Modélisation des structures élémentaires

Classes

- Les classes constituent les briques de base les plus importantes d'un système orienté objet.
- Une classe est la description d'un ensemble d'objets qui partagent les mêmes attributs, les mêmes opérations, les mêmes relations et la même sémantique.
- Une classe implémente une ou plusieurs interfaces.
- Les classes sont utilisées pour capturer le vocabulaire du système que l'on développe.
- Elles peuvent contenir des abstractions du domaine étudié ainsi que des classes pour la réalisation.
- On peut utiliser les classes pour représenter
  - des éléments logiciels,
  - des éléments matériels
  - et même des objets purement conceptuels.
- Les classes bien structurées ont des frontières clairement délimitées et forment une répartition équilibrée des responsabilités au sein du système.

Allouer des responsabilités aux classes

Allouer des responsabilités aux classes

- Une responsabilité est un contrat ou une obligation qu'une classe doit respecter.
- Chacune des exigences fonctionnelles doivent être attribuées à l'une des classes
  - Toutes les responsabilités d'une classe doivent être reliées entre elles.
  - Si une classe a trop de responsabilités, elle devrait être scindée en plusieurs classes
  - Si une classe a aucune responsabilité, alors celle-ci est probablement inutile
  - Lorsqu'une responsabilité ne peut être attribués à aucune classe, alors c'est qu'une nouvelle classe devrait être introduite
- La meilleur façon de modéliser une classe est de définir les responsabilités des éléments dans le vocabulaire. Les techniques comme
  - les cartes CRC
  - Les cas d'utilisation sont très utiles.

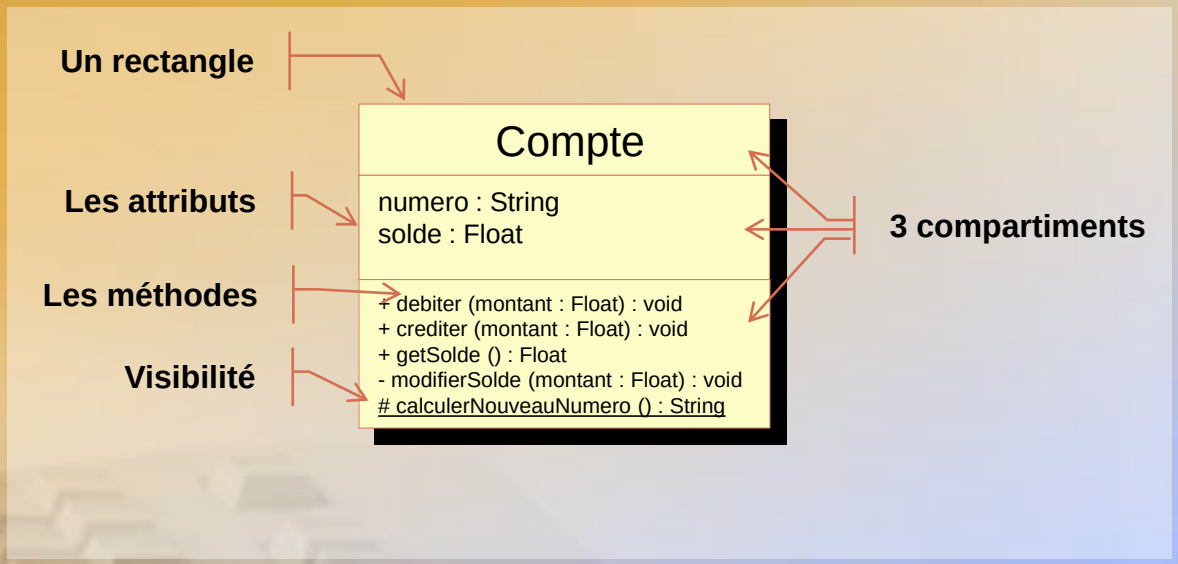
Catégories de responsabilités

- **Catégories de responsabilités**
  - **Fixer et obtenir la valeur d'un attribut**
  - **Créer et initialiser de nouvelle instances**
  - **Sauvegarder et récupérer de l'information persistante**
  - **Détruire des instances**
  - **Ajouter et détruire des liens**
  - **Copier, convertir, transformer, transmettre, afficher**
  - **Calculer des résultats numériques**
  - **Naviguer et rechercher**
  - **Tout autre tâche…**

Classe : Trucs et astuces

- **Une classe bien structurée:**
  - **fournit une abstraction claire de quelque chose tiré du vocabulaire du domaine du problème ou de celui de la solution;**
  - **comprend un petit ensemble de responsabilités bien définies et les réalise parfaitement;**
  - **fournit une séparation nette entre les spécifications de l'abstraction et son implémentation;**
  - **est compréhensible et simple tout en étant extensible et adaptable.**
  - **Lorsqu'on représente une /classe avec UML:**
  - **seules les propriétés de la classe essentielles à la compréhension de l' abstraction dans son contexte sont représentées;**
  - **de longues listes d'attributs et d'opérations sont organisées en fonction de leurs catégories;**
  - **les classes en relation sont représentées sur les mêmes diagrammes de classes.**

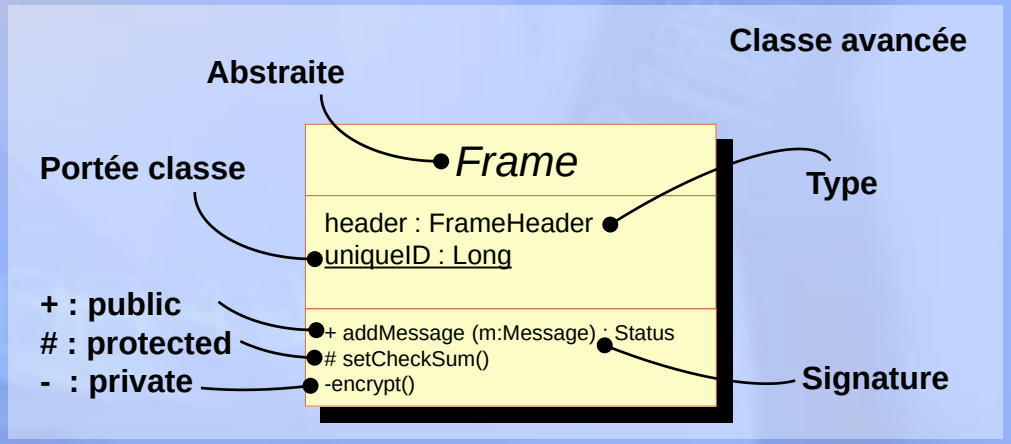
Classe : représentation UML



+ public : tout classificateur extérieur ayant une visibilité sur le classificateur donné peut utiliser cette caractéristique indiquée par le symbole +.

 # protected : seuls les descendants du classificateur peuvent utiliser cette caractéristique indiquée par le symbole #.

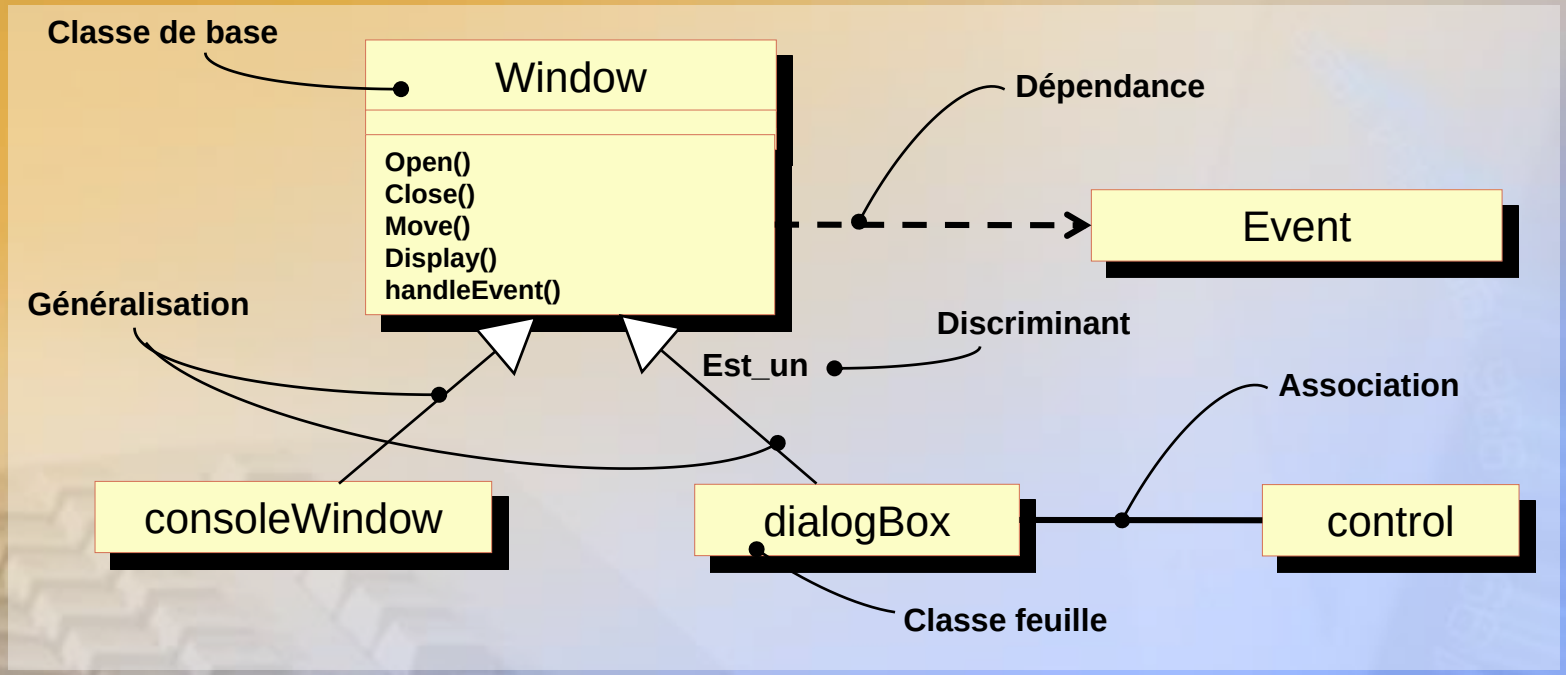
- private : le classificateur, et lui seul, peut utiliser cette caractéristique indiquée par le symbole -



