

Master mention Informatique
Spécialité ISI

Génie Logiciel et Modélisation
M1 / 177UD01

C2 – Les diagrammes de classes

Claudine Piau-Toffolon

Plan du cours

- Les diagrammes de classes
 - Définition
 - Les attributs
 - Les opérations
 - Les associations
 - Arité, multiplicité, nommage, rôle, etc.
 - Les agrégations
 - Les compositions
 - La généralisation
 - Les relations de dépendances
 - Autres types de classe
 - Exemples

Les diagrammes de classes - Définition

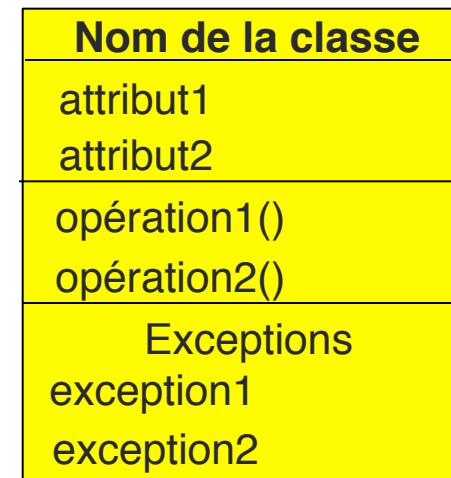
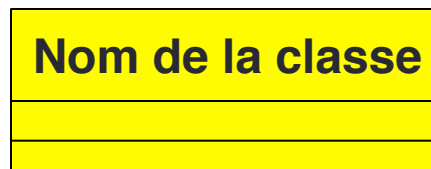
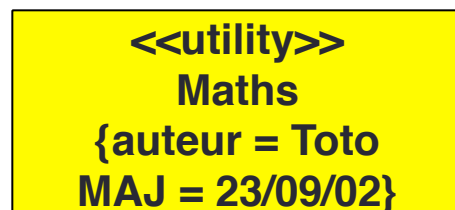
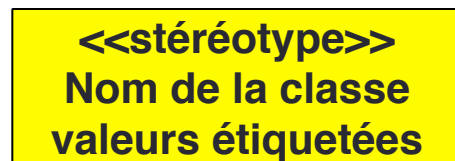
- But :
 - Un diagramme de classe permet de modéliser deux aspects structurels d'un système :
 - les *classes d'objet du domaine* et de leurs relations (niveau analyse)
 - les *classes d'objets internes* et de leurs relations (niveau conception)
 - Structure : packages + classes + relations entre classes de même package ou de packages différents
- Sémantique :
 - Une classe décrit un ensemble d'objets de même nature (*les instances de la classe*)
 - Objets du domaine : représentations simplifiées du monde réel
 - Objets internes : objets purement informatiques sans contreparties réelles
 - Les relations sont modélisées par des associations de différents types
 - Une association décrit un ensemble de lien (*instances de l'association*) entre les objets des classes des extrémités de l'association (binaire, ternaire, n-aire)

Les diagrammes de classes - Utilisation

- Durant l'analyse :
 - Identifier et préciser les concepts du domaine et leurs relations
- Durant la conception :
 - Création d'une solution informatique
 - « Informatisation » des classes du domaine
 - Simplification/modification de ces classes
 - « Intégration » de ces classes avec les classes internes
 - Attribution des responsabilités de chaque classe
 - Choix des conteneurs (liste, tableau, arbre, table de hachage)
 - Utilisation de bibliothèques existantes
 - Applications de *design pattern*
 - Mise en œuvre de framework(s)
 - Structuration en package : architecture logicielle
 - Base pour les diagrammes de composants et de déploiement
- Durant la programmation :
 - Génération automatique des classes du système dans un langage cible (Java, C++, C#...)
 - Création des diagrammes de classe par *reverse engineering*

Les diagrammes de classes – La classe

- Dans le méta-modèle UML, une classe est un élément de classification
 - Remarque : un acteur est une classe (*un acteur peut être dans un diag. de classe*)
- Une classe est représentée par un rectangle et comprend plusieurs compartiments :
 - un *nom* (obligatoire) écrit en caractère gras avec la possibilité de mentionner le paquetage d'origine :
Nom du paquetage :: Nom de la classe
 - un stéréotype optionnel qui spécifie/personnalise la classe
 - une liste optionnelle de valeurs étiquetées (liste attribut valeur)
 - une suite optionnelle d'*attributs*
 - une suite optionnelle d'*opérations*
 - des compartiments optionnels nommés (signaux générés, exceptions traitées...)



Diagrammes de classes – Les attributs (1/2)

- Syntaxe :

[Visibilité] nom_attribut ['[' Multiplicité ']'] ':' Type_attribut [= Valeur_initiale]

avec

Visibilité ::= '+' | '#' | '-' *(signifie respectivement public, protégé et privé)*

Multiplicité ::= (Intervalle | Nombre) [',' Multiplicité]

Intervalle ::= Limite_Inférieure '..' Limite_Supérieure

Limite_Inférieure ::= *entier_positif* | 0

Limite_Supérieure ::= Nombre

Nombre ::= *entier_positif* | '*' *(* signifie illimité)*

Type_Attribut ::= (*classe* | *type_primitif* | *expression_non-précisée_par_UML*)
['{' Mutabilité '}']

