UML – Unified Modeling Language 2/4 : diagrammes statiques

Yannick Prié
Département Informatique – Faculté des Sciences et Technologies
Université Claude Bernard Lyon 1
2009-2010

Objectifs de ce cours

- Apprendre la syntaxe et la sémantique des diagrammes statiques les plus importants
- Améliorer au passage la compréhension de différents principes objets

Plan

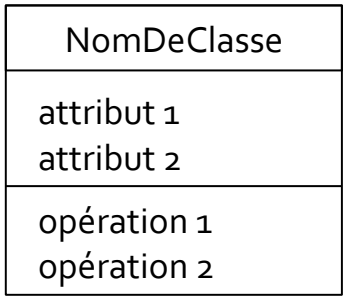
- **Diagrammes de classes**
- Diagrammes d'objets
- Diagrammes de paquetages
- Diagrammes de composants
- Diagrammes de déploiement

Diagrammes de classes : présentation générale

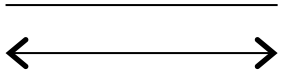
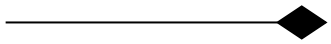
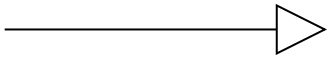
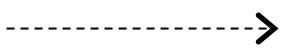
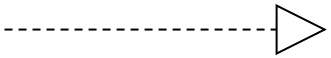
- Diagrammes fondamentaux
  - les plus connus, les plus utilisés
- Présentent la vue statique du système
  - représentation de la structure et des déclarations comportementales
  - classes, relations, contraintes, commentaires…
- Permettent de modéliser plusieurs niveaux
  - conceptuel (domaine, analyse)
  - implémentation (code)

Classes

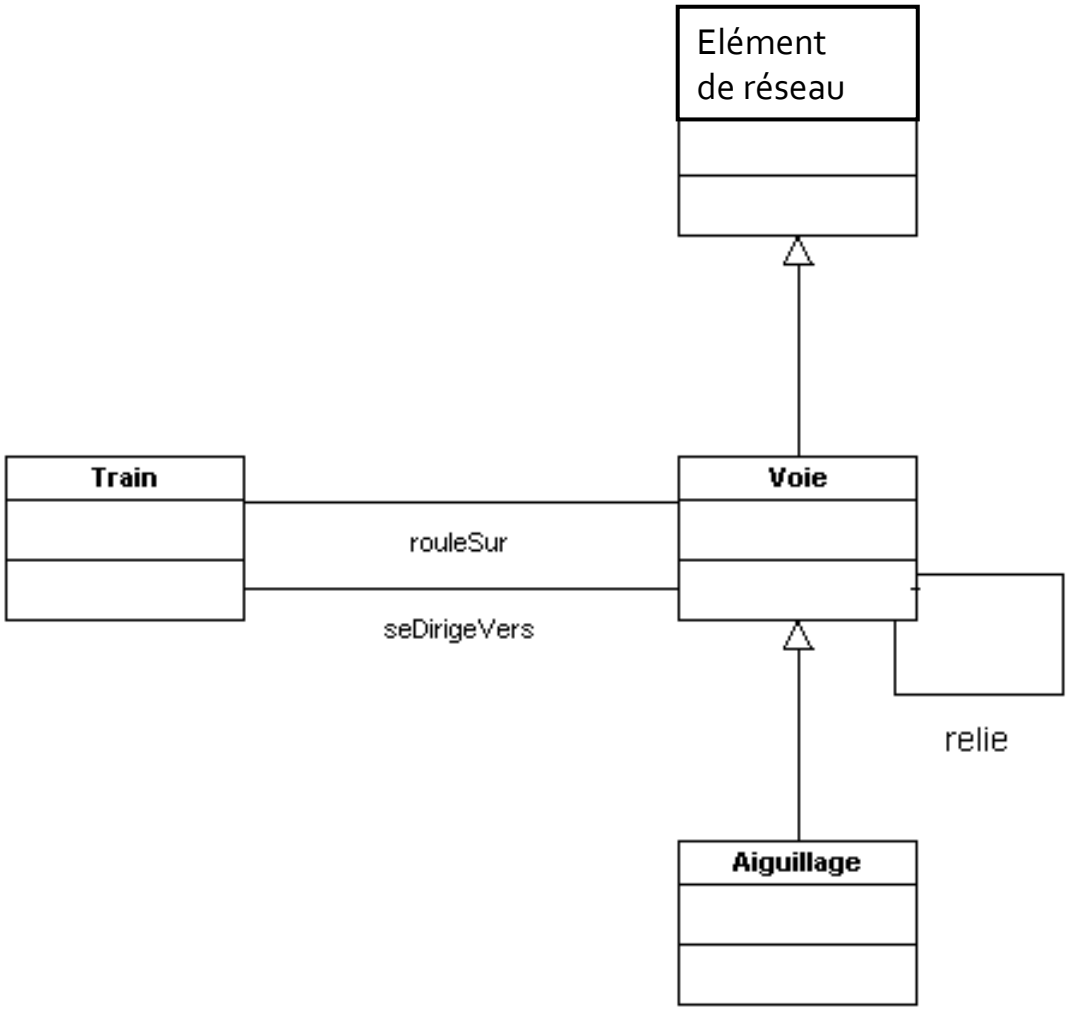
- Descripteurs de jeux d'objets
  - structure / comportement / relations / sémantique communs
- Représentation
  - rectangle à trois compartiments
    - nom
    - attributs
    - opérations
  - plus ou moins de détails suivant les besoins
- Nom : singulier, majuscule (en général)
  - ex. : Fichier, Client, Compte, Chat



Relations entre classes/ liens entre objets

- Association 
  - les instances des classes sont liées
  - possibilité de communication entre objets
  - relation forte : composition 
- Généralisation/spécialisation 
  - les instances de la sous-classe sont des instances de la super-classe (niveau conceptuel)
  - héritage (niveau implémentation)
- Dépendance 
  - la modification d'une classe peut avoir des conséquences sur une autre
- Réalisation 
  - une classe réalise une interface

Un exemple


Elément
de réseau
Train
Voie
rouleSur
seDirigeVers
relie
Aiguillage

Utilisation des diagrammes de classes

- Expression des besoins
  - modélisation du domaine
- Conception
  - spécification : gros grain
- Construction
  - implémentation : précis
  - rétro-ingénierie

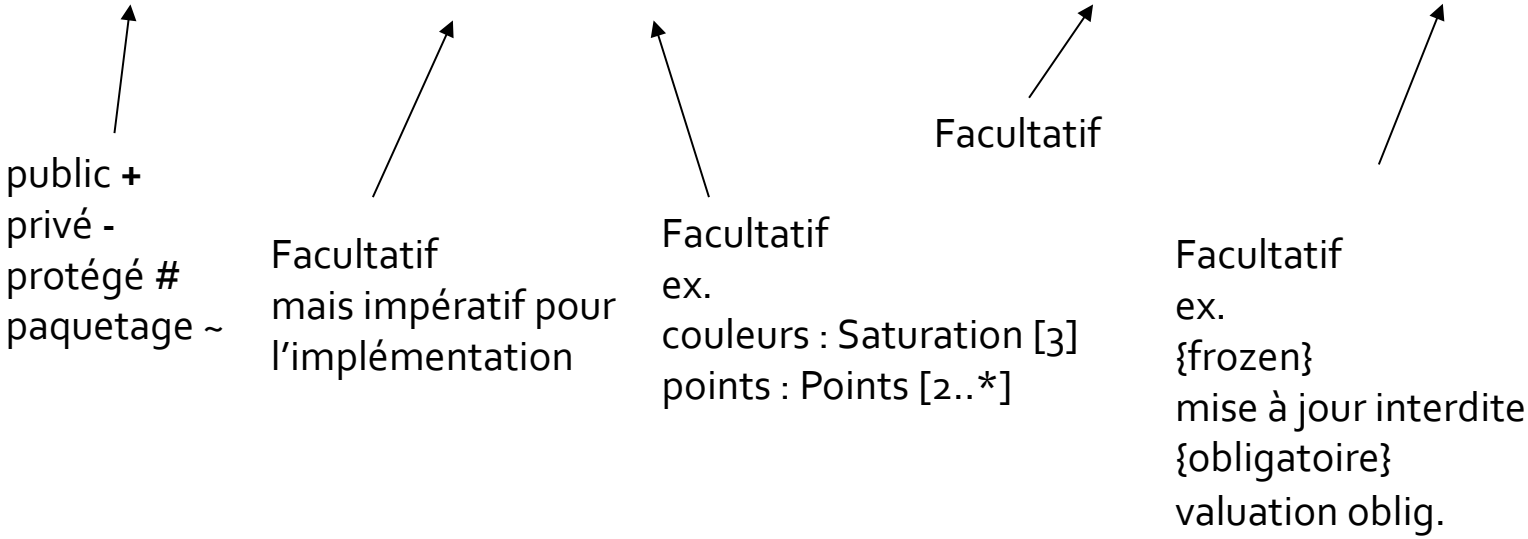
- Les diagrammes de classes permettent de représenter toute modélisation en classes, que ce soit des classes implémentées en machine ou non
- On peut modéliser n'importe quel domaine avec des classes

Petit exercice

- Dessiner un diagramme de classe du domaine avec les classes suivantes
  - étudiant
  - enseignant
  - cours
  - salle de classe

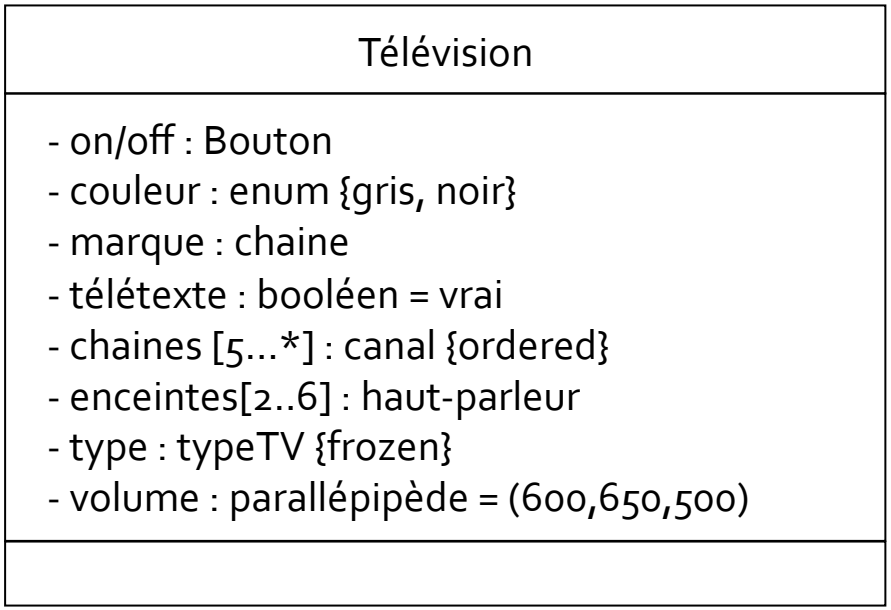
Attributs

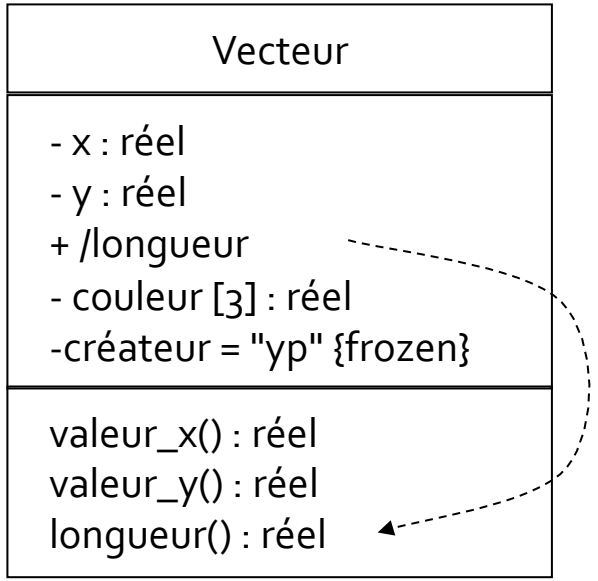
Visibilité nom : type [multiplicité] = valeur_initiale {propriétés}



- Remarques
  - /nom : attribut dérivé (calculé)
  - souligné : attribut statique (de classe)
  - {frozen} : disparu de UML2 ; à utiliser quand-même