Les relations

- La construction d'abstractions enseigne que très peu de classes sont isolées mais qu'au contraire la plupart d'entre elles collaborent avec d'autres classes de différentes manière.
- Modéliser un système nécessite
  - identification des éléments qui composent son vocabulaire,
  - mais également la compréhension des relations qui s'établissent entre eux.
- En modélisation orientée objet, trois sortes de relations sont particulièrement importantes:
  - les *dépendances,* c'est-à-dire les relations d'utilisation entre les classes (comme les relations de raffinement, de trace et de liaison),
  - les *généralisations,* qui relient les classes générales à leurs spécialisations,
  - et les *associations,* qui décrivent les relations structurelles entre objets. Toutes ces relations proposent une manière différente de combiner des abstractions.
- Si elle est trop élaborée, l'écheveau des relations rend le modèle incompréhensible.
- À l'inverse, si elle est trop laxiste, une bonne partie de la richesse du système, à savoir la manière dont les éléments collaborent, n'est pas exploitée.

Représentation



- Une relation est une connexion sémantique entre des éléments.
- Une relation est représentée par une ligne.
- Chaque sorte de relation est représentée par un type de ligne différent.

Dépendance et généralisation

- **Dépendance**
  - Relation d'utilisation qui établit qu'un changement de spécification d'un élément (par exemple, la classe Event) peut en affecter un autre qui l'utilise (par exemple, la classe Window), mais que l'inverse n'est pas nécessairement vrai.

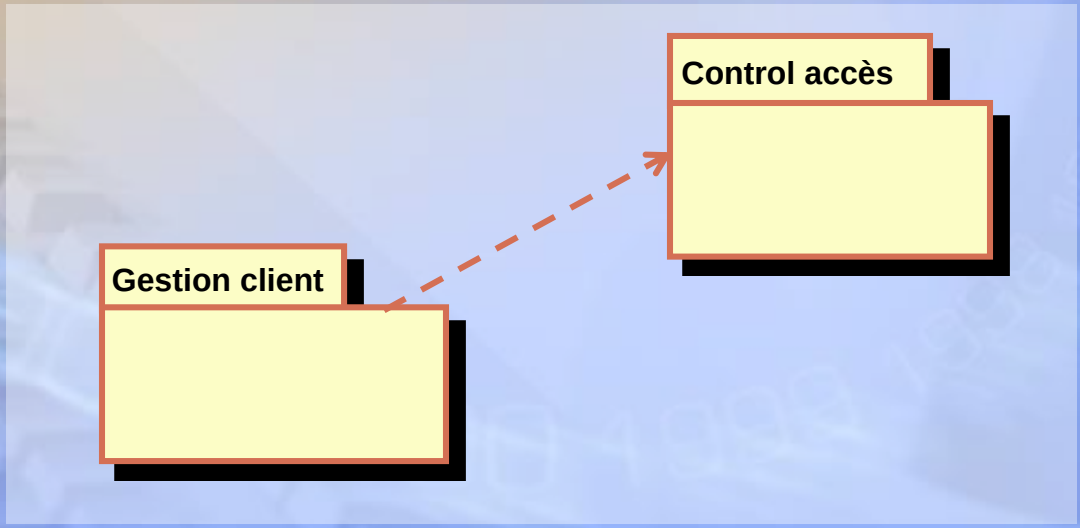
- **Généralisation**
  - relation entre un élément général (appelé super-classe ou mère) et un élément dérivé de celui-ci, mais plus spécifique (désigné par le terme sous-classe ou fille).
  - La généralisation est parfois qualifiée de relation "est_une_sorte_de" :
  - La généralisa!ion implique que des objets de l'enfant puissent être utilisés partout où le parent apparaît, mais pas l'inverse.
  - En d'autres termes, elle indique que l'enfant peut remplacer le parent. L'enfant hérite des propriétés du parent, en particulier de ses attributs et de ses opérations mais, le plus souvent, ceux-ci s'ajoutent à ses propres attributs et à ses propres opérations. Lorsqu'un enfant a une opération dont la signature est la même que celle d'une opération du parent, l'opération de l'enfant prévaut sur celle du parent.
  - Ce phénomène est connu sous le nom de polymorphisme. Comme le

Association

- Une *association* est une relation structurelle qui précise que les objets d'un élément sont reliés aux objets d'un autre élément.
- En reliant deux classes, elle autorise la navigation d'un objet de l'une d'elles à un objet de l'autre, et vice versa.
- Il n'est pas interdit que les deux extrémités de ce genre de relation forment une boucle et se rattachent à la même classe. Cela signifie qu'un objet qui appartient à cette classe peut être connecté à d'autres objets de cette même classe.
- On appelle "association binaire" une association qui relie seulement deux classes entre elles.
- Une association qui relie plus de deux classes, ce qui est moins fréquent, est appelée "association n-aire".
- Une association est représentée par une ligne pleine qui relie une classe à d'autres classes ou à elle-même.
- On utilise les associations pour montrer les relations structurelles.
- 5 décorations s'appliquent aux associations.
    1. Le nom
    2. Le rôle
    3. La multiplicité
    4. La navigabilité
    5. Agrégation

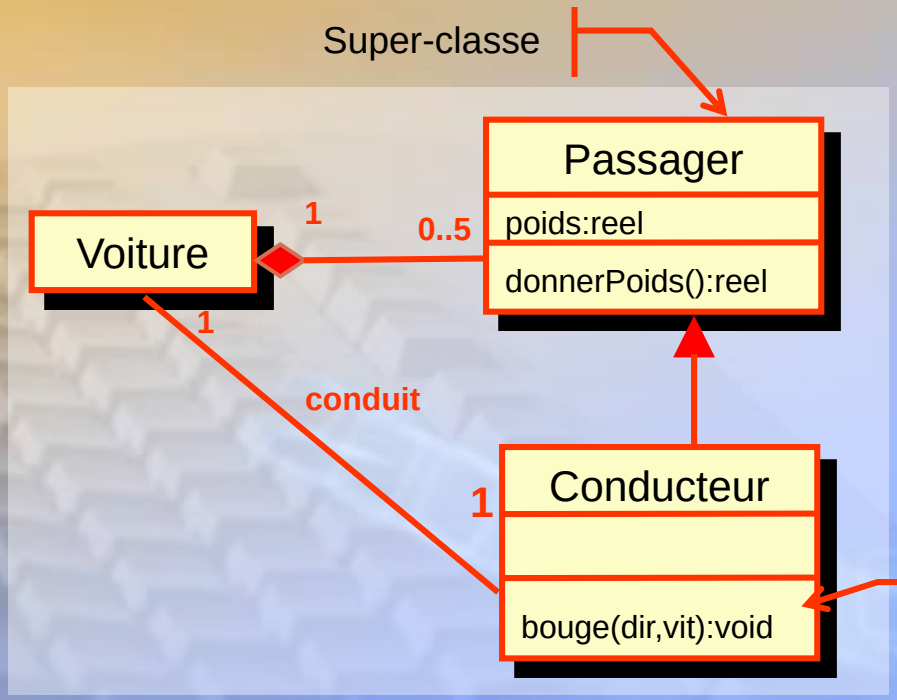
Analyser et valider les dépendances

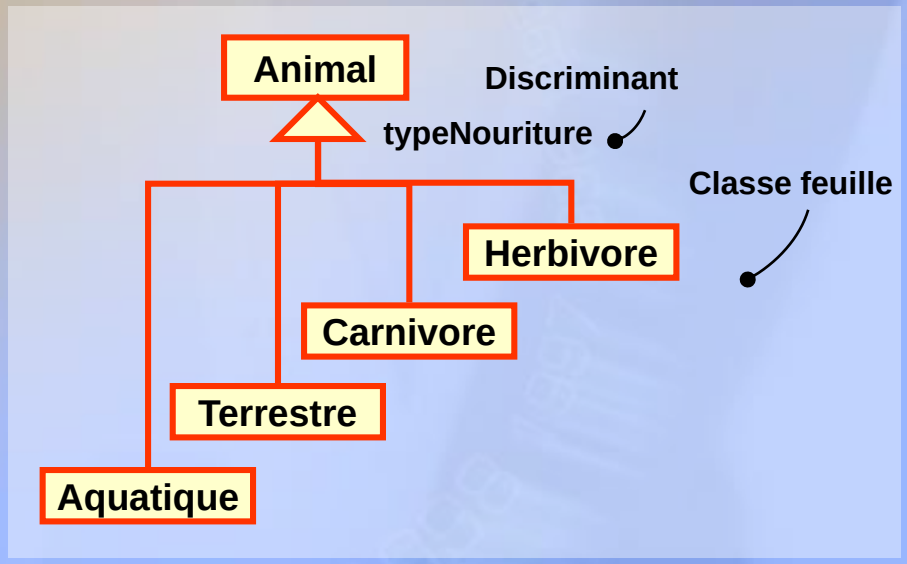
- Relation client - serveur
- Exprime le besoin d'un service
- Sans savoir si le serveur existe et surtout quelle forme il a



