Chapitre 1
Rappels sur la modélisation UML

Introduction

- Approche Objet:
  - Développement des systèmes logiciels complexes
  - Incontournable
  - Evolution incessante : technologies, besoins applicatifs
  - Mais moins intuitive que la programmation fonctionnelle -> besoin de modéliser

- Modélisation:
  - Apporte une grande rigueur
  - Offre une meilleure compréhension
  - Facilite la comparaison des solutions de conception avant leur développement
  - Langages de modélisation: s'affranchir des contraintes des langages de programmation
  - Besoin d'une méthode de description et de développement de systèmes:
    - Années 90: plusieurs dizaines mais aucune ne prédomine

- **UML (Unified Modeling Language):**
  - Unification et normalisation: Booch + OOSE + OMT
  - Notation graphique pour représenter, spécifier, construire et documenter les systèmes logiciels
  - Objectifs:
    - Modélisation de systèmes en utilisant les techniques orientées objet
    - Création d'un langage abstrait compréhensible par l'homme et interprétable par les machines

- **UML (Suite …)**
  - S’adresse à toutes les personnes chargées de la production, du déploiement et du suivi de logiciels (analystes, développeurs, chefs de projets, architectes…)
  - Sert à la communication avec les clients et les utilisateurs du logiciel
  - S’adapte à tous les domaines d’application et à tous les supports
  - Permet de construire plusieurs modèles d’un système, chacun mettant en valeur des aspects différents : fonctionnels, statiques, dynamiques et organisationnels

- **UML (Suite …)**
  - Notation mais pas une méthodologie
  - Processus de développement complets fondés sur UML:
    - Rational Unified Process (RUP)
    - MDA (Model Driven Architecture)
  - Ne convient pas pour la modélisation de processus continus (procédés en physique)
  - N'est pas formel
  - N'est pas un langage de programmation : n'a pas la rigueur syntaxique et sémantique … malgré UML2
    - Produit du code partiel (squelettes de classes)

- **UML (Suite …)**
  - Continuellement enrichi

VERSION	ADOPTION DATE	URL
2.5	May 2015	http://www.omg.org/spec/UML/2.5
2.4.1	July 2011	http://www.omg.org/spec/UML/2.4.1
2.3	May 2010	http://www.omg.org/spec/UML/2.3
2.2	January 2009	http://www.omg.org/spec/UML/2.2
2.1.2	October 2007	http://www.omg.org/spec/UML/2.1.2
2.0	July 2005	http://www.omg.org/spec/UML/2.0
1.5	March 2003	http://www.omg.org/spec/UML/1.5
1.4	September 2001	http://www.omg.org/spec/UML/1.4
1.3	February 2000	http://www.omg.org/spec/UML/1.3
1.2	July 1999	http://www.omg.org/spec/UML/1.2
1.1	December 1997	http://www.omg.org/spec/UML/1.1

- **UML (Suite …)**
  - Couvre toutes les phases du cycle de vie de développement d'un système
  - Indépendant des langages d'implémentation
  - Indépendant des domaines d'application

1. Diagramme de cas d'utilisation (Use Case Diagram)

- Objectifs: Recueillir, analyser et organiser les besoins
- Le maître d'ouvrage:
  - Définir et exprimer les besoins
  - Valider les solutions proposées par le maître d'œuvre
  - valider le produit livré
- Le maitre d'oeuvre:
  - Compétences techniques: savoir-faire va bien au-delà
  - Recueille les besoins auprès du maître d'ouvrage
    - Discussions, rapports techniques, étude de marché, …
    - Question: ai-je toutes les connaissances et les informations pour définir ce que doit faire le système ?

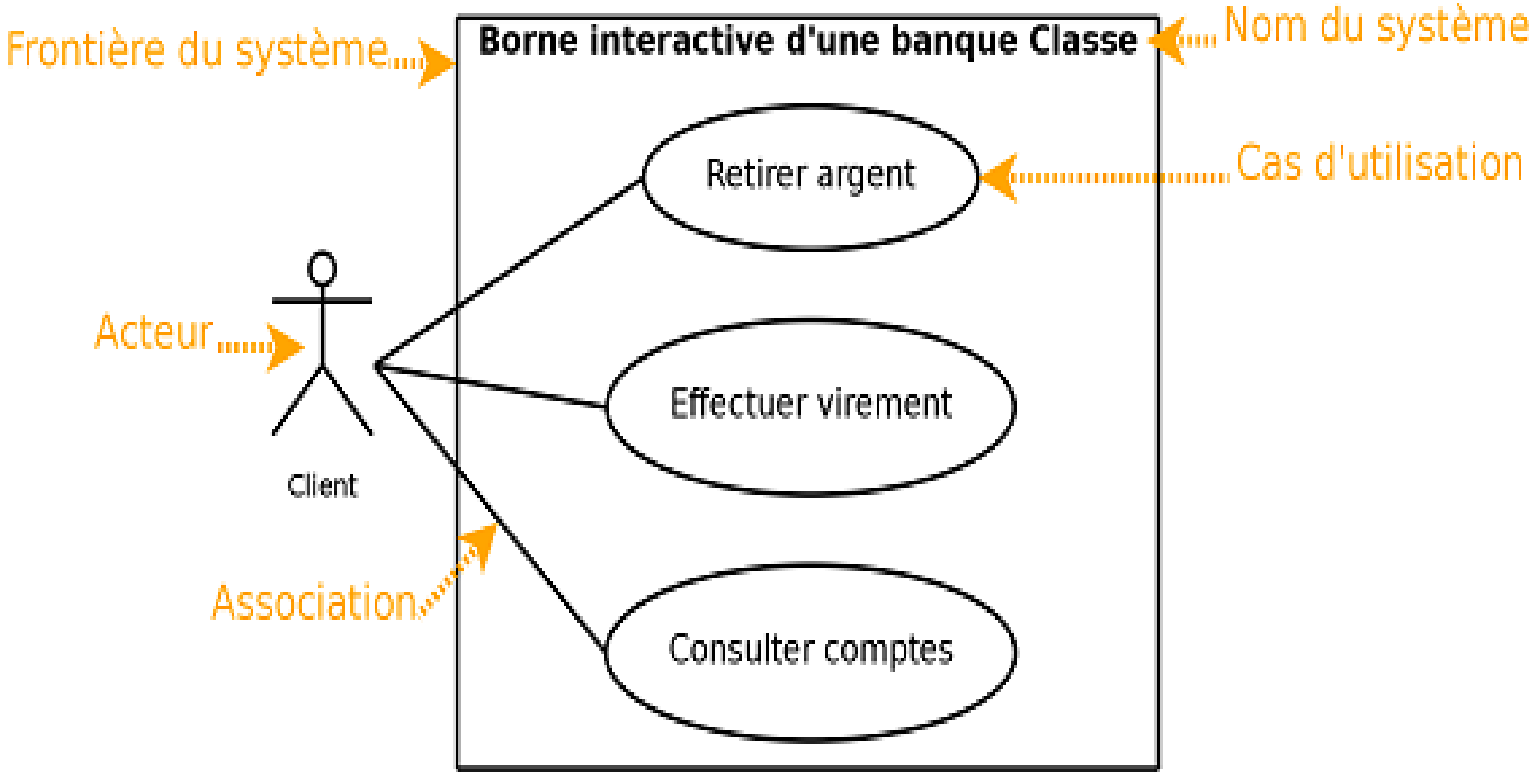
Cas d’utilisation

- **Définition:** Un **cas d’utilisation** est une manière spécifique d’utiliser un système.