Attributs : exemple





Opérations de classes

visibilité nom (liste de paramètres) : type-retour {propriétés}

argument ::= direction nom : type = valeur-défaut

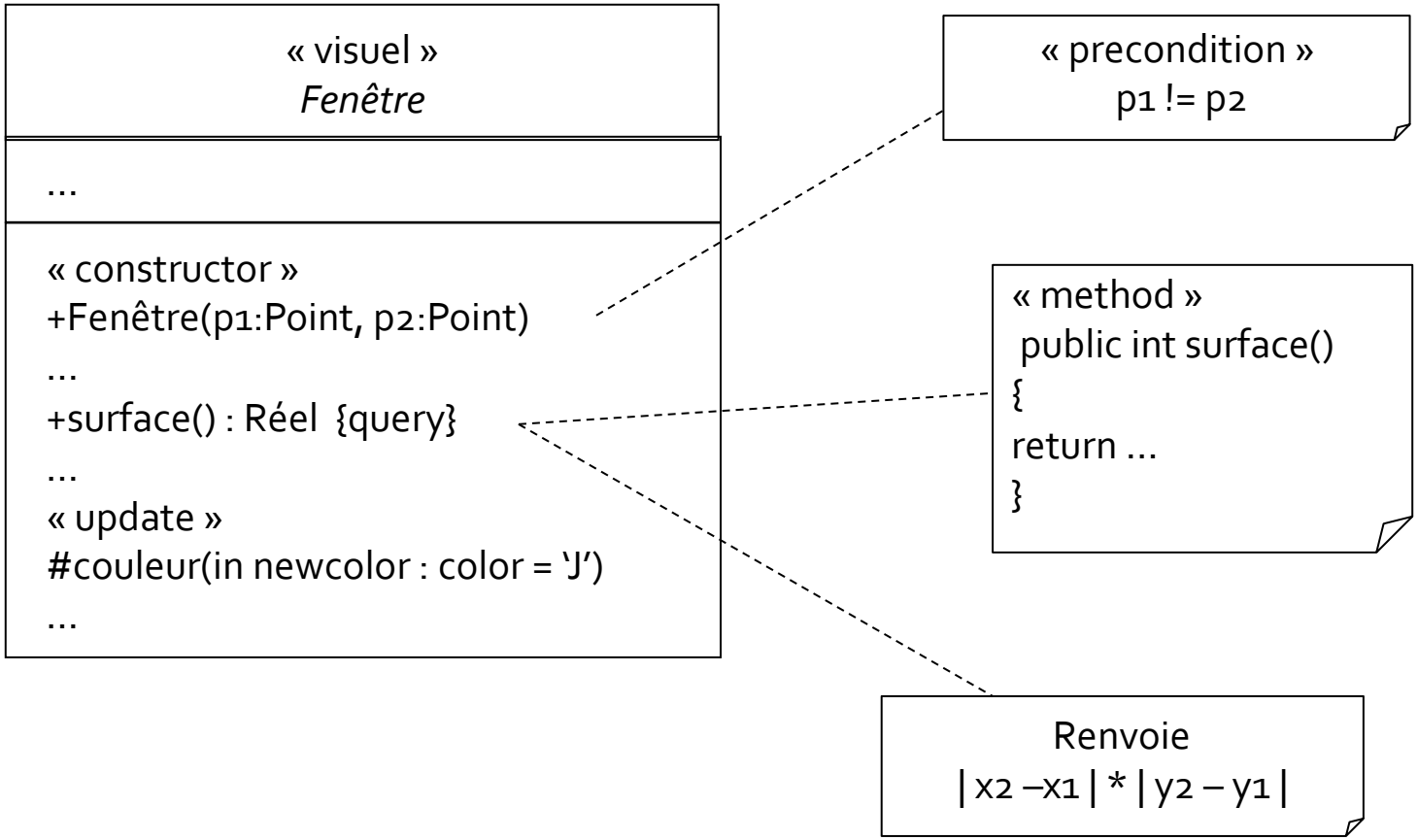
public **+**
privé **-**
protégé **#**
paquetage ~

in | out | inout

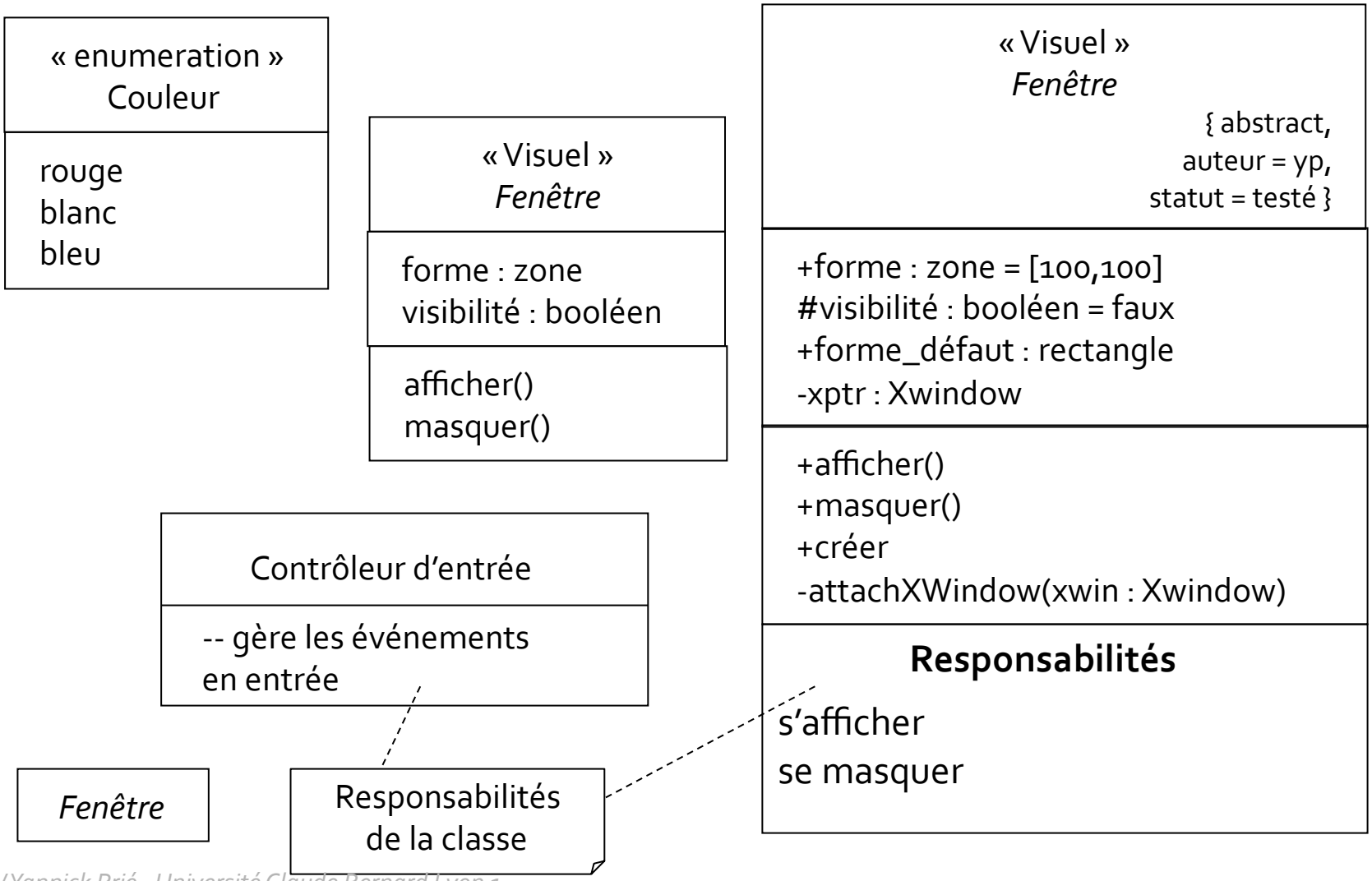
asbtract
query
...

- Remarques
  - notation : *opération abstraite* / opération statique
  - opérations = comportement d'une classe, trouvées en examinant les diagrammes d'interaction
  - méthode = implémentation d'une opération dont elle spécifie l'algorithme ou la procédure associée
  - pré et post-conditions, description du contenu : commentaires + OCL

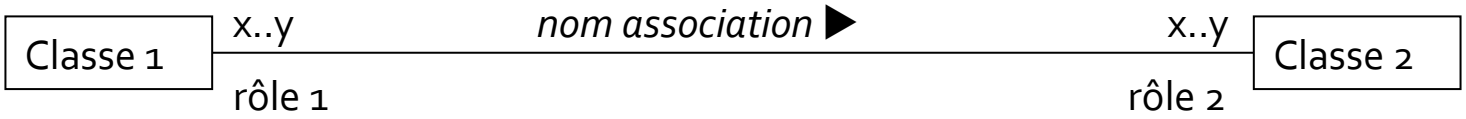
Opérations : exemple



Autres exemples de classes



Associations

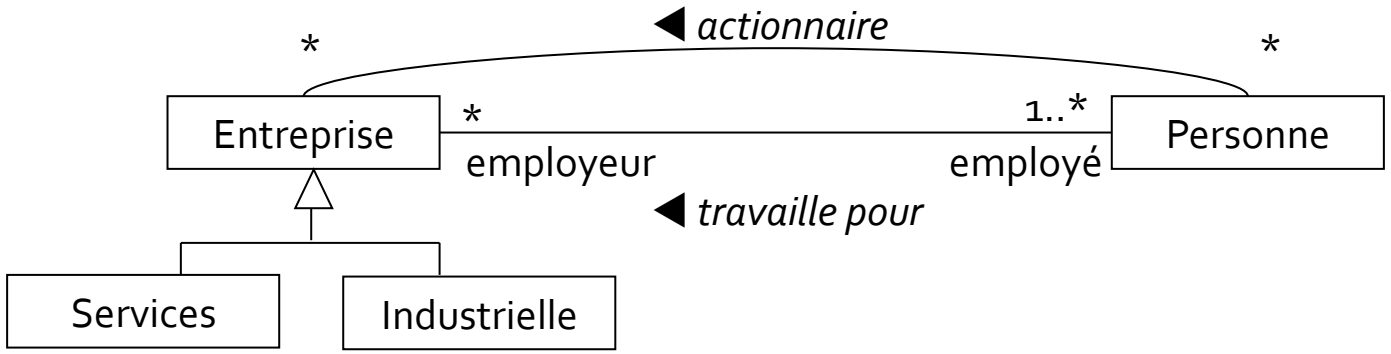


Nom : forme verbale, sens de lecture avec flèche

Rôles : forme nominale, identification extrémité association

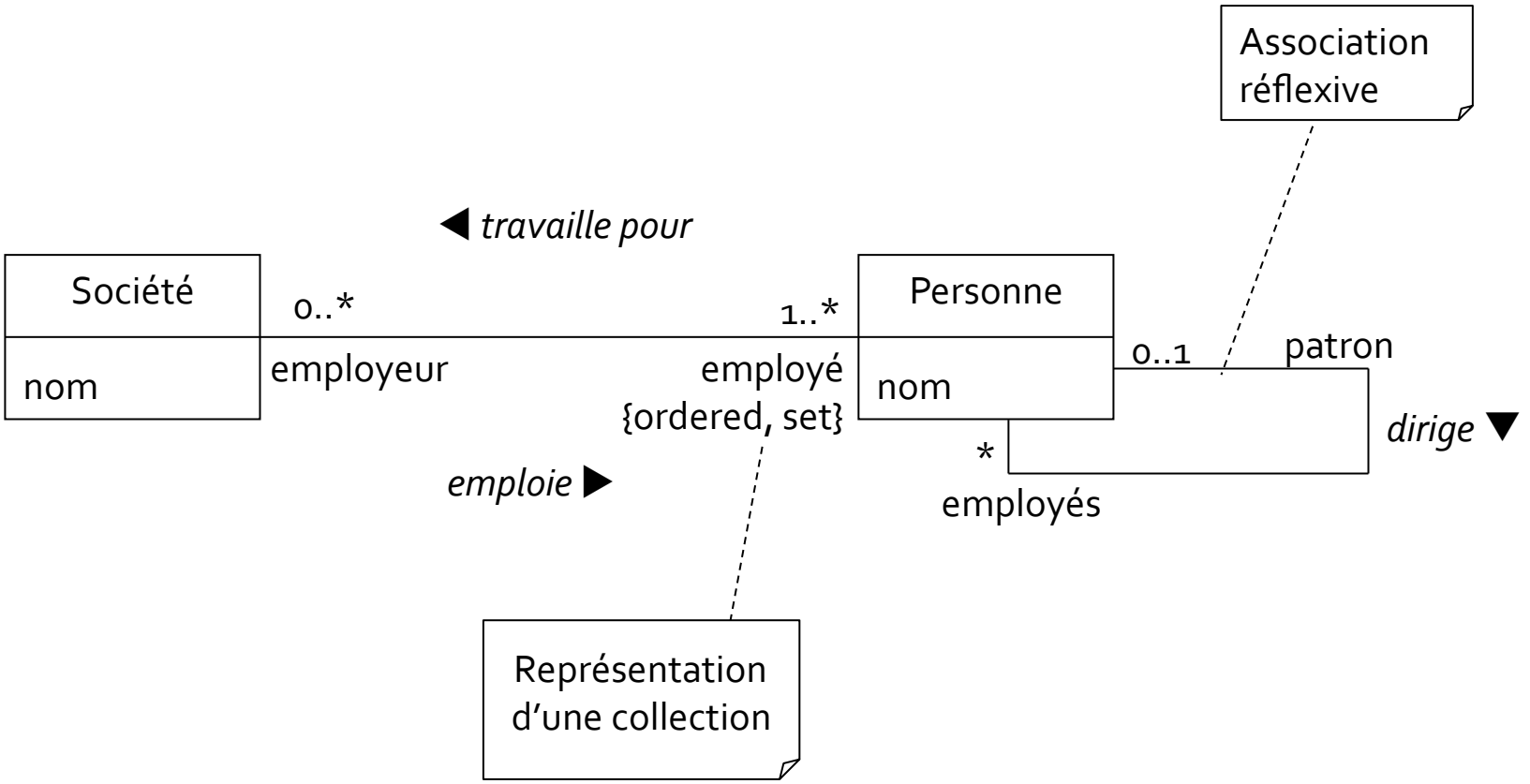
Multiplicité : 1, 0..1, 0..*, 1..*, n..m