Analyser et valider généralisation

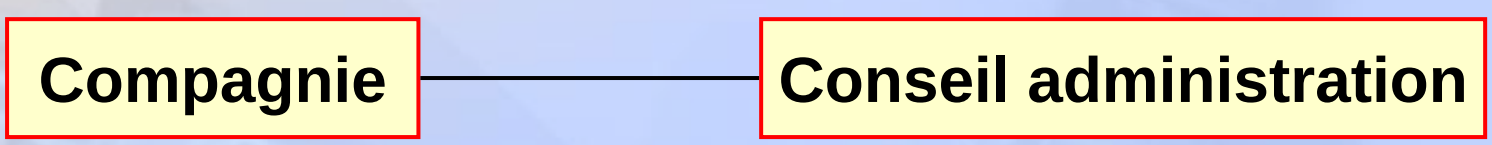
- **Une super-classe se spécialise en sous-classes**
  - **Le *discriminant* est une étiquette décrivant le critère suivant lequel se base la spécialisation**





Analyser et valider les associations (1..1)

- Une à une
  - **A chaque compagnie est associé un conseil d'administration**
  - **Un conseil d'administration gère une seule compagnie**
  - **Une compagnie doit avoir un conseil d'administration**
  - **Un conseil d'administration est toujours attaché à une et une seule compagnie**

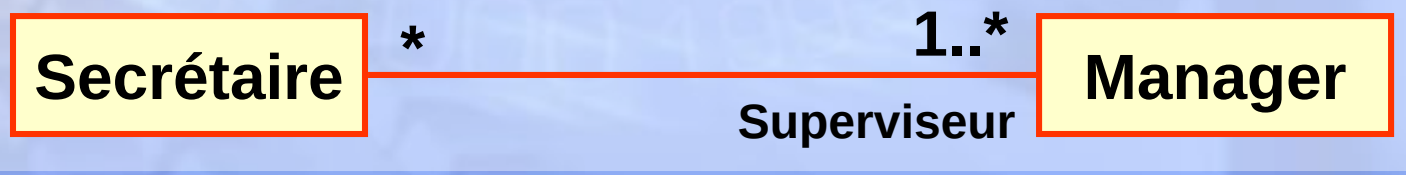


Analyser et valider les associations (1..*)

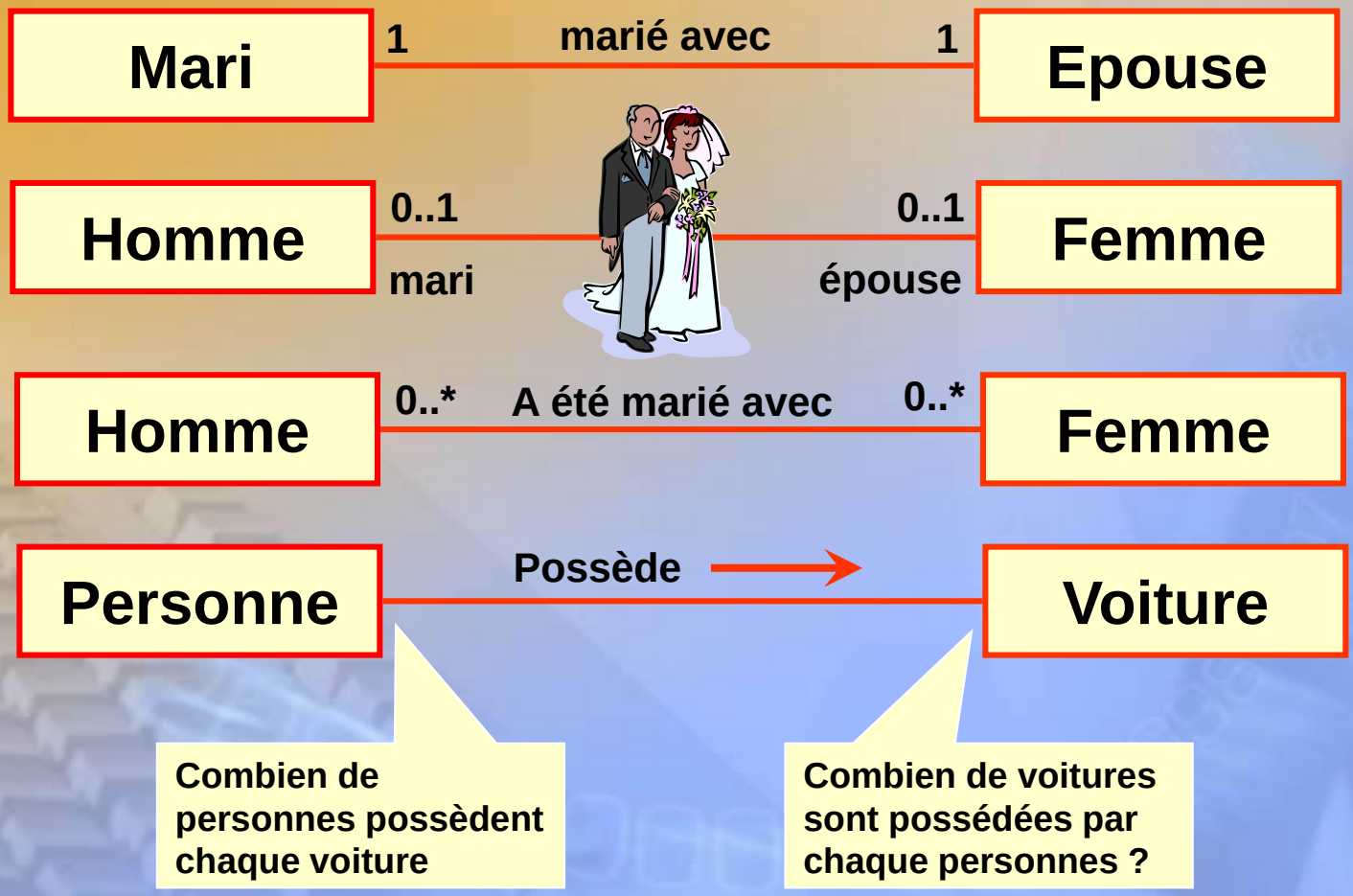
- Une à plusieurs
  - **Une compagnie a plusieurs employés**
  - **Un employé ne peut travailler que pour une seule compagnie**
    - **Qu'en est-il des employés occupant un double emploi!**
  - **Une compagnie peut n'avoir aucun employé**
  - **Un employé associé à une compagnie travaille pour cette compagnie**

Analyser et valider les associations (*..*)

- Plusieurs à plusieurs
  - **Un(e) secrétaire peut travailler pour plusieurs superviseurs**
  - **Un superviseur peut avoir plusieurs secrétaires**
  - **Les secrétaires peuvent travailler en équipes**
  - **Les superviseurs peuvent avoir recours à un groupe de secrétaires**
  - **Certains superviseurs peuvent n'avoir aucun(e) secrétaires**
  - **Est-il possible qu'un(e) secrétaire puisse se retrouver, ne serait-ce que temporairement, sans superviseur?**



Associations (exemples)



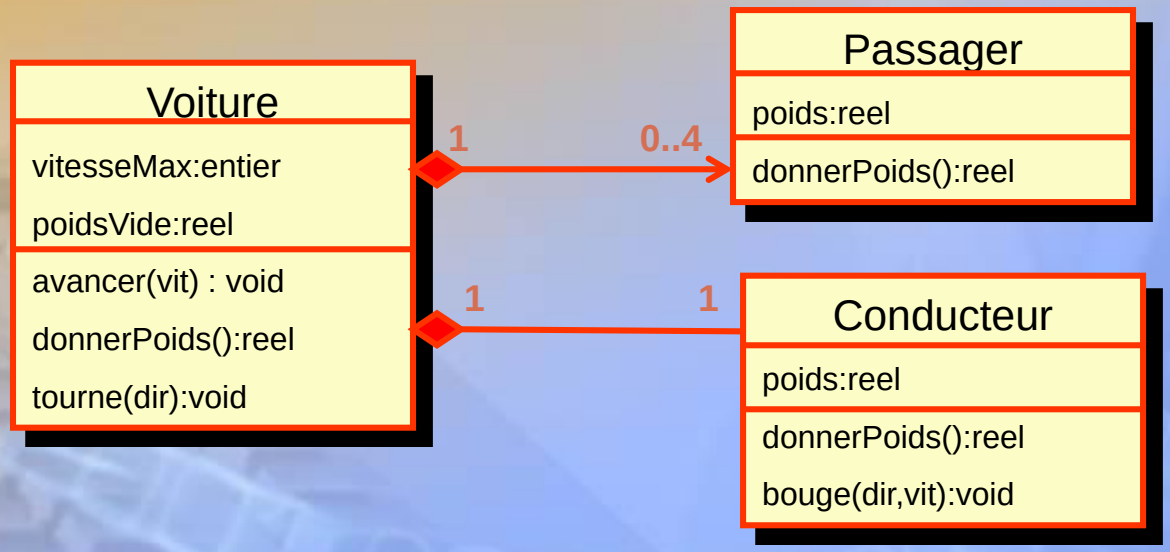
Quelques trucs

- **Une association devrait exister si une classe**
  - ***possède***
  - ***contrôle***
  - ***est connecté à***
  - ***est relié à***
  - ***est une partie de***
  - ***est fait de parties de***
  - ***est membre de***
  - ***a comme membres***

  une autre classe dans le modèle
- **Spécifier ensuite la multiplicité à chaque extrémité de l'association**
- **Étiqueter clairement cette association**