  Les acteurs sont à l’extérieur du système ; ils modélisent tout ce qui interagit avec lui.

  Un cas d’utilisation réalise un service de bout en bout, avec un déclenchement, un déroulement et une fin, pour l’acteur qui l’initie.

- Exemple:


Frontière du système
Borne interactive d'une banque Classe
Nom du système
Retirer argent
Cas d'utilisation
Acteur
Effectuer virement
Client
Association
Consulter comptes

- **Définitions:**
  - Un **stéréotype** représente une variation d'un élément de modèle existant.
  - Un **classeur** est un élément de modélisation qui décrit une unité comportementale ou structurelle. Les acteurs et les cas d'utilisation sont des classeurs.
  - Un cas d'utilisation est dit **« interne »** s'il n'est pas relié directement à un acteur.

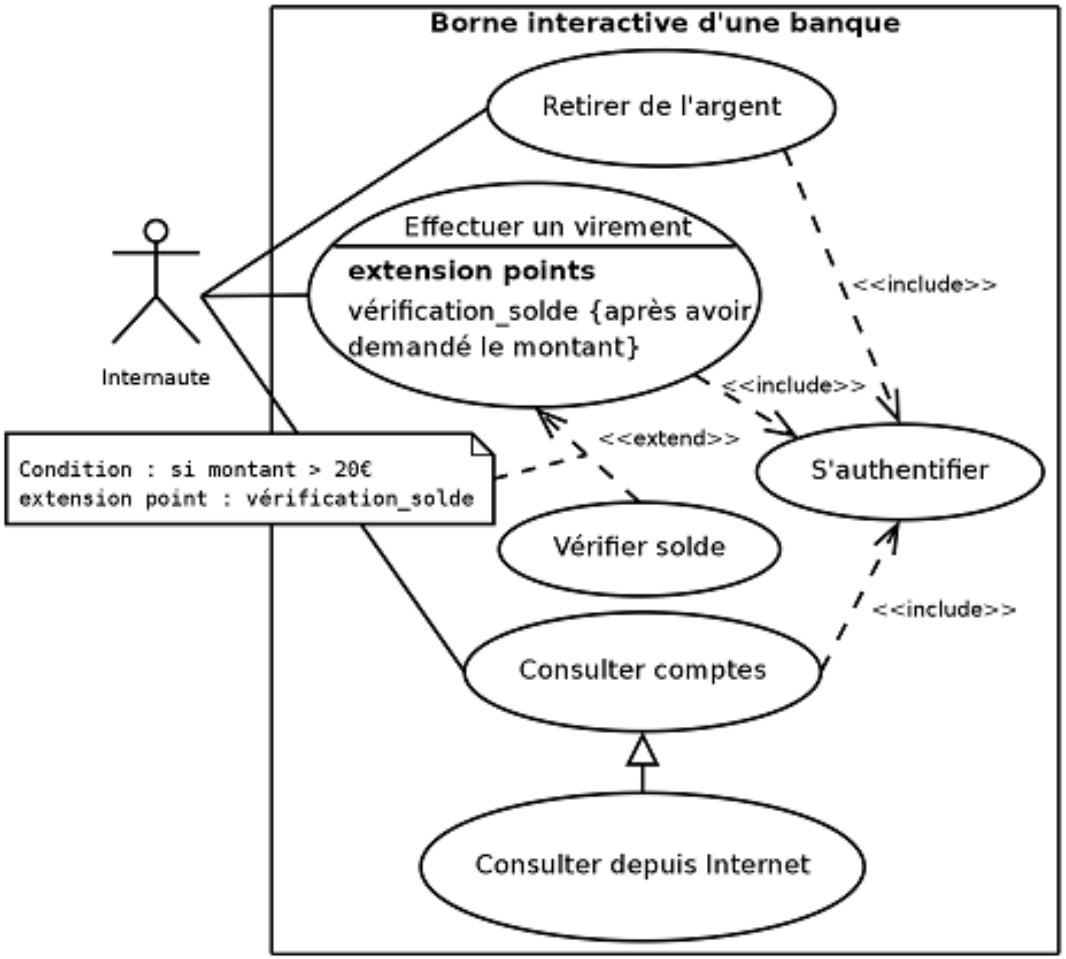
Relations entre cas d'utilisation

- Deux types de relations :
  - Les dépendances stéréotypées
    - La relation d'inclusion
    - La relation d'extension
  - La généralisation/spécialisation

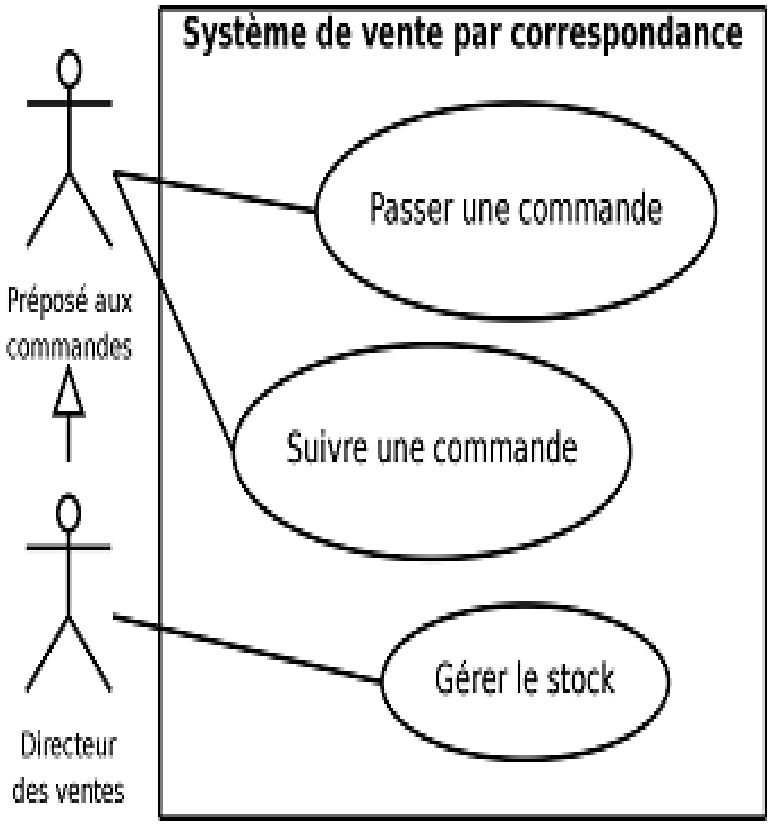
Acteur

- L’acteur est dit « principal » pour un cas d’utilisation lorsque le cas d’utilisation rend service à cet acteur.
- Les autres acteurs sont dits « secondaires ».
- Un acteur principal obtient un résultat observable du système tandis qu’un acteur secondaire est sollicité pour des informations complémentaires.

