FULLSTACK

—

Diagramme de classe d'UML

Xavier Crégut
<Prénom.Nom@enseeiht.fr>

ENSEEIHT
Sciences du Numérique

Qu'est-ce qu'UML ?

- Un language de modélisation visuel ;
- Un langage pour spécifier, visualiser, construire et documenter les produits d'un développement logiciel ;
- Une notation et une sémantique ;
- Des diagrammes pour modéliser les aspects statiques, dynamiques et organisationnels ;
- Une unification des connaissances acquises dans le domaine de l'orienté-objet ;
- Compatible avec toutes les méthodes / procédés de développement ;
- Doit être outillé (outils support) ;
- UML n'est **pas** un langage de programmation.

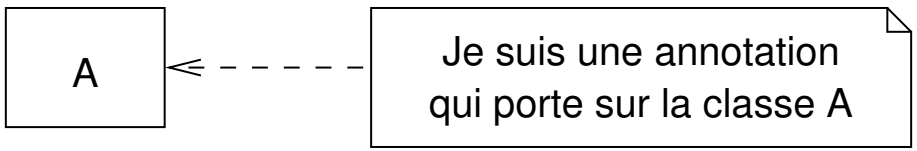
Les constituants d'UML

- **Vue statique du système (architecture du système) : 6 diagrammes structurels**
  - Diagramme de classe
  - Diagramme d'objet
  - Diagramme de paquetage (UML 2)
  - Diagramme de structure composite (UML 2)
  - Diagramme de composant (Vue organisationnelle)
  - Diagramme de déploiement (Vue de déploiement)
- **Vue dynamique (comportement) : 7 diagrammes comportementaux**
  - Diagramme de vue d'ensemble des interactions (UML 2)
  - Diagramme de séquence (fortement changé en UML 2)
  - Diagramme de communication (anciennement collaboration)
  - Diagramme de temps (UML 2)
  - Diagramme de machine à états
  - Diagramme d'activité
- **Autre diagramme :**
  - Diagramme de cas d'utilisation (vue fonctionnelle, interaction utilisateurs/système)
- **Mécanismes d'extension**

Quelques conventions

La notation UML définit pour tous les diagrammes :

- des *commentaires* (ou *annotations*) qui permettent d'ajouter des informations sur un modèle.



  L'annotation est reliée à l'élément décrit par un trait interrompu.

- Les *stéréotypes* sont des mots entre chevrons qui permettent de compléter le vocabulaire UML.

  `«exception», «interface», «instanceOf»...`

- Les *contraintes* : les accolades sont utilisées pour ajouter une contrainte sur un modèle (modification de la sémantique du modèle).

  `{abstract}, {ordered}, {readOnly},` $\{ x = \rho \times cos(\theta) \}$

Unified Modeling Language (UML)

Principales caractéristiques d'UML [1, 2]

- notation graphique pour décrire les éléments d'un développement (objet)
- normalisée par l'OMG [3] (Object Management Group)
- version 1.0 en janvier 1997. Version actuelle : 2.5.1 (2017).
- *s'abstraire du langage de programmation et des détails d'implantation*

Utilisations possibles d'UML [2]

- **esquisse (*sketch*) :** communiquer avec les autres sur certains aspects
  - simplification du modèle : seuls les aspects importants sont présentés
  - échanger des idées, évaluer des alternatives (avant de coder), expliquer (après)
  - n'a pas pour but d'être complet
- **plan (*blueprint*) :** le modèle sert de base pour le programmeur
  - le modèle a pour but d'être complet
  - les choix de conception sont explicités
  - seuls les détails manquent (codage)
- **langage de programmation :** *le modèle est le code !*
  - pousser à l'extrême l'approche « plan »
    $\Longrightarrow$ mettre tous les détails dans le modèle
  - engendrer automatiquement le programme à partir du modèle.