Mots-clés : *set*, *ordered set* (uniques) ; *bag*, *list* (doublons)



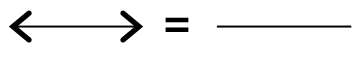
Les associations ont une durée de vie, sont indépendantes les unes des autres, sont héritées, comme les attributs

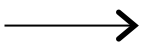
Associations : exemple



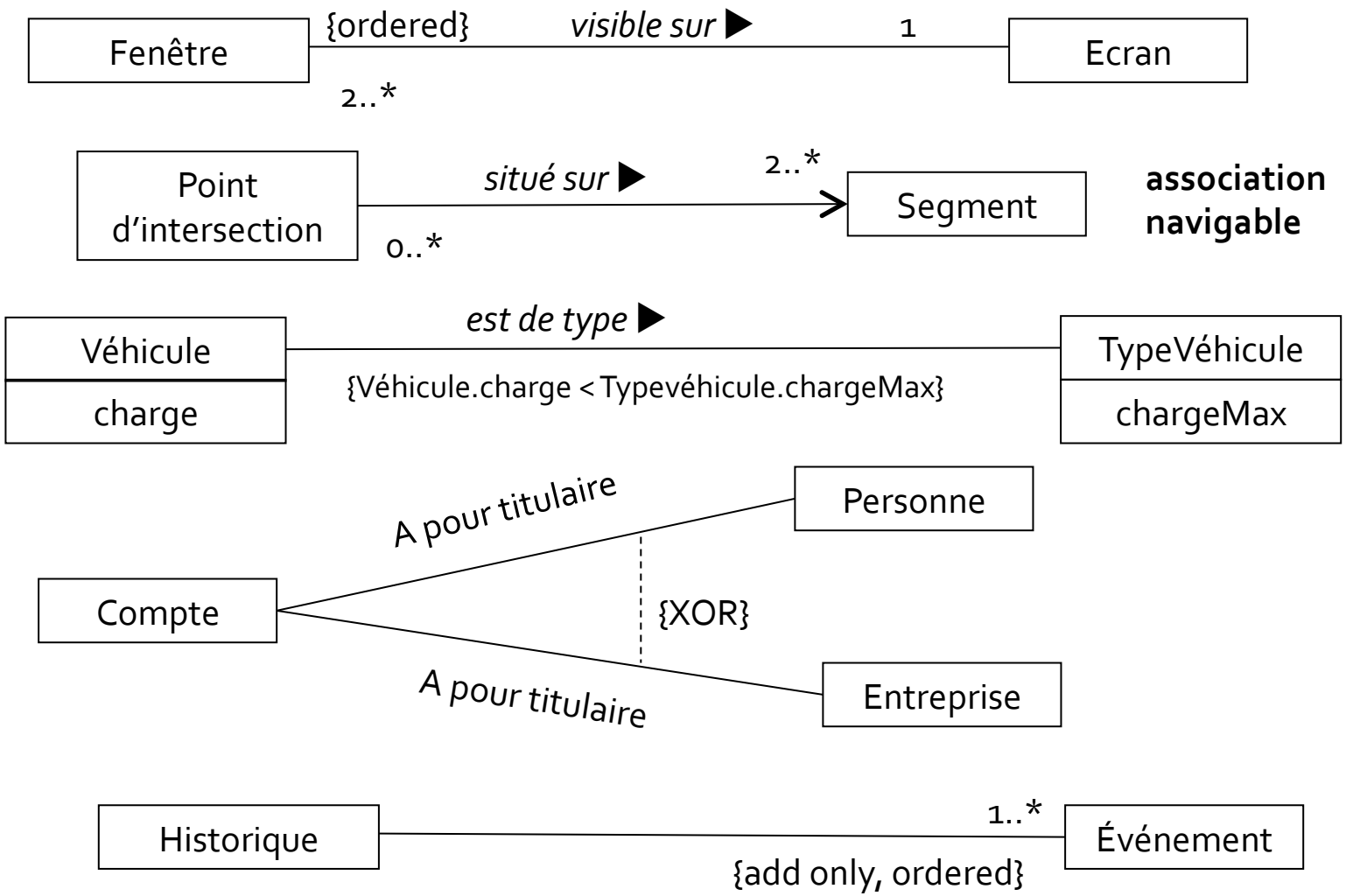
Associations : remarques

- Tout objet doit être accessible via un lien
  - ne peut recevoir de messages sinon
  - liens plus ou moins permanents : voir "Visibilités"
- Multiplicité
  - nombre d'instances d'une classe en relation avec une instance d'une autre classe
  - pour chaque association
    - deux décisions à prendre : deux extrémités
- Directionnalité
  - bidirectionnalité par défaut, evt explicitée
  - restriction de la navigation à une direction



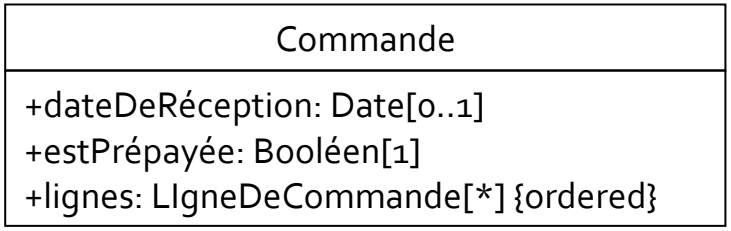


Associations et contraintes

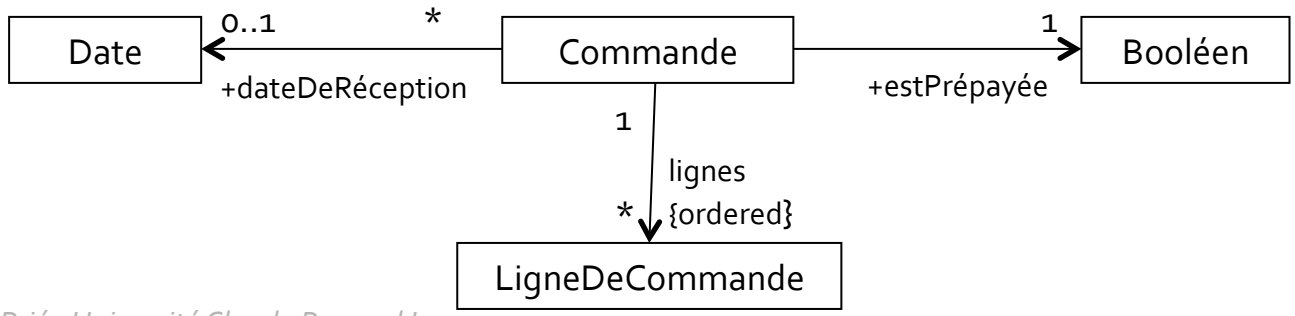


Propriétés : caractéristiques structurelles des classes

- Concept unique regroupant attributs et associations monodirectionnelles : équivalence des représentations
- Pour choisir
  - attribut (texte) pour les types de données
    - objets dont l'identité n'est pas importante
  - association pour insister sur les classes



(Fowler, 2004)



Agrégation et composition

- Associations asymétriques, fortes
- Agrégation
  - non nommée, structure d'arbre sous-jacente (pas de cycle), rôle prépondérant d'une extrémité

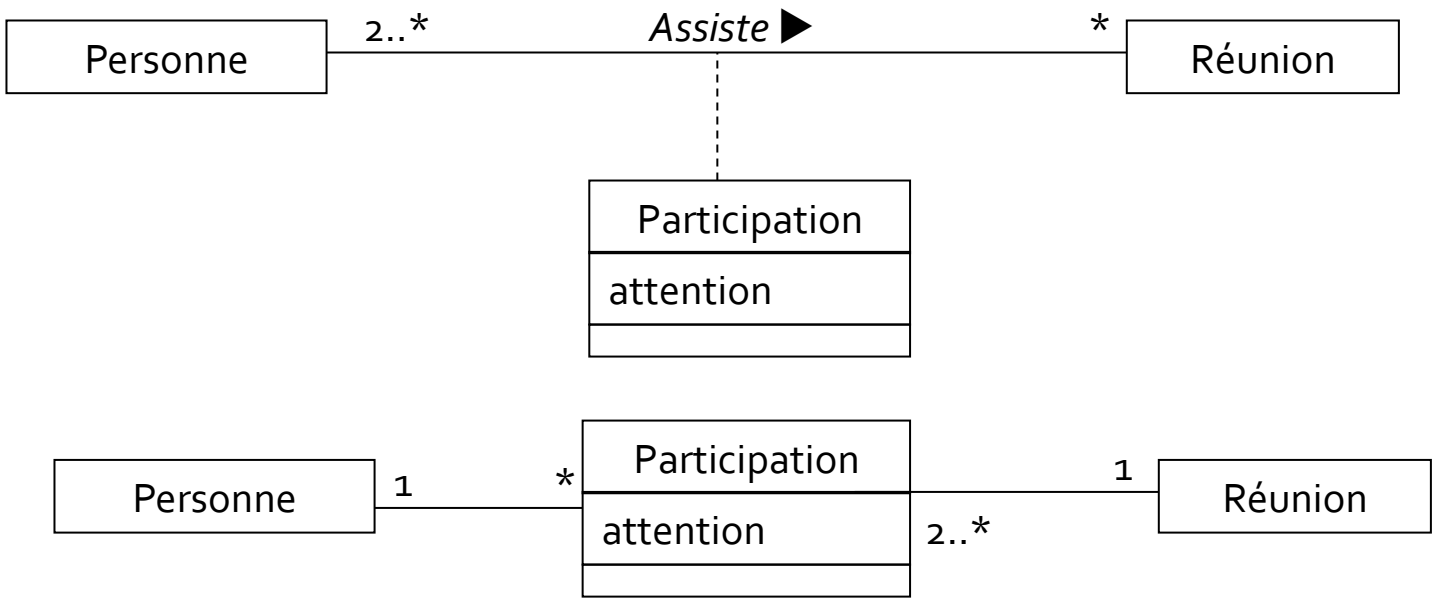
- Composition
  - non partage des éléments composants, création et destruction des composants avec le composite

Composition, agrégation et association

- Quelques questions à se poser
  - asymétrie et lien de subordination entre instances des deux classes (agrégation/composition) ou indépendance des objets (association) ?
  - propagation d'opérations ou d'attributs du tout vers les parties ? (agrégation/composition)
  - création et destruction des parties avec le tout ? (composition)
- Remarques importantes
  - dans le doute, toujours utiliser une association : c'est la moins contrainte
  - pour certains experts, il faut oublier l'agrégation
    - agrégation = "placebo denué de sens"

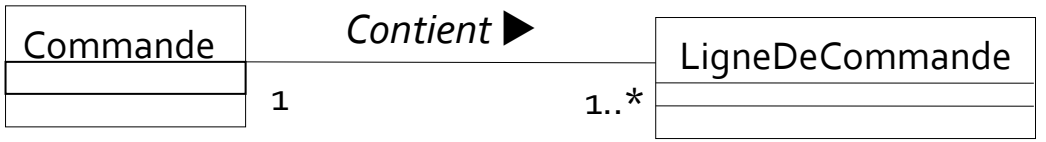
Classes d'association

- Pour ajouter attributs et opérations à des associations
- Quelques indices pour l'utilisation
  - un attribut est lié à une association
  - la durée de vie des instances de la CA dépend de l'association
  - association N..N entre deux classes + informations liées à l'association