- Exemple:


Borne interactive d'une banque
Retirer de l'argent
Effectuer un virement
extension points
vérification_solde {après avoir demandé le montant}
<<include>>
<<include>>
<<extend>>
Internaute
Condition : si montant > 20€
extension point : vérification_solde
S'authentifier
Vérifier solde
<<include>>
Consulter comptes
Consulter depuis Internet

- **Relations entre acteurs**
  - La seule relation possible entre deux acteurs est la généralisation : un acteur A est une généralisation d'un acteur B si l'acteur A peut être substitué par l'acteur B (tous les cas d'utilisation accessibles à A le sont aussi à B, mais l'inverse n'est pas vrai).

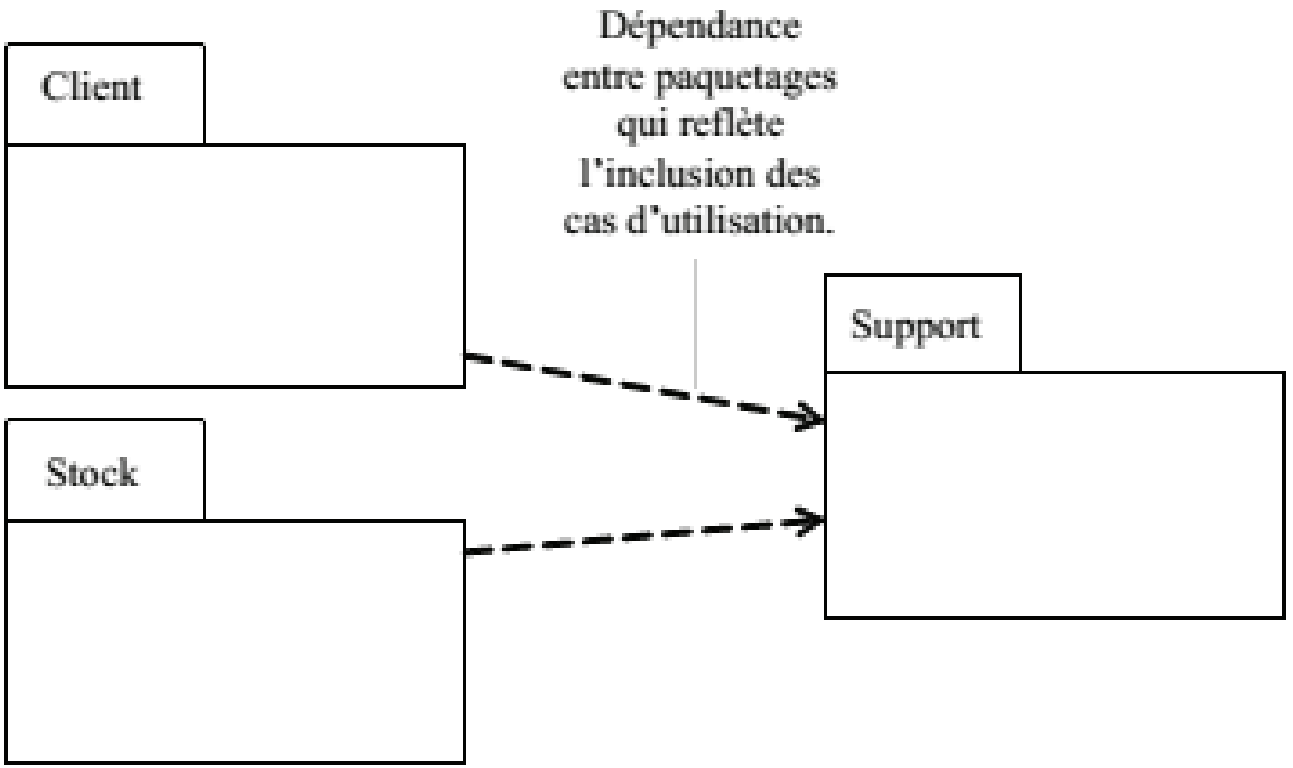


Regroupement des cas d'utilisation en paquetages

- **Définition:** Un **paquetage** permet d'organiser des éléments de modélisation en groupe.

  Un paquetage peut contenir des classes, des cas d'utilisations, des interfaces, etc.
- UML permet de regrouper des cas d'utilisation dans un « paquetage ».
- Le regroupement peut se faire par acteur ou par domaine fonctionnel.
- Un diagramme de cas d'utilisation peut contenir plusieurs paquetages et des paquetages peuvent être inclus dans d'autres paquetages.

Exemple:

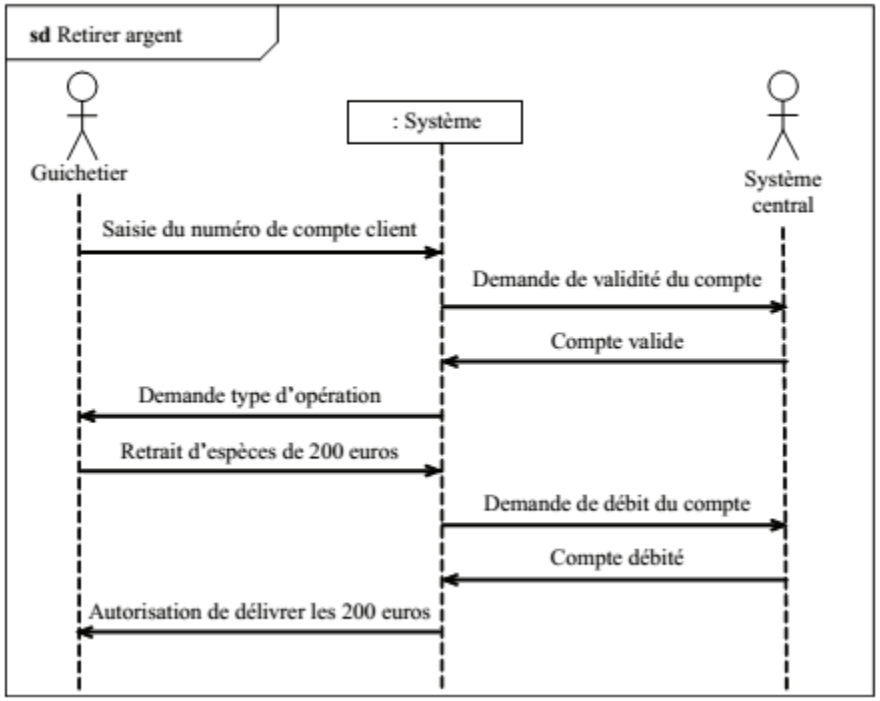

Dépendance
entre paquetages
qui reflète
l'inclusion des
cas d'utilisation.
Client
Support
Stock

Espace de noms

- UML est soumis à des règles de nommage qu'il faut strictement respecter.
- Pour accéder au contenu de paquetages imbriqués les uns dans les autres, il faut utiliser les **deux-points** comme séparateur des noms de paquetage. Par exemple, si un paquetage B inclus dans un paquetage A contient une classe X, il faut écrire **A::B::X** pour pouvoir utiliser la classe X en dehors du contexte du paquetage B.