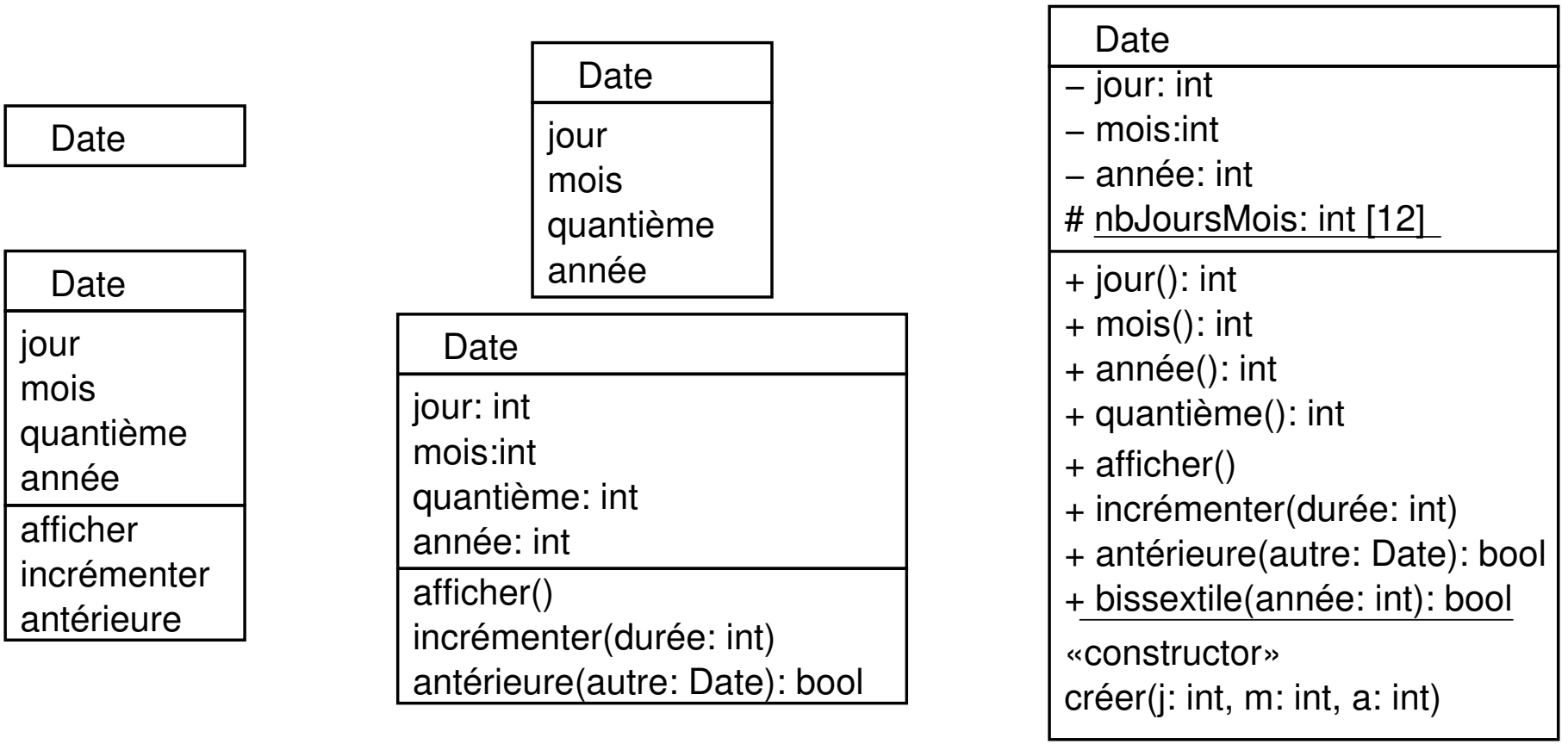
Représentation d'une classe

Date
− jour : int − mois : int − année : int
+ getJour() : int + getMois() : int + getAnnée() : int + afficher() + ajouter(nbJours : int) + distance(autre : Date) : int − incrémenter() + estBissextile(année : int) : boolean
+ Date(jour : int, mois : int, année : int)

- Ordre des rubriques :
  - nom de la classe
  - attributs
  - opérations
  - constructeurs
- Droits d'accès :
  - public : +
  - privé : −
  - protected : #
  - paquetage : (rien)
- membre de classe : souligné

- Les constructeurs peuvent être mis avec les opérations, précédés du stéréotype «create»
- Pour le premier diagramme (analyse), il est conseillé de faire requêtes/commandes plutôt que attributs/opérations.

Différents niveaux de description d'une classe



Description des attributs

La forme générale de déclaration d'un attribut est :

```
[visibilité] nom [: type] [multiplicité] [= valeur-initiale] [{propriétés}]
```

```
origine                  -- seulement le nom
+ origine                -- visibilité et nom
origine: Point = (0, 0)  -- nom, type et initialisation
nom : String [0..1]      -- nom, type et multiplicité
id: Integer {frozen}     -- nom, type et propriété
```

Les visibilités sont : + (**public**), – (**private**), # (**protected**) et ~ (paquetage).
Les propriétés sont :

- `changeable` : pas de restriction sur les modifications (défaut) ;
- `frozen` : pas de modification possible après initialisation (const de C++) ;
- `addOnly` : (si multiplicités > 1) seules de nouvelles valeurs peuvent être ajoutées, les anciennes ne peuvent être ni enlevées, ni modifiées.

Description des opérations

La forme générale de déclaration d'une opération est :

```
[visibilité] nom [(liste-paramètres)] [: type-retour]  [{propriétés}]
```

Les paramètres sont séparés par des virgules et sont de la forme :

```
[direction] nom  : type  [= valeur-par-défaut]
```

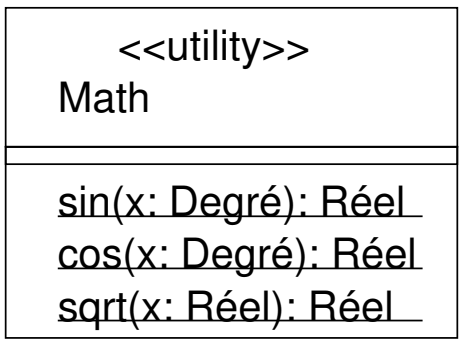
Les *directions* sont **in**, **out** et **inout** (sémantique d'Ada : entrée seule sans modification, sortie seule, entrée et sortie).
Les propriétés sont :

- `leaf` : opération non polymorphe (ne peut pas être redéfinie) ;
- `query` : fonction pure (sans effet de bord) ;
- `sequential`, `guarded`, `concurrent` : propriétés liées au parallélisme.

Classes de différentes natures

Des stéréotypes permettent de préciser la signification d'une classe :

- «`utility`» : une classe qui correspond à une bibliothèque et rassemble essentiellement des méthodes de classe.

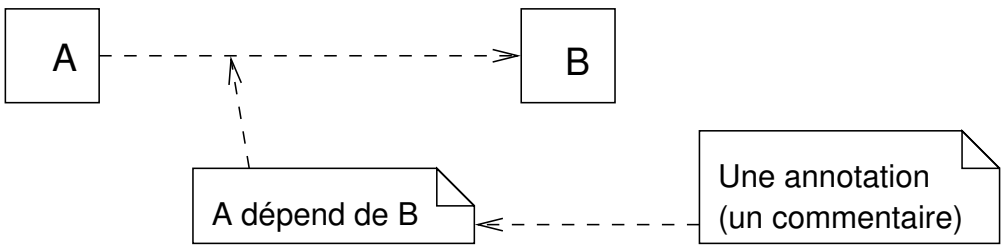


- «`exception`» : une classe qui décrit une exception.
- «`interface`» : il s'agit en fait d'une interface et non d'une classe.
  Remarque : On peut également utiliser «`type`».
- et bien d'autres...