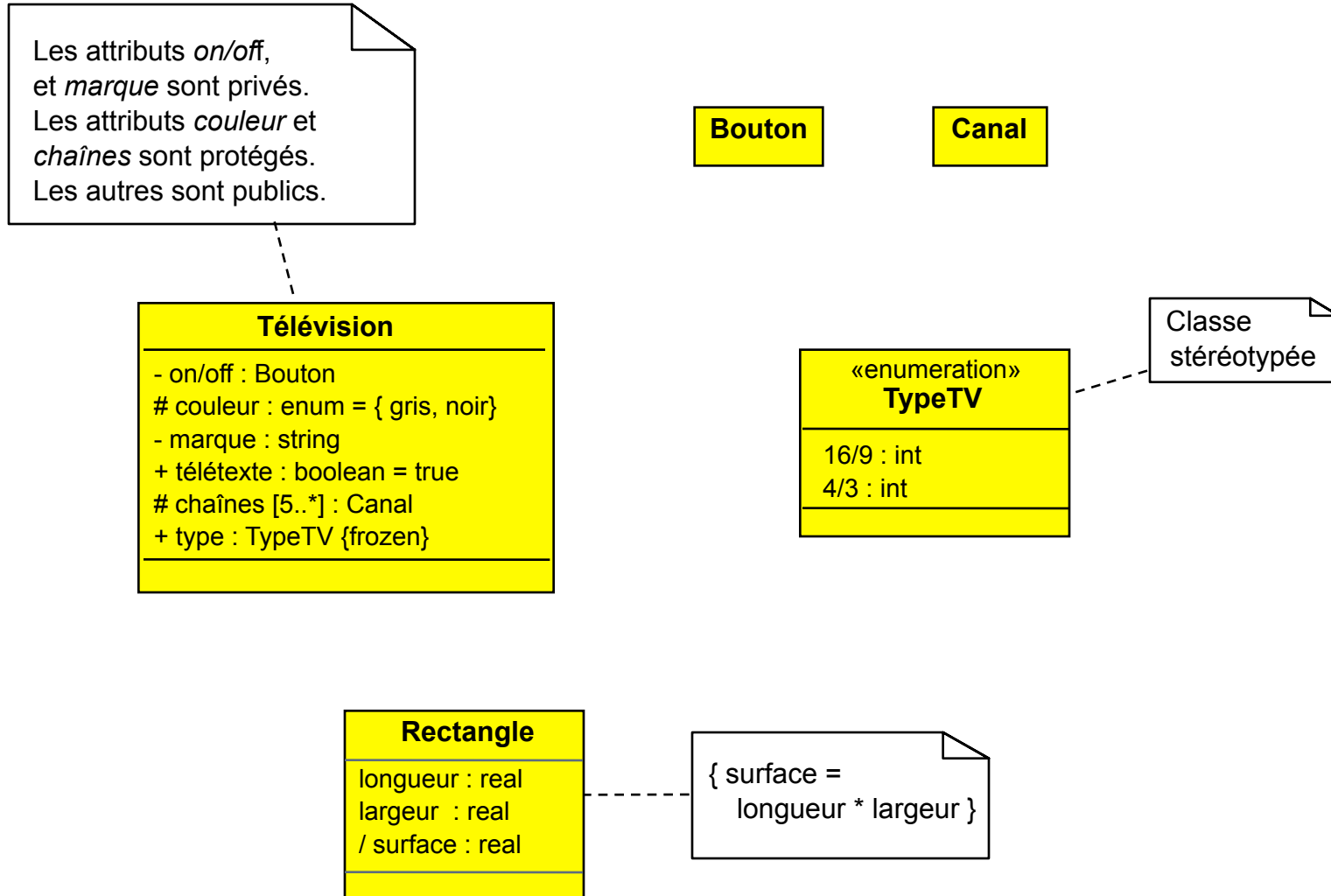
Mutabilité ::= '*frozen*' | '*changeable*' | '*addOnly*'

Valeur_initiale ::= *la_valeur_initiale_de_l'attribut*

- Attribut dérivé :

- Attribut redondant pouvant être calculé à partir des autres attributs
- Indiqué par le caractère / avant le nom de l'attribut
- A la conception un attribut dérivé devient un attribut normal ou une méthode

Diagrammes de classes – Les attributs (2/2)



Diagrammes de classes – Les opérations (1/2)

- Syntaxe

[Visibilité] Nom_Opération '(' [Arguments] ')' [':' Type_Retourné] ['{' Propriété '}']

avec

Visibilité ::= '+' | '#' | '-'

Arguments ::= [Direction] Nom_Argument ':' Type_Argument ['=' Valeur_Par_Défaut] [',' Arguments]

Direction ::= 'in' | 'out' | 'inout'

(signifie respectivement en entrée seule non-modifiable, en sortie seule, en entrée modifiable)

Type_Retourné : précise le type (ou une liste de type) retourné par l'opération

Propriété : cinq propriétés d'une opération (par défaut à *true*)

'isQuery' ['=' 'true' | 'false'] : l'opération n'altère pas l'état de l'instance

'abstract' ['=' 'true' | 'false'] : l'opération est non-implémentée

'leaf' ['=' 'true' | 'false'] : l'opération ne peut pas être surchargée

'root' ['=' 'true' | 'false'] : l'opération est définie pour la première fois

concurrency ::= 'sequential' | 'guarded' | 'concurrent'