Description des cas d'utilisation

- Diagramme de séquence, activités, …



- Description textuelle …

Conclusion

- Le diagramme de cas d'utilisation est un premier modèle d'un système.
- L'objectif de cette phase de la modélisation est d'identifier les frontières du système et les interfaces qu'il doit offrir à l'utilisateur.
- Le système est encore une boîte noire à l'architecture et au mode de fonctionnement interne inconnus.
- En revanche, son interface avec le monde qui l'entoure est partiellement connue.
- Modéliser la structure interne d'un système: diagramme de classes.

2. Diagramme de classes

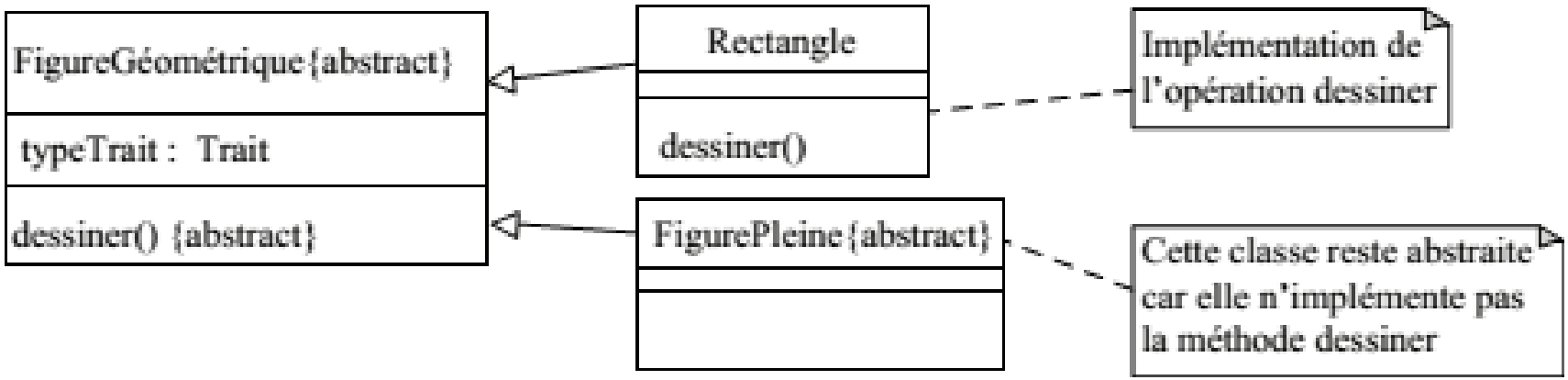
- **Définition:** Une **classe** est une description d'un ensemble d'objets ayant une sémantique, des attributs, des méthodes et des relations en commun.
- Un objet est une instance d'une classe.
- **Instance** plus générale que **Objet**

Classes et méthodes abstraites

Définitions:

- Une méthode est dite abstraite lorsqu'on connaît son entête mais pas la manière dont elle peut être réalisée.
- Une classe est dite abstraite lorsqu'elle définit au moins une méthode abstraite ou lorsqu'une classe parent contient une méthode abstraite non encore réalisée.

- **Exemple:**


FigureGéométrique{abstract}
typeTrait : Trait
dessiner() {abstract}
Rectangle
dessiner()
Implémentation de l'opération dessiner
FigurePleine{abstract}
Cette classe reste abstraite car elle n'implémente pas la méthode dessiner

Nom de la classe

- Les informations les plus souvent utilisées sont :
  - le nom de la classe,
  - Les paquetages qui la contiennent,
  - le stéréotype éventuel de la classe,
  - l'auteur de la modélisation,
  - la date,
  - le statut de la classe (elle est validée ou pas).
  - Abstraite ou concrète : par défaut, une classe est considérée comme concrète. Sinon, ajoutez le mot-clé **abstract** pour indiquer qu'elle est abstraite.
- Le nom de la classe doit évoquer le concept décrit par la classe.
- Il commence par une majuscule. Quand le nom est composé, chaque mot commence par une majuscule et les espaces blancs sont éliminés.

Encapsulation

- Principe du coffre-fort
- Définir les droits d'accès aux propriétés d'un classeur
- 4 niveaux d'encapsulation d'une propriété d'une classe: publique, privé, protégé, visible dans le paquetage
- Les modificateurs d'accès ou de visibilité
  - Le mot-clé **public** ou le caractère **+**.Propriété visible partout
  - **Aucun caractère**, **ni mot-clé**. Propriété visible uniquement dans le paquetage où la classe est définie
  - Le mot-clé **protected** ou le caractère **#**. Propriété visible dans la classe et par tous ses descendants.
  - Le mot-clé **private** ou le caractère **-**. Propriété visible uniquement dans la classe.

Attributs

- Représentent les données encapsulées dans les objets.
- Un attribut est défini par un nom, un type de données et une visibilité.
- Le nom de l'attribut doit être unique dans la classe.
- Syntaxe de la déclaration d'un attribut:
  <modificateur d'accès> [/]<NomAttribut> :
  <NomClasse>['['multiplicité']'] [= valeur(s) initiale(s)]

Les attributs de classes

- Parfois, il est nécessaire de définir un attribut qui garde une valeur unique et partagée par toutes les instances de la classe.
- Les instances ont accès à cet attribut, mais n'en possèdent pas une copie.
- un attribut de classe est souligné
- En Java, comme en C++, un attribut de classe s'accompagne du mot-clé static.

Les attributs dérivés

- Symbolisés par l'ajout d'un slash (**/**) devant leur nom
- Peuvent être calculés à partir d'autres attributs et des formules de calcul
- Exemple: Modélisation de la classe Personne
  - l'âge est calculé à partir de la date de naissance et la date du jour
  - On note: /âge : integer

Méthodes et Opérations

- Le comportement d'un objet est modélisé par un ensemble de méthodes.
- Chaque méthode correspond à une implémentation bien précise d'un comportement.
- UML distingue la spécification d'une méthode, qui correspond à la définition de son entête, de son implémentation.
- La spécification est appelée « opération » et l'implémentation est appelée « méthode ».
- On peut associer plusieurs méthodes à une opération.

Méthodes et Opérations

- Dans une classe, une opération (même nom et mêmes types de paramètres) doit être unique.
- Quand le nom d'une opération apparaît plusieurs fois avec des paramètres différents, on dit que l'opération est **surchargée**.
- Il est impossible que deux opérations ne se distinguent que par la valeur retournée.
- Syntaxe:

  <modificateur d'accès><nomDeLaMéthode ([paramètres])> : [<valeurRenvoyée>][{propriété}]

Opérations de classe

- Comme pour les attributs de classe, il est possible de définir une opération qui ne dépend pas des valeurs propres de chaque objet, mais qui porte sur les attributs de la classe ou uniquement sur les paramètres de l'opération.
- L'opération devient propriété de la classe, et non de ses instances.
- Elle n'a pas accès aux attributs des objets de la classe.
- Graphiquement, une méthode de classe est soulignée.