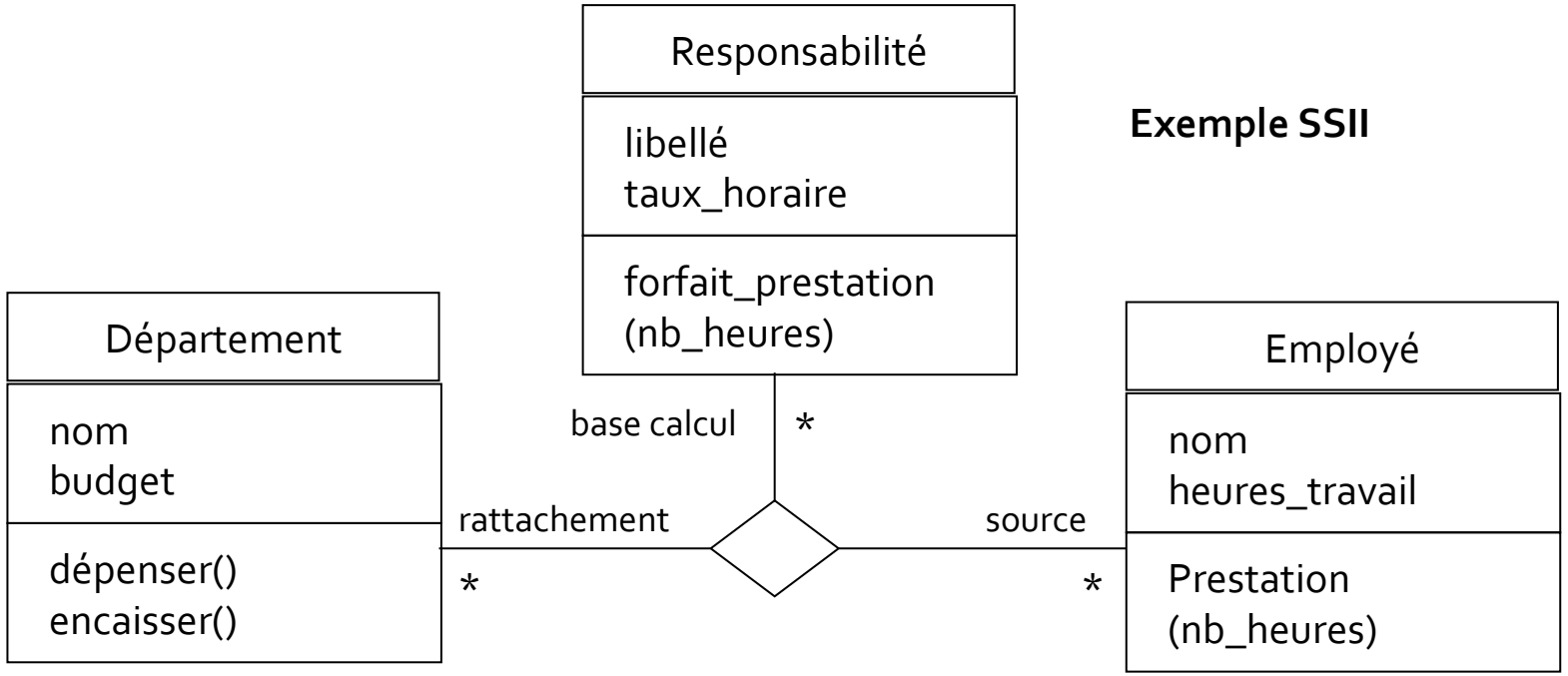
Associations qualifiées

- Equivalent UML des dictionnaires
- Sélection d'un sous-ensemble des objets qui participent à l'association à l'aide d'une clé.
  - cet attribut est propriété de l'association



Associations n-aire

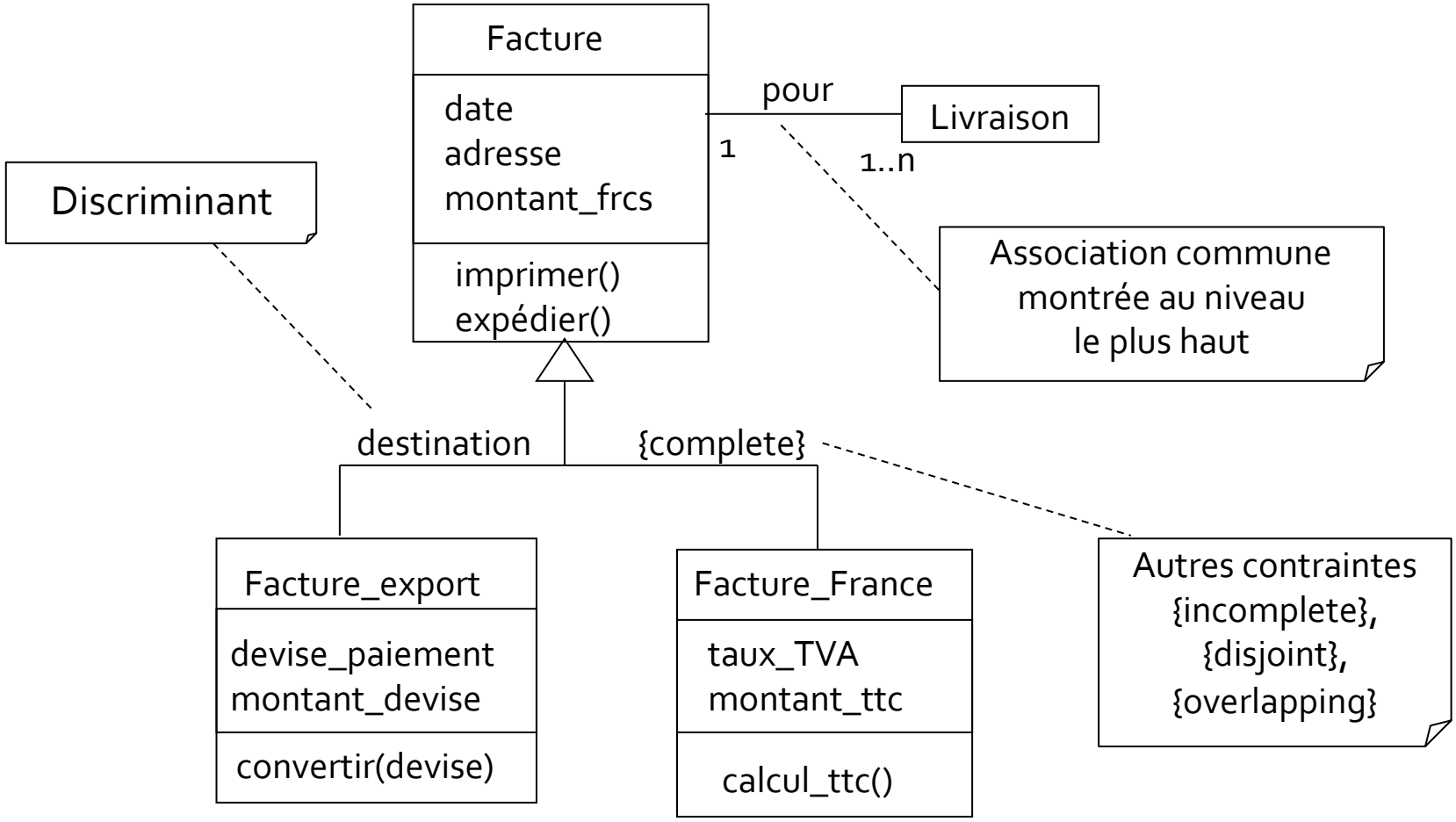
- Groupe de liens entre au moins trois instances
- Instance de l'association = n-uplet des attributs des instances impliquées



Généralisation / spécialisation

- Deux interprétations
  - niveau conceptuel
    - organisation : un concept est plus général qu'un autre
  - niveau implémentation
    - héritage des attributs et méthodes
- Pour une bonne classification conceptuelle
  - principe de substitution / conformité à la définition
    - toutes les propriétés de la classe parent doivent être valables pour les classes enfant
  - « A est une sorte de B » (mieux que « A est un B »)
    - toutes les instances de la sous-classe sont des instances de la super-classe (définition ensembliste)
- Spécialisation
  - relation inverse de la généralisation

Hiérarchie de classes



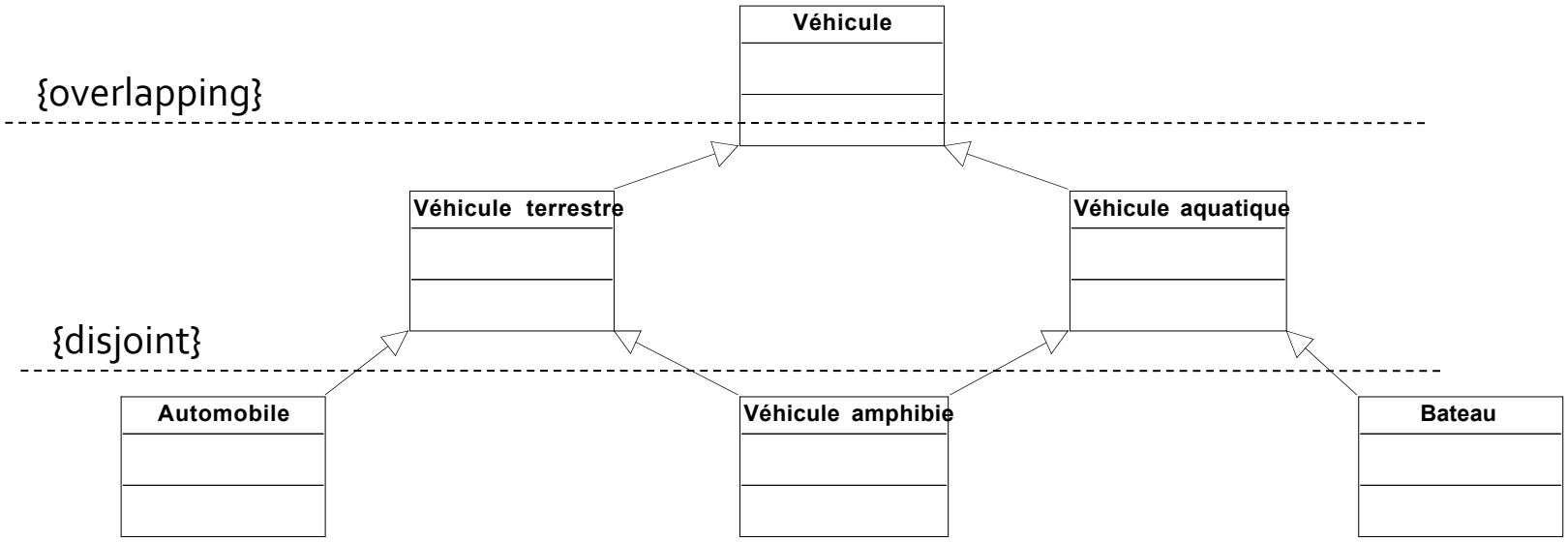
Conseils pour la classification conceptuelle

(Larman, 2005)

- Partitionner une classe en sous-classes
  - la sous-classe a des attributs et/ou des associations supplémentaires pertinents
  - par rapport à la superclasse ou à d'autres sous-classes, la sous-classe doit être gérée, manipulée, on doit agir sur elle ou elle doit réagir différemment, et cette distinction est pertinente
  - le concept de la sous-classe représente une entité animée (humain, animal, robot) qui a un comportement différent de celui de la superclasse, et cette distinction est pertinente
- Définir une super-classe
  - les sous-classes sont conformes aux principes de substitution et « sorte-de »
  - toutes les sous-classes ont au moins un même attribut et/ou une même association qui peut être extrait et factorisé dans la superclasse