'sequential' : appels simultanés non-prévus

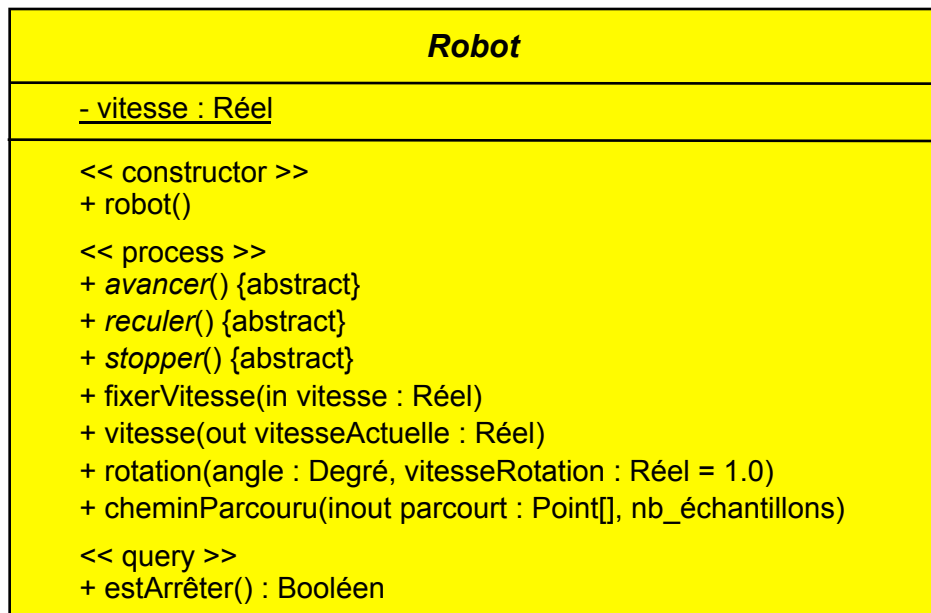
'guarded' : appels simultanés traités en séquence

'concurrent' : appels simultanés traités en parallèle

- Portée des attributs et des opérations :

- Par défaut, la portée d'un attribut ou d'une opération est celle de l'instance
- La portée peut être celle de la classe (attribut ou opération notée en soulignée)

Diagrammes de classes – Les opérations (2/2)



La classe *Robot* est une classe abstraite (stéréotype ou noté italique)
Les opérations *avancer*, *reculer* et *stopper* sont abstraites.

L'opération *fixerVitesse* a un argument en entrée de type Réel.

L'opération *vitesse* a un argument en sortie de type Réel.

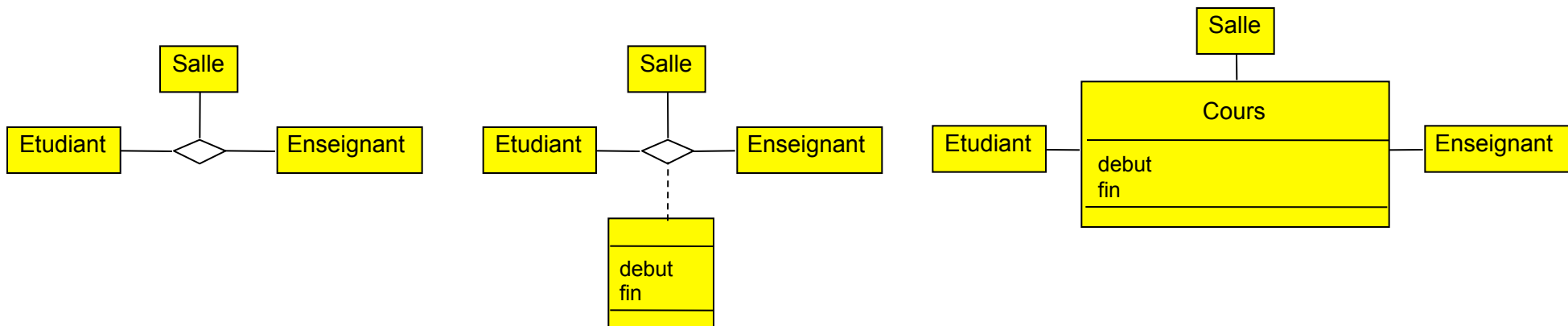
L'opération *rotation* possède deux arguments dont le second a une valeur par défaut.

L'opération *estArrêter* retourne une valeur de type booléen, les autres opérations ne retournent rien.

L'attribut *vitesse* est un attribut de classe.

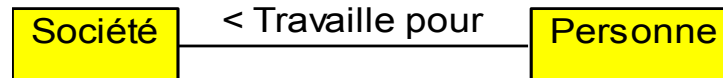
Diagrammes de classes – Les associations (1/7)

- Une association symbolise une relation structurelle entre classes d'objets : elle informe de l'existence d'un lien sémantique durable entre les objets instances des classes associées
- Arité des associations :
 - Généralement, une association est binaire. Elle est figurée par une ligne continue
 - Pour représenter une association n-aire, on utilise un losange
- On peut également avoir recours à une **classe-association** lorsque l'association possède des attributs
- Attention aux redondances entre attribut et association !