La navigabilité

- **Une flèche qui restreint la navigation dans un sens**



Agrégations

Association

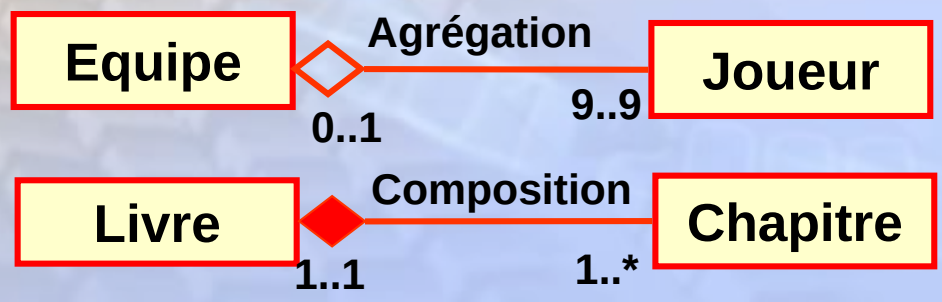
Les objets se connaissent et peuvent travailler ensemble

Agrégation

1. **Protège l'intégrité de la configuration**
2. **Fonctionne comme une entité unique**
3. **Contrôle par un objet – propagation vers le bas**

Composition

Chaque partie ne peut être membre que d'un seul agrégat.



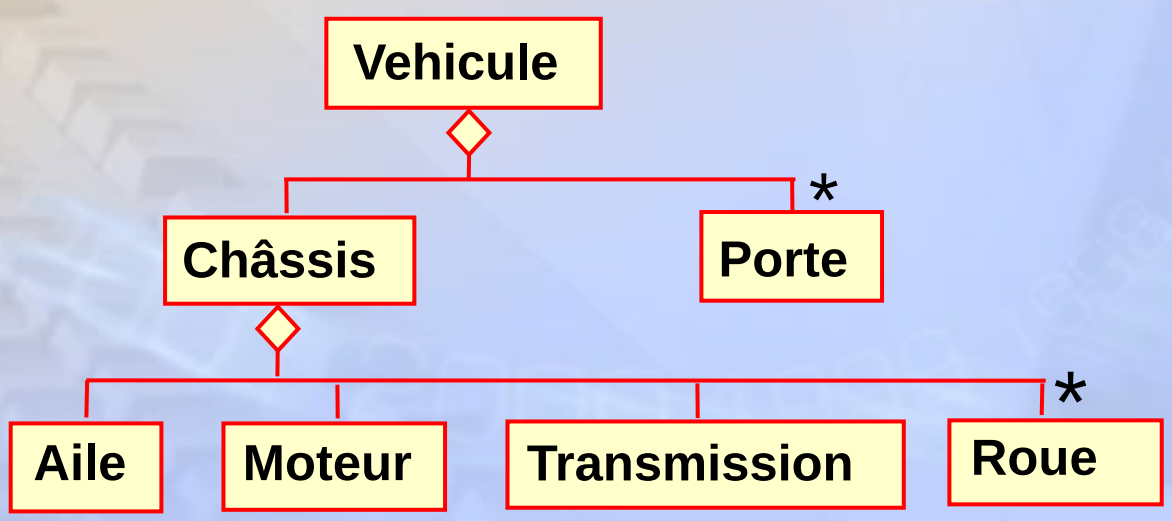
Une agrégation est une forme spéciale représentant une relation 'partie-tout'.

- **Le 'tout' est souvent appelé l'ensemble ou l'agrégat**
- **Le symbole désignant l'agrégation se place du côté de la parti**

Quand faut-il utiliser l'agrégation?

- **En général, une association peut être représentée comme une agrégation si:**
  - **il y a une relation de type *est-une-partie-de***
  - **il y a une relation de type *est-composé-de***
  - **Lorsque quelque chose contrôle l'agrégat, il contrôle aussi ses parties**

Une hiérarchie d'agrégation

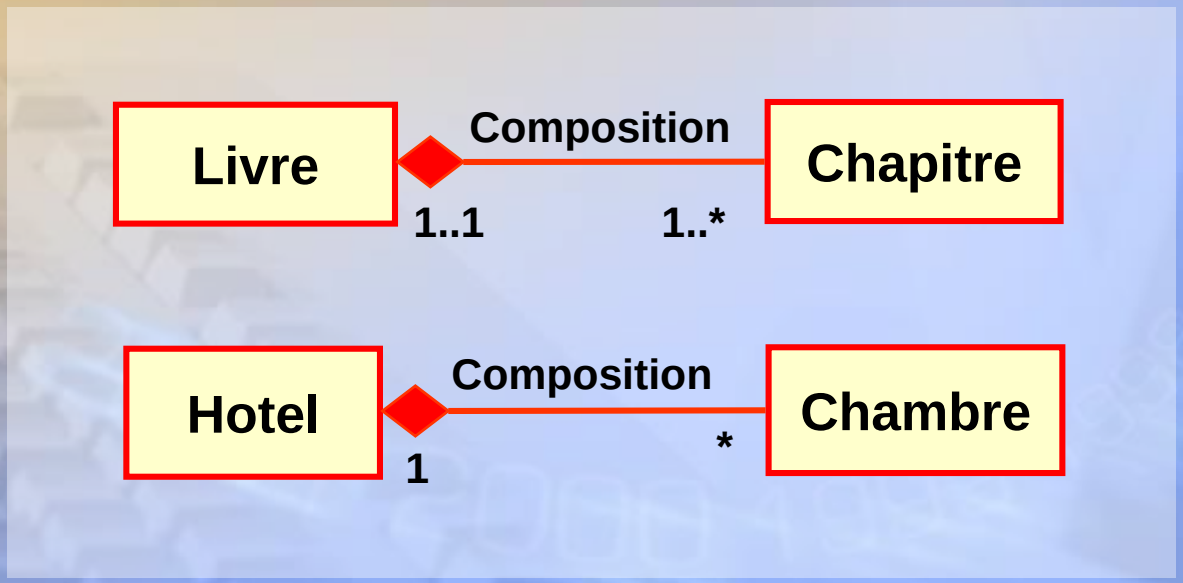


Propagation

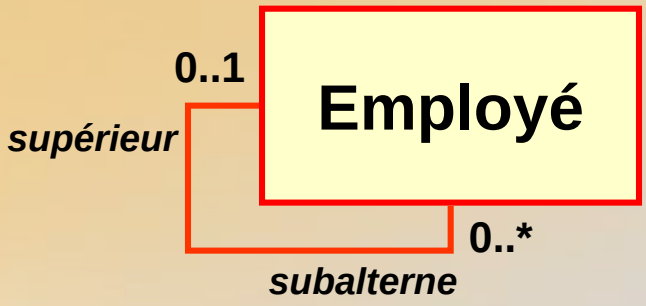
- La propagation est un mécanisme statuant que lorsqu'une opération est effectuée sur le tout elle doit aussi s'appliquer à ses parties
- La propagation permet aussi aux propriétés des parties de se propager vers le tout
- La propagation est à l'agrégation ce que l'héritage est à la généralisation.
  - La différence majeure est que:
    - L'héritage est un mécanisme implicite
    - La propagation doit être programmée

Composition

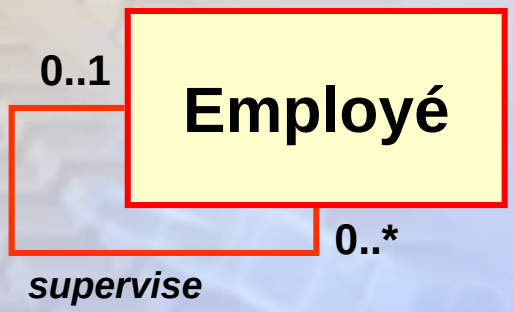
- Une *composition* est une forme forte d'agrégation
  - Si l'agrégat est détruit, alors ses parties le sont aussi