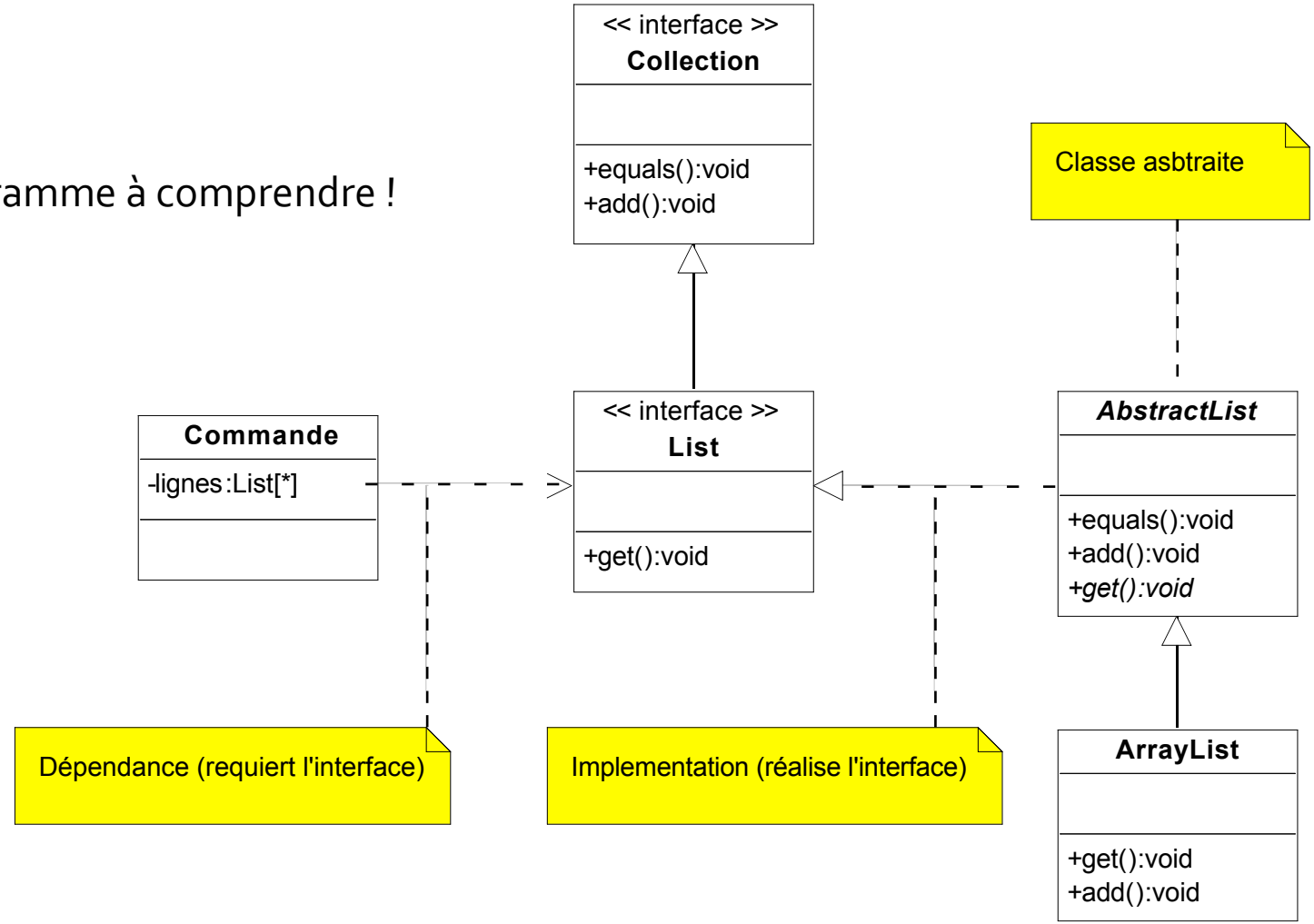
Généralisation multiple

- Autorisée en UML
- Attention aux conflits : il faut les résoudre
- Possibilité d'utiliser aussi délégations ou interfaces

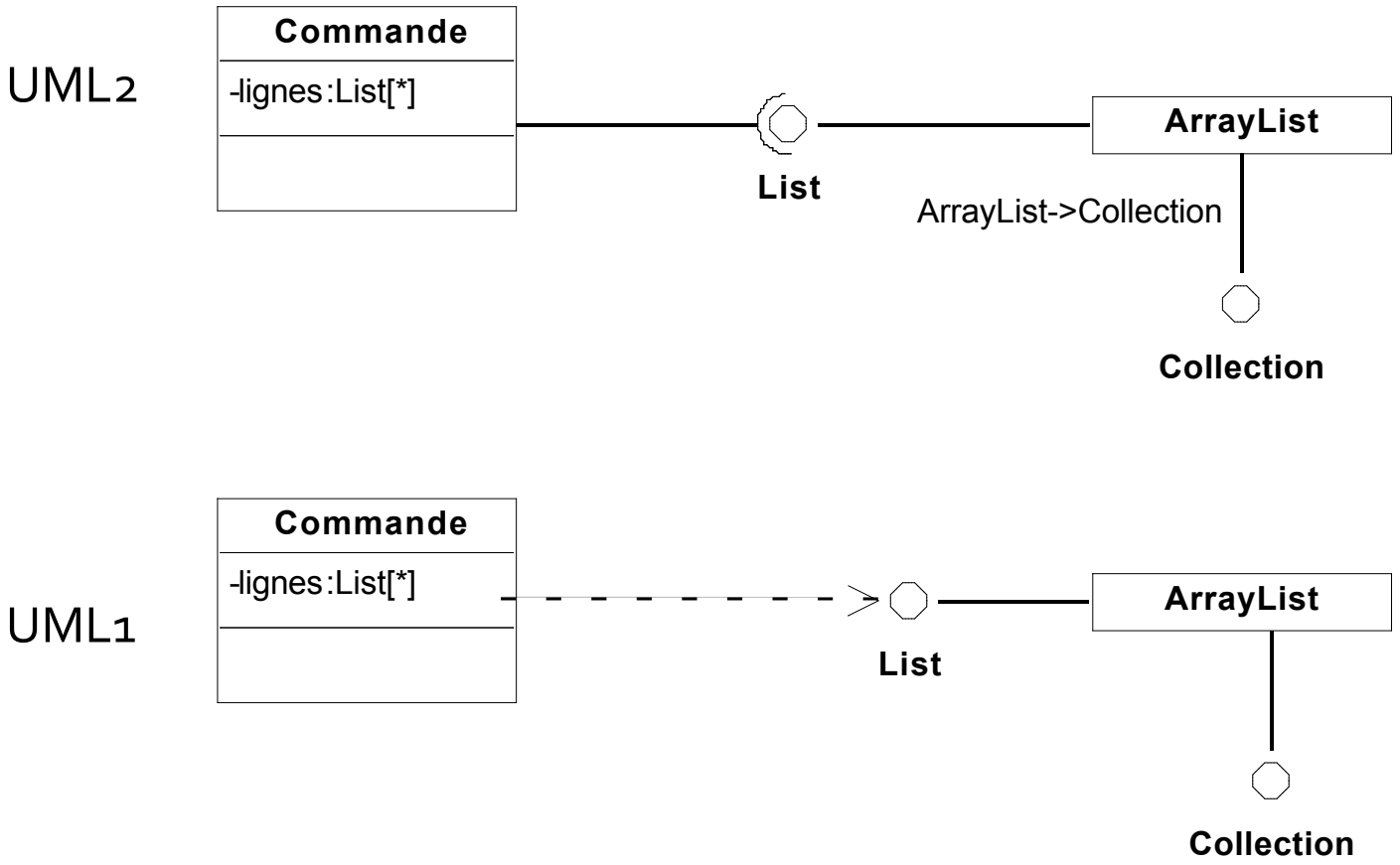


Interfaces et classes abstraites

Un diagramme à comprendre !

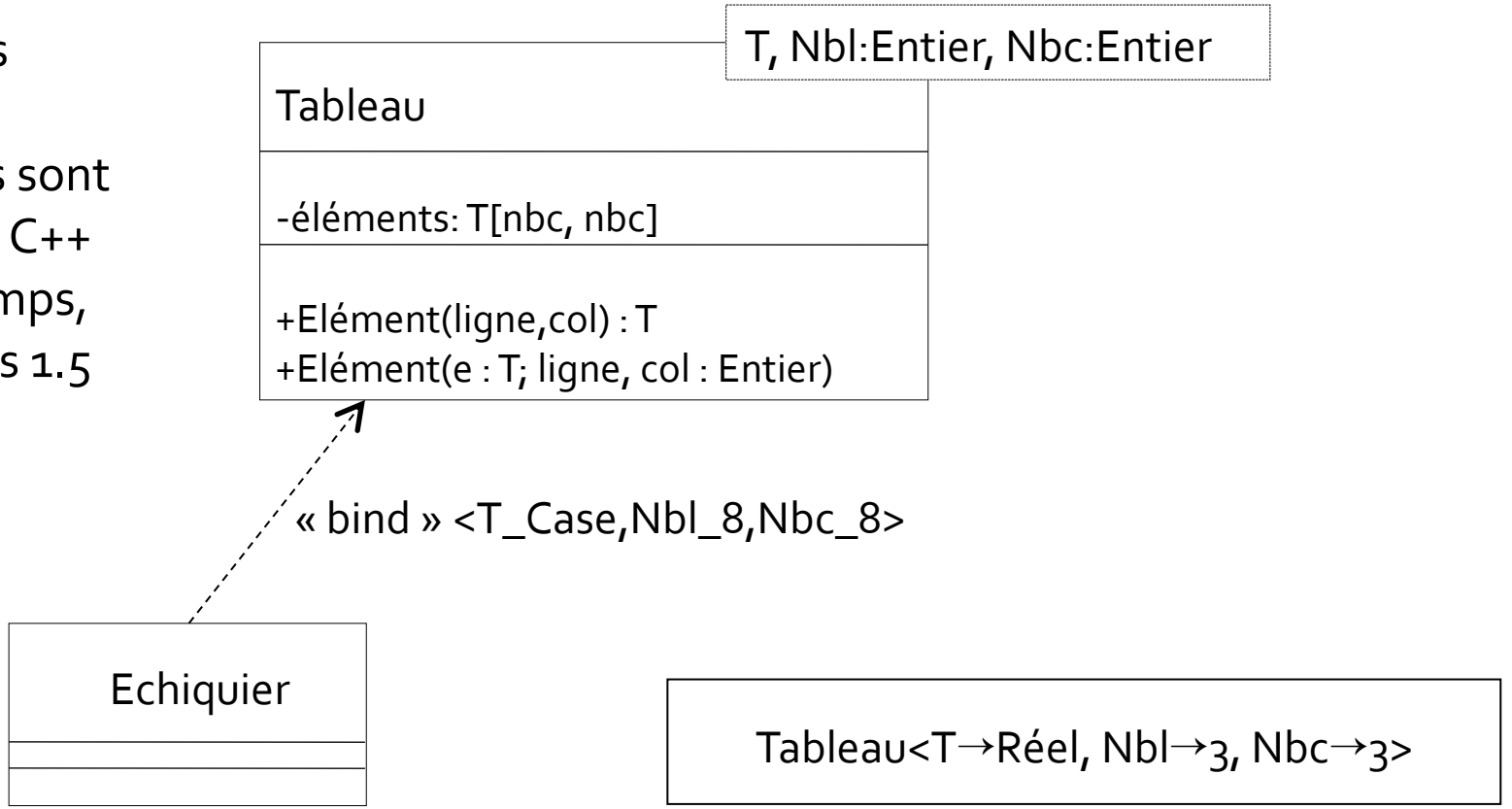


Interface et utilisation : notation



Classes paramétrables (templates)

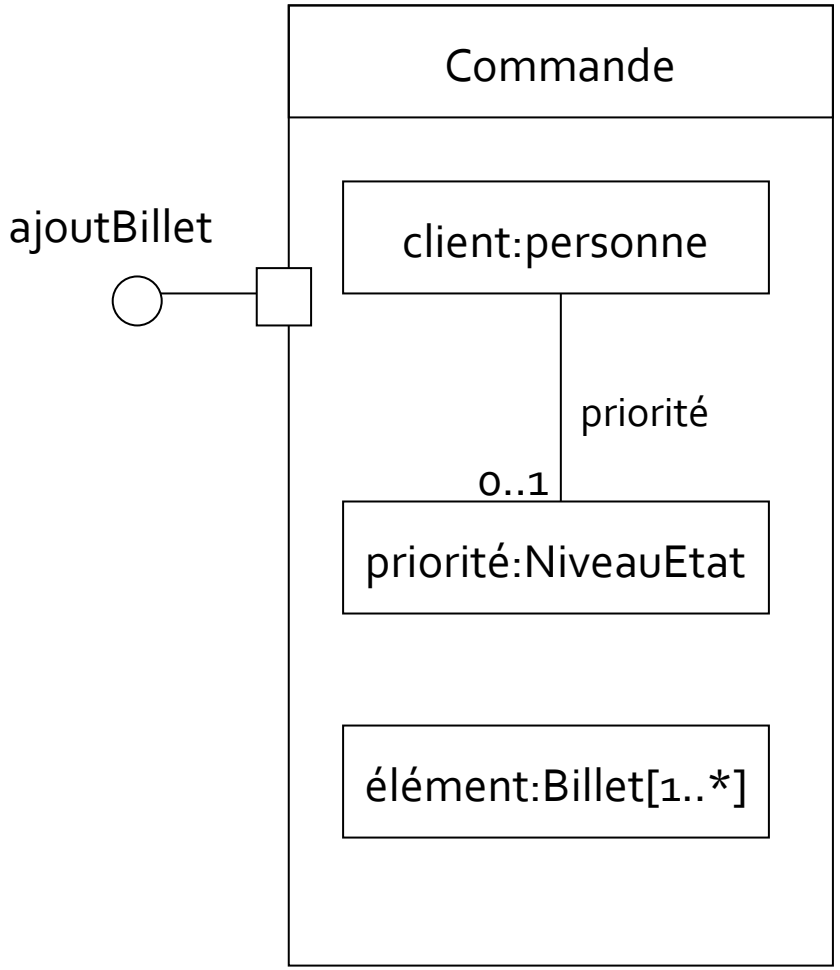
Remarque : les classes paramétrables sont disponibles en C++ depuis longtemps, en JAVA depuis 1.5