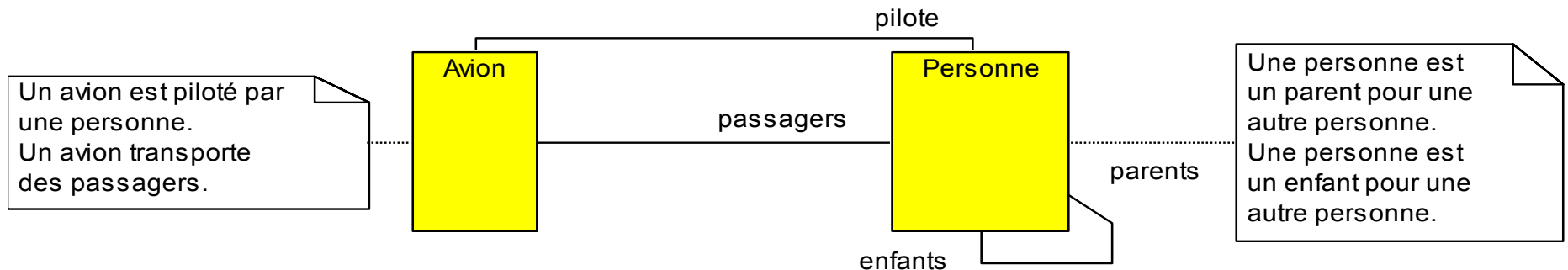


Diagrammes de classes – Les associations (2/7)

- Nommage des associations :
 - Nommer une association permet de préciser la sémantique du lien entre les classes de l'association
 - Par convention, on utilise une forme verbale
 - Une association anonyme n'a aucun intérêt d'un point de vue représentation
 - La lecture se fait par défaut de gauche à droite mais elle peut être imposée en utilisant les symboles < et > ou ◀ et ▶

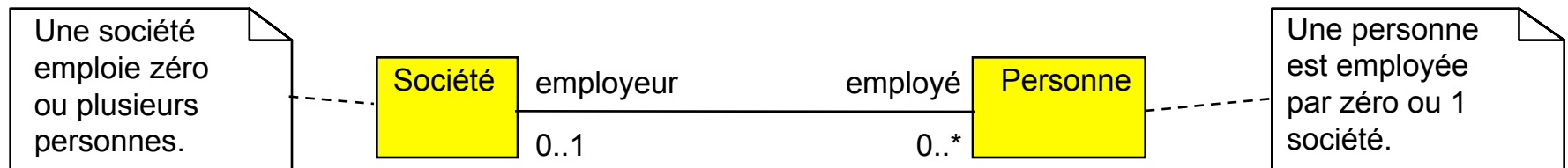


- Rôles des extrémités d'une association :
 - Un rôle (côté cible) décrit comment la classe source voit la classe cible
 - Un rôle est un attribut de la classe source mais figuré sur l'extrémité opposé
 - Une association peut être réflexive (lien entre objets de même classe)



Diagrammes de classes – Les associations (3/7)

- Multiplicité des associations :
 - Les extrémités peuvent porter une indication de **multiplicité**
 - La multiplicité indique le nombre d'objets de la classe cible avec lesquels un objet de la classe source peut être associé simultanément

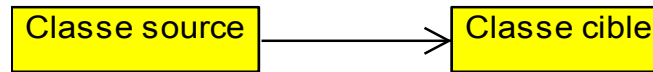


- UML définit les multiplicités suivantes :
 - 1 : un et un seul (*multiplicité par défaut*)
 - 0..1 : zéro ou 1
 - N : exactement N (*entier naturel*)
 - M..N : de M à N (*M et N entiers naturels et $M < N$*)
 - * : zéro ou plus
 - 0..* : zéro ou plus
 - 1..* : au moins 1

Diagrammes de classes – Les associations (4/7)

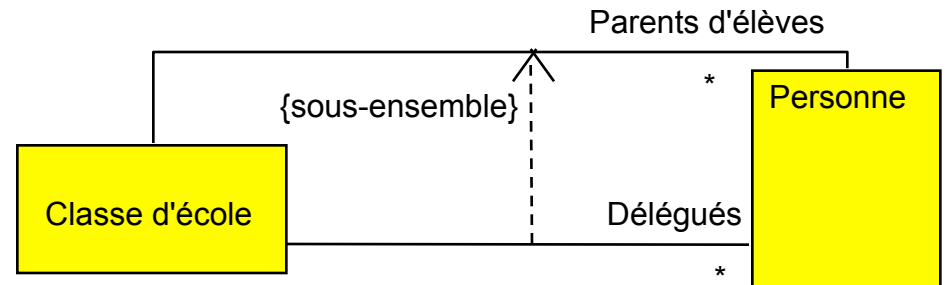
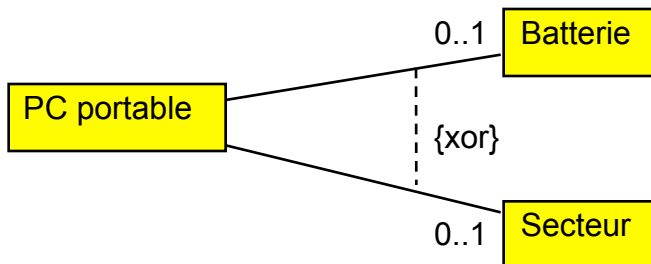
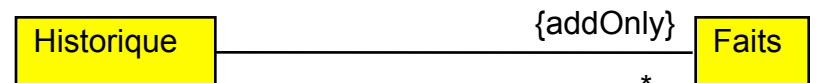
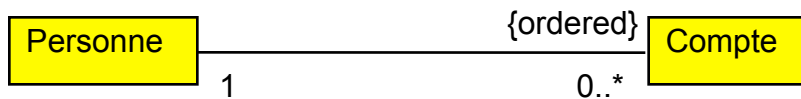
- Navigabilité sur les associations :