Compartiments complémentaires d'une classe

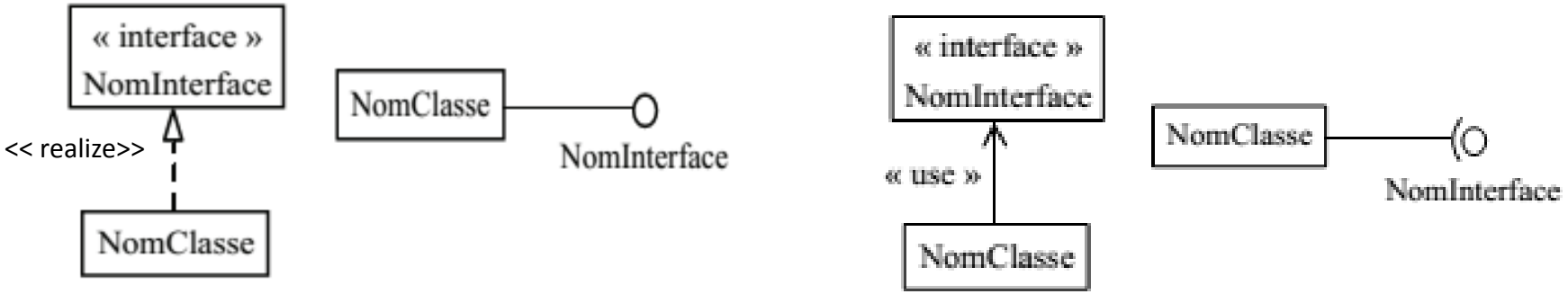
- Classe: construction incrémentale, évolutive
- Informations complémentaires:
  - les responsabilités,
  - les contraintes générales,
  - les exceptions, etc.
- 2 nouveaux compartiments: le compartiment des responsabilités et celui des exceptions.
  - Disparaîtront à la phase de génération de code ; ils seront transformés en un ensemble d'attributs et de méthodes.

Interfaces

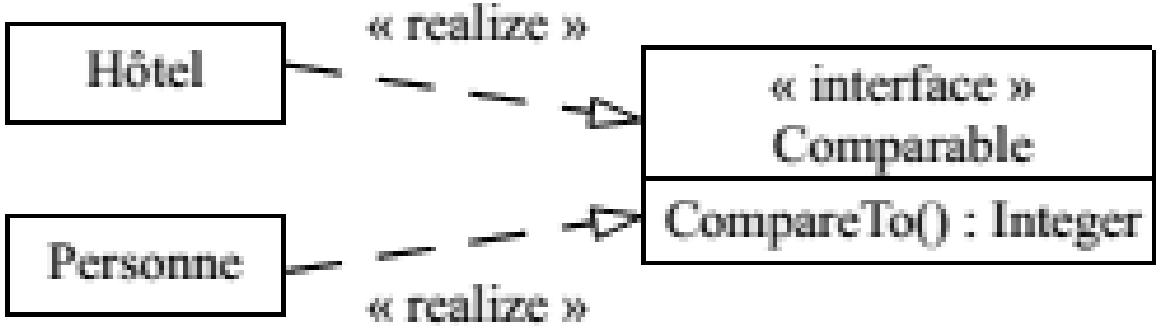
- But: regrouper un ensemble d'opérations assurant un service cohérent offert par une classe
- Utilisée pour classer les opérations en catégories sans préciser la manière dont elles sont implémentées
- Contrairement à une classe, une interface ne spécifie pas une structure interne et ne précise pas les algorithmes permettant la réalisation de ses méthodes.
- N'a pas d'instances directes
- Pour être utilisée, elle doit être réalisée par une classe

Interfaces

- Graphiquement:



- Exemple:



Relations entre classes

- Les relations entre classes expriment les liens sémantiques ou structurels.
- Les relations les plus utilisées sont :
  - l'association,
  - l'agrégation,
  - la composition,
  - la dépendance et
  - l'héritage.

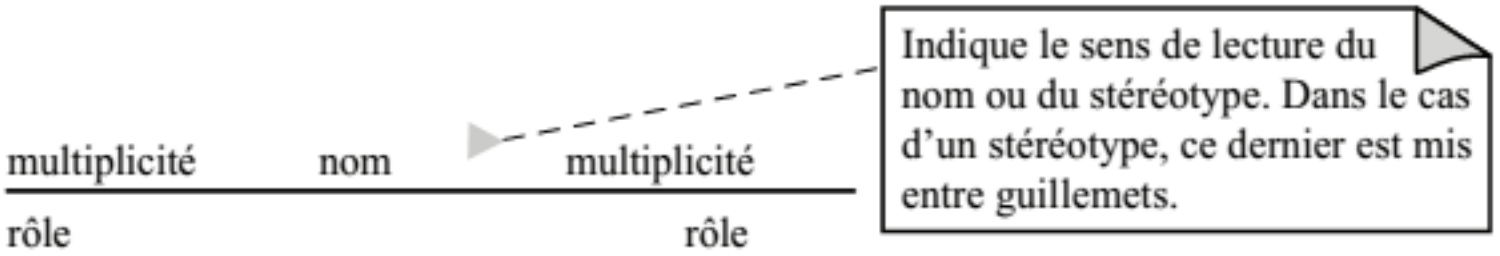
Multiplicité

- Permet le contrôle du nombre d'objets intervenant dans chaque instance de la relation.
- Les principales multiplicités normalisées sont :
  - « plusieurs » (*),
  - « exactement n » (n),
  - « au minimum n » (n..*) et
  - « entre n et m » (n..m).