Deux notations de la paramétrisation d'une classe paramétrable

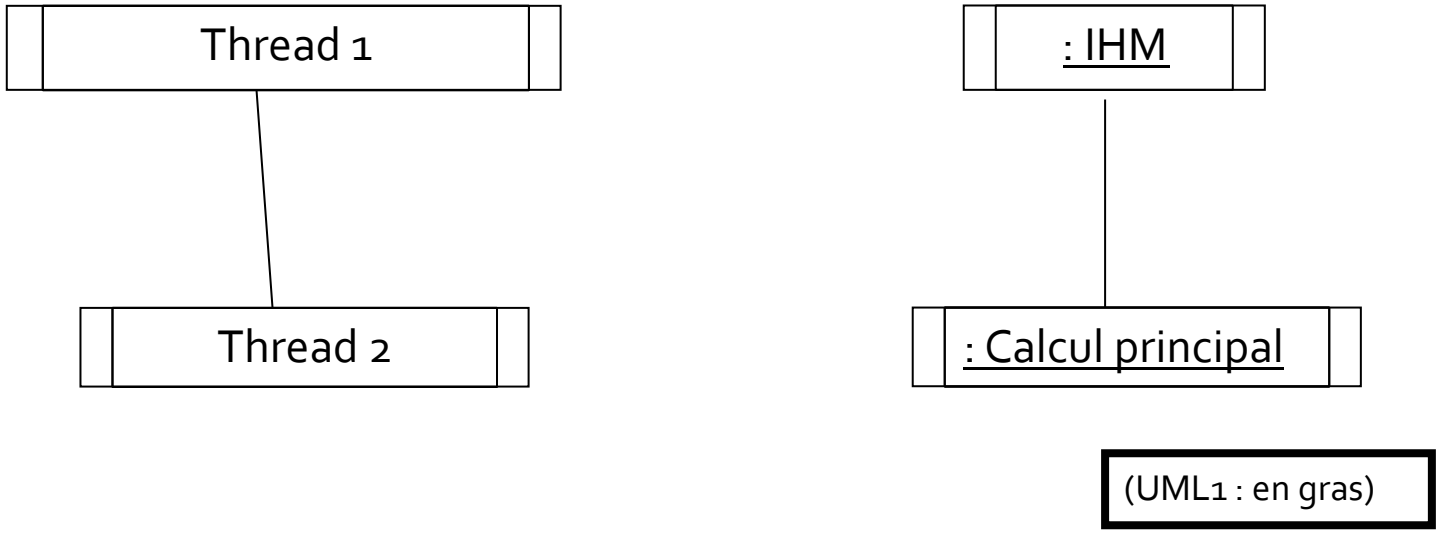
Classes structurées

- Description de la structure d'implémentation interne d'une classe
- Contient
  - ports : points de connexion avec l'environnement (evt. interne)
  - parties : fragment structuré de la classe
  - connecteurs : connexions de deux parties au sein de la classe



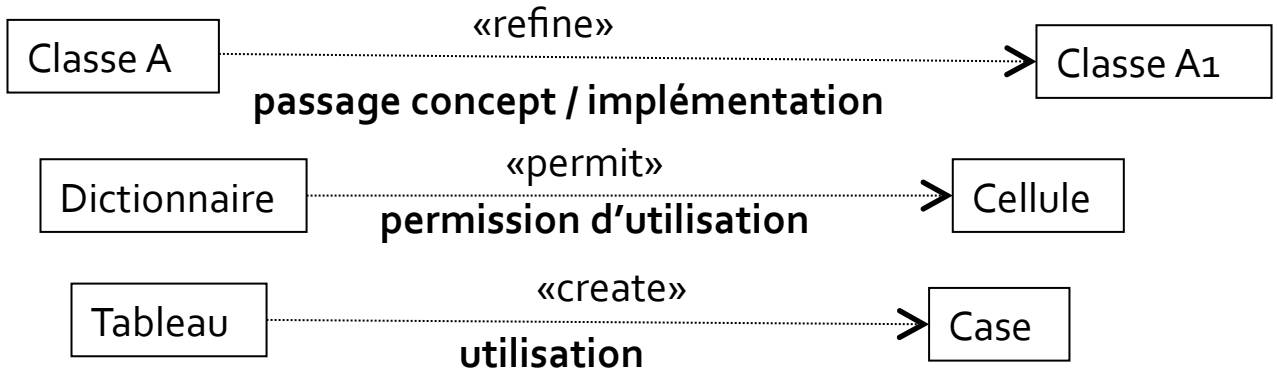
Classes actives

- Classe dont les instances sont des objets actifs
  - possèdent leur propre thread
  - dans un environnement multitâche



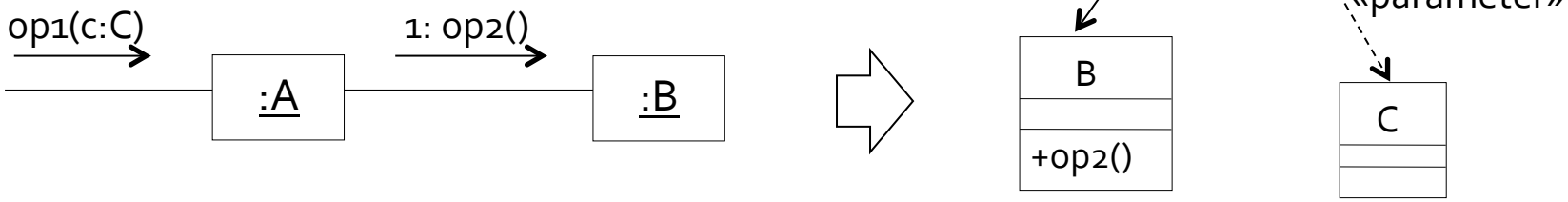
Relations de dépendance

- 3 grands types
  - abstraction : différents niveaux d'abstraction
    - ex. «refine», «trace», «derive»
  - permission d'utilisation (cf. friend en C++)
    - ex. «permit»
  - utilisation
    - ex. «use», «create», «call», «parameter»
- Conseil
  - utiliser une dépendance pour tout ce qui n'est pas spécifié



Liens, visibilité et dépendances

- Lien => possibilité d'envoyer un message d'un objet à un autre
- Deux types de liens
  - Lien durable : visibilité d'attribut ou globale
    - se matérialise par une association entre classes
  - Lien temporaire : visibilité paramètre ou locale
    - résulte d'une utilisation temporaire d'un objet par un autre
    - se matérialise par une dépendance entre classes
      - ex. passage de paramètre : « parameter », variable locale à une méthode : « local »
    - attention à ne pas oublier ces dépendances !

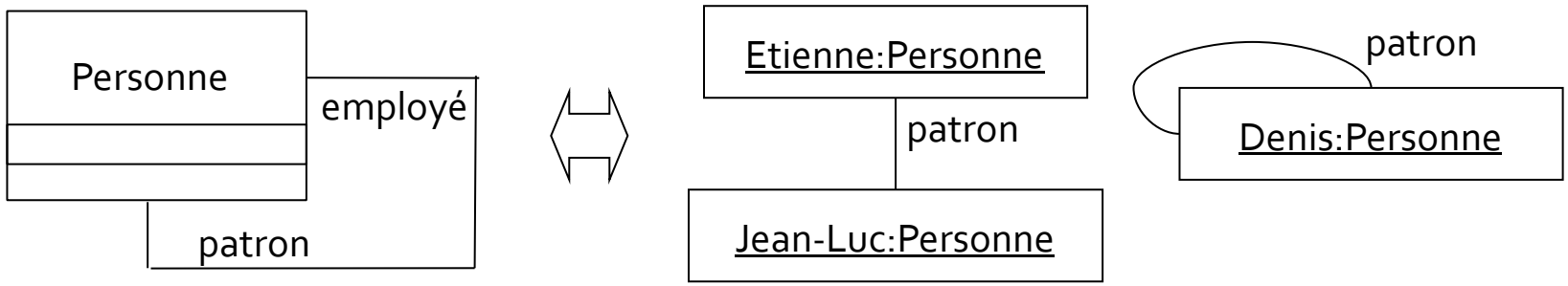


Plan

- Diagrammes de classes
- **Diagrammes d'objets**
- Diagrammes de paquetages
- Diagrammes de composants
- Diagrammes de déploiement

Diagrammes d'objets

- Pour représenter un instantané du système
  - les objets et leurs liens
  - objets = spécification d'instances
- Quand les utiliser ?
  - pour montrer un contexte
    - collaborations sans messages
  - quand une structure complexe est trop difficile à comprendre avec un diagramme de classe
    - ex. : récursivité, associations multiples, etc.



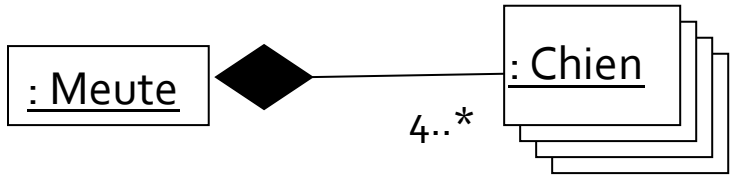
Objets

- Nom de l'objet souligné
- Objets anonymes ou non
- Objets classifiés ou non

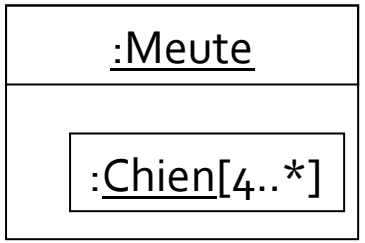
Nom objet	Nom objet : classe	: classe
Rex	Rex : Chien	: Chien

Objets dans une collection

- Multi-objet (UML1)
  - modéliser un jeu
    - comme un objet unique avec des opérations sur le jeu
    - comme jeu d'objets individuels avec leurs opérations



- Classe structurée (UML2)

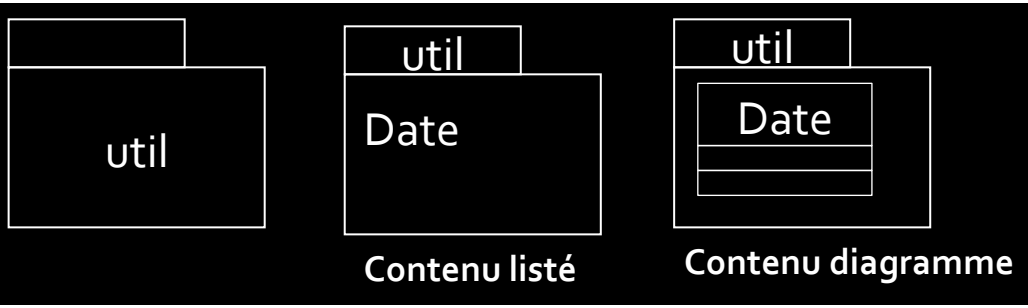


- Utiles pour les diagrammes de communication pour s'adresser
  - à l'objet qui représente la collection (Meute)
  - aux objets dans la collection (Chien)

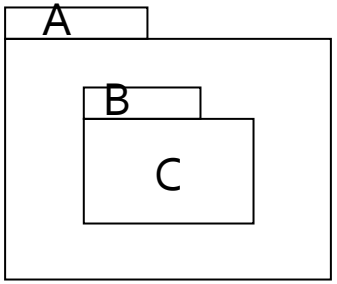
Plan

- Diagrammes de classes
- Diagrammes d'objets
- **Diagrammes de paquetages**
- Diagrammes de composants
- Diagrammes de déploiement

Paquetage



- Mécanisme général pour
  - organiser les éléments et les diagrammes du modèle, notamment les classes
    - partitionner, hiérarchiser
    - clarifier
  - les nommer
    - un paquetage définit un espace de nom
    - deux éléments ne peuvent avoir le même nom dans un paquetage
- Un paquetage
  - contient des éléments
    - y compris d'autres paquetages : hiérarchie
  - peut importer d'autres paquetages
  - peut posséder des interfaces