- Une association est normalement bidirectionnelle
- Une flèche permet de préciser que les objets de la source voient les objets de la classe cible mais pas l'inverse
- Cela permet de réduire les couplages entre classes ou de modéliser une dissymétrie des besoins de communication



- Contraintes sur les associations :

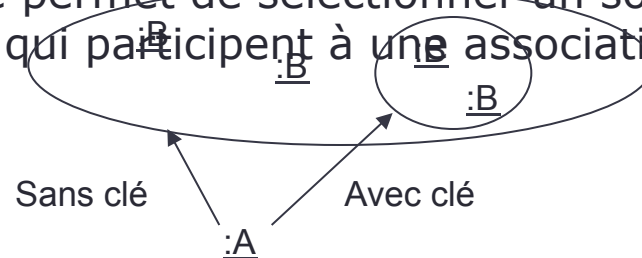
- Exprime des contraintes logiques sur ou entre les associations
- Les contraintes sont en langue naturelle ou en **OCL** (cf. dernier cours)
- Exemples :



Diagrammes de classes – Les associations (5/7)

- Qualification des associations :

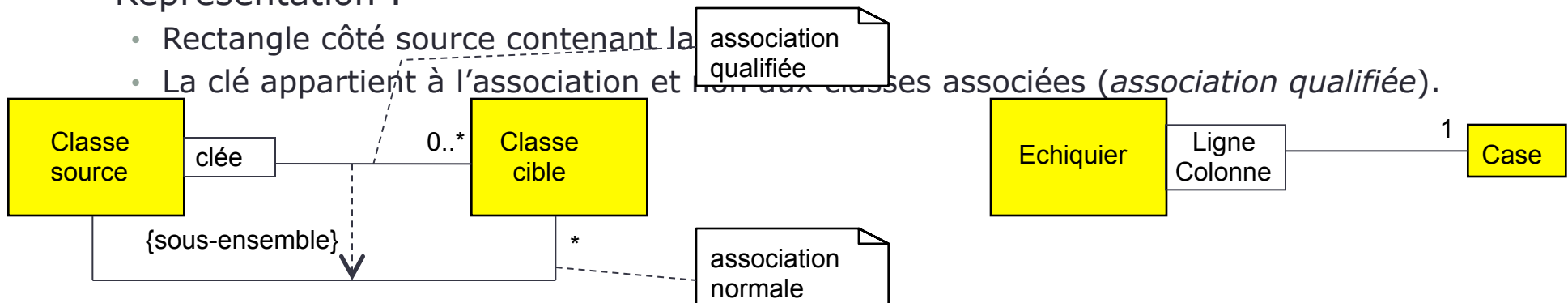
- La qualification par clé permet de sélectionner un sous-ensemble d'objets parmi l'ensemble des objets qui participent à une association



- La clé est formée d'un ou plusieurs attributs de la classe cible

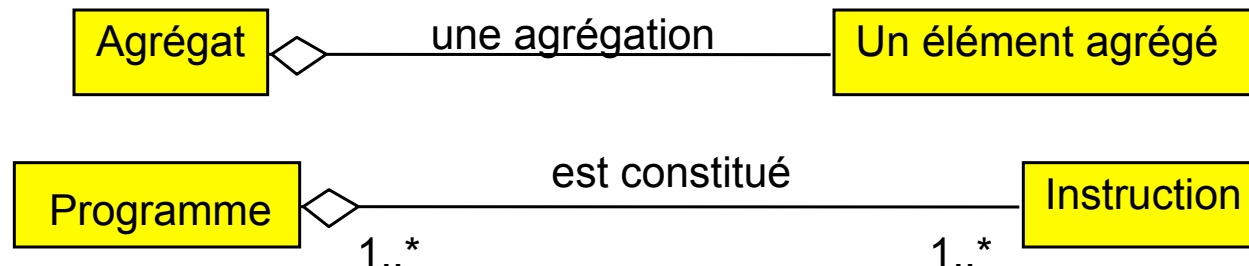
- Représentation :

- Rectangle côté source contenant la
- La clé appartient à l'association et non aux classes associées (*association qualifiée*).



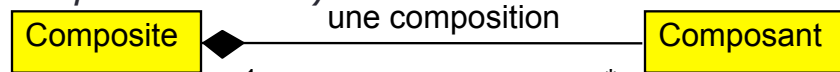
Diagrammes de classes – Les associations (6/7)

- Les agrégations :
 - Sémantique : association non-symétrique dans lequel une classe joue un rôle plus important par rapport à l'autre classe de l'association
 - L'agrégat est constitué d'agrégés
 - L'agrégation est une relation « tout/partie » faible
 - Les parties sont partageables ou échangeables
 - Exemple : un train (le *tout*) est constitué de wagons (les *parties*), mais ces wagons peuvent être utilisés pour former d'autres trains à un autre moment
 - Point de vue conception :
 - Les objets agrégés sont juste référencés par l'objet agrégat qui peut y accéder mais n'en est pas propriétaire
 - Les objets agrégés peuvent ainsi avoir une durée de vie différente de celle de l'objet agrégat
 - Représentation UML :

