Le diagramme des classes

A.Osmani

Site officiel UML : www.uml.org

Sommaire

Rappel : Intérêt

Chaque langage orienté objets donne un moyen spécifique d'implémenter le paradigme objet.

Une méthode objet (comme UML) permet de définir le problème à haut niveau sans rentrer dans les spécificités du langage (pointeurs ou pas, héritage multiple ou pas, etc.)

De plus, le fait de programmer dans un langage objet, ne fait pas de vont un concepteur objet

- Il est tout à fait possible de concevoir un programme syntaxiquement juste, dans un langage comme C++ ou java, sans adopter une approche objet.

Une méthode objet permet

- Une conception abstraite d'un modèle
- Implémentation à l'aide d'un langage orienté objet

Rappel : les cas d'utilisation

Un cas d'utilisation décrit un service rendu par le système

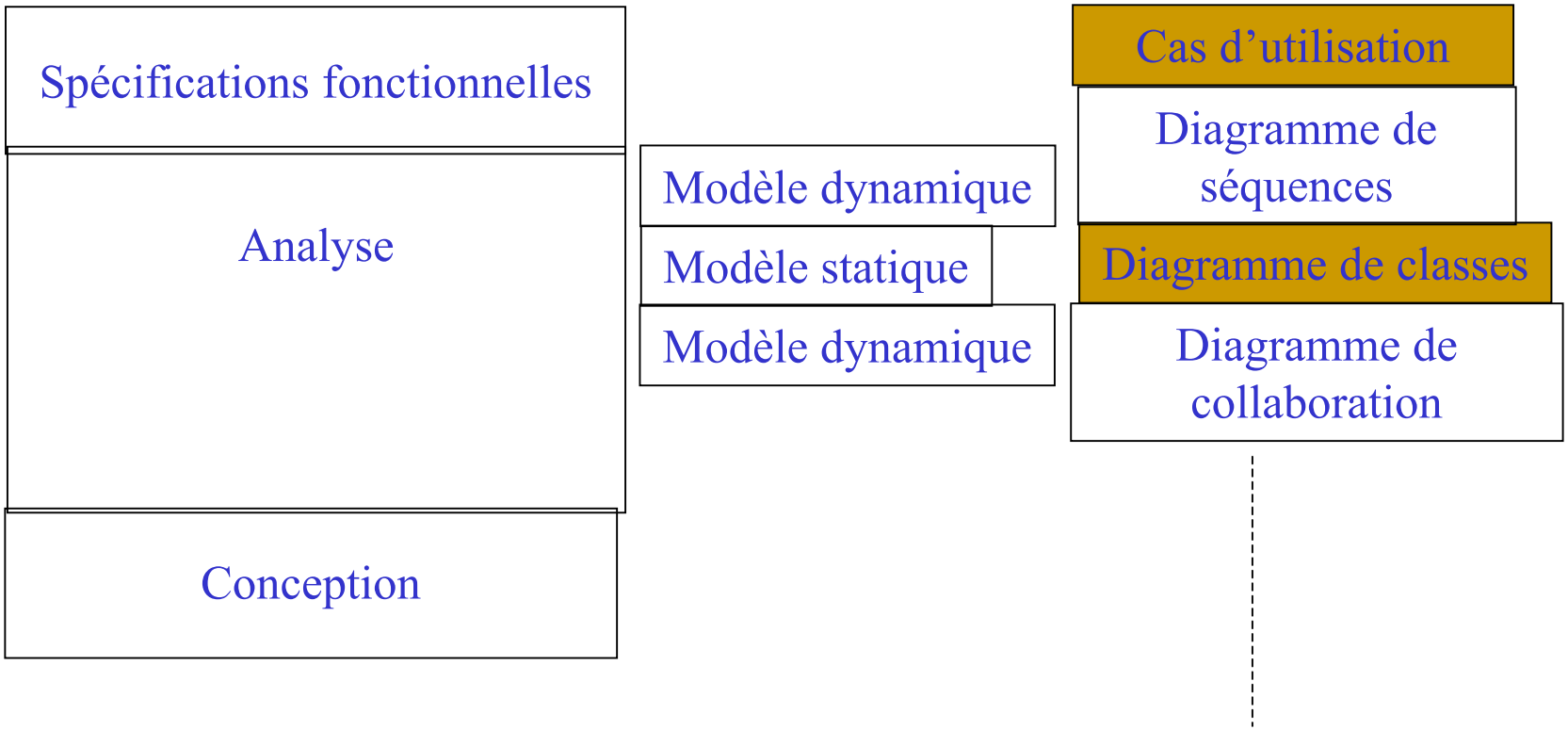
- Il exprime les interactions entre le système et les acteurs

Un acteur est une entité externe au système. Un acteur

- Attend des services du système
- Interagit avec le système par échange de messages
- Peut être une personne ou un autre système

NB : une même personne physique peut correspondre à plusieurs acteurs dans le diagramme des cas d'utilisation

Le diagramme des classes en UML



Quelques notions préliminaires

- Une instance est une concretisation d'un concept abstrait.

Exemples :

- Concept : amitié. Instance : Jean est l'ami de François
- Concetp : stylo. Instance : le stylo que vous utilisez en ce moment est une instance du concept stylo : il a sa propre forme, sa propre couleur, son propre niveau d'usure, etc.

- Un objet est une instance d'une classe (une instance n'est pas toujours un objet).

Exemple et contre exemple :

- Classe Chat : objet ou classe : le chat du voisin du troisième étage.
- Si on dispose du concept « travailler pour » qui lie deux classes Personne et entreprise alors l'instance de « travailler pour » qui lie Jean avec l'entreprise SNCF n'est pas un objet.

- Package : est un groupement de n'importe quel type d'éléments.

Exemple :

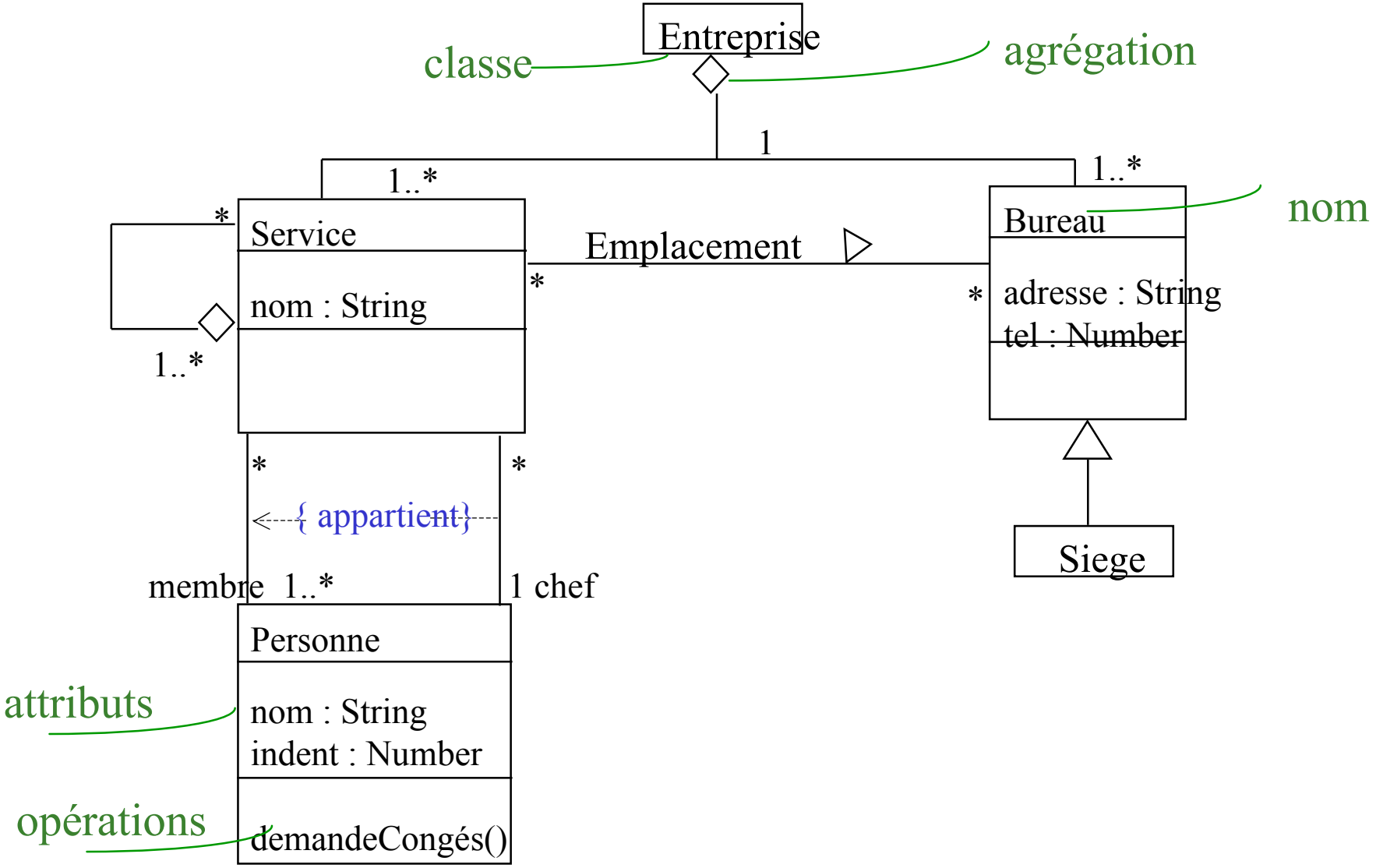
- le package java.io regroupe toutes les classes liées aux entrées et sorties.

- Stéréotype : permet de modéliser de manière compact plusieurs éléments du modèle.

Exemple :

- au lieu de décrire tous les constructeurs d'une classe Personne, on utilisera le stéréotype « constructeurs ».

Exemple de diagramme de classes



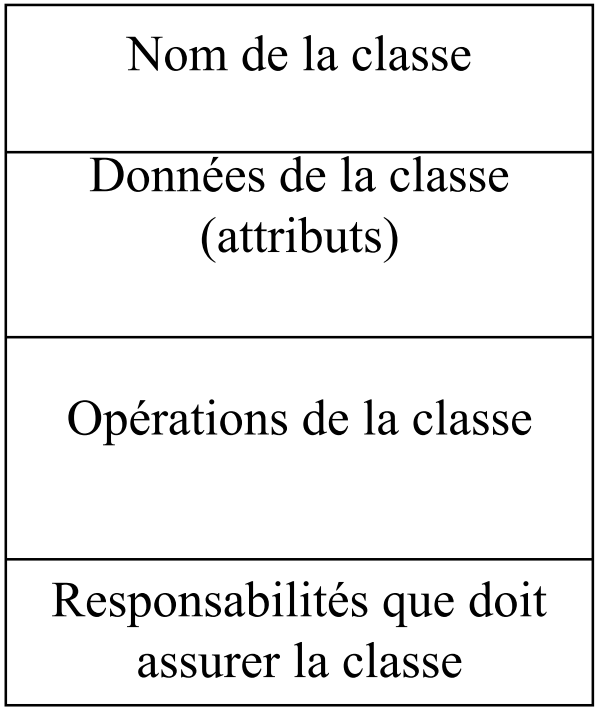
Définition d'une classe

C'est la description d'un ensemble d'objets ayant des attributs, des opérations, des relations et une sémantique communs.

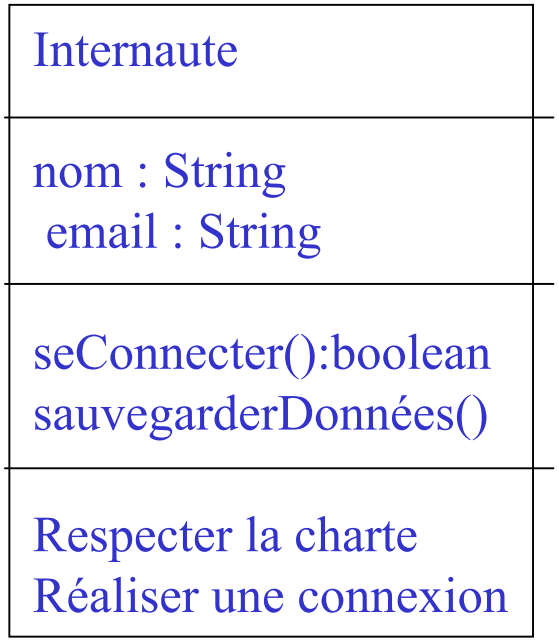
Une classe est la modélisation de la structure d'un objet. Elle regroupe l'ensemble des caractéristiques qui compose un objet.