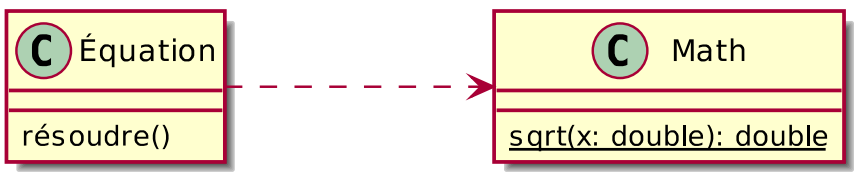
Relation de dépendance

On dit qu'il y a relation de dépendance entre une classe A et une classe B si la classe A fait référence à la classe B dans son texte. On dit que A dépend de B.

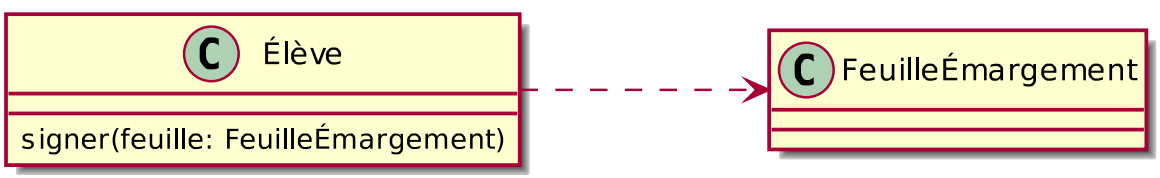
La relation de dépendance est représentée par une flèche en traits interrompus dont l'origine est la classe A et la cible la classe B.



Exemple : La classe Equation dépend de la classe Math puisque sa méthode « résoudre » utilise la méthode « sqrt » de Math.



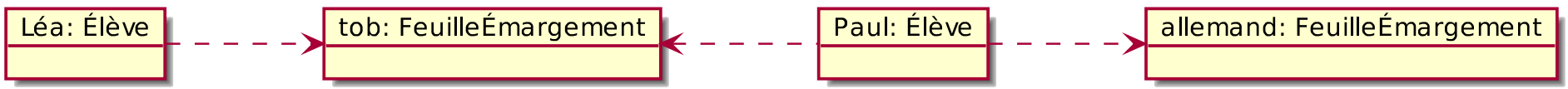
Exemple : Un élève doit signer la feuille d'émargement.



Lien de dépendance

En général, si une classe A dépend d'une classe B, il y a un lien de dépendance entre un objet de la classe A et un objet de la classe B.

Exemple : Léa, Paul... doivent chacun signer la feuille d'émargement de TOB. Plus tard, Paul signe la feuille d'émargement d'Allemand.
Il y a un lien entre l'objet Léa et l'objet feuille TOB, un lien entre l'objet Paul et le même objet feuille TOB, etc. Il y a un lien entre l'objet Paul et la feuille Allemand.



Remarque : Un lien est instance d'une relation (comme un objet est instance d'une classe).

Le lien (par extension la relation de dépendance) est momentané si B apparaît comme variable locale ou paramètre d'une méthode m de A. Le lien n'existe que pendant l'exécution de m.
Les relations de dépendance sont rarement représentées en UML (souvent peu pertinentes).

Exemple : Il n'y a un lien entre Léa et la feuille d'émargement de TOB que le temps de signer la feuille (exécution de la méthode signer).

Relation structurelle

Une relation de dépendance est dite relation structurelle si elle dure dans le temps.
C'est par exemple le cas quand B est le type d'un attribut de la classe A.

Exemple : Une personne travaille dans une entreprise.

- Un objet personne est lié à un objet entreprise
- Ce lien dure dans le temps
- La classe Personne pourrait avoir un attribut dont le type est Entreprise (et inversement).

UML propose trois relations de plus en plus précises :

- association (la plus générale)
- agrégation
- composition (la plus précise)

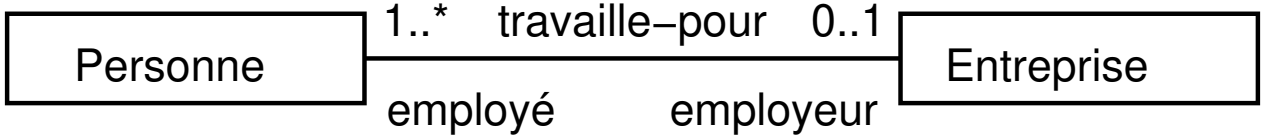
Ces relations permettent de décrire l'architecture de l'application.

Relation d'association

Définition : Une relation d'association est une relation entre deux classes qui traduit un couplage faible : chaque classe pourrait être considérée indépendamment de l'autre.

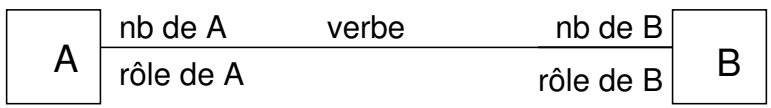
Propriété : Les objets des deux classes sont relativement indépendants et leurs durées de vie ne sont pas liées.

Exemple :



- Classes (et objets) aux extrémités de la relation indépendantes :
  - On pourrait parler de Personne sans parler d'Entreprise
  - On pourrait parler d'Entreprise sans parler de Personne
- Durées de vie des objets non liées :
  - Si une entreprise disparaît, les personnes continuent à exister.
  - Si une personne disparaît, l'entreprise continue à exister.

Forme générale :

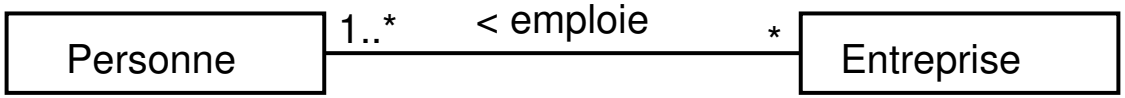


Expliquer une relation : nom et rôles

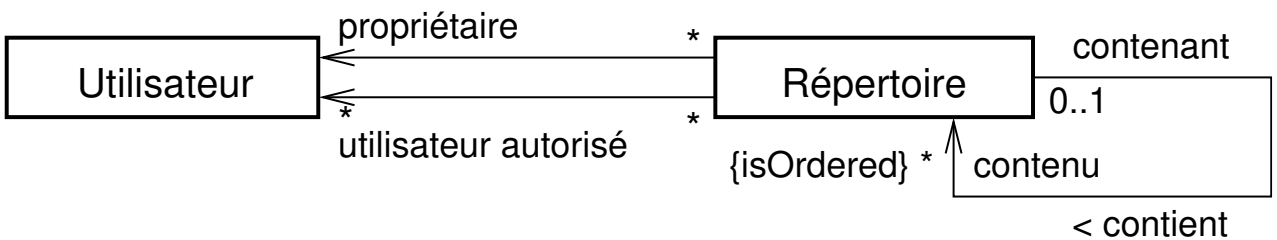
Il est essentiel de savoir ce que représente une relation et donc de préciser sa signification.