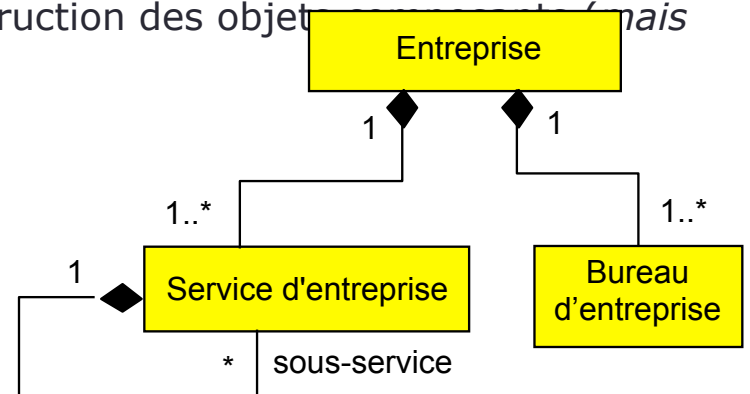
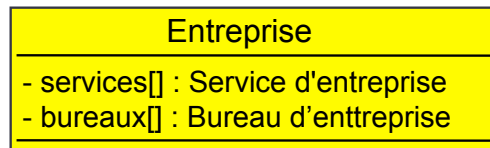


Diagrammes de classes – Les associations (7/7)

- Les compositions :
 - Sémantique : une composition est une association avec un couplage plus fort qu'une agrégation
 - Le composite est constituée de composants
 - La composition est une relation « tout/partie » forte
 - Les parties **ne sont pas** partageables ou échangeables
 - Exemple : les chambres (*les parties*) d'un hôtel (*le tout*) ne sont pas partageables ou échangeables entre plusieurs hôtels
 - Point de vue conception :
 - Les objets composants sont possédés par l'objet composite
 - La destruction de l'objet composite entraîne la destruction des objets composants (mais pas l'inverse).

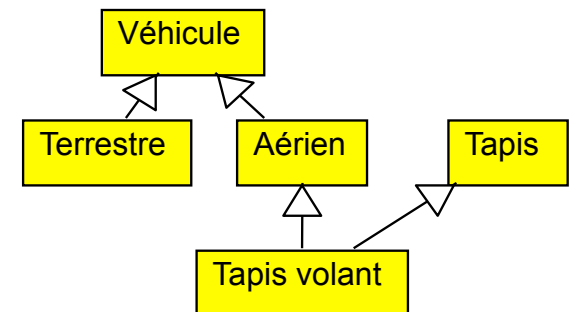
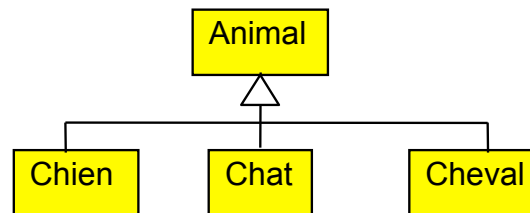
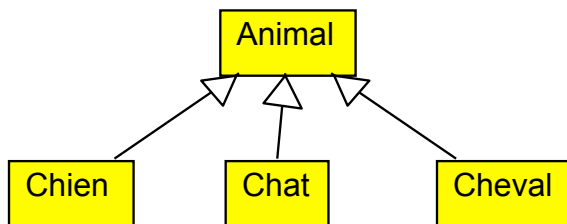


- Représentation UML :




Diagrammes de classes – La généralisation

- Généralisation : désigne une relation de classification entre un élément général et un élément spécifique
- Deux types de généralisation : simple ou multiple
 - La généralisation simple :
 - Cette relation binaire est du type « est une sorte de »
 - Les sous-classes **héritent des attributs, des opérations, des associations et des contraintes** définis dans la classe mère
 - La généralisation multiple :
 - Les sous-classes héritent simultanément de plusieurs classes mères
- Représentation :



Diagrammes de classes – Les relations de dépendances (1/2)

- Relation unidirectionnelle qui modélise une dépendance sémantique dans laquelle un changement de la cible implique un changement de la source 
- Elle est représentée par une ligne fléchée pointillée de la source vers la cible
- Elles s'appliquent à différents éléments de modélisation UML :
 - Dépendances entre classes et objets :
 - « *bind* » : la source instancie le template cible
 - « *friend* » : la source a une visibilité spéciale de la cible
 - « *instanceof* » : l'objet source est une instance de la classe cible
 - « *instantiate* » : la classe source crée des instances de la classe cible
 - « *refine* » : degré d'abstraction de la source est plus fin que celui de la cible
 - « *use* » : la cible utilise la source (variable, argument, valeur de retour)
 - Dépendances entre paquetages :
 - « *access* » : le paquetage source a le droit de référencer les éléments du paquetage cible
 - « *import* » : les éléments public du paquetage cible deviennent des éléments du paquetage source