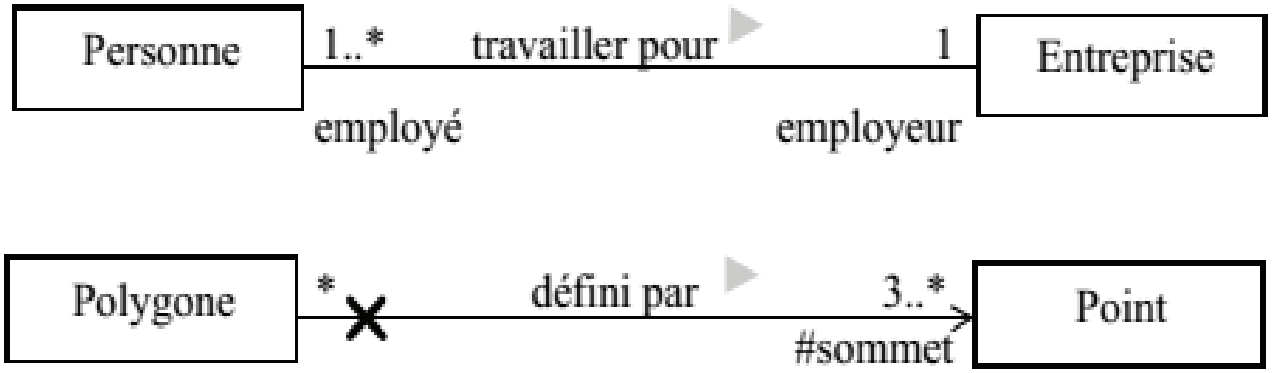
Association

- Une association représente une relation sémantique entre les objets d'une classe.
- Elle est représentée graphiquement par un trait plein entre les classes associées.
- Elle est complétée par un nom qui correspond souvent à un verbe à l'infinitif, avec une précision du sens de lecture en cas d'ambiguïté.
- Parfois, un stéréotype qui caractérise au mieux l'association remplace le nom.
- Chaque extrémité de l'association indique le rôle de la classe dans la relation et précise le nombre d'objets de la classe qui interviennent dans l'association.
- Quand l'information circule dans un sens uniquement, le sens de la navigation est indiqué à l'une des extrémités.
- Il est possible d'exprimer la non-navigabilité d'une extrémité par l'ajout d'une croix sur l'association

Association

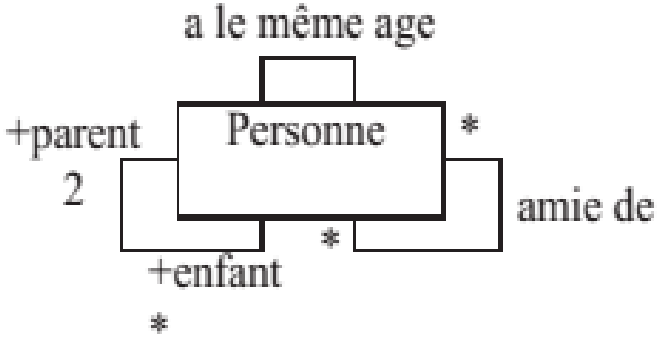


- Exemples:

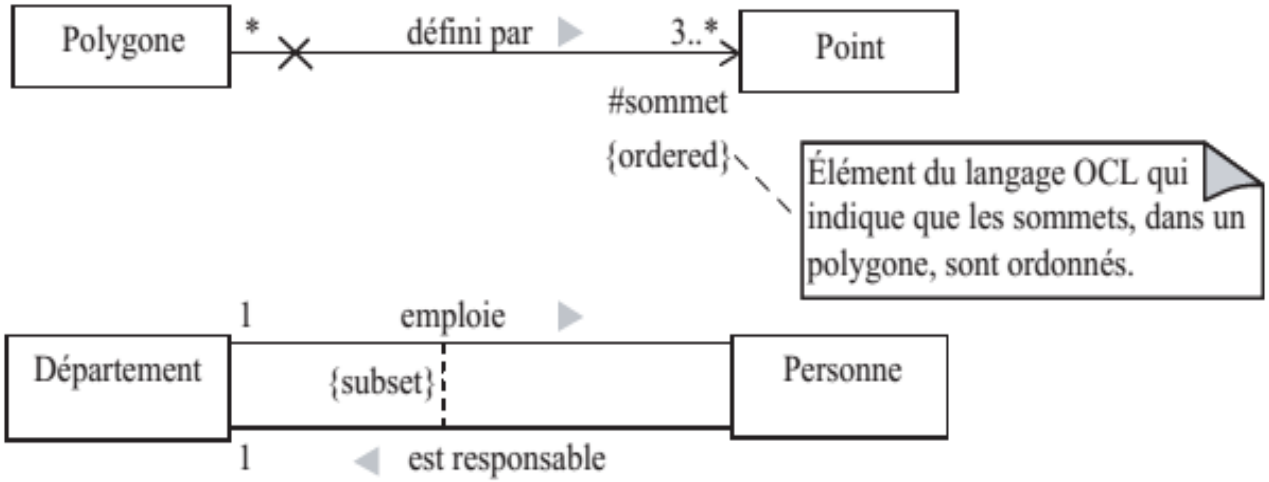


Association

- Exemples d'associations réflexives :



- Exemple d'association avec contraintes OCL:

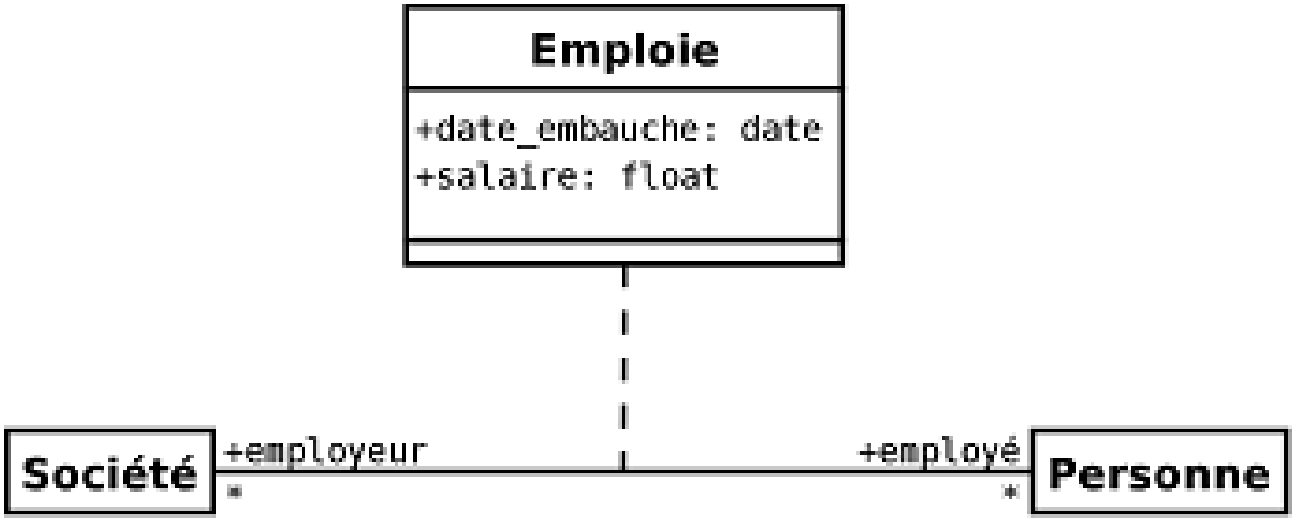


Classe-Association

- Une association peut être raffinée et avoir ses propres propriétés, qui ne sont disponibles dans aucune des classes qu'elle lie. Comme, dans le modèle objet, seules les classes peuvent avoir des propriétés, cette association devient alors une classe appelée « classe-association ».