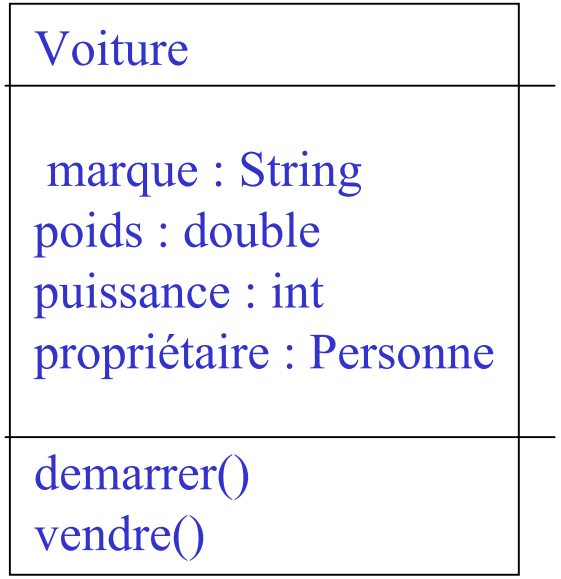
Une classe est composée d'un nom, d'attributs et de méthodes. Lors de la modélisation ces informations ne sont pas forcement connues. D'autres compartiments sont ajoutés : Responsabilités, exceptions, etc.

Graphiquement, une classe est représentée par un rectangle contenant quatre compartiments :



Exemples de classes :

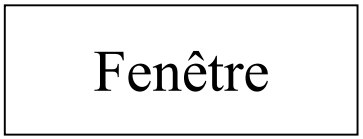




Deux objets d'une même classe peuvent avoir tous les attributs égaux sans pour autant être un seul et même objet.
Exemple : vous avez les mêmes polycopies (mêmes valeurs d'attributs) tout en étant différent.

Le nom d'une classe

Le nom doit être unique dans le système. On peut écrire le nom tout seul ou précédé de son chemin. Le chemin définit le package dans lequel se trouve la classe.

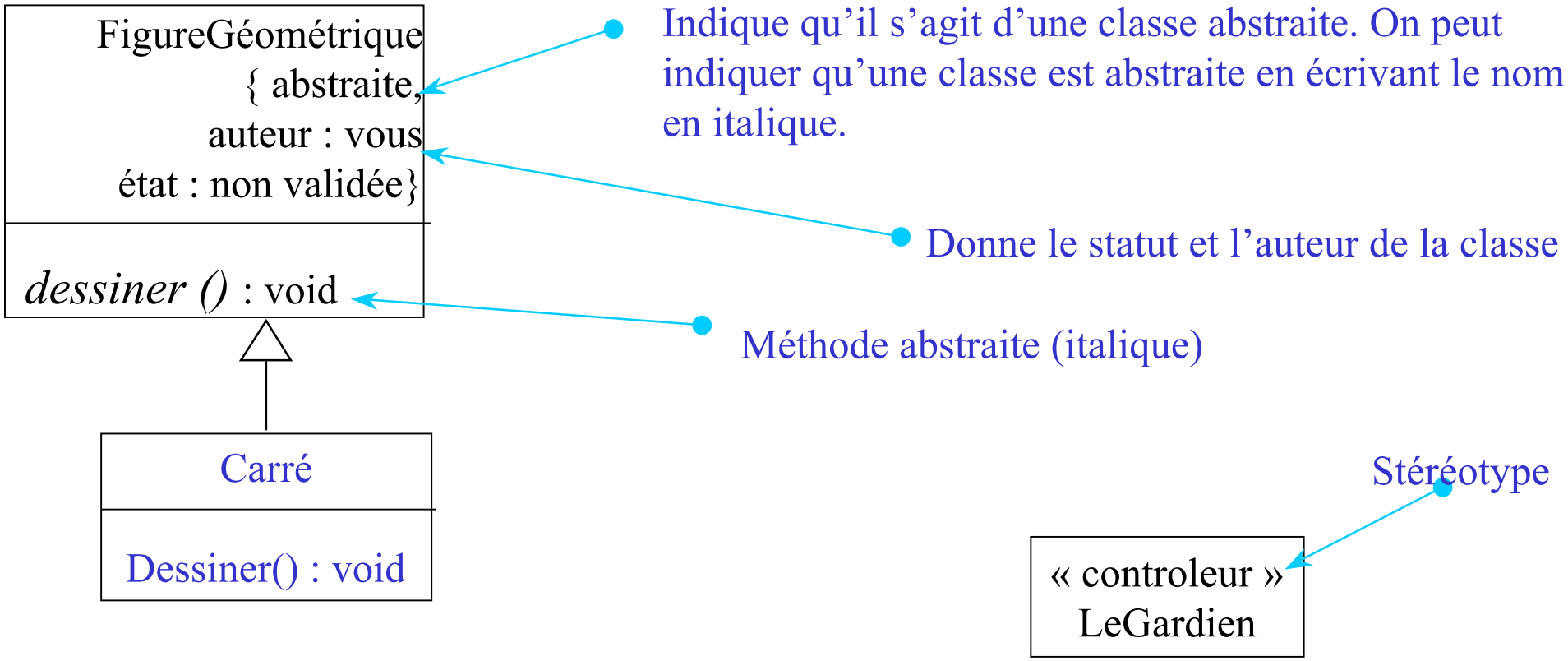


noms simples

noms avec le chemin

Le nom d'une classe

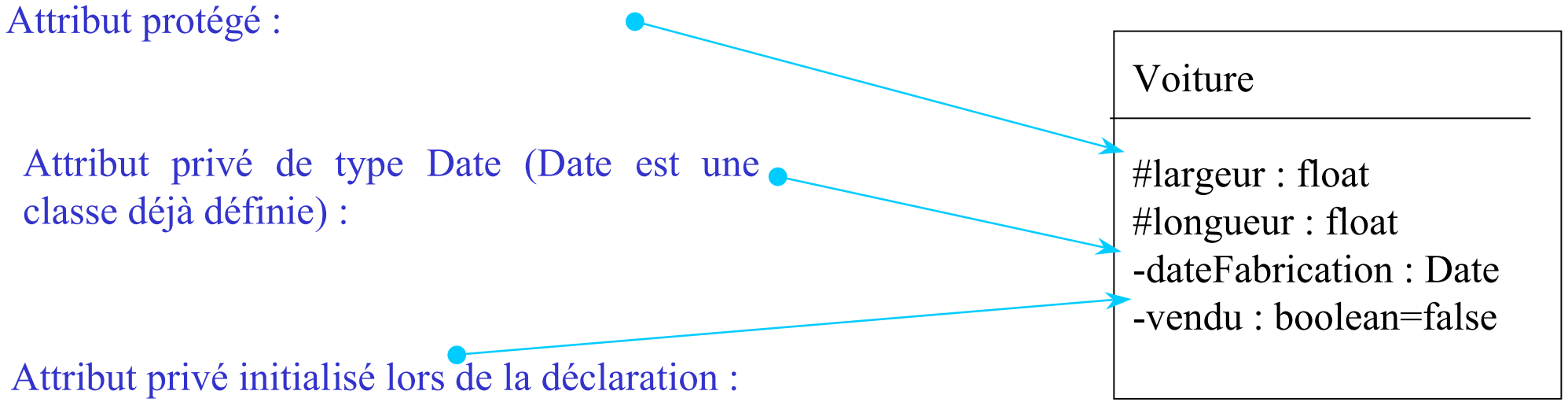
Le compartiment nom de la classe peut comporter des indications sur L'auteur, le statut et le type de classe. Il peut également comporter un stéréotype si la classe n'est pas encore bien spécifiée.



Les attributs d'une classe et modificateurs d'accès

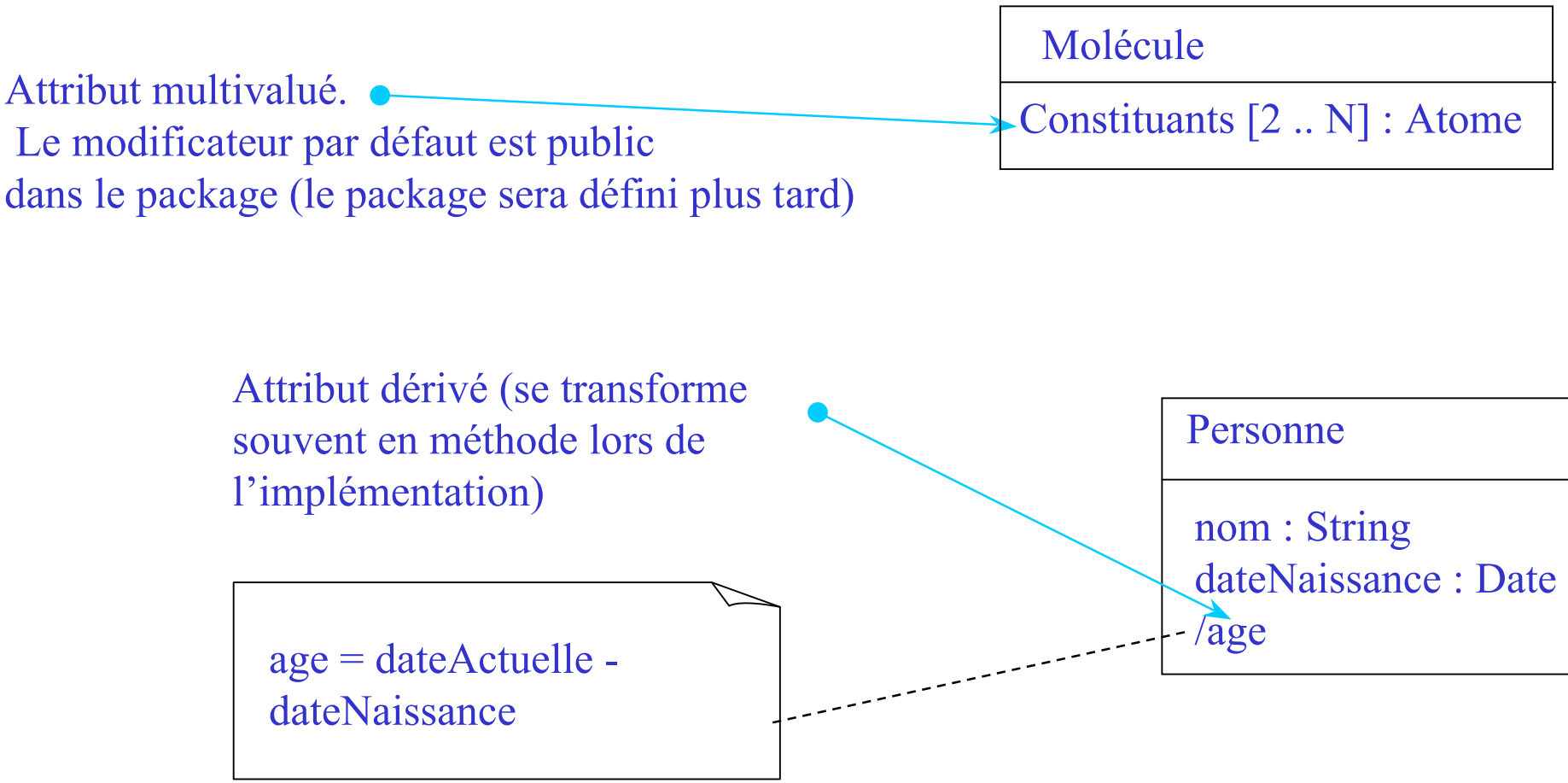
Un attribut décrit une propriété de la classe. Une classe comporte un ensemble d'attributs.