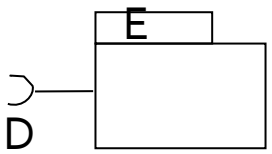
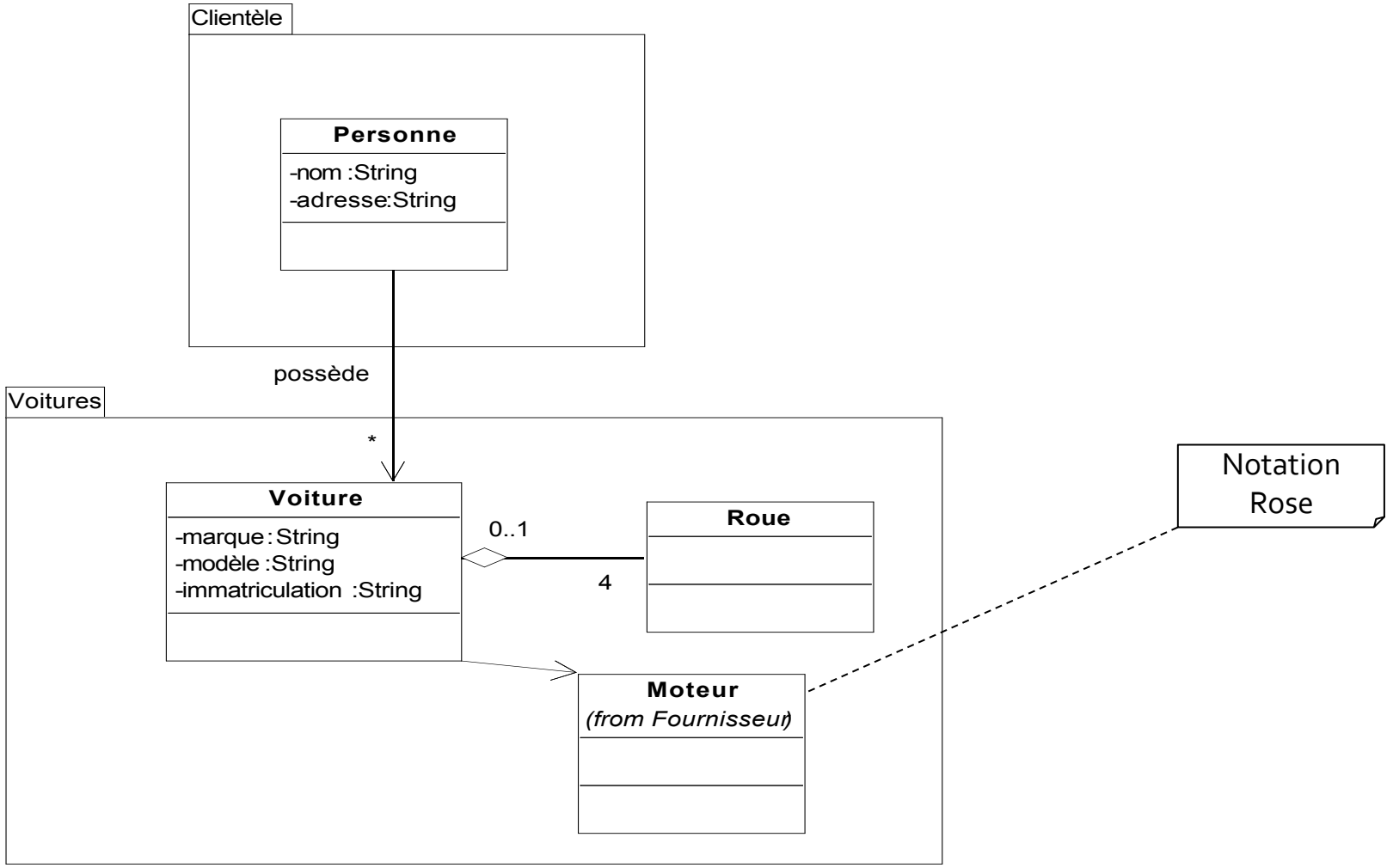
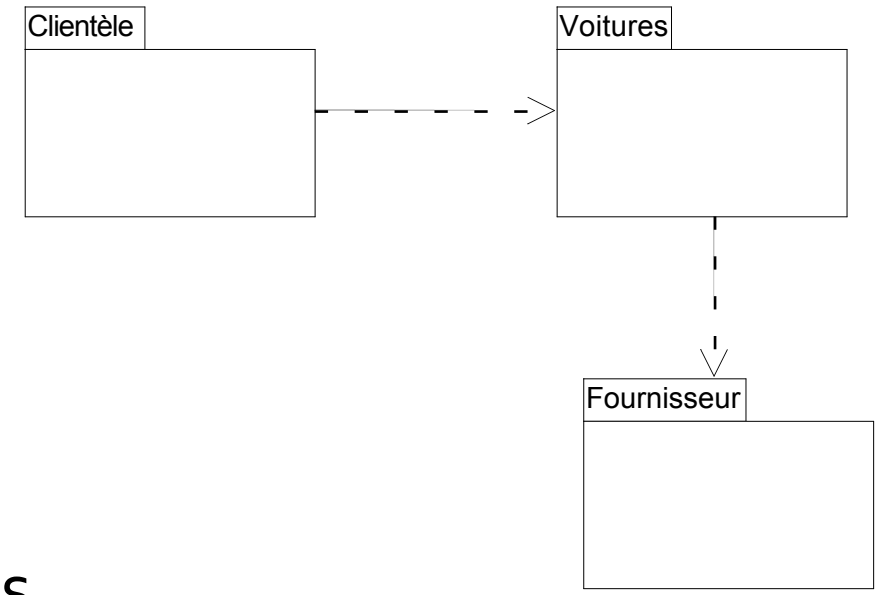


Diagramme de paquetage



Dépendances entre paquetages

- Découlent des dépendances entre éléments des paquetages
  - notamment les classes
- Les dépendances ne sont pas transitives
  - modifier Fournisseur n'oblige pas obligatoirement à modifier Clientèle



Utilisation des diagrammes de paquetages

- Organisation globale du modèle mis en place
  - hiérarchies de paquetages contenant diagrammes et éléments
- Organisation des classes en paquetages pour
  - contrôler la structure du système
    - comprendre et partager
    - obtenir une application plus évolutive et facile à maintenir
      - ne pas se faire déborder par les modifications
      - viser la généricité et la réutilisabilité des paquetages
  - avoir une vue claire des flux de dépendances entre paquetages
    - il s'agit de les minimiser

Paquetage et nommage

- Noms pleinement qualifiés
  - équivalent à chemin absolu
    - ex. paquetage java::util, classe java::util::Date
- Stéréotypes de dépendance
  - « import » : les éléments passent dans l'espace de nommage
    - ex. classe Date depuis le paquetage qui importe
  - « access » : sont accessibles
    - ex. classe java::util::Date depuis le paquetage qui importe

Principes du découpage en paquetages

- Cohérence interne du paquetage : relations étroites entre classes
  - fermeture commune
    - les classes changent pour des raisons similaires
  - réutilisation commune
    - les classes doivent être réutilisées ensemble
- Indépendance par rapport aux autres paquetages
- Un paquetage d'analyse contient généralement moins de 10 classes

Bien gérer les dépendances

- Les minimiser pour maintenir un *couplage faible* (voir patterns)
  - dépendances unidirectionnelles
    - cf. associations navigables
  - pas de cycles de dépendances
    - ou au moins pas de cycles inter-couches
  - stabilité des dépendances
    - plus il y a de dépendances entrantes, plus les interfaces de paquetage doivent être stables

Paquetages : remarques variées

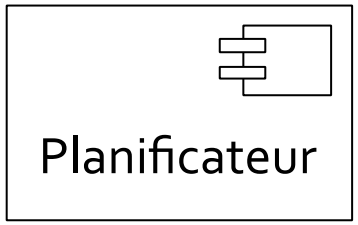
- Les paquetages peuvent être considérés comme
  - soit simples regroupements
  - soit des véritables sous-systèmes opérationnels
    - comportement + interfaces
- Paquetage vu de l'extérieur
  - classe publique gérant le comportement externe (cf. pattern Façade)
  - interfaces
- Pour un Paquetages utilisé partout (très stable)
  - mot-clé « global »
- Utilité pratique d'un paquetage Commun
  - regrouper les concepts largement partagés, ou épars
- Lien entre paquetages et couches (niveaux)
  - une couche est composée de paquetages

Plan

- Diagrammes de classes
- Diagrammes d'objets
- Diagrammes de paquetages
- **Diagrammes de composants**
- Diagrammes de déploiement

Composants

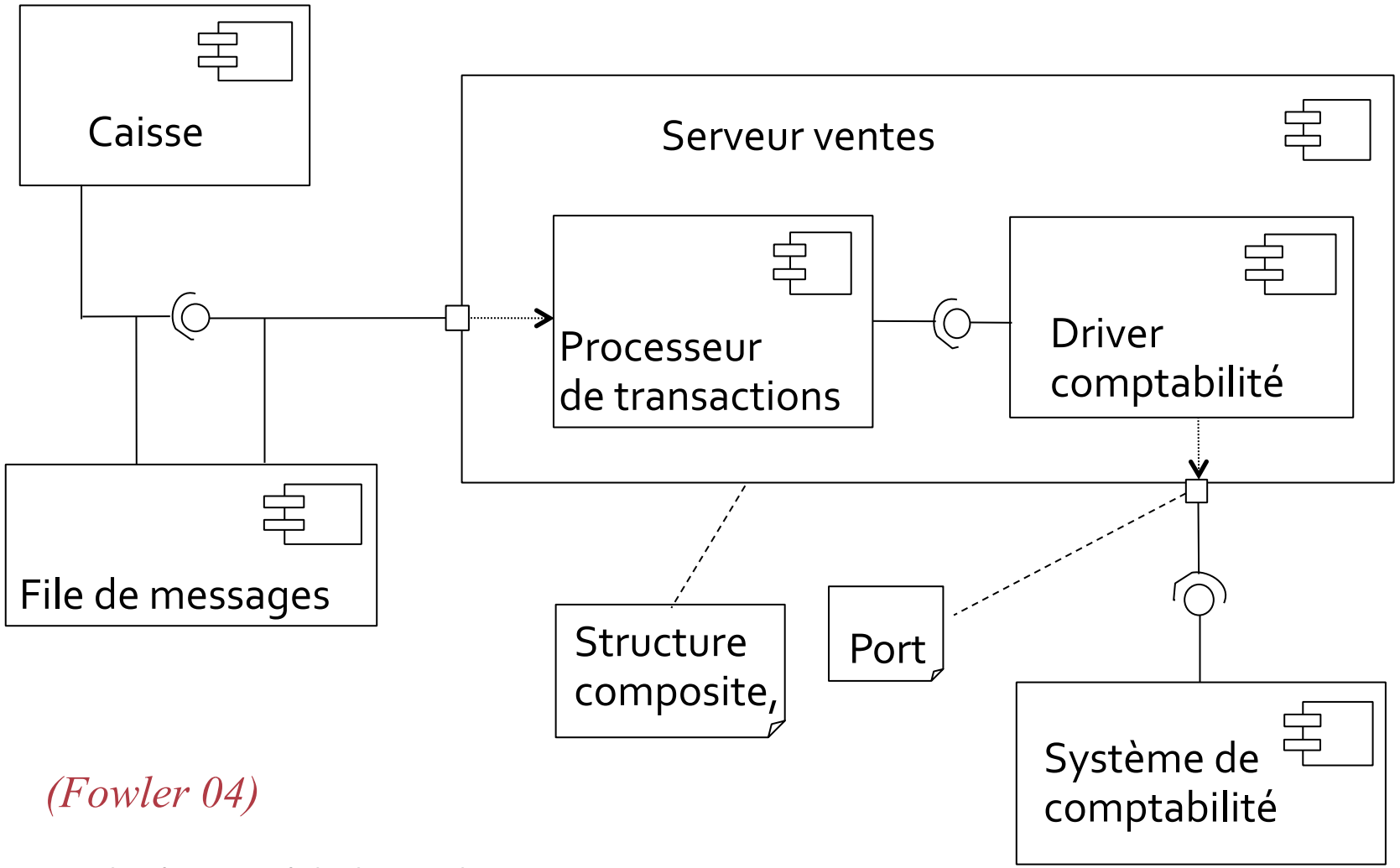
- Partie modulaire de conception d'un système qui masque son implémentation derrière un jeu d'interfaces externes
  - partie remplaçable d'un système qui se conforme à des interfaces et fournit la réalisation de ces interfaces
  - doit être compris comme un élément qu'on peut acheter, associer à d'autres composants (cf. HiFi)
- Division en composants = décision technique et commerciale (Fowler)
- Représentation



Diagrammes de composants

- Objectif
  - représenter l'organisation et les dépendances entre les composants logiciels
  - décrire les composants et de leurs relations dans le système en construction
- Diagramme de composants
  - inclusion des composants
  - relations de fourniture et d'utilisation d'interfaces

Diagramme de composants



(Fowler 04)

Remarque

- UML1 : composant = n'importe quel élément, y compris fichiers, bibliothèque, etc.
- UML2 : utiliser les artefacts (voir plus loin) pour représenter des structures physiques (jar, dll…)

→ attention : vérifier quelle syntaxe est utilisée dans les diagrammes de composants que vous avez sous les yeux