Classe-Association

- Exemple:

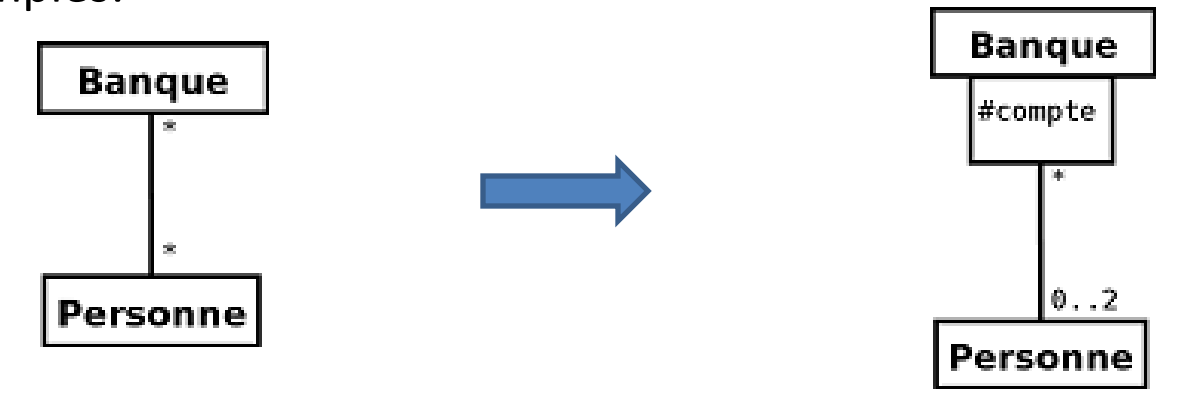


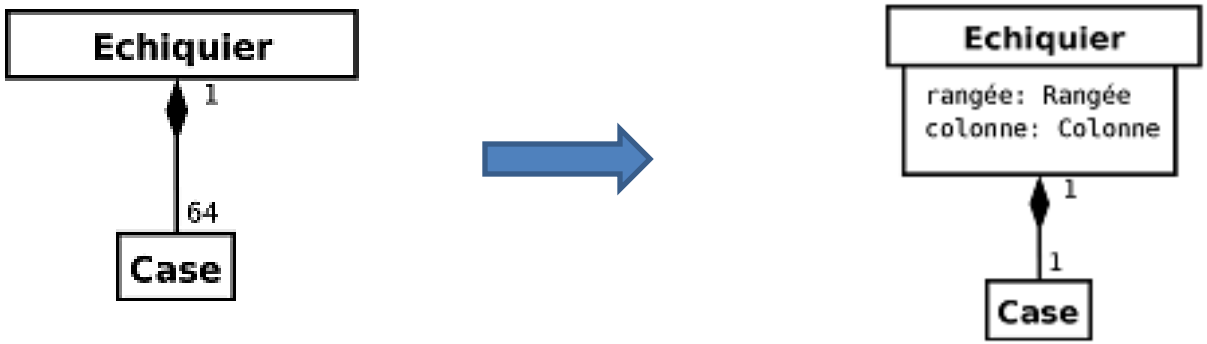
Association qualifiée

- Restreindre la portée de l'association à quelques éléments ciblés (comme un ou plusieurs attributs) de la classe.
- Ces éléments ciblés sont appelés qualificatif.
- Un qualificatif permet donc de sélectionner un ou des objets dans le jeu des objets d'un objet (appelé objet qualifié) relié par une association à un autre objet.
- L'objet sélectionné par la valeur du qualificatif est appelé objet cible.
- L'association est appelée association qualifiée.
- Un qualificatif agit toujours sur une association dont la multiplicité est plusieurs (avant que l'association ne soit qualifiée) du côté cible.

Association qualifiée

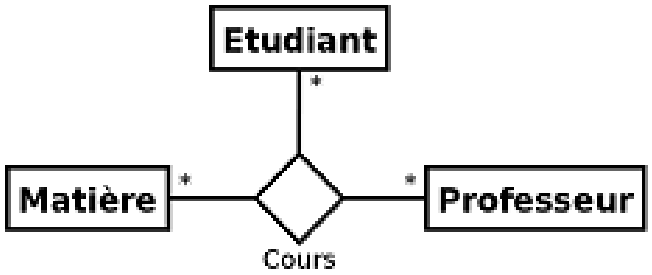
- Exemples:





Association n-aire

- Une association n-aire lie plus de deux classes
- On représente une association n-aire par un grand losange avec un chemin partant vers chaque classe participante
- Le nom de l'association apparaît à proximité du losange.
- Exemple:



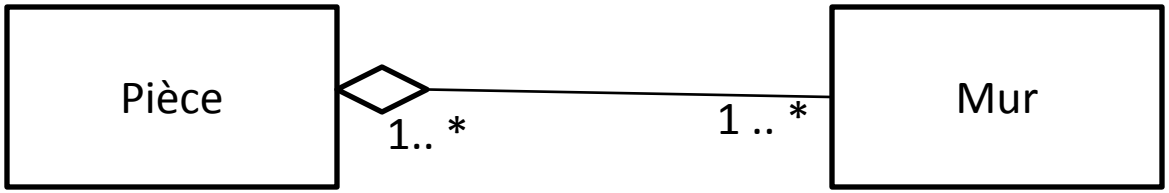
Relation d'agrégation

- Relation structurelle entre deux classes de même niveau conceptuel
- Modélise une relation tout/partie: une classe constitue un élément plus grand (tout) composé d'éléments plus petit (partie)
- Une agrégation est une association qui représente une relation d'inclusion structurelle ou comportementale d'un élément dans un ensemble
- Graphiquement, on ajoute un losange vide du côté de l'agrégat
- Contrairement à une association simple, l'agrégation est transitive.

Relation d'agrégation

- Elle ne contraint pas la navigabilité ou les multiplicités de l'association.
- Elle n'entraîne pas de contraintes sur la durée de vie des parties par rapport au tout.

- Exemple:

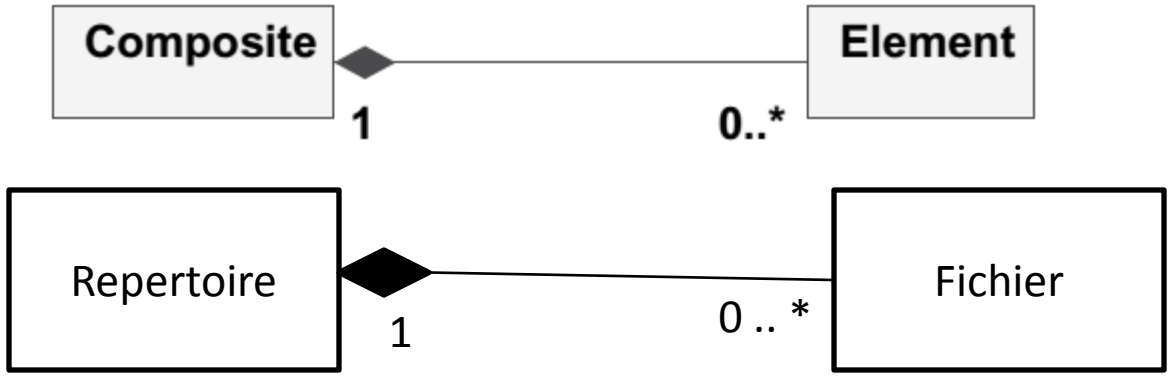


Relation de composition

- Appelée également « agrégation composite »
- Décrit une contenance structurelle entre instances.
- L'élément composite est responsable de la création, de la copie et de la destruction de ses composants.
- La destruction ou la copie de l'objet composite implique respectivement la destruction ou la copie de ses composants.
- Une instance de la partie appartient toujours à au plus une instance de l'élément composite.