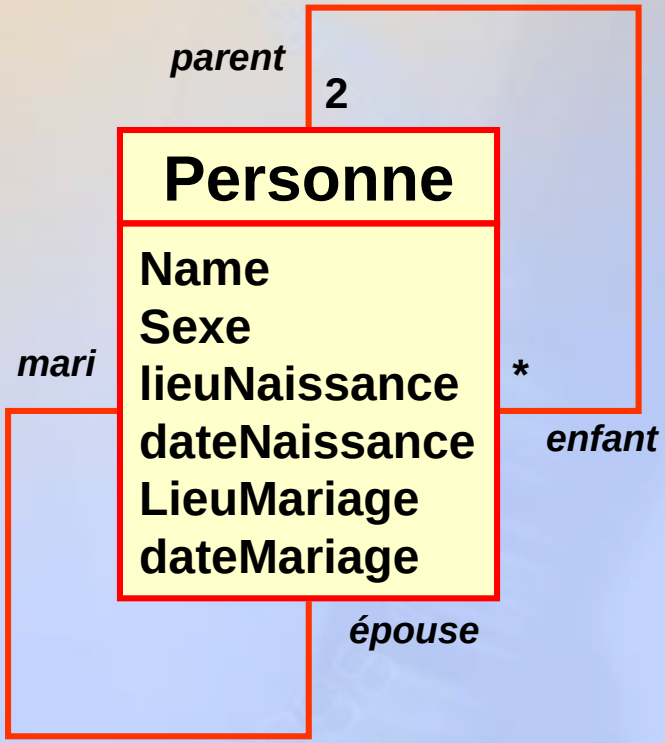
Associations réflexives



Utilisation de noms de rôles

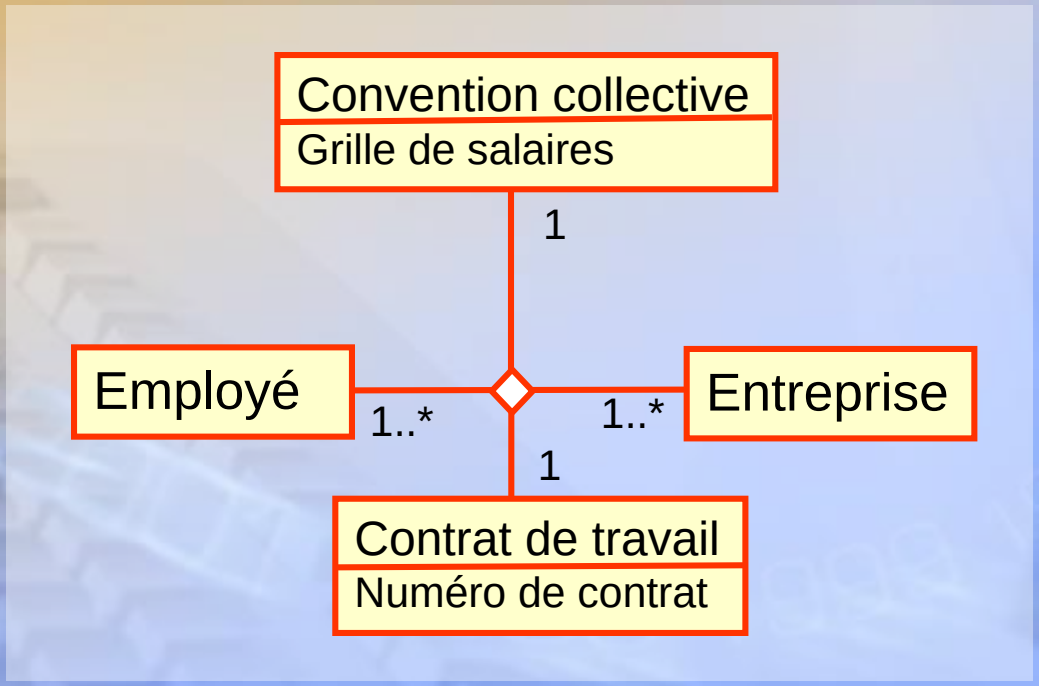


Utilisation d'un nom d'association



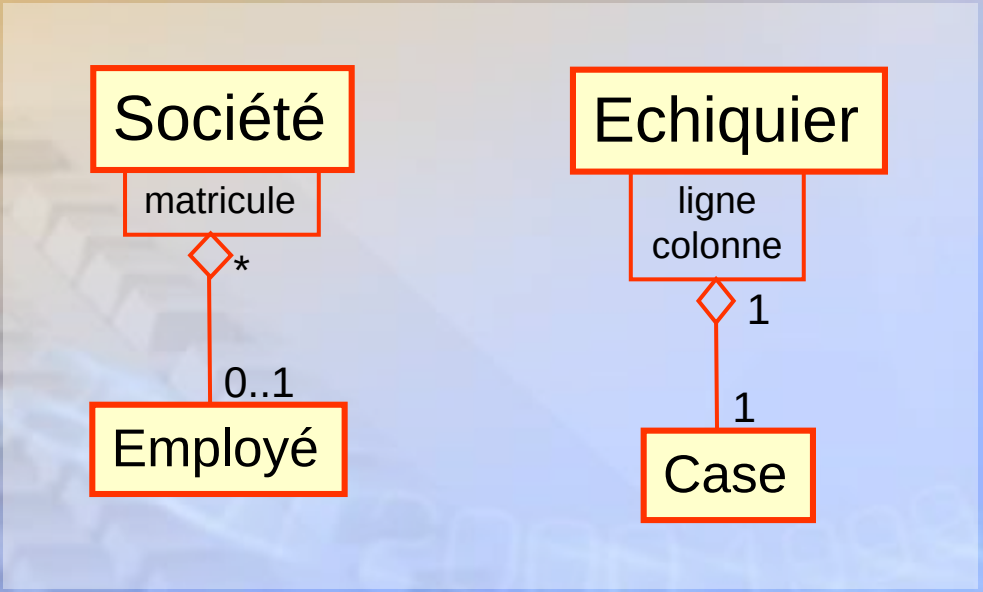
Associations n-aires

- **Permet de représenter des associations multi-membres**
- **A n'utiliser qu'en analyse**
- **Pose un problème de représentation pour la conception**



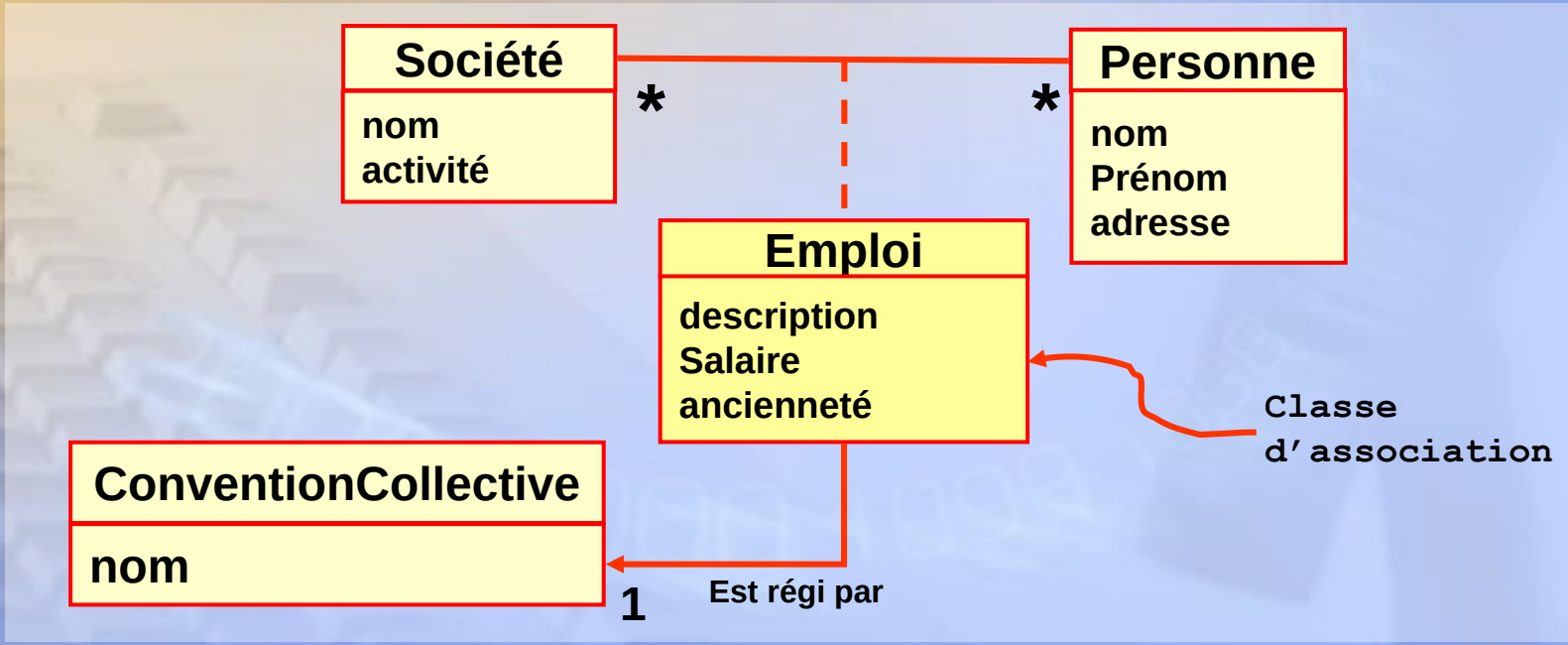
Association qualifiée

- **Attribut d'association dont les valeurs partitionnent l'ensemble des objets reliés à un objet à travers une association.**
  - **Exemple : le matricule permet d'identifier une personne dans une société.**
  - **Exemple : ligne et colonne permettent d'identifier une case sur un échiquier**