On peut nommer la relation :

- C'est un verbe (travaille-pour, emploie, etc.)
- Le nom est placé au milieu de la relation
- La lecture se fait de gauche à droite ou de haut en bas.
  Une direction peut indiquer le sens de lecture quand il n'est pas naturel.



On peut donner le rôle joué par les objets dans la relation (extrémités de la relation)

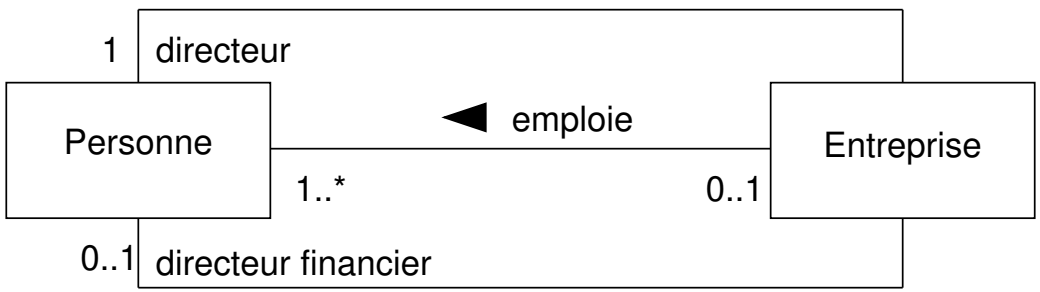


- Les rôles peuvent avoir des droits d'accès (comme les attributs).
- Si le rôle est omis, c'est le nom de la classe qui est utilisé.

Multiplicité (ou cardinalité)

Objectif : indiquer combien d'objets peuvent/doivent prendre part à la relation :

- forme générale : min..max
  - exemple : 1..4
  - le nombre d'objet est compris entre min et max inclus
  - la valeur de max peut être « * » : nombre quelconque (« illimité »)
- un entier (ex : 4) : le nombre exact d'objets (simplification d'écriture de 4..4)
- `0..1` : optionnel
- $*$ ou `0..*` : 0 ou plusieurs (un nombre quelconque y compris 0)
- `1..*` : au moins 1 (un nombre quelconque mais au moins 1)
- une multiplicité omise crée une ambigüité (souvent considérée comme 1..1)



Navigation

Définition : La navigation est le fait de traverser une relation (en fait un lien) :
à partir d'une objet d'une extrémité, on peut atteindre les objets de l'autre extrémité.

Exemple : Depuis une personne, Xavier, on obtient l'entreprise où Xavier travaille (N7).

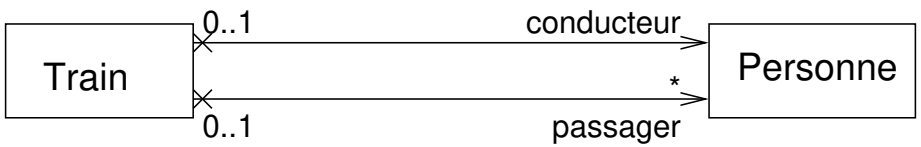
Propriété : La relation d'association est bidirectionnelle : navigable dans les deux sens.

Exemple : Partant de Xavier, on trouve son employeur, N7, et inversement, de N7 on trouve Xavier parmi les employés.

Principe : Rôles (et noms) d'association sont utilisés pour la navigation (termes en bleu)

Sens de navigation : Expliciter dans quel sens une relation peut être traversée.

- Flèche à l'extrémité d'une relation : la relation peut être traversée dans le sens de la flèche
- Croix à une extrémité : interdit le sens de navigation qui mène à la croix

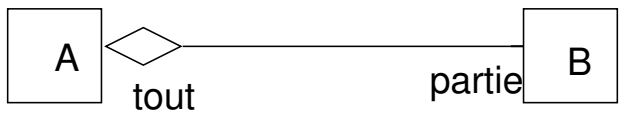


- d'un objet train on peut obtenir le conducteur ou les passagers
- d'une personne on ne peut pas retrouver le train (Train n'apparaît pas dans Personne)

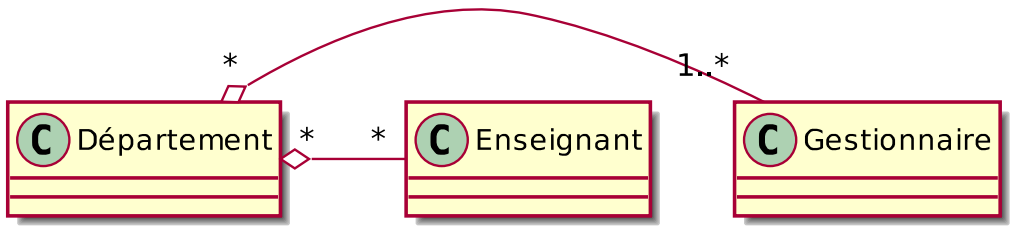
Relation d'agrégation

Définition : La relation d'agrégation est un cas particulier d'association où une classe est prépondérante par rapport à l'autre : ses objets (le tout) agrégent d'autres objets (les parties).

Notation : On utilise un losange vide à l'extrémité de la relation, côté tout.



Exemple : Un département de formation s'appuie sur des gestionnaires et des enseignants.



Propriété : La relation d'agrégation permet le *partage* : le même objet peut appartenir à plusieurs liens d'agrégation.