Les types des attributs et leurs initialisation ainsi que les modificateurs d'accès peuvent être précisés dans le modèle



Les attributs d'une classe

- attribut dérivé : c'est un attribut destiné à devenir une méthode.
- attribut multivalué : il définit un élément composé de plusieurs objets.

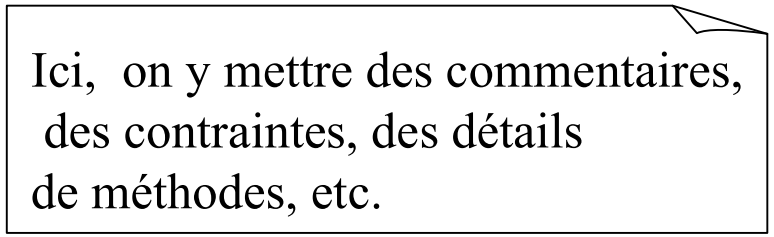


Annotations

Il existe un seul type d'annotation, il est dit **note**.

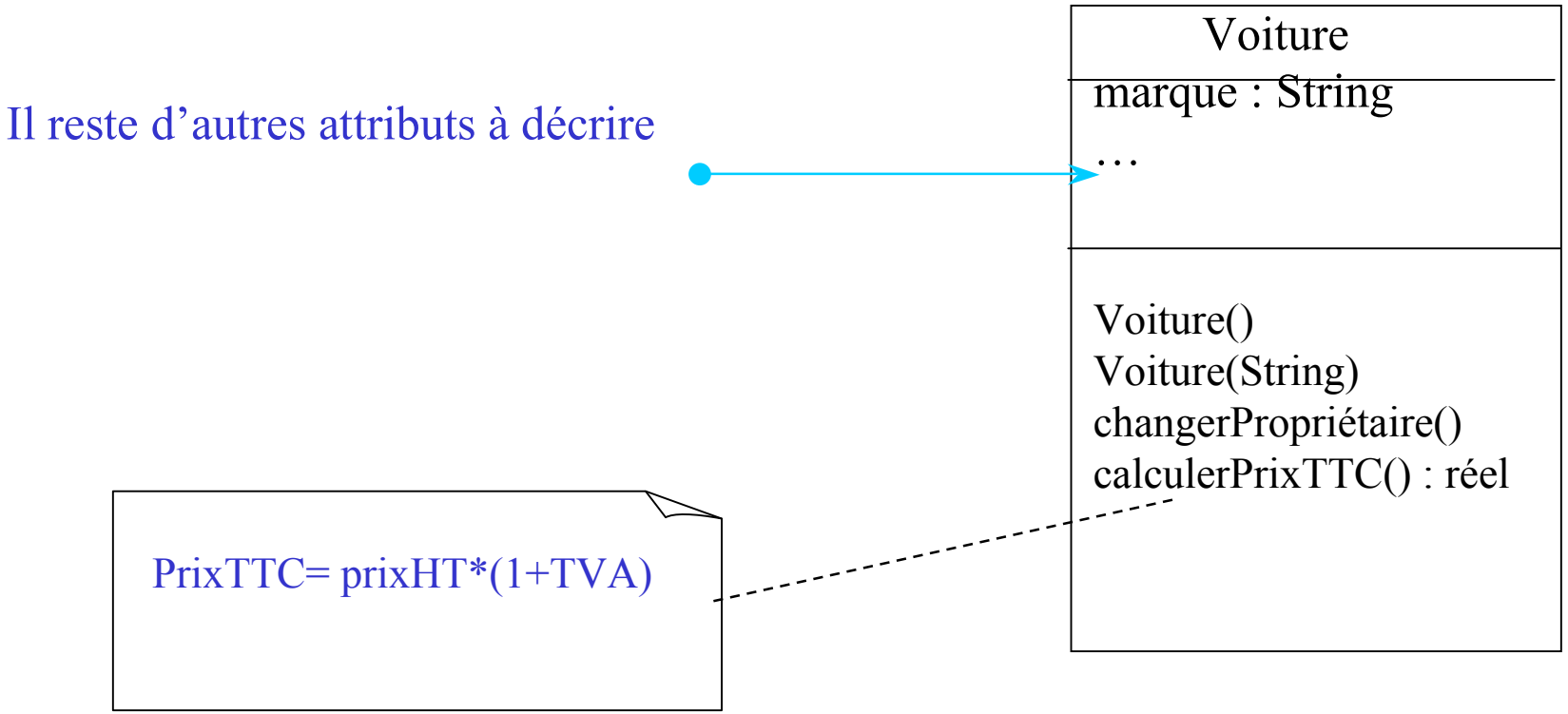
Une note est un symbole permettant de décrire des contraintes, des commentaires ou des algorithmes associés à un élément du modèle.

Graphiquement, une note est représentée par la figure :



Les opérations

Une opération est un service offert par la classe. Elle permet de changer les du modèle. Une classe dispose d'un ensemble d'opérations (qui peut être vide).

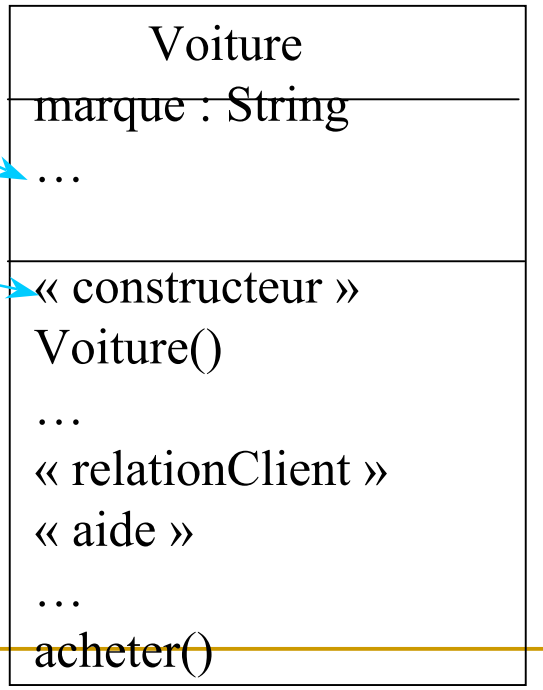


Les opérations

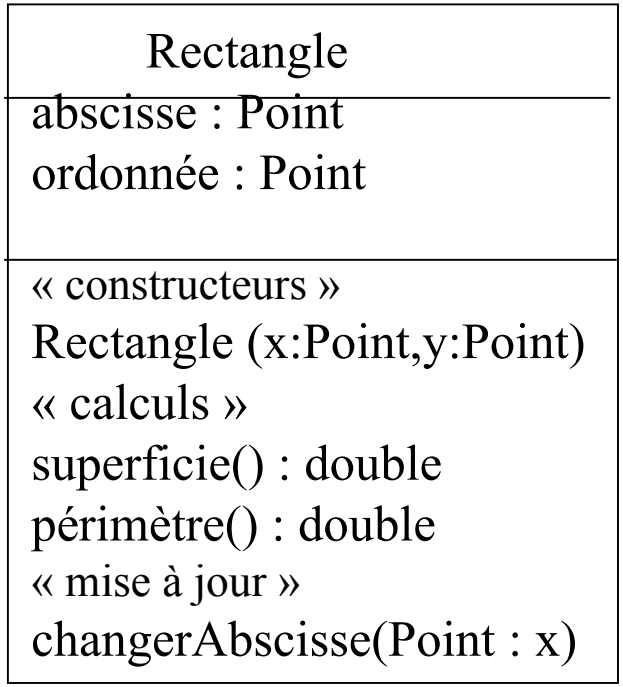
Quand le nombre d'attributs et/ou d'opérations est élevé, UML dispose d'un moyen nous dispensant de tout écrire : laisser le compartiment vide, mettre trois points de suspension ou bien utiliser des stéréotypes comme le montre l'exemple suivant :

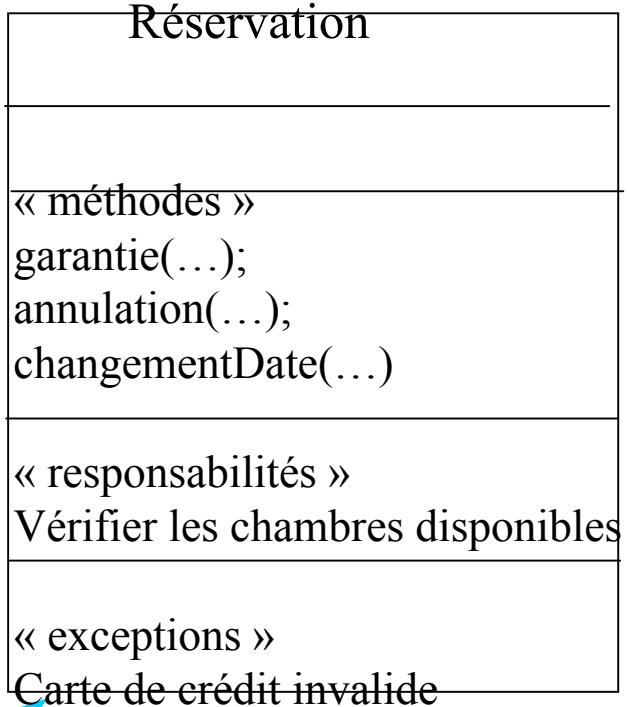
Il reste d'autres attributs à décrire

Stéréotype : il permet d'étendre la sémantique des éléments de modélisation, il s'agit d'un mécanisme d'extensibilité d'UML. Pour ce qui nous concerne, un stéréotype permet de modéliser de manière compacte plusieurs méthodes ou attributs.



Les opérations





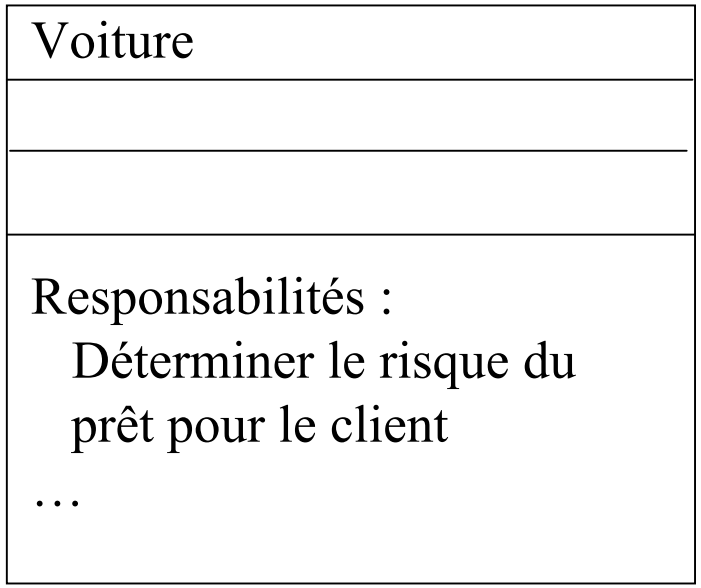
Pour assurer la prise en charge des exceptions dès la phase de conception du diagramme de classes un compartiment destiné à cet effet est ajouté à la classe.

Responsabilités

La responsabilité est l'obligation ou le contrat d'une classe. La classe voiture est responsable des valeurs des attributs largeur et longueur et de l'exécution des opérations internes.

Au début de la modélisation, souvent seules les responsabilités sont connues. A la fin de la modélisation, toutes les responsabilités doivent être concrétisées par les opérations de la classes.

Graphiquement, les responsabilités sont définies dans un compartiment dédié.