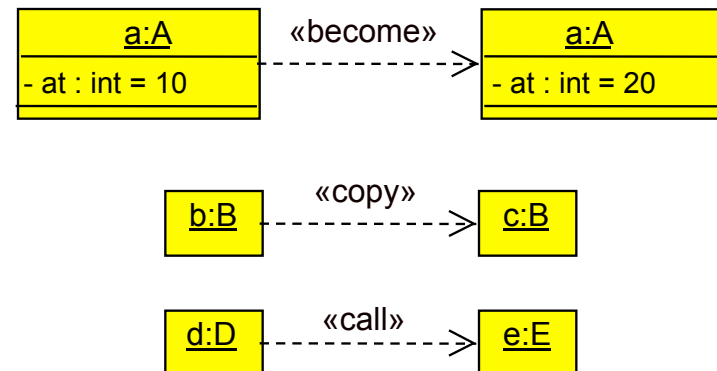
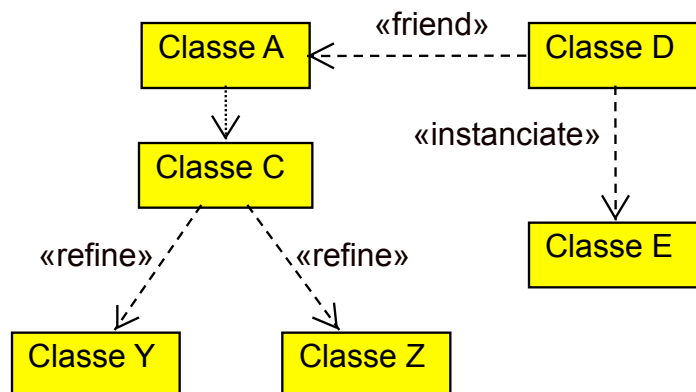
Diagrammes de classes – Les relations de dépendances (2/2)

Dépendances entre cas d'utilisation :

- « *extend* » : le cas d'utilisation source étend le comportement du cas d'utilisation cible
- « *include* » : le cas d'utilisation cible incorpore le comportement du cas d'utilisation source

• Dépendances entre objets :

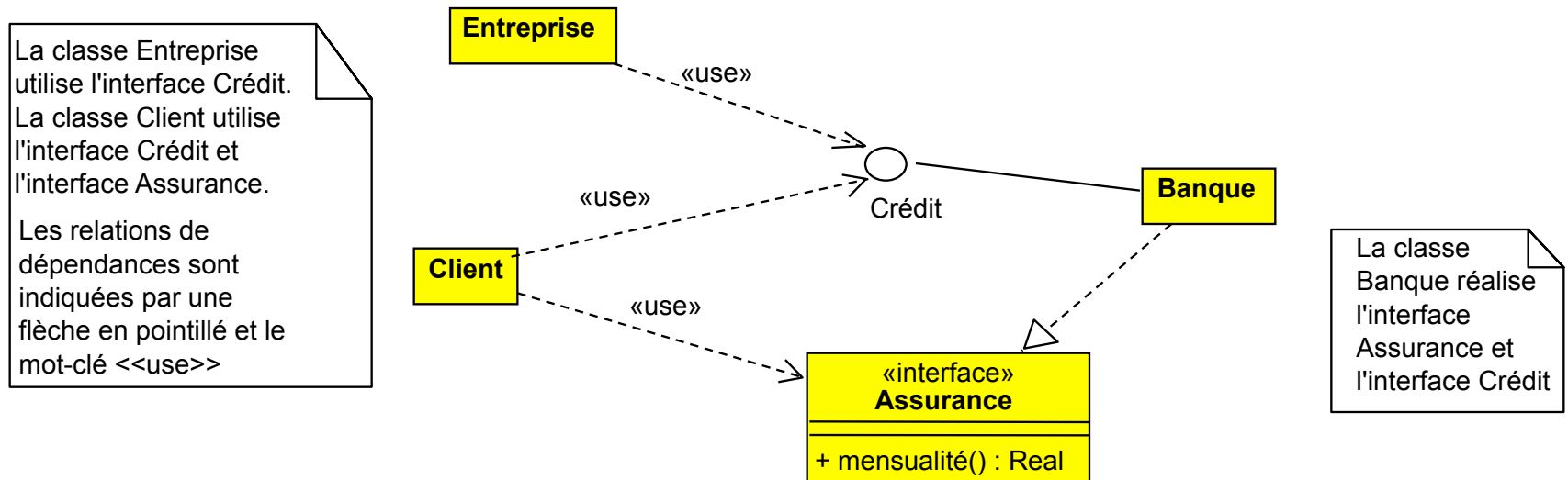
- « *become* » : indique un changement de propriété de l'objet source
- « *call* » : l'objet cible appelle une opération de l'objet source
- « *copy* » : l'objet cible est une copie de l'objet source



Diagrammes de classes - Les autres classes (1/3)