Exemple : Le même enseignant peut intervenir dans plusieurs départements.

Remarque : Le terme utilisé en UML est *aggregation* ou *shared aggregation*.

Relation d'agrégation (2)

Voici quelques indices d'une relation d'agrégation :

1. un objet correspond au tout, les autres aux parties
   - La promotion 1SN et les étudiants.
   - Toulouse INP et les trois écoles historiques : N7, A7 et NSAT.
2. les opérations du tout se propagent sur les parties
   - L'INP et ses écoles : les statistiques demandées à l'INP sont d'abord demandées par l'INP aux écoles puis consolidées au niveau INP.
   - Les opérations de Segment utilisent les opérations de Point.
3. il est difficile de parler du tout sans parler de ses parties
   - Segment et ses Points extrémités
   - Itinéraire et les Villes étapes

Attention : La relation d'agrégation est subjective. Elle apporte une nuance au modèle, explicite un point de vue.

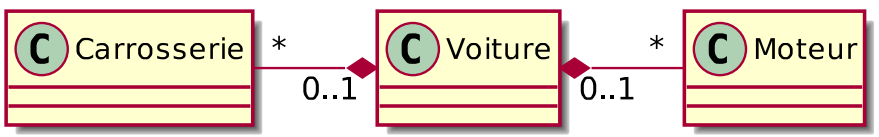
Exemple : Entre Groupe et Étudiant qui est le tout ?
Le groupe qui regroupe plusieurs étudiants ?
L'étudiant qui est inscrit à plusieurs groupes (1SN, langue, sport, BDE...) ?
La réponse dépend du point de vue (gestionnaire du groupe ou étudiant) !

Relation de composition

Définition : La relation de composition est un cas particulier de la relation d'agrégation. Elle dénote un couplage plus fort qui lie la durée de vie des objets : quand le tout est détruit, les parties sont aussi détruites.

Notation : On utilise un losange plein à l'extrémité de la relation, côté tout.

Exemple : Une voiture « est composée » d'une carrosserie, un moteur, etc.



Vocabulaire : On parle d'aggrégation forte.
L'objet du côté du losange (A, Voiture) est appelé l'agrégat ou composite.

Propriété : La composition interdit le partage. Un même objet ne peut appartenir qu'à un seul lien de composition (peu importe la relation de composition dont le lien est instance).

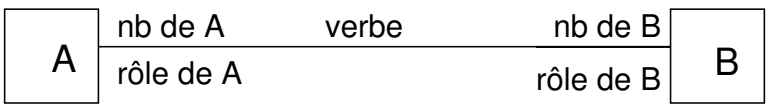
Exemple : Un même moteur ne peut pas être lié à une voiture et un camion !

Remarque : Un épaviste pourrait récupérer des éléments de la voiture avant sa destruction !
Remarque : La relation de composition est particulièrement importante pour les langages sans ramasse-miette car elle permet de savoir qui doit libérer la mémoire occupée par les objets.
En réalité, elle est importante dans tout système.

Relation entre classes : synthèse

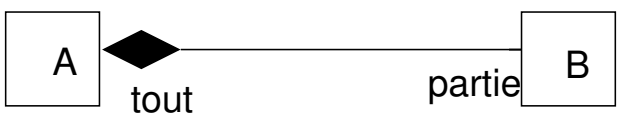
Une application est constituée de plusieurs classes dont le couplage est caractérisé par des relations dites de délégation :

- **association :** couplage faible correspondant à une relation symétrique entre objets relativement indépendants (durées de vie non liées) ;



- **agrégation :** association non symétrique avec couplage plus fort
  - relation tout et parties
  - relation de subordination : propagation des opérations de A vers B
  - on ne peut pas parler de A sans parler de B

- **composition :** agrégation forte (par valeur) :
  - La durée de vie des objets « partie » est liée à celle du « tout »
  - Pas de partage possible.



Relations entre classes : exercices

Exercice 1 Dessiner un diagramme de classes faisant apparaître un site web, des pages HTML et un « webmaster ».