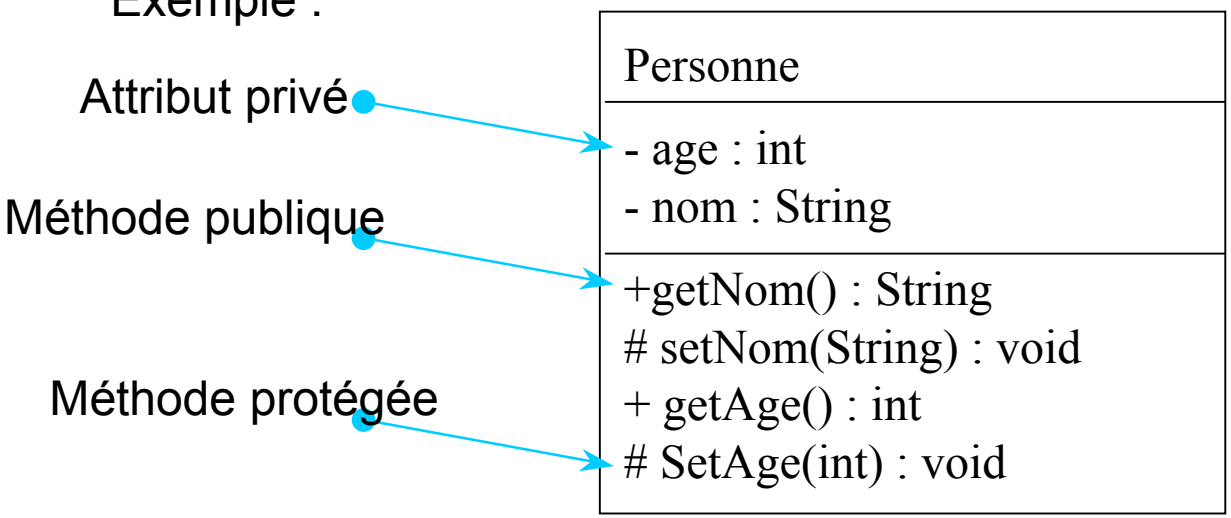


Encapsulation (Visibilité des attributs)

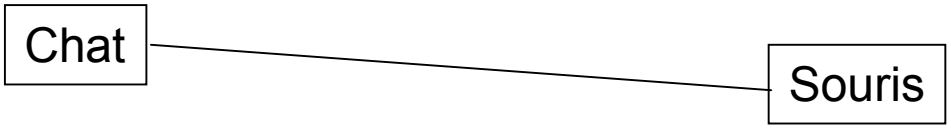
L'encapsulation un concept important du paradigme objet. L'encapsulation permet de regrouper les attributs et les méthodes dans une structure en cachant leur implémentation. Elle intérdit l'accès aux données en dehors des services proposés (modificateurs d'accès, etc.)

- L'encapsculation permet de garantir l'intégrité des données contenues dans un objet.

L'utilisateur définit des méthodes d'accès aux attributs de l'objet. Ces m éthodes sont désignées par le vocable : interface d'accès à l'objet.
Exemple :



Relations entre classes



Un lien existe entre deux classes (ou même deux objets) dès lors que qu'une classe a un rapport directe avec l'autre (appartenance, utilisation, contenance, association, dépendance, comosition, etc.)

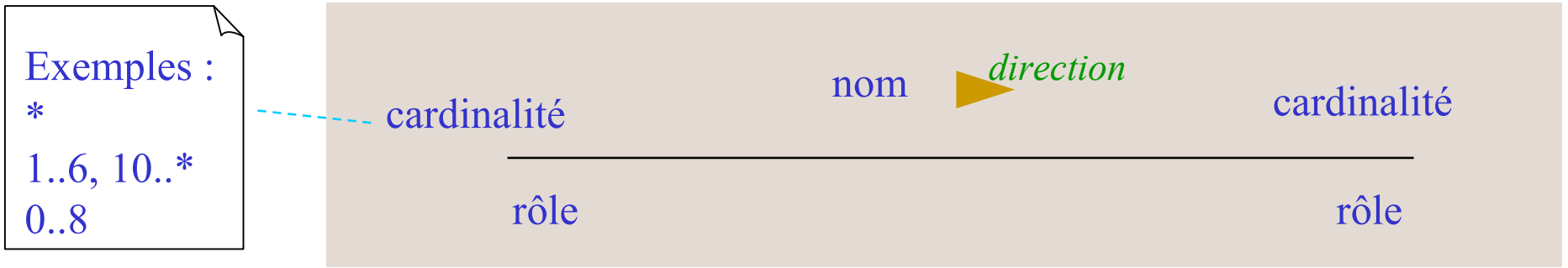
Sommaire :

1. Association
2. Héritage
3. Dépendance
4. agrégation

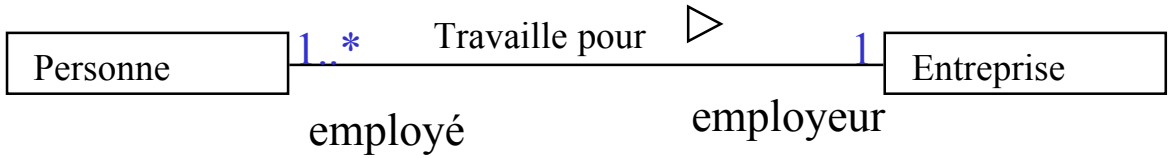
Relation association

C’est une relation structurelle entre objets. Une association est souvent utilisée pour agréger les liens.

Graphiquement, une association est décrite par un trait comme le montre le schéma suivant :

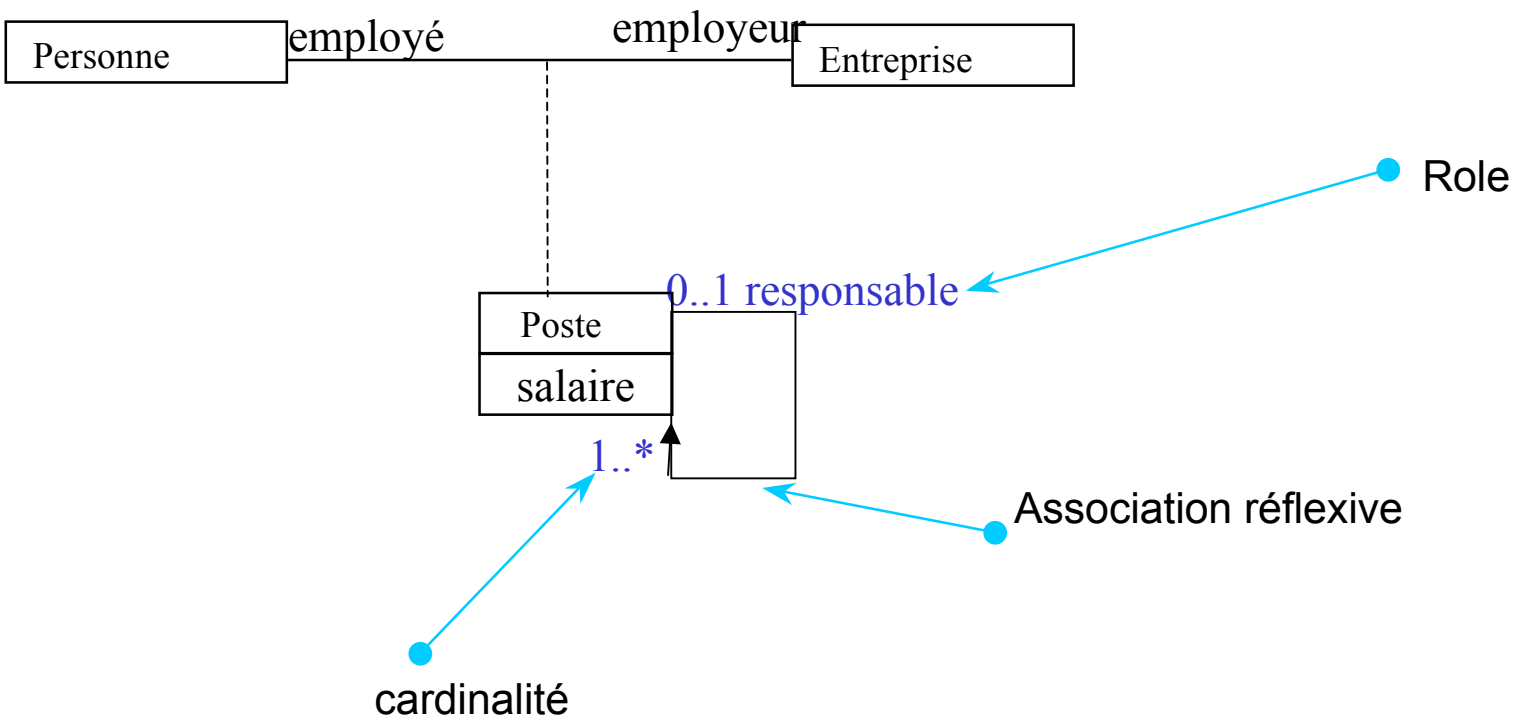


Exemple :association binaire



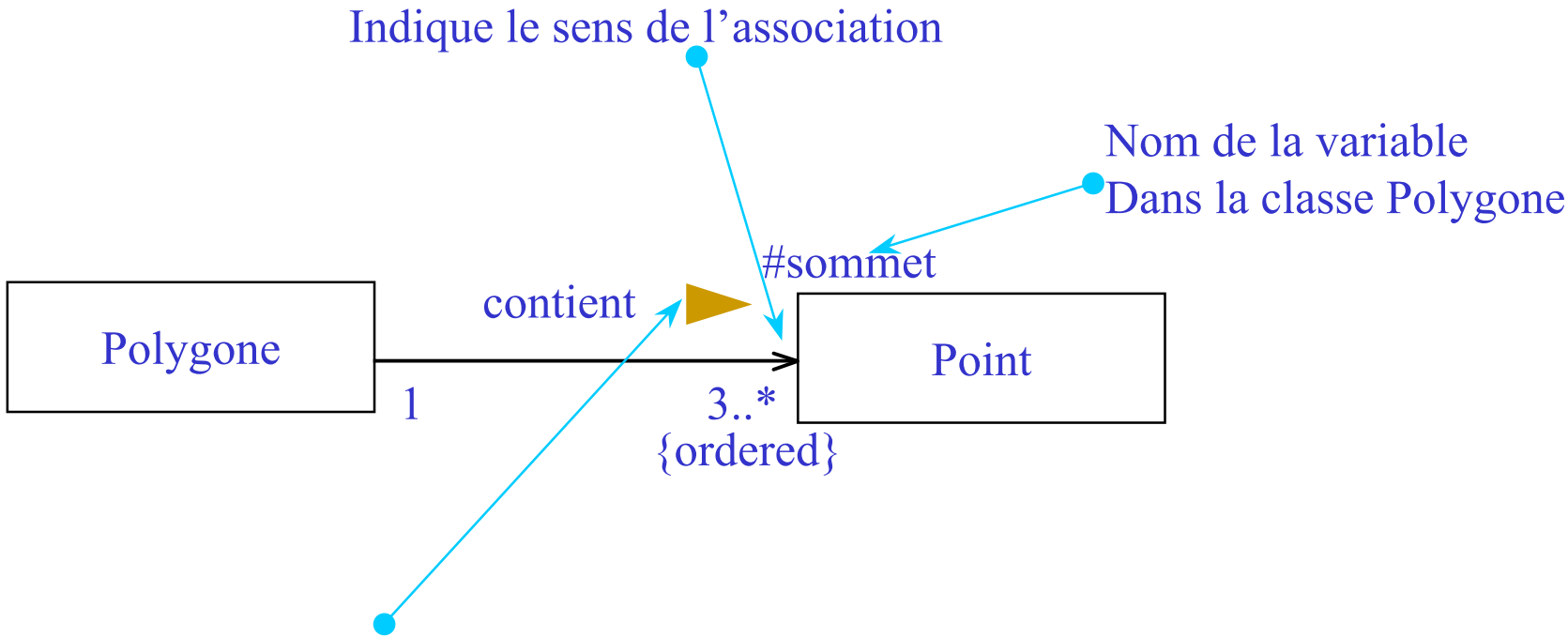
Une classe association

une classe association est une association qui possède les propriétés d'une classe, comme le montre l'exemple suivant :



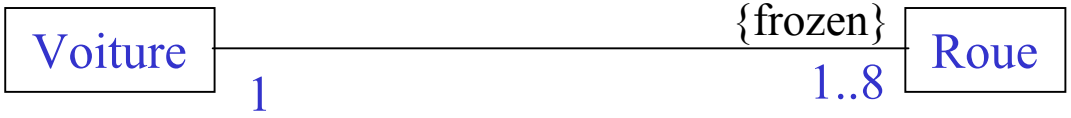
Association : la navigabilité

Permet d'indiquer le sens de navigation de l'association comme l'indique l'exemple suivant : un polygone est composé de plusieurs points. Un point « n'est pas sensé savoir » qu'il appartient à un polygone



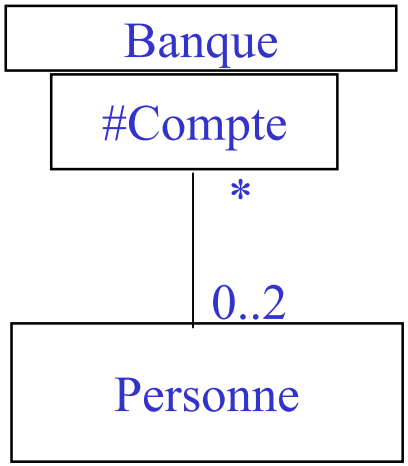
Association : changement structurel

Dans le cas ou les liens entre les classes ne change pas, rien n'est ajouté à l'association sinon il faut ajouter le mot clé {frozen} pour indiquer qu'aucune modification n'est possible après la création et l'initialisation des objets. Le mot clé {addOnly} indique que seul l'ajout sera possible, etc.