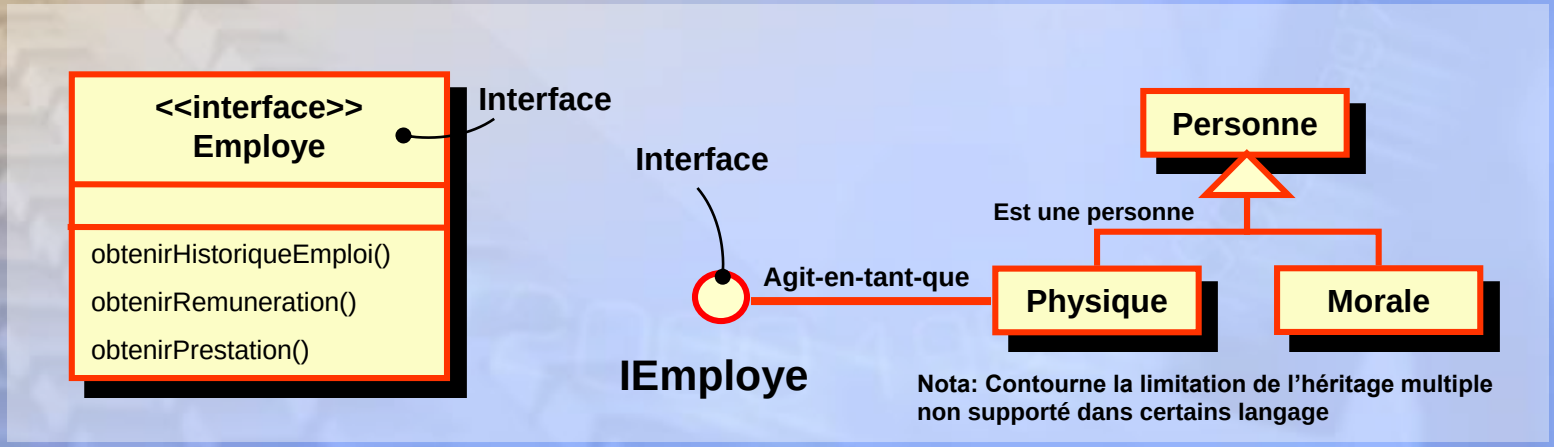
Classes-Associations

- Une classe-association est une association qui est aussi une classe.
- Les classes-associations sont utilisées lorsque les associations doivent porter des informations
- Il est toujours possible de se passer des classes-associations.



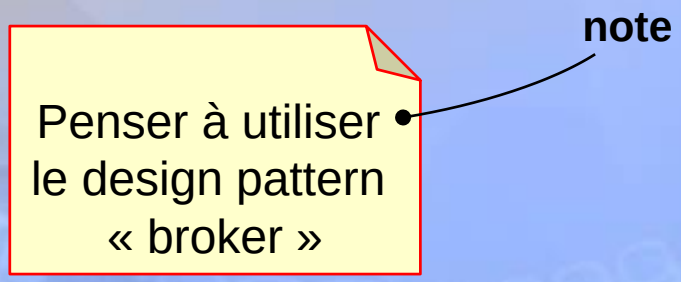
Les interfaces

- Une interface définit une limite entre la spécification des actions d'une abstraction et l'implémentation de la manière dont cette abstraction les exécute.
- Une interface est un ensemble d'opérations utilisées pour décrire un service d'une classe ou d'un composant.
- On utilise les interfaces pour :
  - visualiser,
  - spécifier, construire
  - documenter les points de soudure d'un système.
- Une interface bien structurée apporte une séparation nette entre les vues externes et internes d'une abstraction.



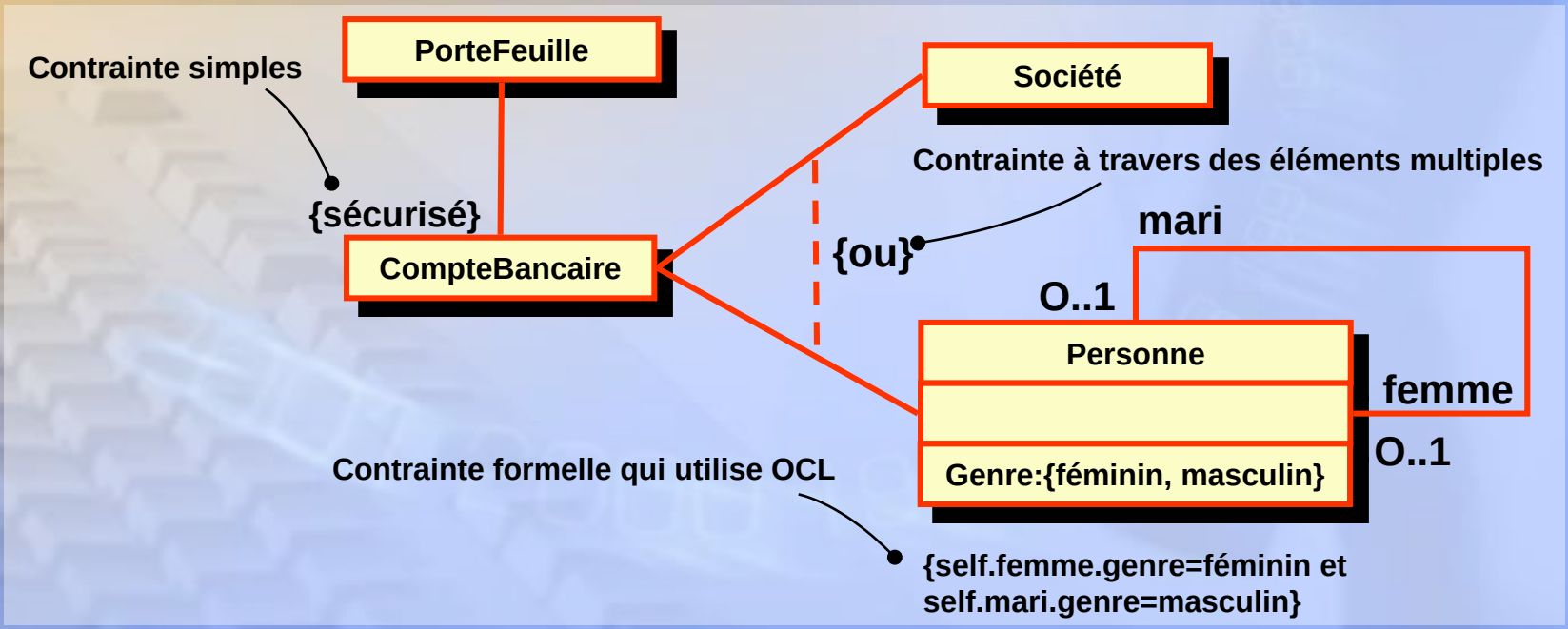
Notes

- **Les notes**
  - **Les notes constituent les décorations indépendantes les plus importantes.**
  - **Ce sont des symboles graphiques utilisés pour représenter les contraintes et les commentaires rattachés à un élément ou à un ensemble d'éléments.**
  - **Elles sont utilisées pour ajouter des informations à un modèle, comme des exigences, des observations, des révisions ou des explications.**



Contraintes

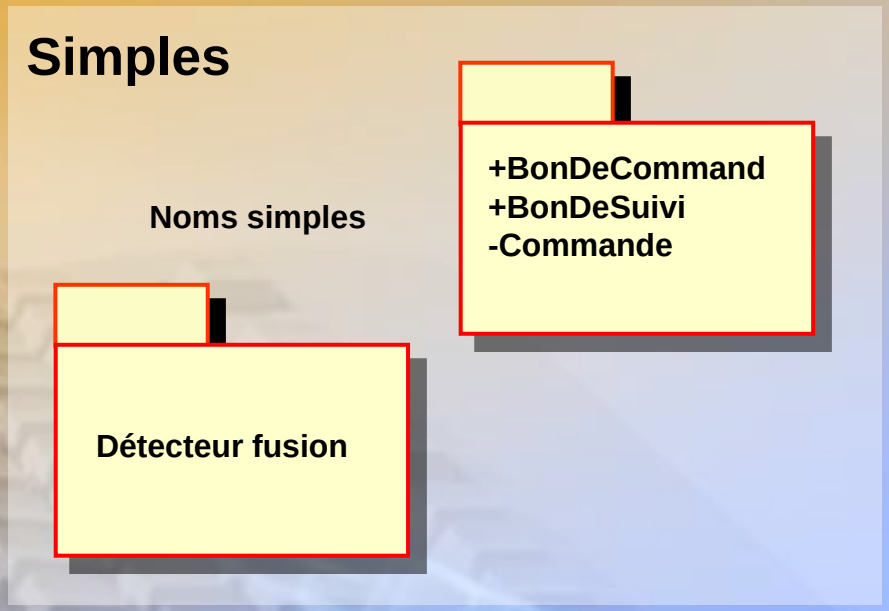
- **Contrainte**
  - **En UML, chaque élément possède sa propre sémantique.**
  - **Les contraintes permettent d'ajouter de nouvelles règles sémantiques ou de changer celles qui existent. Elles précisent des conditions indispensables pour que le modèle soit correctement formé.**