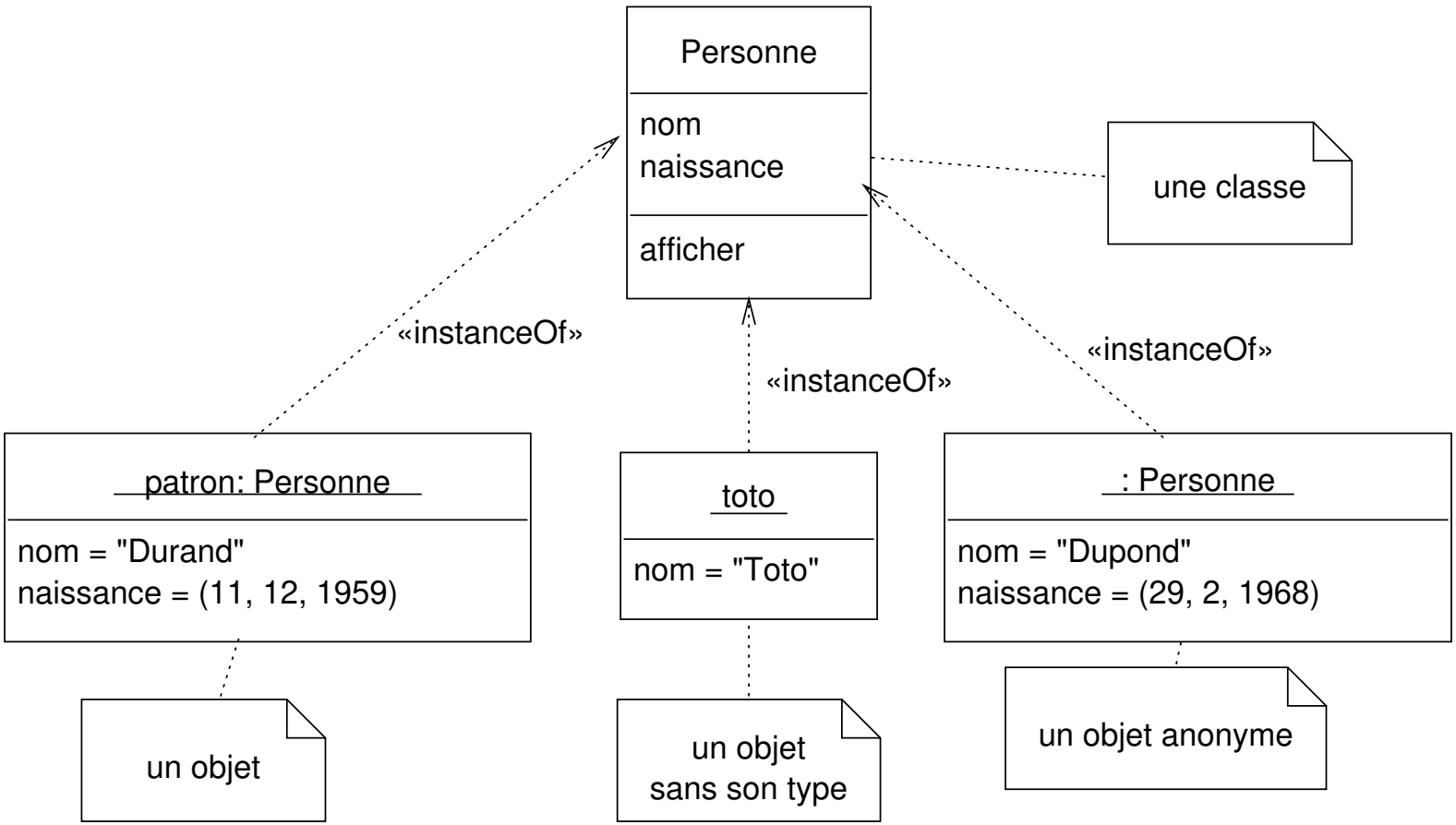
Exercice 2 Indiquer quelles relations il serait possible de définir entre :

- Livre et Page
- Une UE et les étudiants inscrits à l'UE
- Itinéraire et Gare
- Segment et Point

Exercice 3 Que penser d'attributs de types primitifs ?
Et plus généralement de la rubrique « attribut » d'UML.

Règle : Dans une classe, je jamais mettre un attribut dont le type est une classe !
Il faut le représenter par une relation entre les deux classes, le nom de l'attribut étant utilisé comme rôle.
Exceptions : On peut utiliser des attributs de type classe pour les classes considérées comme des types de base (String, Date, etc.) et quand il s'agit d'une relation de composition.

Représentation des objets



Un objet est représenté sous la forme « nom : Type » souligné. Les deux sont optionnels.
On peut aussi faire apparaître la valeur des attributs de l'objet (*slot* en UML)
Remarque : le souligné se perd...

Diagramme d'objet

Diagramme d'objet : Une configuration possible respectant un diagramme de classe.

Exemple de diagramme de classe :

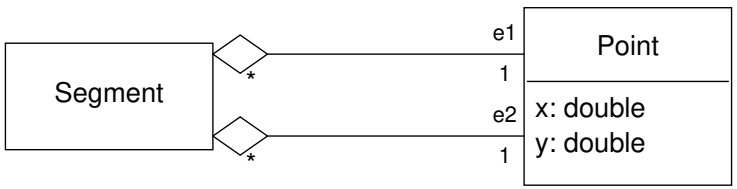
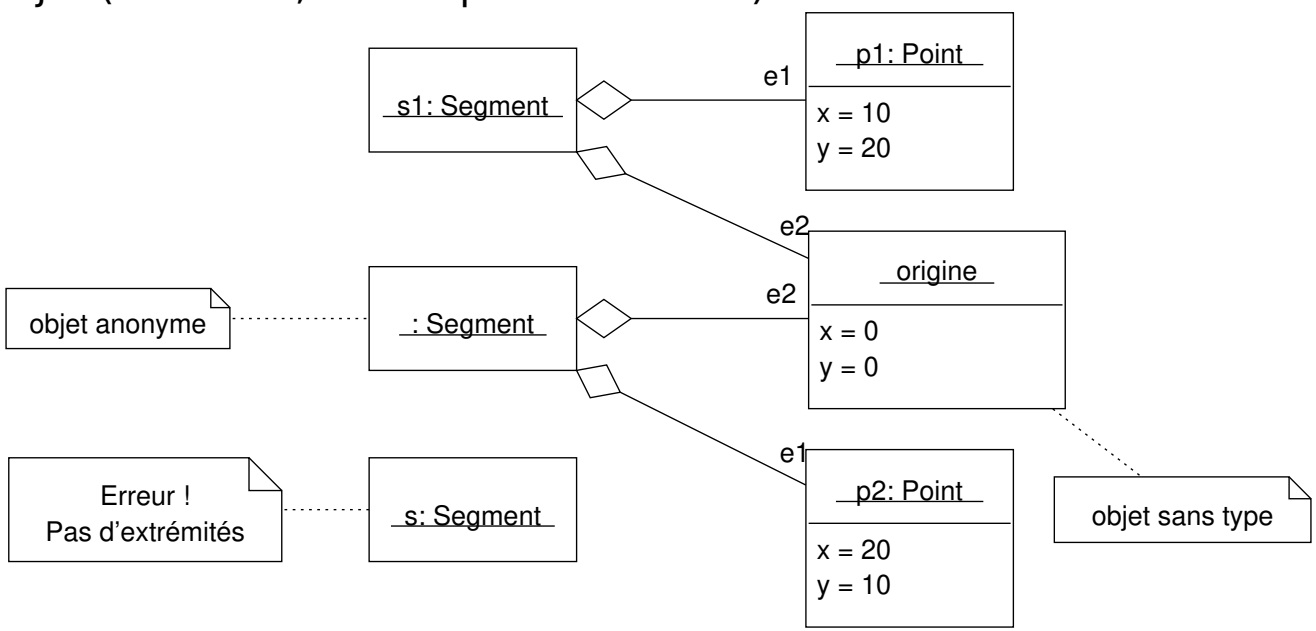


Diagramme d'objet (attention, s n'est pas conforme !) :



Conformité

Un diagramme d'objet est conforme à un diagramme de classe ssi :

- tout objet du diagramme d'objet est instance d'une classe du diagramme de classe
  - chaque attribut de l'objet correspond à un attribut de la classe
  - la valeur des attributs de l'objet respecte le type de l'attribut dans la classe
- tout lien du diagramme d'objet est instance d'une relation du diagramme de classe
- le nombre de liens respecte les multiplicités exprimées sur le diagramme de classe
- la relation de composition est respectée :
  - un objet ne peut apparaître extrémité que d'un seul lien de composition !