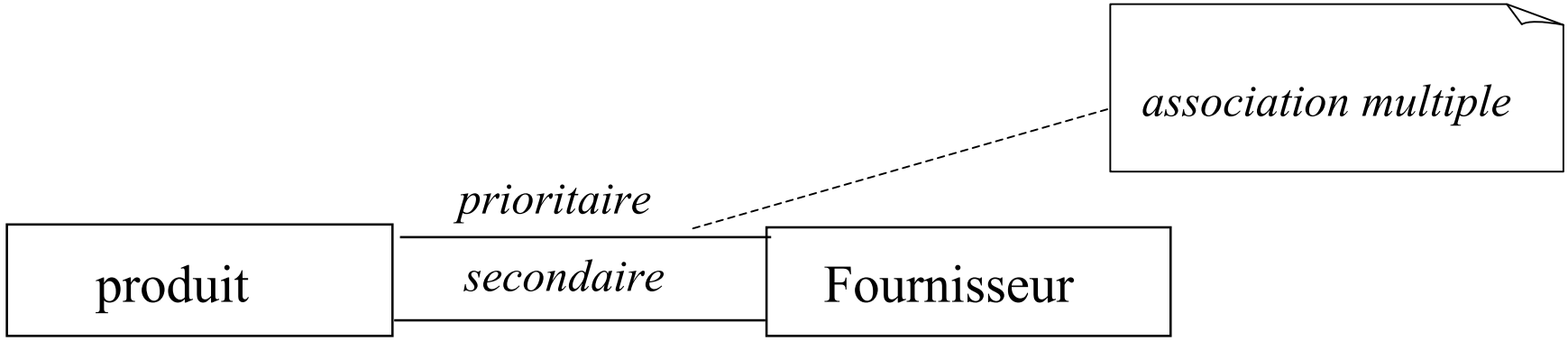


Association : « qualifieur » (en anglais qualifier)

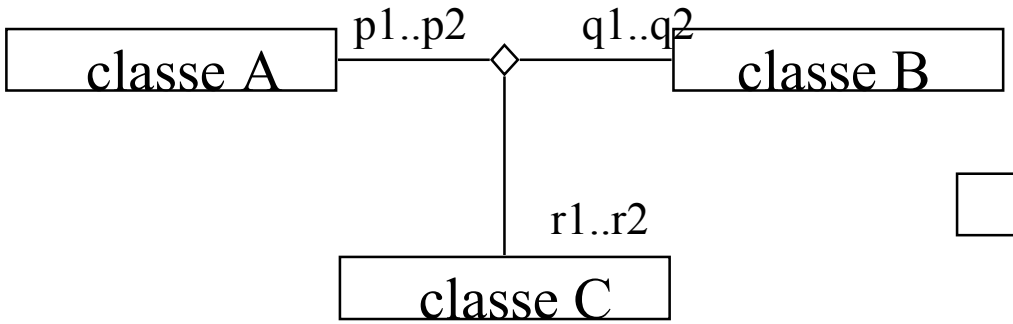
Attribut ou ensemble d'attributs permettant de partitionner l'ensemble des instances d'une classe pour mieux définir les associations. Par exemple, les clients sont associés à une banque car ils y possèdent un compte et non pour autre chose d'autre. On représente cette association comme suit :

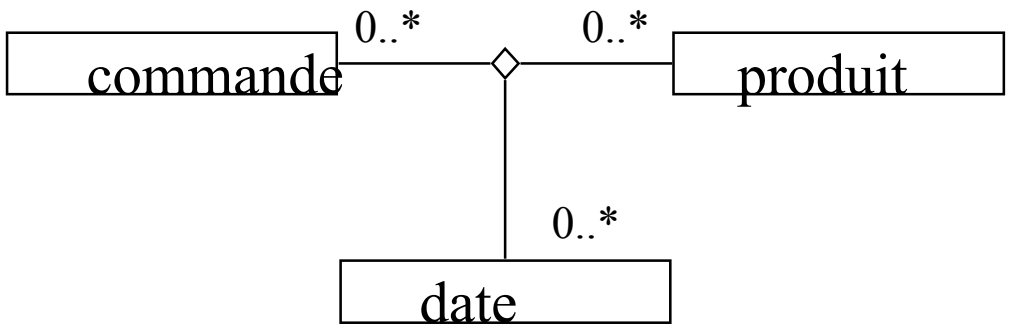


Association : relations multiple et n-aire



pour chaque produit à acheter en distingue un fournisseur prioritaire et des fournisseurs secondaires

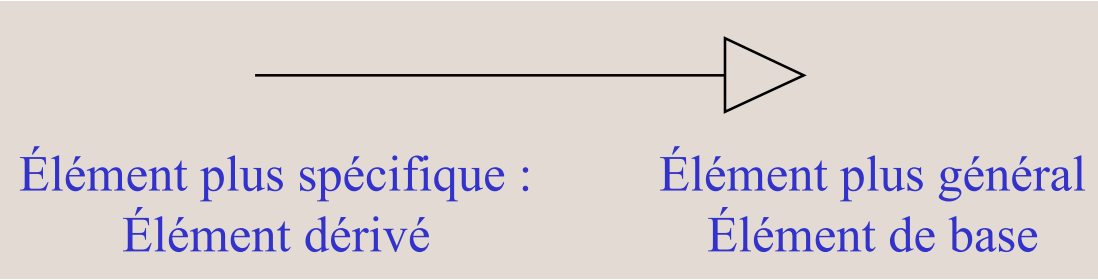




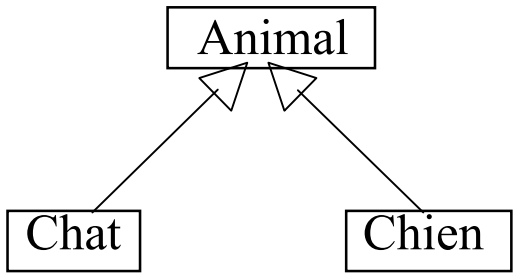
Pour un couple d'instances des classes A et B, il y a au minimum r1 instances et au maximum r2 instances de la classe C .

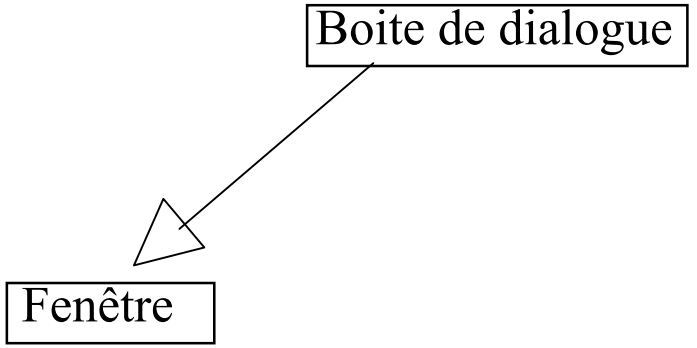
Relation d'héritage

C'est une relation de spécialisation/généralisation. Les éléments spécialisés héritent de la structure et du comportement des éléments plus généraux.



Exemple :





Relation d'héritage

Indice pour utiliser la relation d'héritage : si vous pouvez dire qu'un objet de la classe A « est un » objet de la classe B alors vous pourrez utiliser la relation d'héritage.
Exemple : un chat est un animal, un étudiant est une personne, un marteau est un outil particulier, etc.

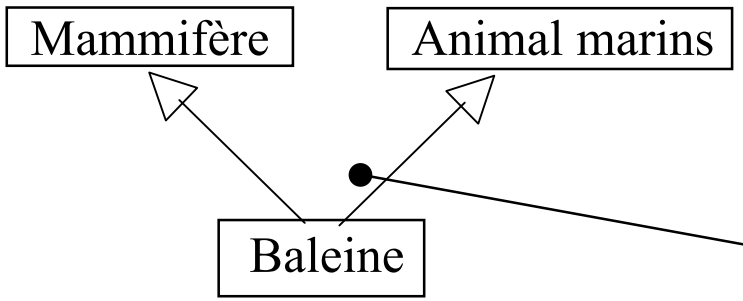
Conséquences :

- Toutes les données et les méthodes non privées des classes de base peuvent être utilisées par les classes dérivées comme s'il s'agit de leurs propres données et méthodes.

Notations à connaître
Relation d'héritage multiple

Représente les cas ou une classe dérive d'au moins deux classes de base.

Exemple :



héritage multiple (une classe dérive de deux classes de base)

L'héritage multiple est implémenté dans la plupart des langages objets (comme le c++). Les difficultés liées à la gestion de l'héritage des propriétés de plusieurs classes de base, on conduit les concepteurs du langage java à ne pas prendre en compte cet aspect de la conception objet.

Relation d'héritage multiple

Très souvent, la relation de généralisation est suffisante sans décoration. UML définit, cependant, un stéréotype et quatre contraintes pour cette relation

implementation : indique que l'enfant hérite de l'implantation du parent mais qu'il ne rend pas ses interfaces publiques et ne les supporte pas non plus, violant ainsi le principe de remplaçabilité.

en C++ ceci correspond à l'héritage avec le modificateur private.

les contraintes :

- ☞ complète : tous les enfants sont décrits dans le modèle. les enfants supplémentaires sont interdits
- ☞ incomplète : d'autres enfants peuvent être ajoutées au modèle

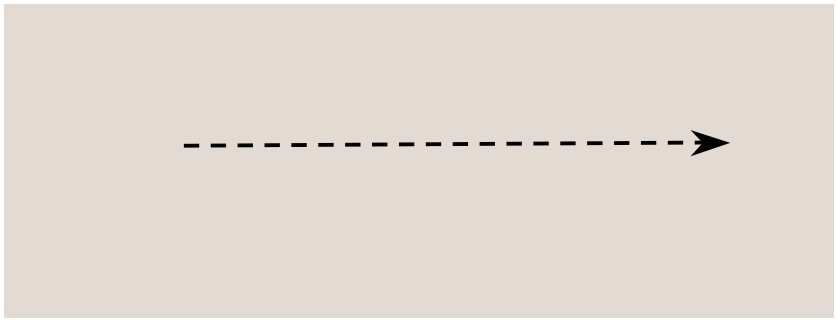
Pour l'héritage multiple on utilise deux contraintes

- ☞ disjoint : les objets d'un parent ne peuvent pas avoir plus d'un enfant donné comme type(statique)
- ☞ overlapping : les objets d'un parent peuvent avoir plus d'une enfant comme type(dynamique)

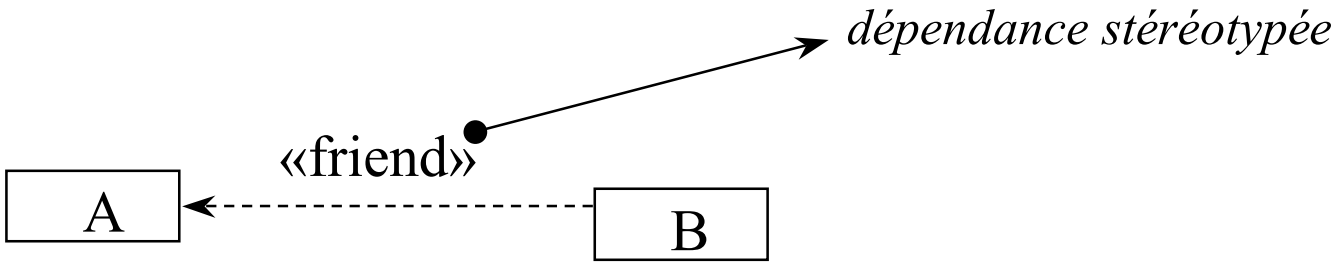
Relation de dépendance

La dépendance est une relation sémantique entre deux classes. Elle réfère aux éléments du modèle et ne nécessite pas d'instances (d'objets) pour être interprétée. Elle indique que le changement de la cible implique le changement de la source.