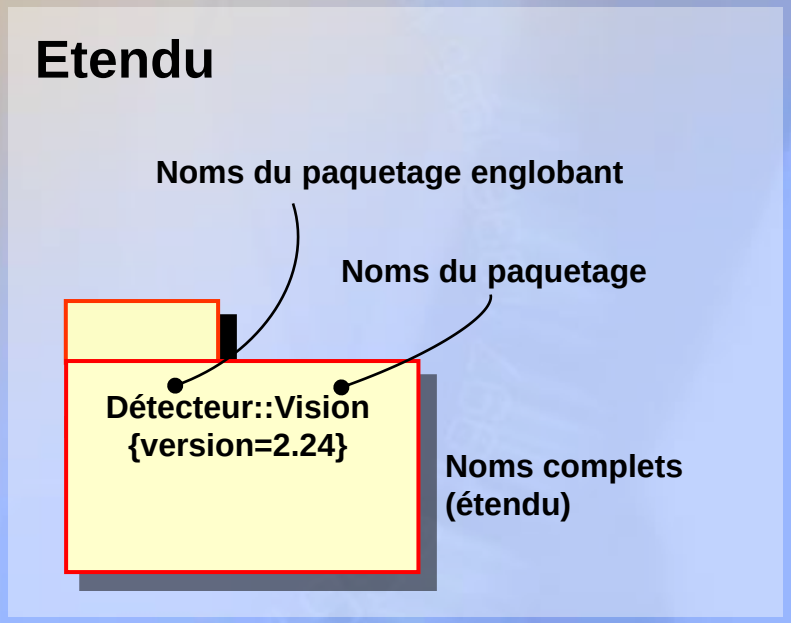


Paquetages

- La visualisation, la spécification, la construction et la documentation de grands systèmes impliquent la manipulation d'un nombre potentiellement élevé
  - de classes,
  - d'interfaces,
  - de composants,
  - de nœuds,
  - de diagrammes et autres éléments.
- En UML, un paquetage est un mécanisme générique pour organiser des éléments de modélisation en groupes.
- Grâce aux paquetages, il est possible
  - de manipuler un ensemble d'éléments comme un groupe.
  - On peut contrôler la visibilité des éléments au sein du paquetage,
  - On peut également utiliser les paquetages pour présenter différentes vues de l'architecture d'un système.
- Les paquetages bien conçus regroupent des éléments proches sur le plan sémantique et qui ont tendance à évoluer ensemble.
- Un paquetage est un mécanisme d'intérêt général permettant d'organiser les éléments en groupes.
  - Ils permettent de disposer les éléments des modèles de manière à en faciliter la compréhension
  - de contrôler l'accès aux contenus afin de pouvoir surveiller les points de soudure dans l'architecture du système.

Paquetages simples et étendu





Instances

- Les termes " instance " et " objet " sont largement synonymes et peuvent se substituer l'un à l'autre dans la plupart des cas.
- Une instance est une manifestation concrète d'une abstraction à laquelle un ensemble d'opérations peut être appliqué et qui peut avoir un état qui en stocke les effets.
- On utilise les instances pour modéliser des éléments concrets ou prototypes qui existent dans le monde réel.

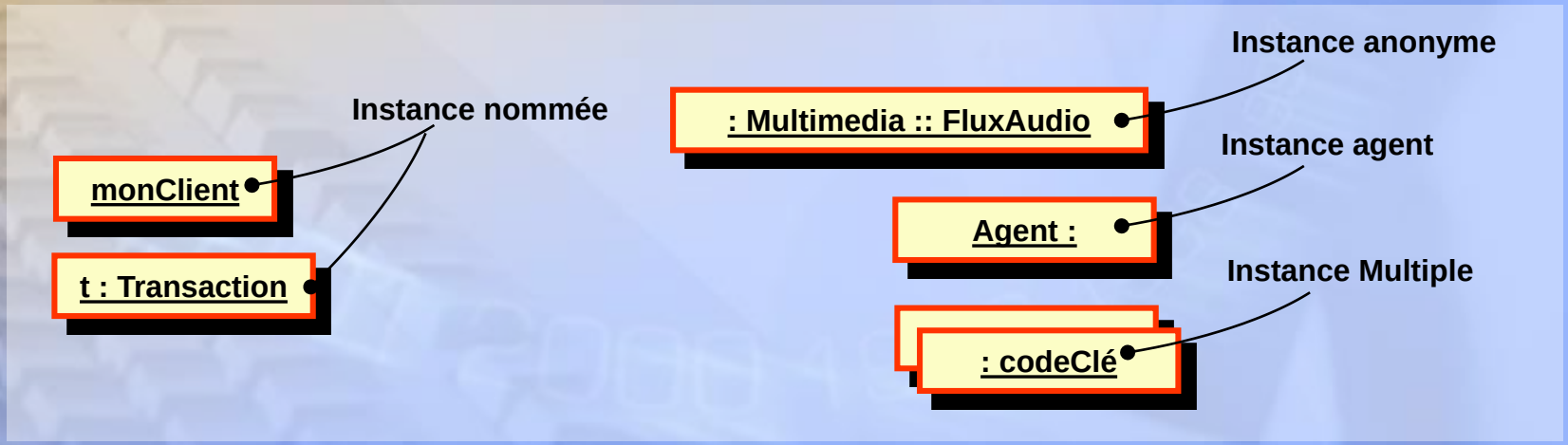


Diagramme d’objet

Diagramme d'objet

- Les diagrammes d'objets modélisent les instances d'éléments qui apparaissent sur des diagrammes de classes.
- Ils montrent un ensemble d'objets et leurs relations à un moment donné.
- On utilise les diagrammes d'objets pour modéliser les vues de conception statiques ou de processus statiques.
- En UML, on utilise les diagrammes de classes pour visualiser les aspects statiques des briques de base d'un système.
- Les diagrammes d'interaction servent à visualiser les aspects dynamiques du système, qui comprennent les instances de ces briques de base et les messages qui peuvent être échangés entre elles.

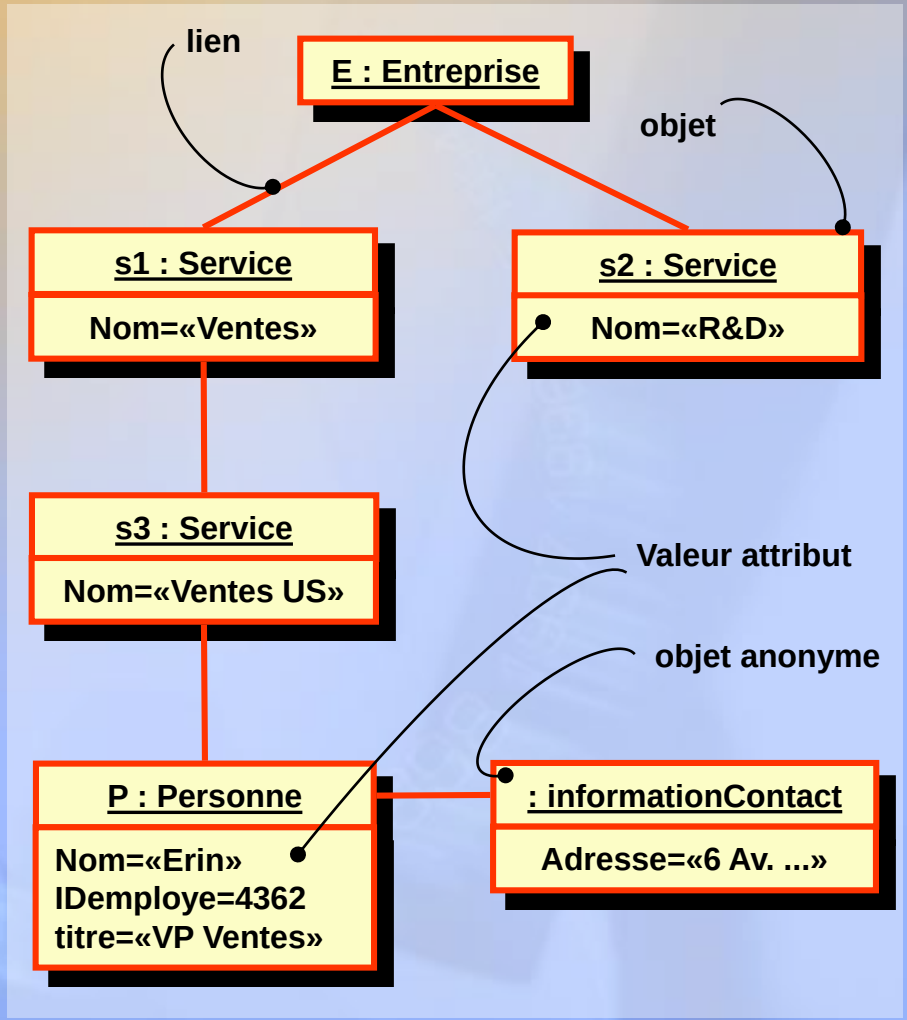


Diagramme de classe

Modélisation des structures élémentaires statiques

Diagramme de classe

- Les diagrammes de classes sont les diagrammes les plus courants dans la modélisation des systèmes orientés objet. Ils représentent un ensemble de
  - classes,
  - d'interfaces
  - de collaborations
  - ainsi que leurs relations.
- On utilise des diagrammes de classes pour modéliser la vue de conception statique d'un système.
- Pour l'essentiel, cela implique
  - la modélisation du vocabulaire du système,
  - la modélisation de collaborations
  - ou la modélisation de schémas.
- Les diagrammes de classes sont aussi le fondement de diagrammes apparentés: diagrammes de composants et diagrammes de déploiement.
- Les diagrammes de classes sont importants non seulement pour
  - visualiser,
  - consruire,
  - spécifier
  - et documenter des modèles structurels.