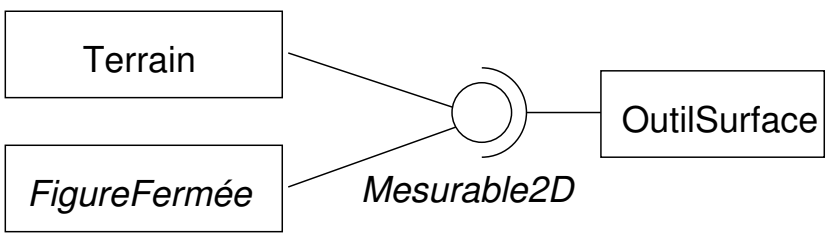
Quelques remarques sur le diagramme d'objet :

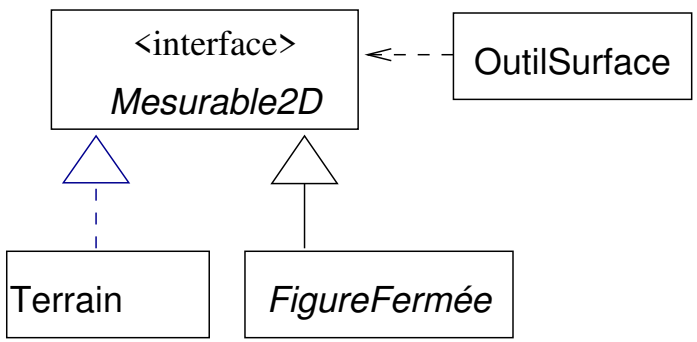
- Généralement, on ne fait pas apparaître les losanges (agrégation, composition)
- Les rôles sont souvent omis (si pas d'ambiguïté)

Relation de généralisation/spécialisation

Quelques remarques :

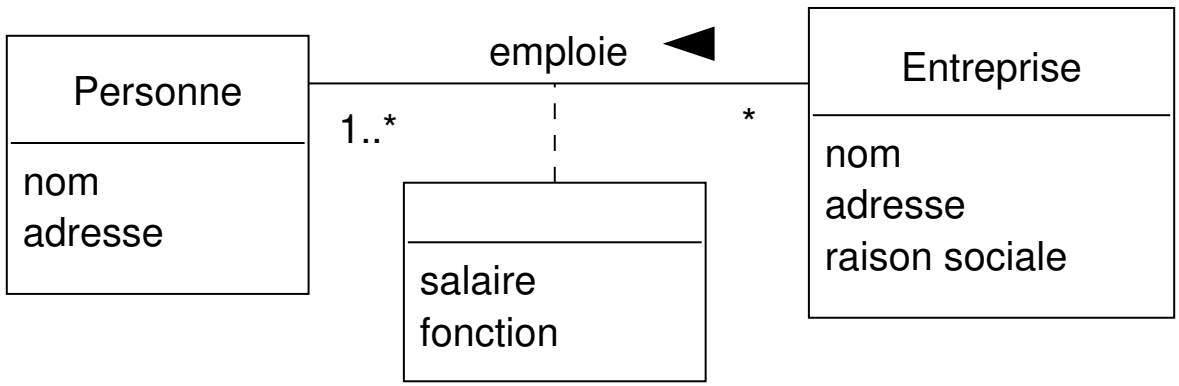
- La relation de généralisation/spécialisation est une relation de sous-typage.
- Une classe abstraite, une méthode abstraite ou une interface sont notées en italique. Il est cependant possible d'utiliser la contrainte `{abstract}`.
- `{polymorphic}` indique qu'une méthode est polymorphe (défaut).
- `{leaf}` interdit la redéfinition ou la dérivation (**`final`** en Java).
- les interfaces peuvent être représentées par un simple cercle avec leur nom dessous (notation dite balle-et-support ou sucette).





UML : Attribut et classe d'association

Définition : Un attribut d'association est un attribut qui caractérise la relation et pas seulement une de ses classes extrémités.



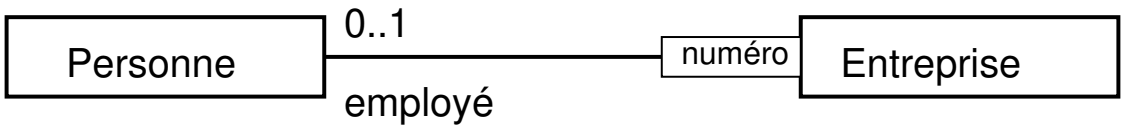
Remarque : Dans le cas d'une multiplicité 1, il est possible *mais non souhaitable* d'attacher l'attribut d'association à la classe (salaire sur Personne si elle ne peut travailler que dans une seule entreprise).

Remarque : Les attributs d'association peuvent être promus au rang de classe. Ici, une classe Poste, avec des méthodes telles que travailler, etc.

Traduction en code : Plusieurs possibilités : dans une classe, dans l'autre, dans les deux, en réifiant la relation, etc.