Graphiquement, une dépendance est décrite comme suit :



Exemple :



Relation de dépendance : types de dépendances

Un ensemble de dépendances est défini dans le standard UML (version 1.4). Parmi ces dépendances on retient :

friend : on accorde une visibilité spéciale de la source dans cible.
bind : la source instancie le template cible en utilisant les paramètres donnés.
derive : la source peut être calculée à partir de la cible
instanceof : la source est une instance (objet) de la classe cible
refine : le degré d'abstraction de la source est plus fin que celui de la cible

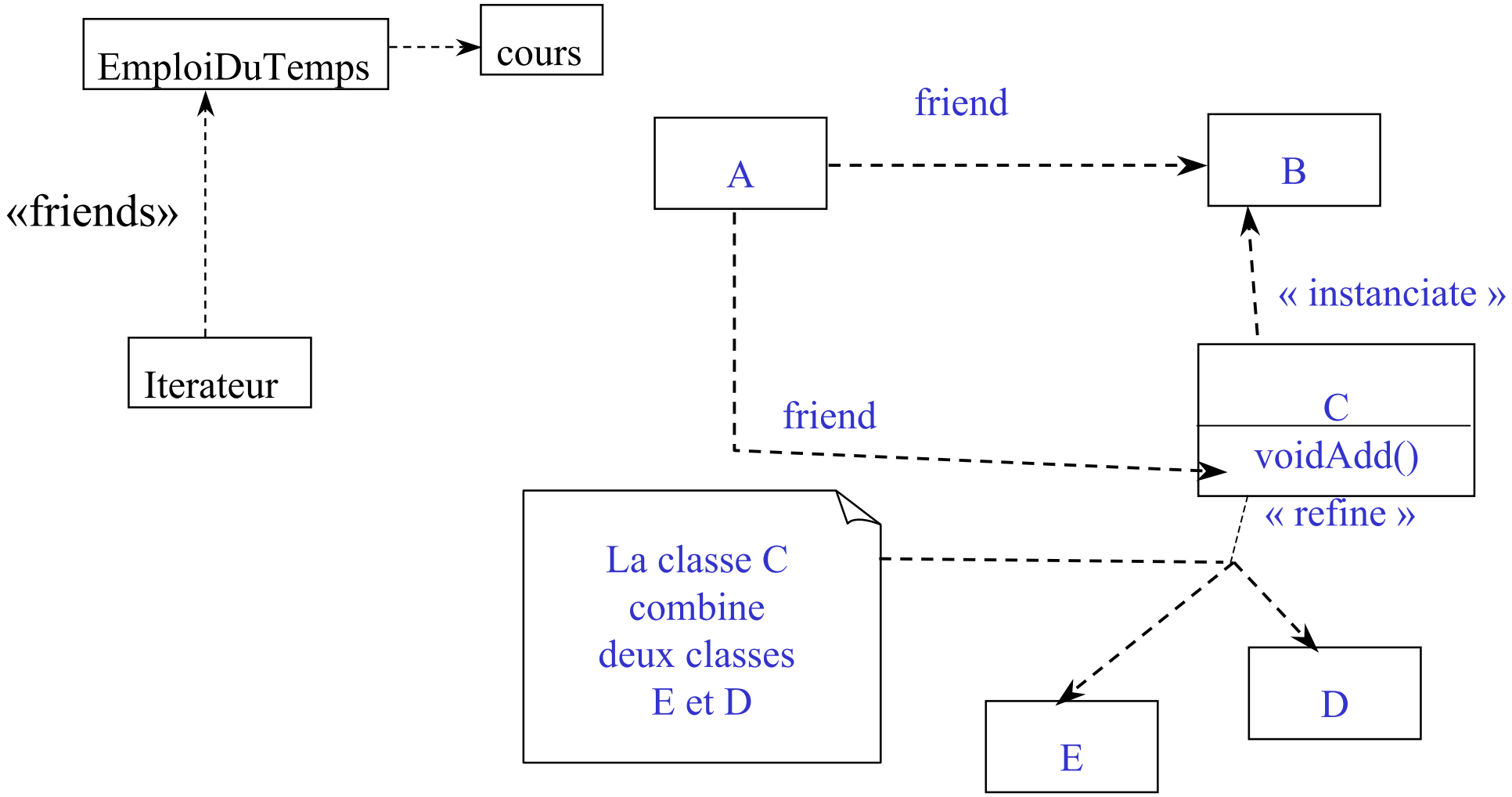
access : le package source à le droit de référencer les éléments du package cible
import : la partie publique du package cible fait partie de l'espace de nommage de la source

extend : la cas d'utilisation cible étend le cas d'utilisation source
include : le cas d'utilisation source inclus le cas d'utilisation cible

copy : l'objet source est une copie exacte et indépendante de la cible

trace : indique que la cible est une version précédente de la source.

Relation de dépendance : exemples



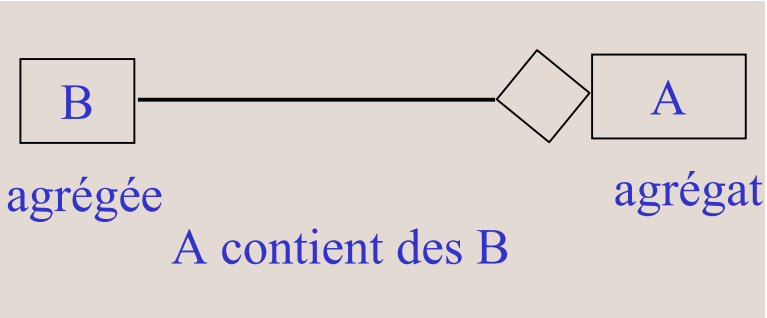
Relation d'agrégation

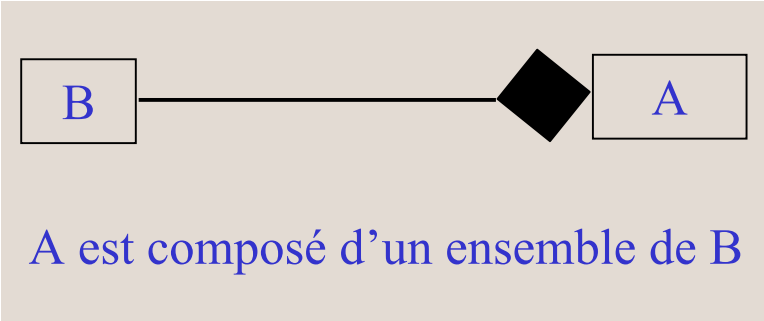
L'agrégation décrit une relation de contenance ou de composition.

On distingue, donc, deux types d'agrégation :

- L'agrégation simple (contenance) : le conteneur existe sans le contenu
- L'agrégation forte (composition) : le contenu est une composant nécessaire du conteneur. Dans ce cas l'agrégat ne peut pas être multiple.

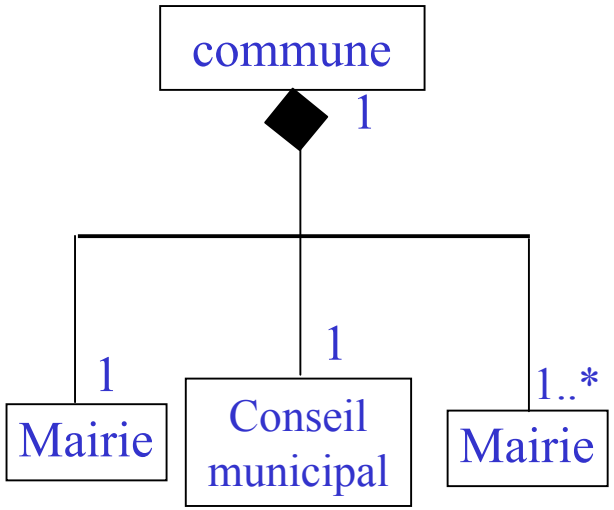
Si l'objet de la A est composé ou contient au moins un ou plusieurs objets de la classe B alors on représentera graphiquement ces agrégations comme suit :