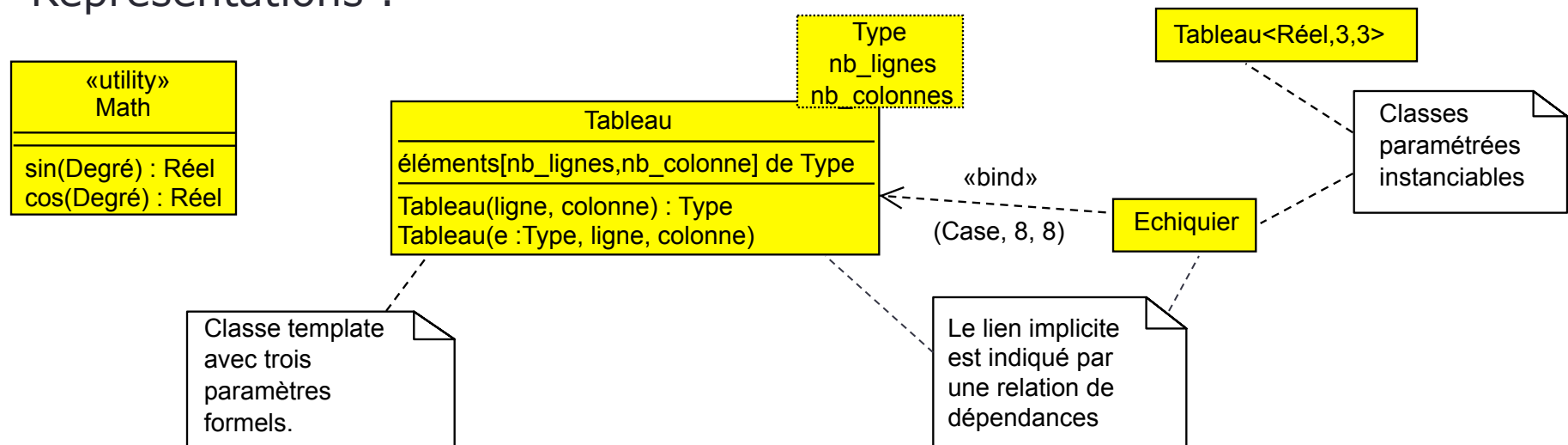
- Les interfaces

- Les interfaces permettent de mettre en place un système de contrat entre classes fournisseurs et classes consommateurs
- Une classe interface = déclaration d'un ensemble d'obligations (opérations publics sans implémentation) que les classes fournisseurs doivent implémenter
- Une classe fournisseur implémente les services déclarés d'une classe interface : elle réalise l'interface (*définit toutes les opérations de l'interface*)
- Une classe peut réaliser plusieurs interfaces (une sorte d'héritage multiple des services à réaliser)
- Une interface peut être réalisée par plusieurs classes
- Les interfaces peuvent avoir des liens de généralisation
- Représentation : un cercle ou une classe stéréotypée



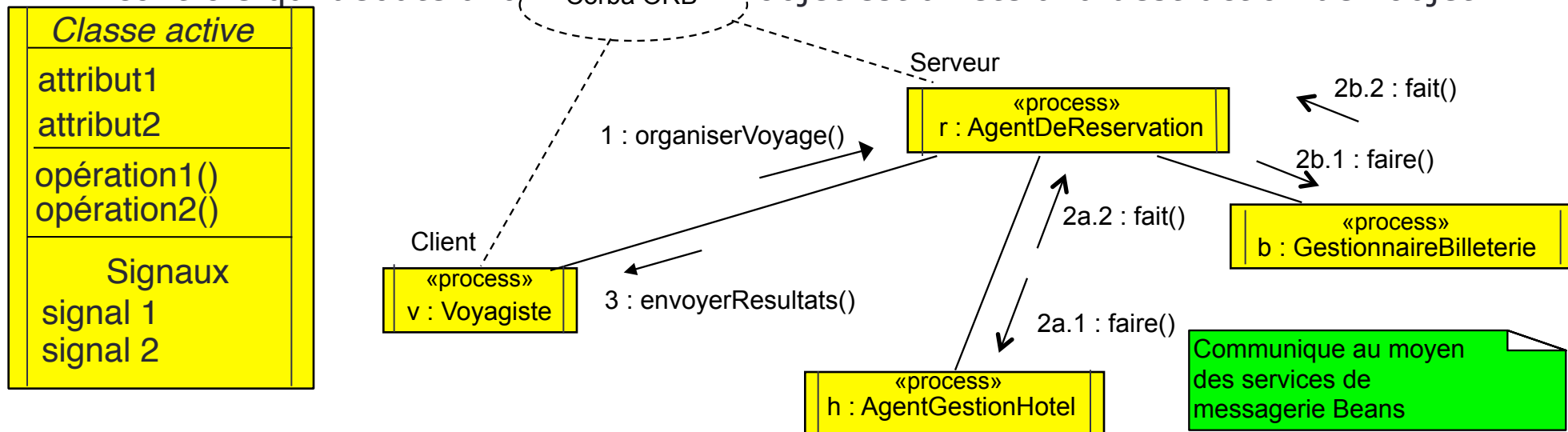
Diagrammes de classes - Les autres classes (2/3)

- Les classes paramétrables
 - Une classe paramétrable (appelée aussi *classe template*) est un modèle de classe (sans instance d'objet) possédant des paramètres
 - Les paramètres formels doivent être liés aux paramètres effectifs afin que la classe paramétrée puisse être instanciée
- Les classes utilitaires
 - Regroupement au sein d'une classe non-instanciable d'attributs et d'opérations sans qu'il y ait notion de classe
 - La portée de ces attributs et de ces opérations est celle de la classe et non celle de l'instance
- Représentations :



Diagrammes de classes - Les autres classes (3/3)

- Les classes actives :
 - Une classe active modélise un flot de contrôle.
 - Flots de contrôle :
 - Processus (*ou flot lourd*) qui peut s'exécuter en concurrence avec d'autres processus
 - Thread (*ou flot léger*) qui peut s'exécuter en concurrence avec d'autres threads au sein d'un même processus
 - Un objet actif (*instance d'une classe active*) est une *réification* d'un flot de contrôle qui débute à la *Corba ORB* l'objet est arrêté à la destruction de l'objet.



Diagrammes de classes – Exemples (Révisions)

- Considérons les six phrases suivantes :
 1. Un répertoire contient des fichiers.
 2. Une pièce contient des murs.
 3. Les modems et les claviers sont des périphériques d'entrée/sortie.
 4. Une transaction boursière est un achat ou une vente.
 5. Un compte bancaire peut appartenir à une personne physique ou morale.
 6. Deux personnes peuvent être mariées.

→ **Question :**

Déterminer la relation statique appropriée dans chaque phrase de l'énoncé précédent.