UML : Qualificatif

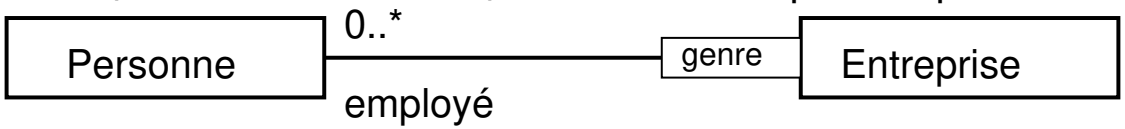
Un *qualificatif* est un attribut spécial placé sur une extrémité d'une relation.



Le qualificatif « numéro » est une clé de la classe Entreprise qui permet d'atteindre un jeu d'objet Personne (ici au plus un).

Intérêt : La qualification améliore la précision sémantique de la relation :

- elle *réduit la multiplicité* effective ($*$ est transformée en `0..1`, `numéro` est ici équivalent à une clé au sens base de données)
  Le qualificatif *genre* (femme ou homme) réduit la multiplicité qui reste $*$

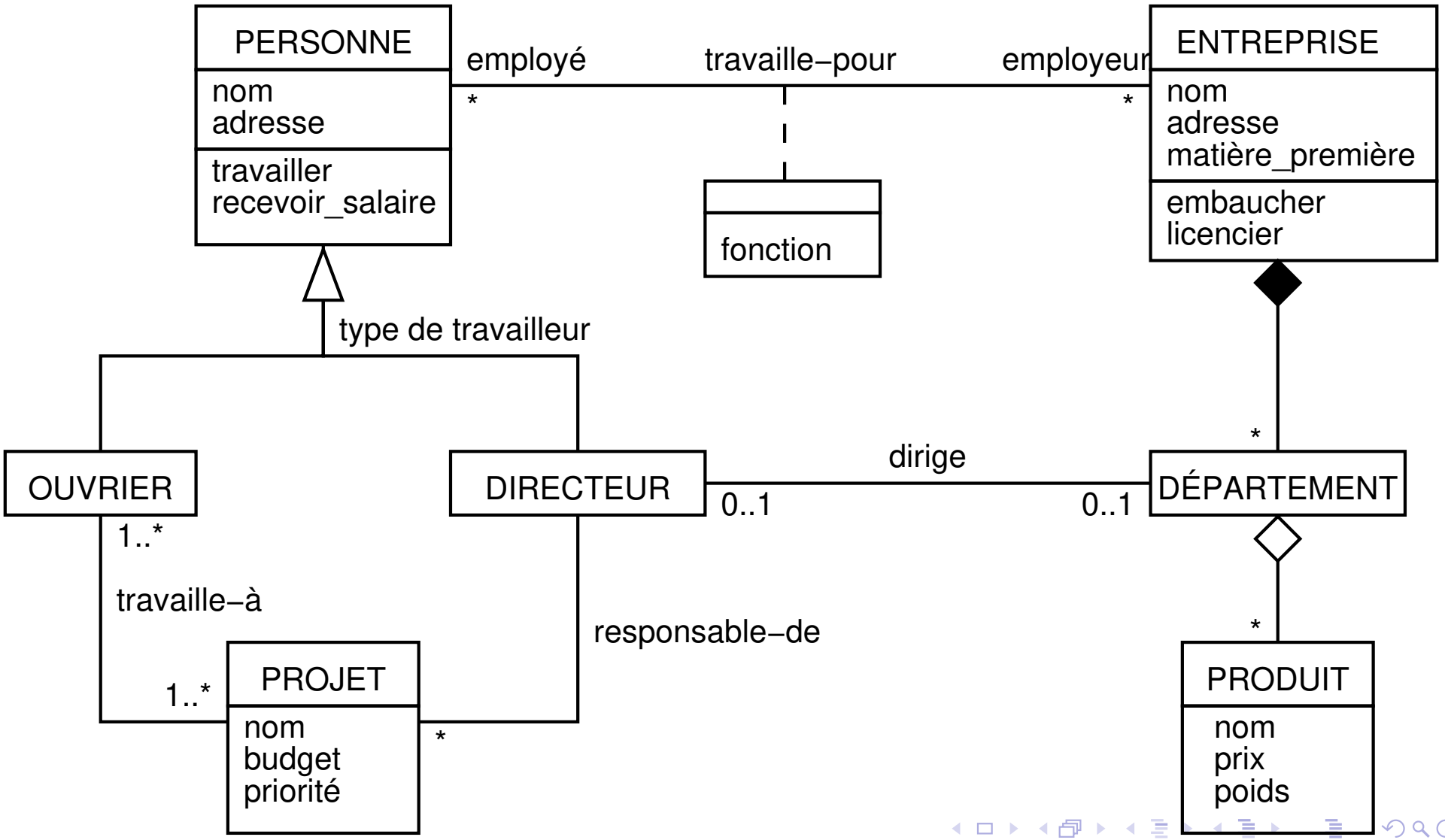


- elle *améliore la navigation* dans le réseau des objets (pour désigner une personne, il suffit d'avoir son numéro)

Remarque : Ceci permet de montrer que le numéro de la personne dans l'entreprise est unique. Comment l'indiquer si numéro est un attribut de Personne ou de la relation ?

Traduction en code : Un qualificatif UML peut se traduire par un tableau associatif (Map)

Que nous apprend ce diagramme de classe ?



Exercice : Réservation de vols

Exercice 4 : Système de Réservation de Vols (SRV)

Dans cet exercice, nous nous intéressons à un système simplifié de réservation de vols [1] pour une agence de voyage. L'interview des experts métiers a permis de résumer leurs connaissance du domaine dans les paragraphes suivants.

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

4.1. Pour chaque phrase précédente, indiquer ce que l'on apprend sur le système en utilisant le vocabulaire UML et construire le diagramme de classe correspondant.

1. Source : `http://www.eyrolles.com/Chapitres/9782212092806/chap03.pdf`

Références

[1] P.-A. Muller and N. Gaertner, *Modélisation objet avec UML*.
Eyrolles, 2è ed., 2003.

[2] M. Fowler, *UML 2.0*.
CampusPress Référence, 2004.

[3] OMG, "UML Resource Page." http ://www.omg.org/uml/.