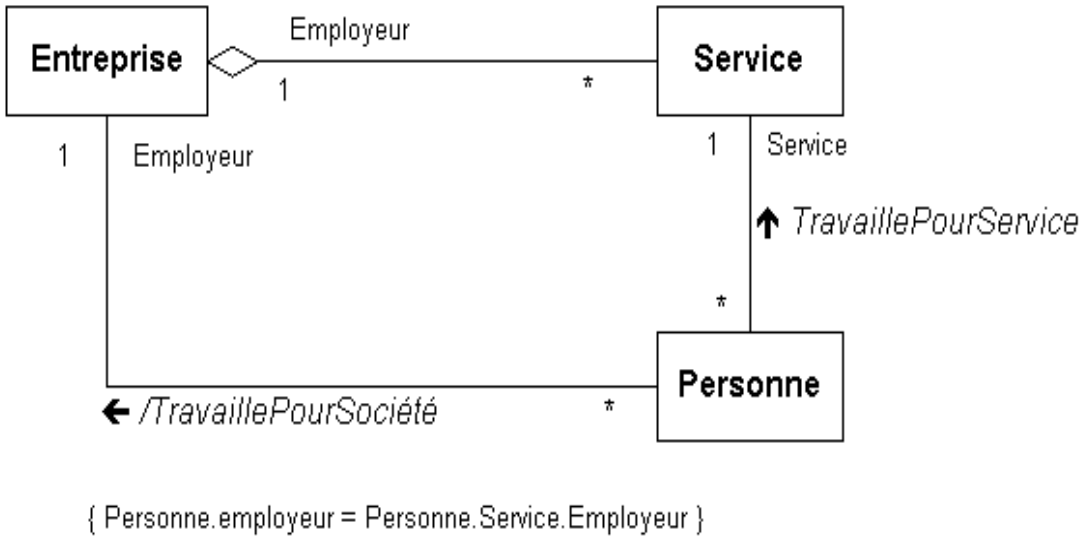




Relation d'agrégation : exemples

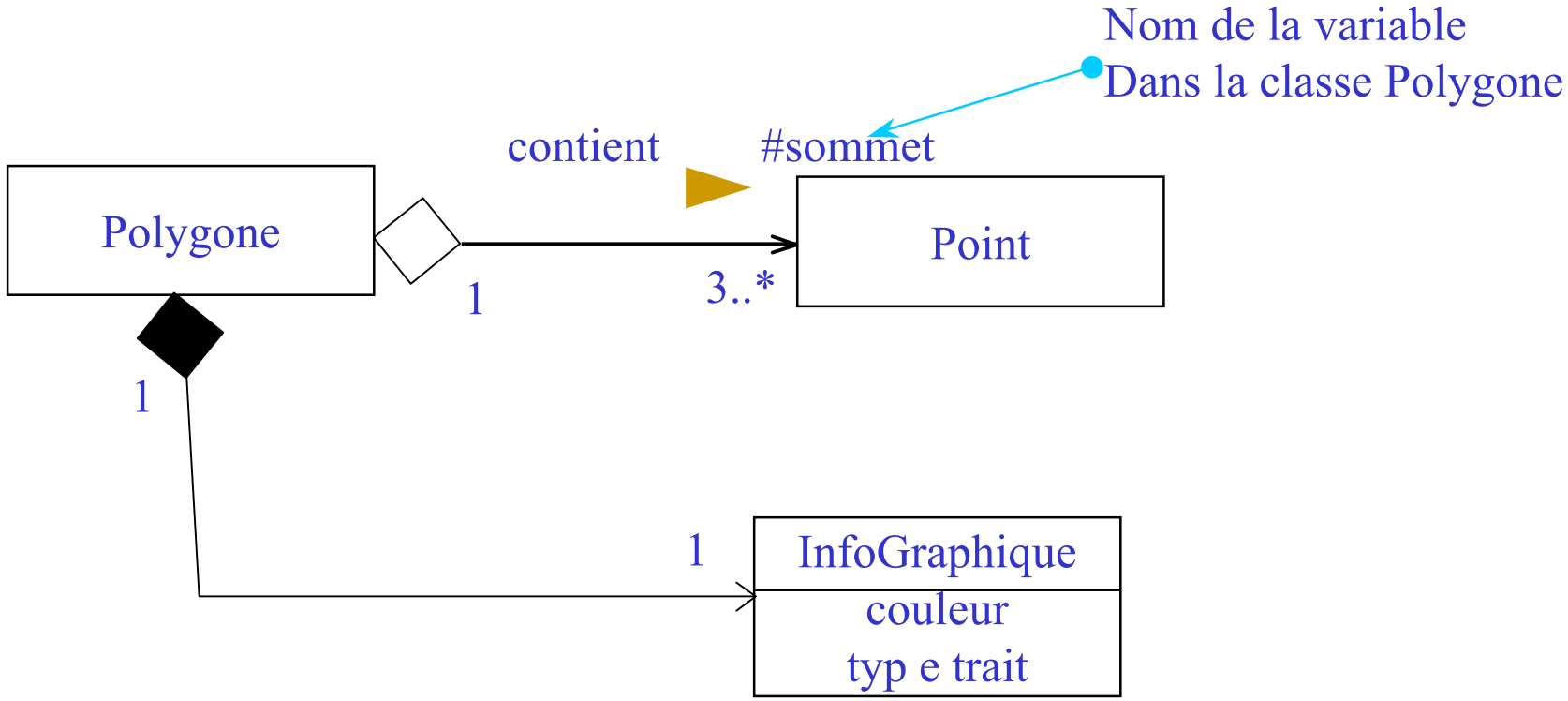
Dans le cas d'une composition entraîne la suppression des objets agrégés.





agrégation : la navigabilité

Permet d'indiquer le sens de navigation de l'association comme l'indique l'exemple suivant : un polygone est composé de plusieurs points. Un point « n'est pas sensé savoir » qu'il appartient à un polygone



Difficultés liées à la relation d'agrégation

Toute agrégation peut être modélisée par une association, mais la description sera moins précise. L'inverse n'est pas toujours vrai.
Pour identifier une agrégation, voici quelques indices :

- Le nom de l'association est proche de « est composé de »
- Il existe une différence de granularité entre les deux classes d'objets
- La suppression d'un objet agrégat induit la destruction d'autres objets agrégés
- La modification des données d'un attribut agrégé conduit à la modification des données des objets agrégés (à ne pas confondre avec dépendance)
- La définition d'une opération (méthode en java) d'un objet agrégat repose sur les opérations des objets agrégés.

Questions

Donner les relations entre les classes d'objets suivantes :

- Un immeuble et des étages
- Une entreprise et des salariés

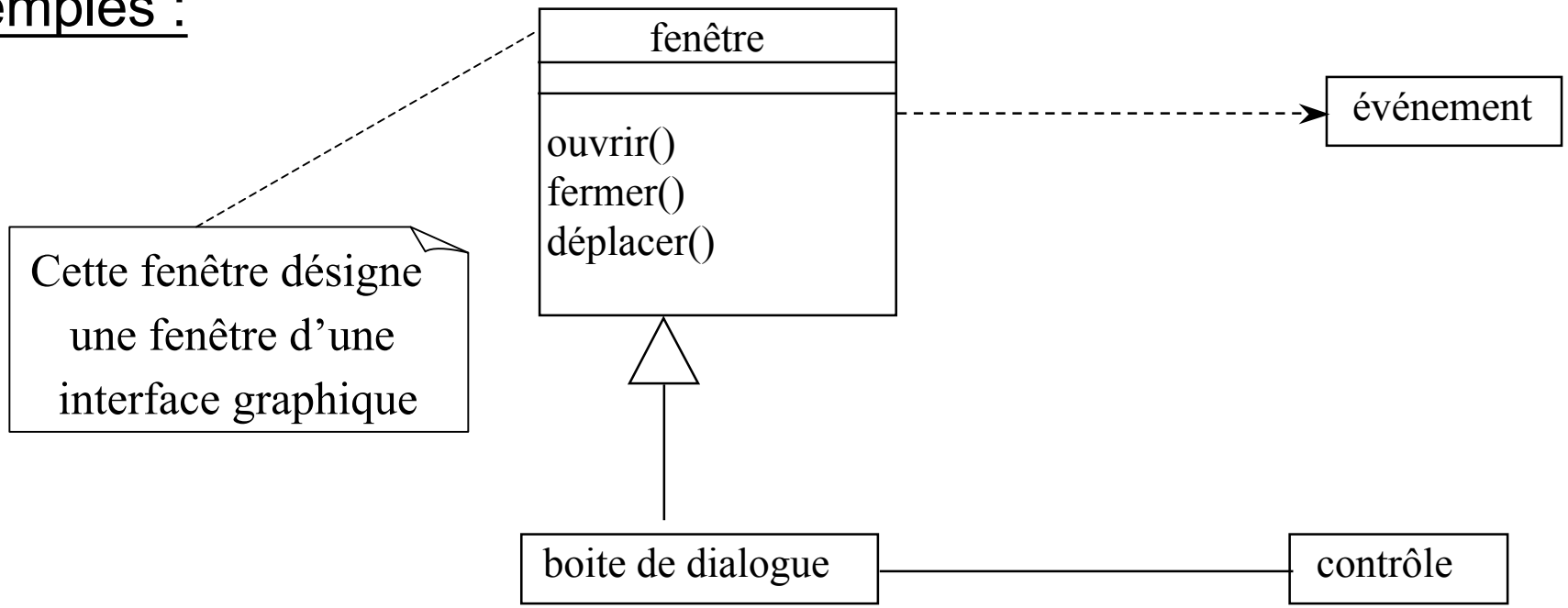
Le diagramme de classes

Ensemble de classes avec leurs relations.

C'est le plus utilisé en MOO (modélisation orientée objets)

Le diagramme des classes donne une vue statique du système/logiciel à modéliser. Il met en relation des classes, des interfaces, des types de donnéesn, des types énumérés, etc.

Exemples :



Exemple de diagramme de classes

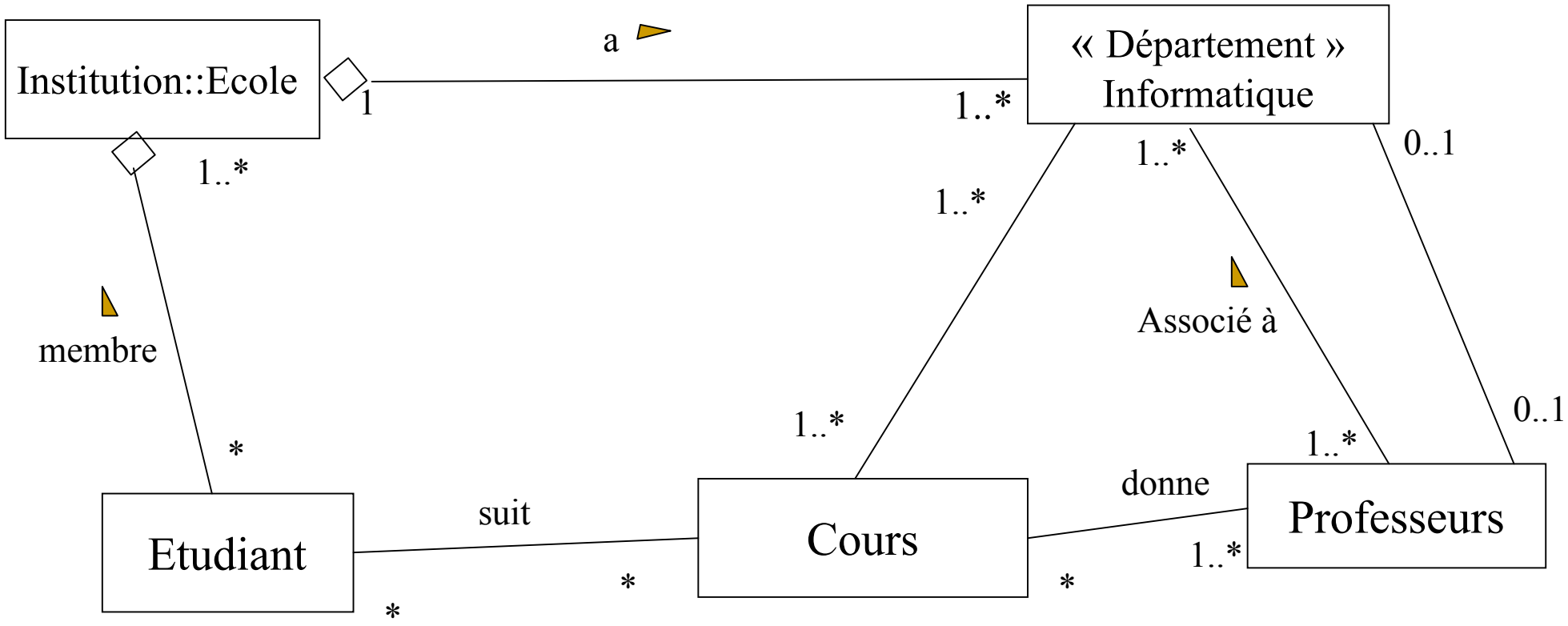


Diagramme des classes : quelques recommandations

Parmi les classes candidates il faut écarter les classes :

- Redondantes (exemple Personne, Individu)
- Trop spécifiques (qui représentent une valeur ou un attribut par exemple)
- Qui représente un opération ou une fonction

Regrouper les classes en packages (exemple java.lang, java.io, etc.)

Exemple :

L'étude du cachier des charge de la gestion d'une agence de voyage fait apparaître les classes suivantes :

- Aéroport
- Adresse du client
- Client
- Compagnie aérienne
- Destination
- Fenetre de choix
- Escale
- liste des réservations
- Fichier client
- Passager
- Facture
- Opérateur d'agence
- Réservation
- Passager privilégié
- ville
- vol
- Moyen de paiement

Quelles sont les classes pertinentes en analyse ?