Relation de composition

- Graphiquement, : Une composition se distingue d'une association par l'ajout d'un losange plein du côté du composite.
- La multiplicité du côté composite ne doit pas être supérieure à 1
- Exemple:

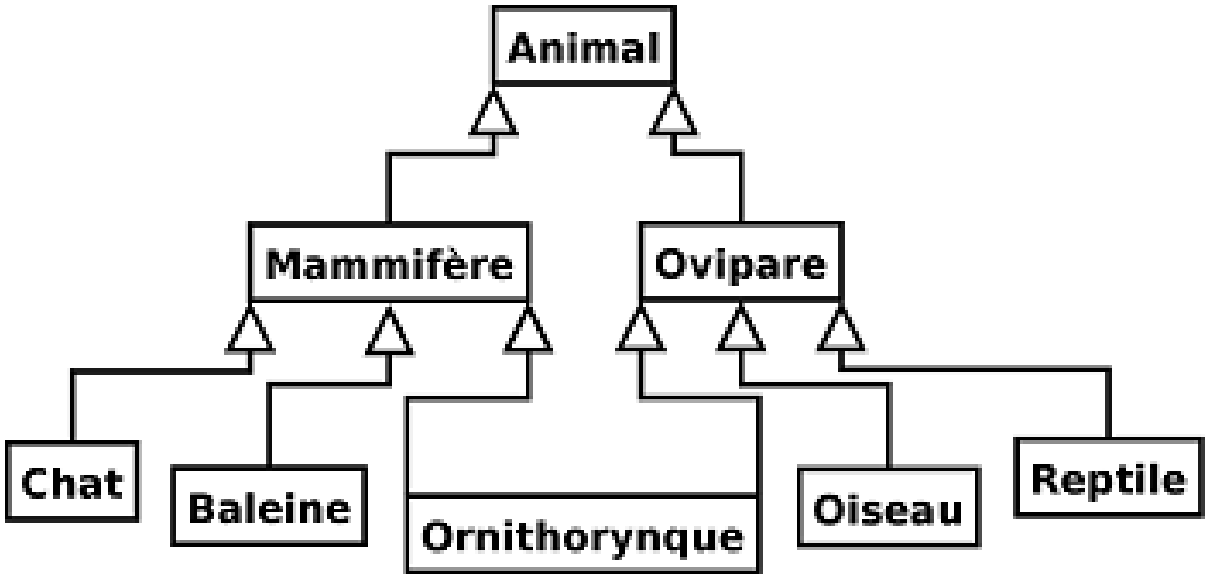


Relation d’héritage

- La généralisation décrit une relation entre une classe générale (classe de base ou classe parent) et une classe spécialisée (sous-classe).
- La classe spécialisée est intégralement cohérente avec la classe de base, mais comporte des informations supplémentaires (attributs, opérations, associations).
- En UML: relation d’héritage

Relation d'héritage

- Exemple:



Relation d'héritage
Caractéristiques

- La classe enfant possède toutes les caractéristiques de ses classes parents, mais elle ne peut accéder aux caractéristiques privées de ces dernières.
- Une classe enfant peut redéfinir (même signature) une ou plusieurs méthodes de la classe parent.
  - Sauf indication contraire, un objet utilise les opérations les plus spécialisées dans la hiérarchie des classes.
- Toutes les associations de la classe parent s'appliquent aux classes dérivées.

Relation d'héritage Caractéristiques

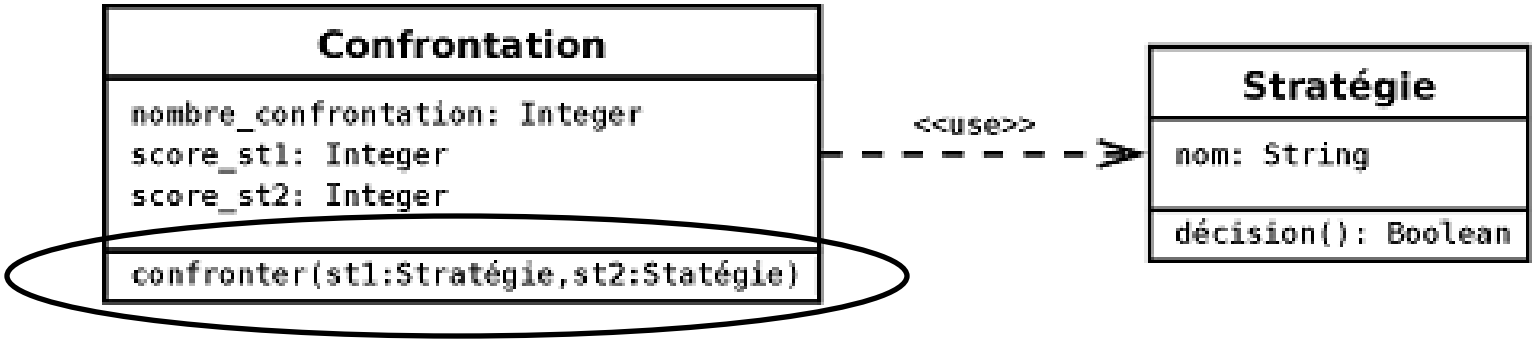
- Une instance d'une classe peut être utilisée partout où une instance de sa classe parent est attendue. Par exemple, en se basant sur le diagramme précédent, toute opération acceptant un objet d'une classe Animal doit accepter un objet de la classe Chat.
- Une classe peut avoir plusieurs parents, on parle alors d'héritage multiple.
  - Implémentation possible en C++ et pas en Java.
- La relation d'héritage n'est pas propre aux classes, elle s'applique à d'autre éléments du langage UML comme les paquetages, les acteurs ou les cas d'utilisation.

Relation de dépendance

- Relation unidirectionnelle exprimant une dépendance sémantique entre des éléments du modèle.
- Représentée par un trait discontinu orienté.
- Indique que la modification de la cible peut impliquer une modification de la source.
- Souvent stéréotypée pour mieux expliciter le lien sémantique entre les éléments du modèle

Relation de dépendance

- On utilise souvent une dépendance quand une classe en utilise une autre comme argument dans la signature d'une opération.
- Exemple:



Conclusion

- Le diagramme de classes montre un aspect structurel (approche objet) du système, tandis que celui des cas d'utilisation montre un côté fonctionnel.
- D'autres aspects: autres modèles (autres diagrammes).
- Diagramme de classes: modèle dominant dans l'Ingénierie Dirigée par les Modèles.