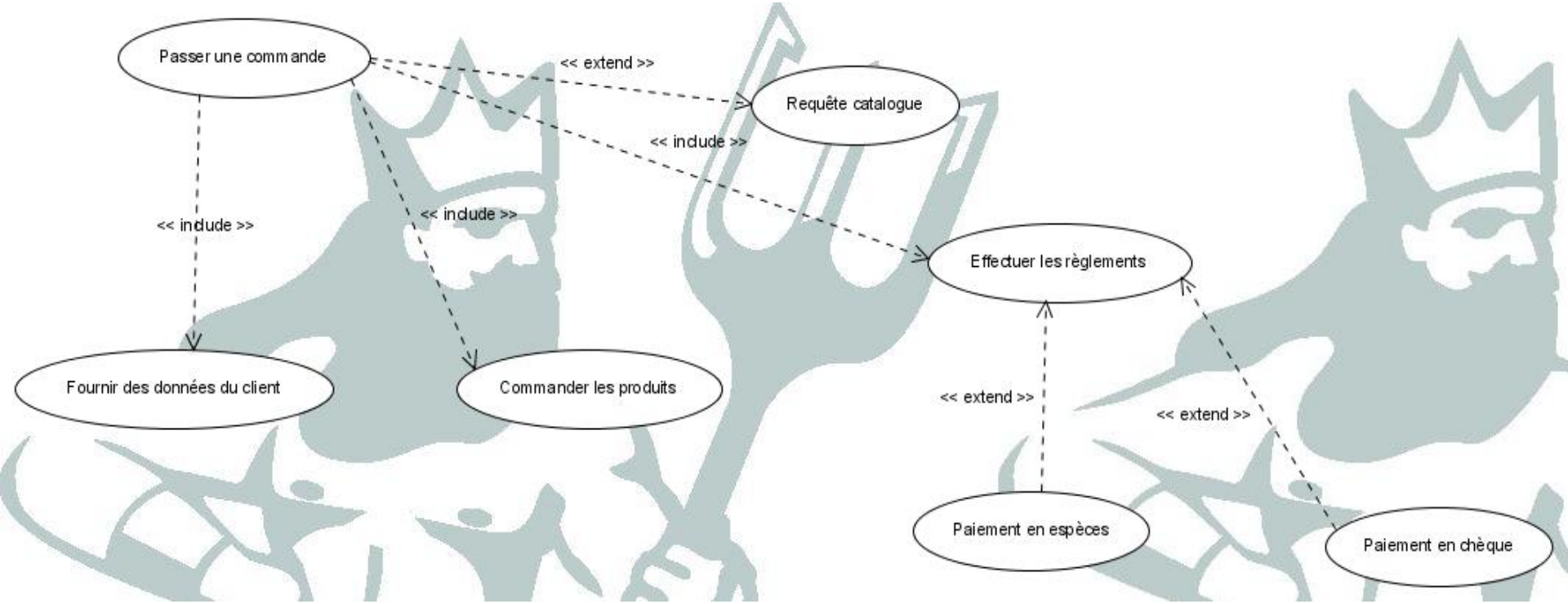
- Relation d ’utilisation: Un cas d’utilisation contient le comportement d’un autre (sous fonction).



- Relation d ’ extension: Un cas d ’ utilisation précise un autre cas.



# Cas d'utilisation



# Cas d'utilisation

- Le nom du cas d'utilisation
- L'acteur primaire du cas d'utilisation
- Le système concerné par le cas d'utilisation
  - Le niveau du cas d'utilisation
  - Objectif utilisateur
  - Sous-fonction (pour les cas d'utilisation étendant un autre ou destinés à être inclus)
- Les opérations
- Les points d'extension (afin de programmer les conditions)

# Cas d'utilisation

| | | |
|---|---|---|
| Cas d'utilisation | | Retrait d'espèces |
| Acteur primaire | | Agent du guichet de la banque |
| Système | | Système informatique de la banque |
| Niveau | | Objectif utilisateur |
| Opérations | | |
| | 1 | Saisir le numéro de compte du client |
| | 2 | Vérifier le solde |
| | 3 | Saisir le montant du retrait |
| | 4 | Attendre l'argent |
| | 5 | Terminer la transaction |
| Extensions | | |
| | 2.A | Le solde est insuffisant et le client a droit à un découvert |
| | 2.A.1 | Continuer |