Notions avancées

1. Les packages
2. Les modificateurs d'accès
3. Les classes d'activités
4. Nouveautés UML 2.0
   - Signaux
   - Ports
   - connecteurs

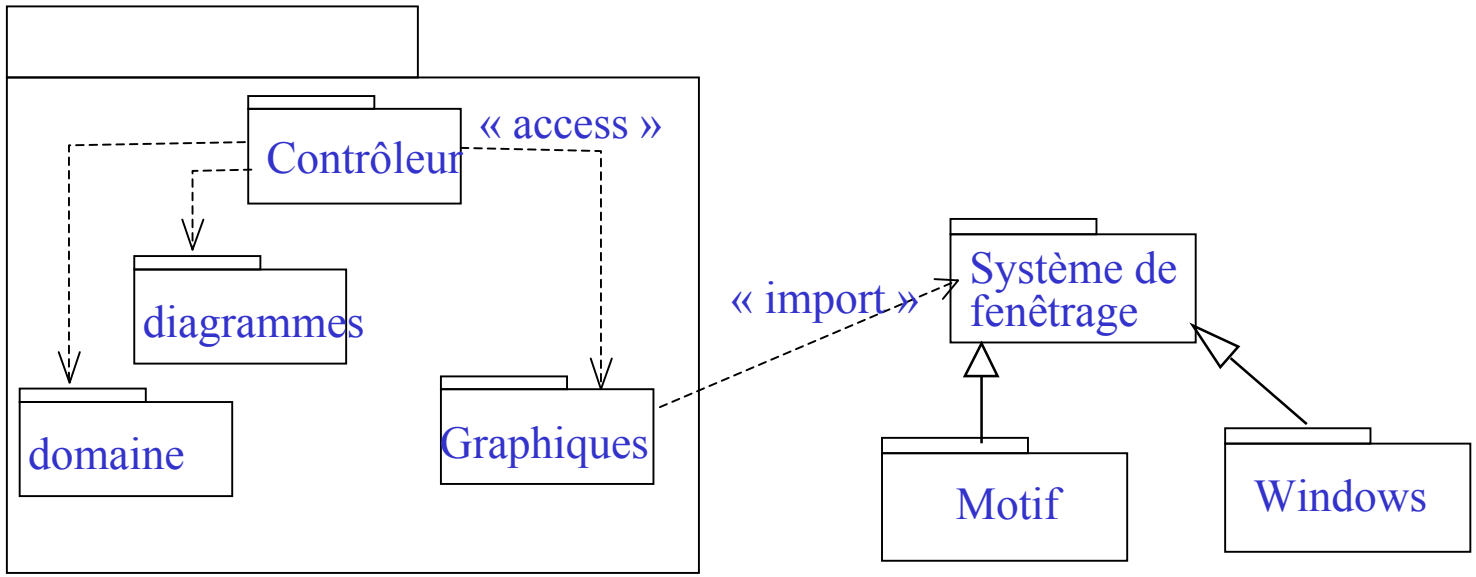
Les packages

- Un package est un groupement d'élément du modèle.
- Les packages eux- mêmes peuvent être emboîtés les uns dans les autres
- Tous les éléments du langage UML peuvent être organisés en package

Les packages constituent les éléments de base pour le contrôle de la configuration, la sauvegarde et les contrôles d'accès.

La hiérarchie des package est décrite par un arbre. Les packages peuvent appelés d'autres en utilisant les stéréotypes «import» et «access» (dépendance).

Graphiquement un package est représenté comme suit :

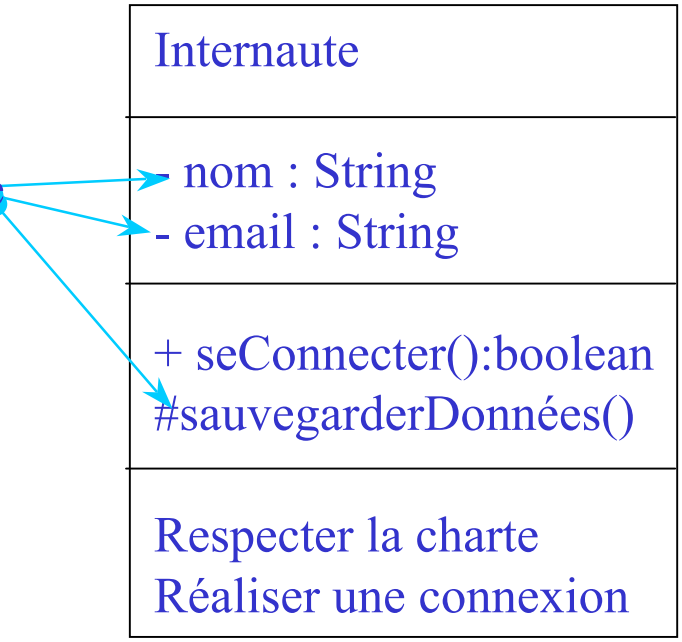


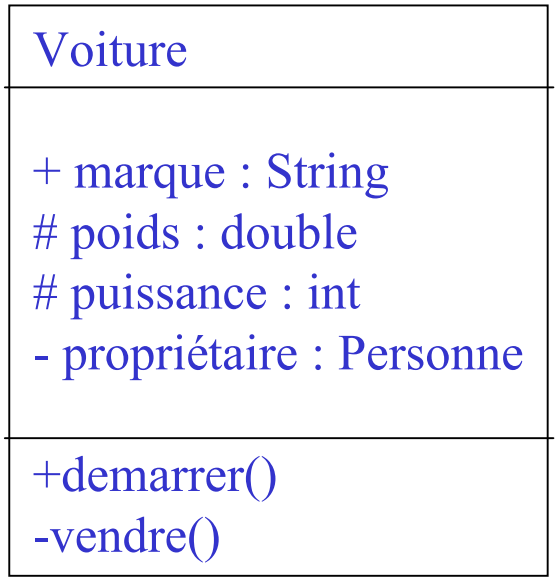
Exemples de classes : les modificateurs d'accès

les niveaux de protections des membres peuvent être ajoutés pour clarifier les droits d'accès aux membres.

Modificateurs d'accès

- : privé (seuls les objets de la classe peuvent avoir accès)
: protégé (les objets des classes dérivées peuvent aussi les utilisés)
+ : public (pas de restriction d'accès)



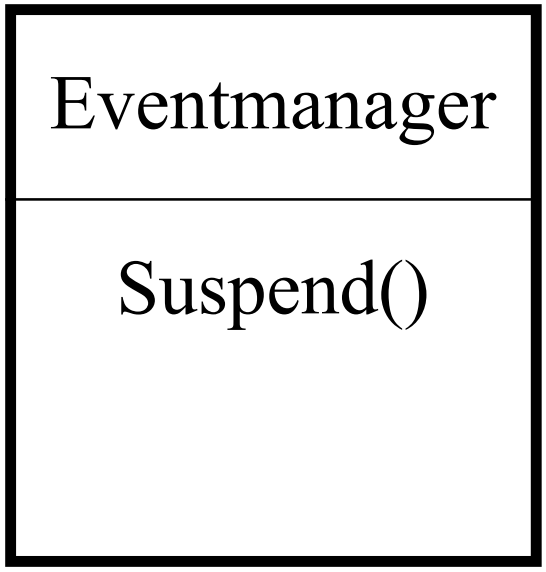


Les classes d'activité

C'est une classe dont les objets sont des processus ou des threads.

Une classe d'activité est une simple classe à l'exception que ses objets représentent des éléments qui sont en concurrence avec d'autres éléments.

Graphiquement elles sont représentées comme des classes avec des lignes en gras.



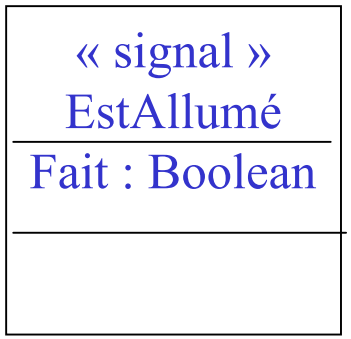
Les signaux

Le signal modélise un message échangé entre deux objets. Les signaux sont asynchrones.

Un signal est caractérisé par

- Un nom
- Un enemble d'attributs. Les attributs correspondent aux données transportées par le signal.

Graphiquement, un signal est représenté par un classe dont le nom est stéréotypé par le mot « signal ».

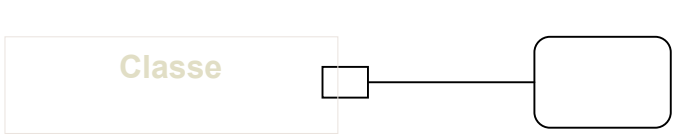


Les ports

Un port est déclaré sur une classe, en général, sur une classe active. Il porte un nom.

Il existe deux types de ports :

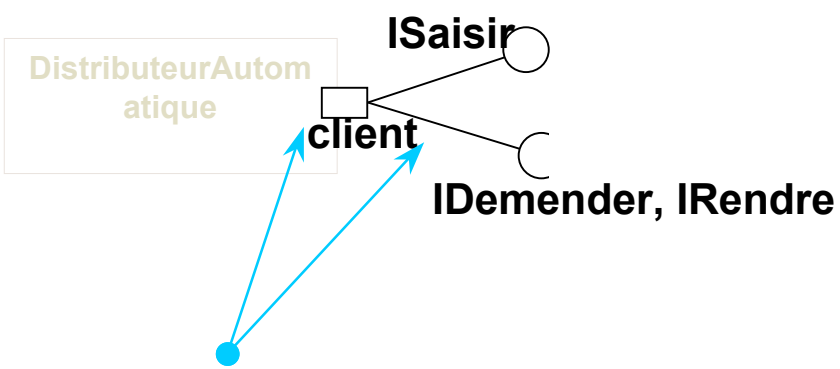
- Port protocole : sert à décrire la connectique interne et externe de la classe
- Port comportemental : il est directement associé à la machine à état (automates) de la classe active qui porte ce port. Tous les messages envoyés sur ce port sont consommés par la machine à état de la classe.



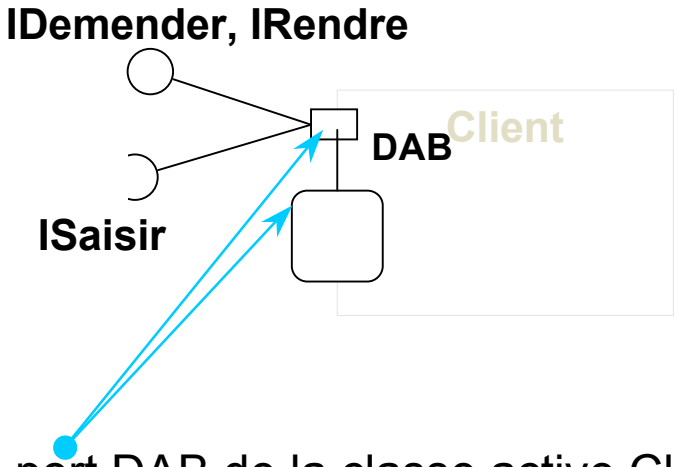
Port comportemental

Port protocole

Les ports (exemple port prototole)



le port client est un port protocole. Il traduit le point d'entrée dans le système DistributeurAutomatique pour les interfaces Client et comment ce point d'entrée se distribue sur les composants du DistributeurAutomatique. Il n'a aucune signification comportementale. Notez que les ports n'ont pas forcément besoin d'être reliés par des connecteurs. Dès l'instant où deux ports se complètent, "s'emboîtent", l'outil considère qu'ils sont connectés.



Le port DAB de la classe active Client est un port comportemental. Il traduit le point d'entrée de la classe, mais surtout indique le comportement typique d'un client de distributeur automatique. La classe Client contient la description d'une machine à états.

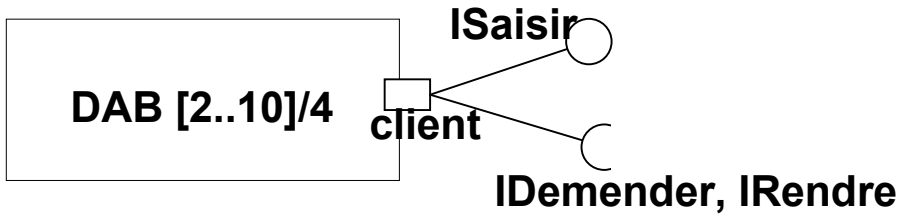
Les Parties (Part en anglais)

Une Partie désigne les instances qui composent une classe.

Une partie est représenté dans un diagramme d'architecture (nouveau diagramme proposé par UML2.0). Ce diagramme est également appelé diagramme structure composite.
Une partie représente une ou plusieurs instances d'une classe grâce à des contraintes de multiplicité. Les multiplicités sont importantes à préciser d'abord dans un but descriptif mais également pour avoir une simulation réaliste lorsque l'outil propose la fonctionnalité de simulation du modèle. Une partie est considérée comme un attribut de la classe active qui le contient.

Exemple :

- ✓ partie avec multiplicité. La multiplicité s'exprime ainsi [cardinalité_minimale..cardinalité_maximale] / cardinalité_initiale. La cardinalité initiale représente le nombre d'instances créées automatiquement lors de la création de la classe contenant le part.



Les connecteurs s'apparentent aux associations. Ils apparaissent dans le diagramme d'architecture et permettent de connecter les parties via les ports.

Connecteur (Part en anglais)