Plan

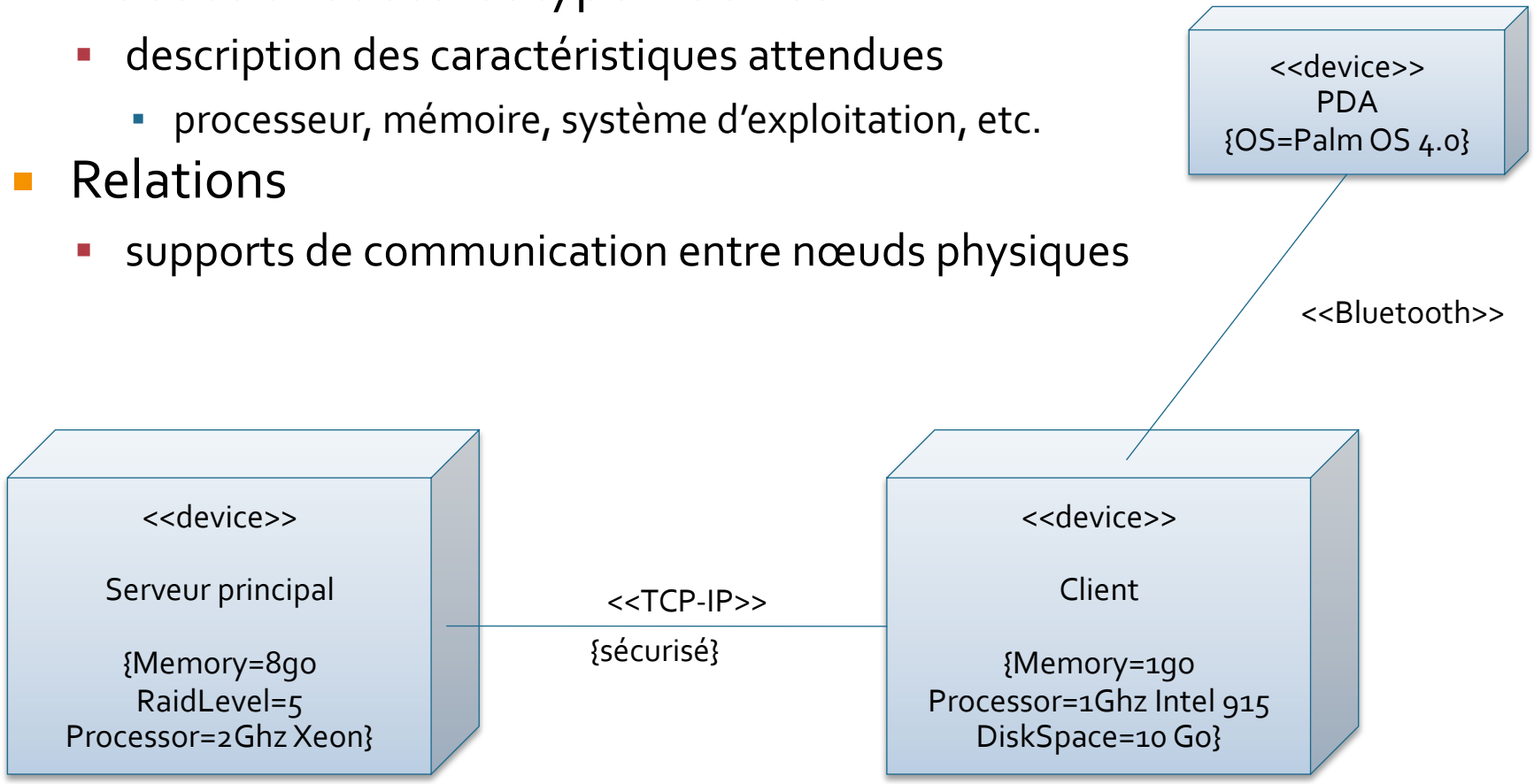
- Diagrammes de classes
- Diagrammes d'objets
- Diagrammes de paquetages
- Diagrammes de composants
- **Diagrammes de déploiement**

Diagramme de déploiement

- Objectif
  - expliquer comment un système décrit statiquement dans un modèle sera concrètement déployé sur une architecture physique distribuée
  - Modéliser l'environnement d'exécution d'un projet
- Pour cela
  - rendre compte de la disposition physique des différents éléments matériels qui entrent dans la composition d'un système
  - rendre compte de la disposition des programmes exécutables et des composants sur ces matériels

Représentation de l'environnement matériel de déploiement

- Noeuds avec stéréotype « device »
  - description des caractéristiques attendues
    - processeur, mémoire, système d'exploitation, etc.
- Relations
  - supports de communication entre nœuds physiques

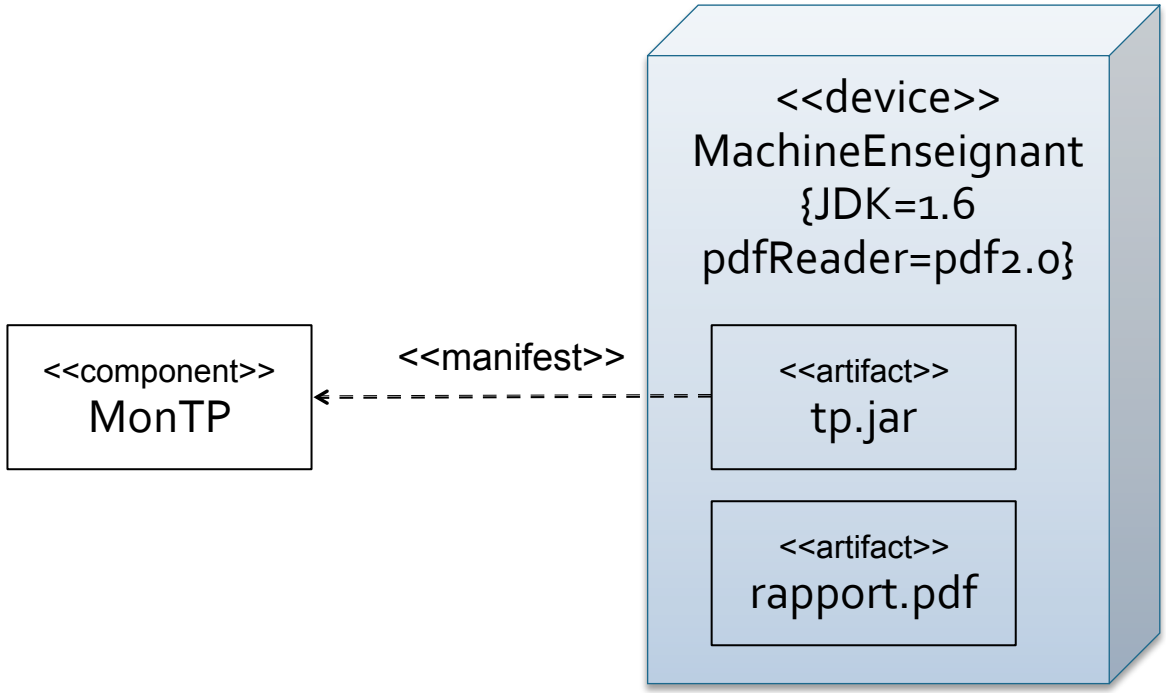


Représentation du logiciel déployé

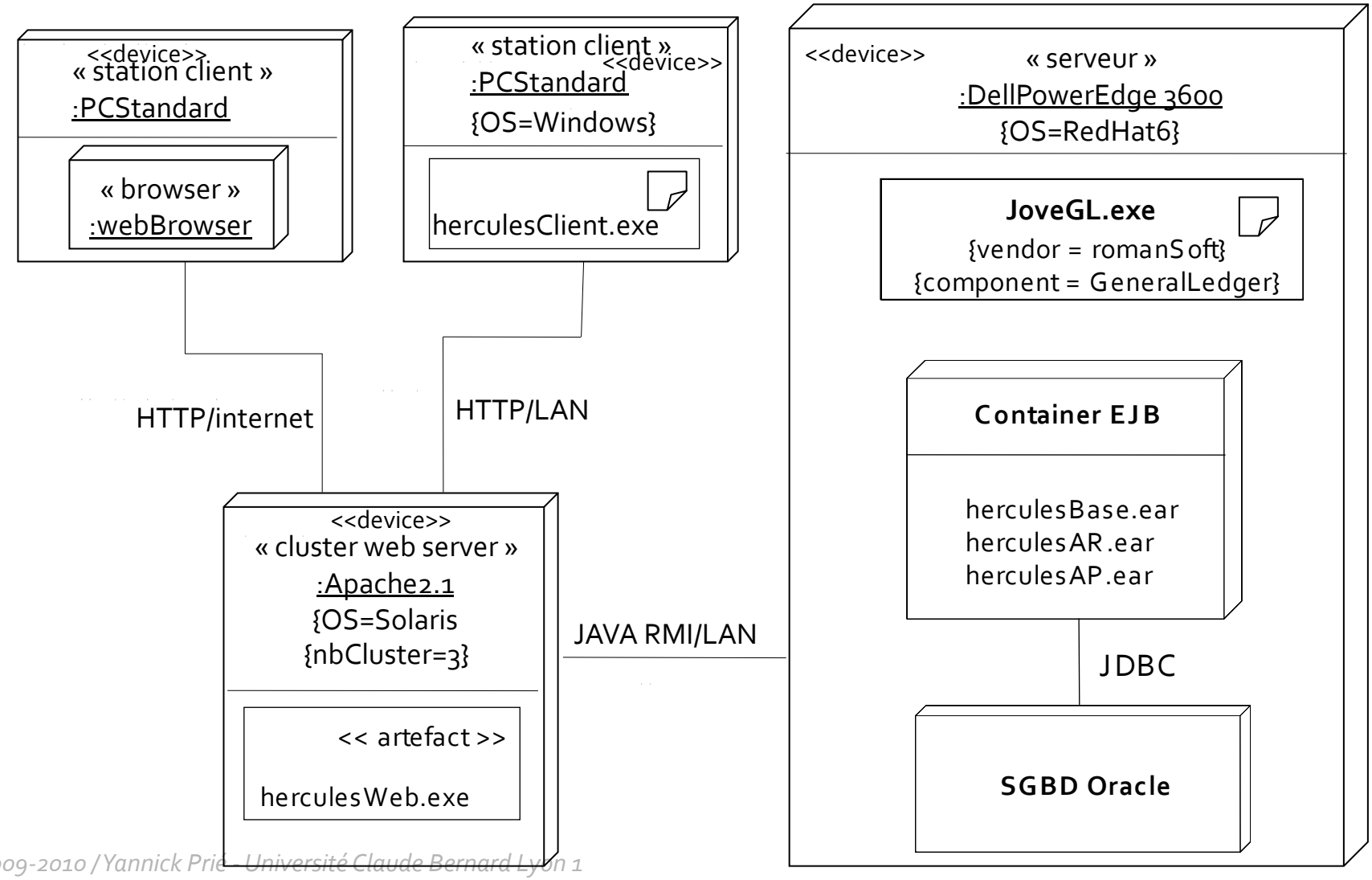
- Artefact = élément d'information impliqué dans le système
  - Fichiers, logiciel, modèle
    - e.g. fichier de configuration, bibliothèque, exécutable, script, table de BD, etc.
  - un artefact est la manifestation (manifest) d'un élément du modèle
    - e.g. une classe (élément du modèle) a pour manifestations un fichier source .java, un fichier compilé .class et un fichier .html lié à la javadoc.

Diagramme de déploiement

- Les artefacts sont déployés sur les nœuds de l'environnement matériel

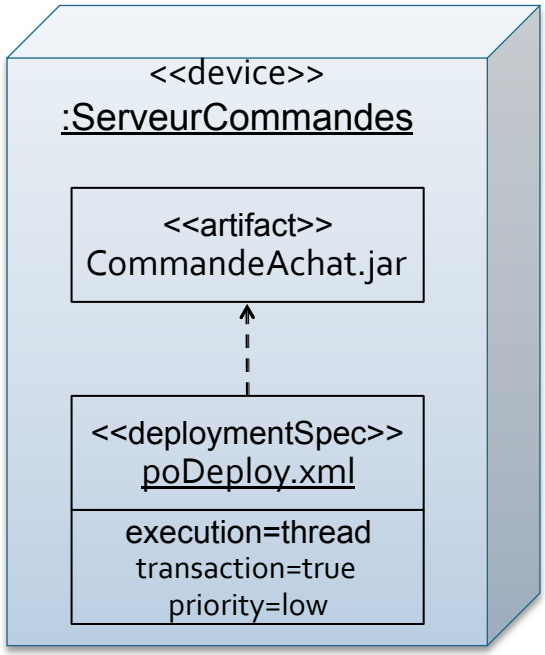


Exemple de diagramme de composants



Déploiement : raffinements

- Les nœuds et les artefacts sont instanciables
  - possibilité de réfléchir à un déploiement abstrait (nœuds, artefacts) et à un déploiement concret (instances de nœuds, instances d'artefacts)
  - exemple
    - tel programme de telle version compilée à telle date tourne tourne sur telle machine physique réelle
- Possibilité de spécifier des caractéristiques du déploiement d'un artefact



A suivre

- UML 3/4 : diagrammes dynamiques et d'interactions
- UML 4/4 : concepts avancés