Diagramme de classe

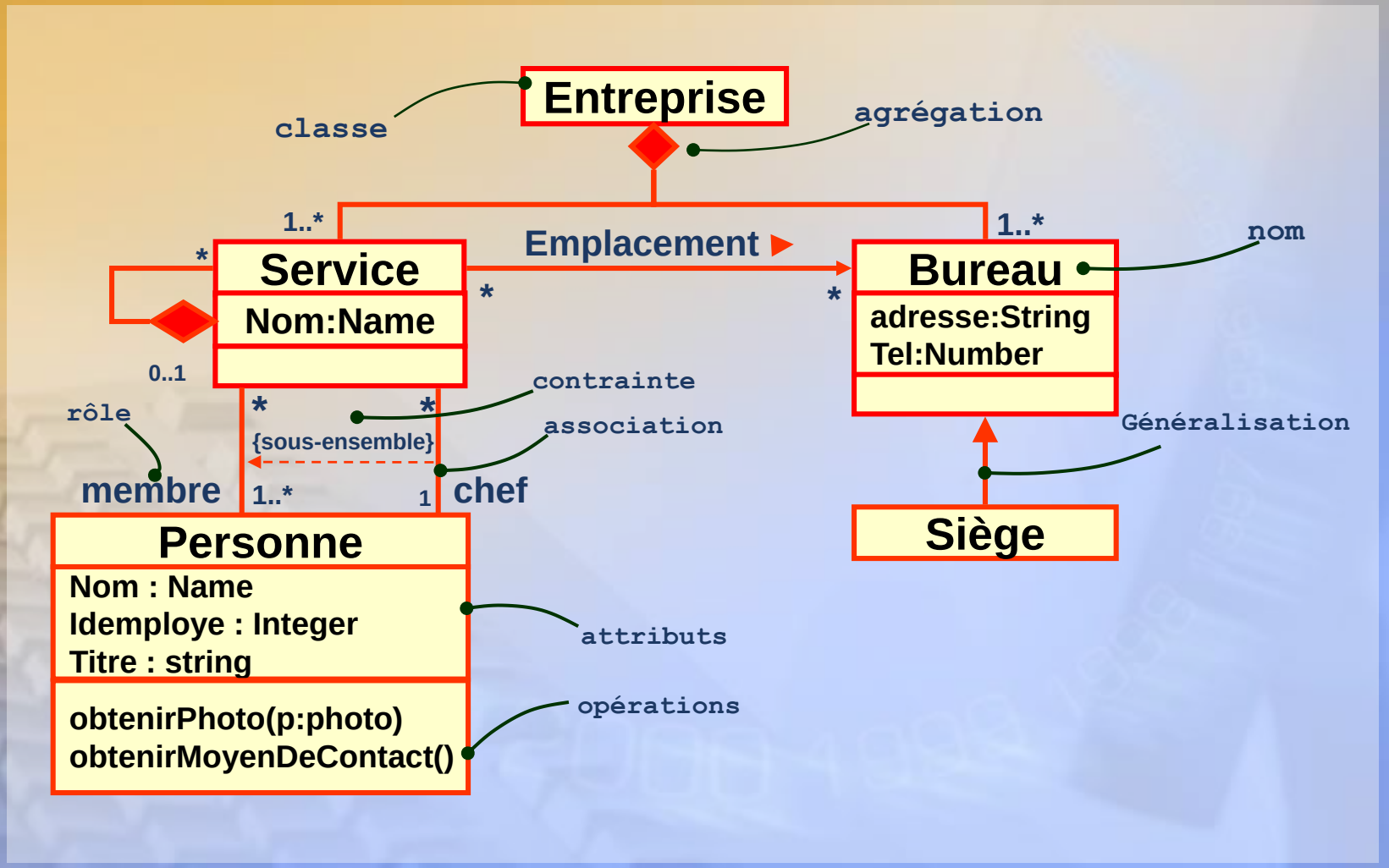
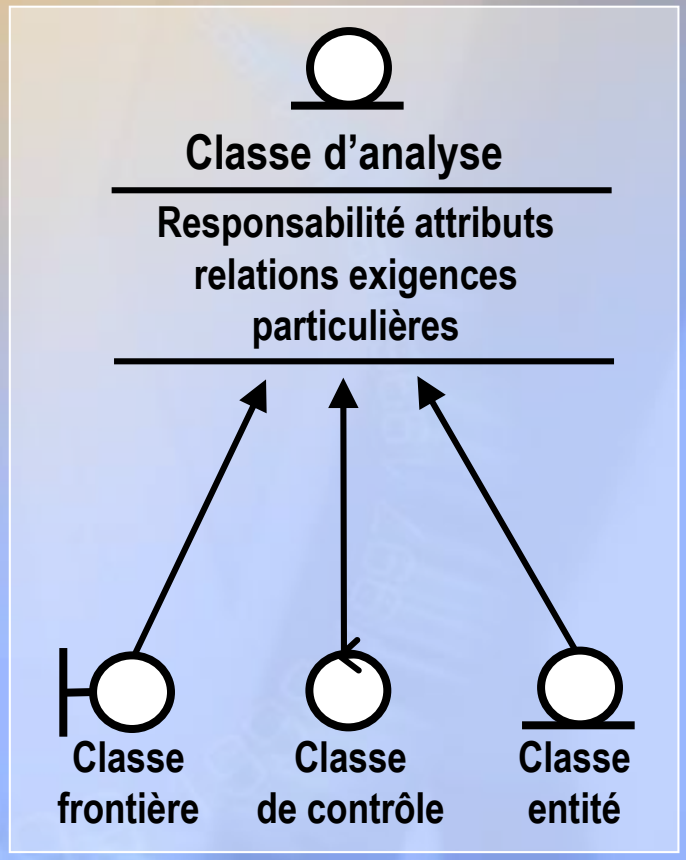


Diagramme de classe et classe d'analyse

- **UML fournit trois stéréotype de classes standard, que nous utilisons dans l'analyse.**
- **Exemple :**



- **Classe frontière:**
  - **Interaction avec les acteurs**
- **Classe contrôle:**
  - **Information de longue durée**
- **Classe entité:**
  - **Coordonne le séquencement, les transactions et le contrôle d'autres objet.**

Exemple

- **Cas d'utilisation: régler le facture**

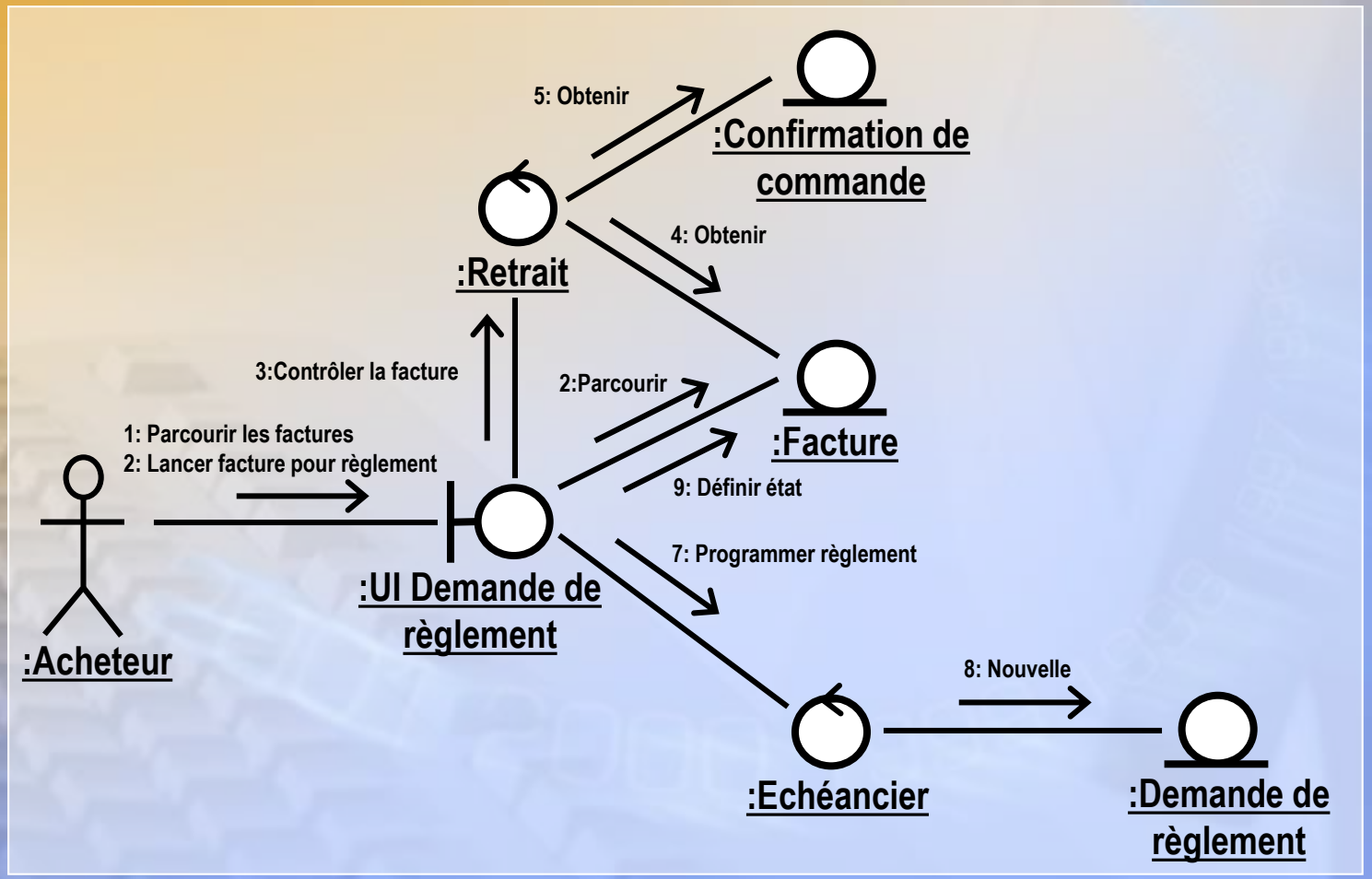


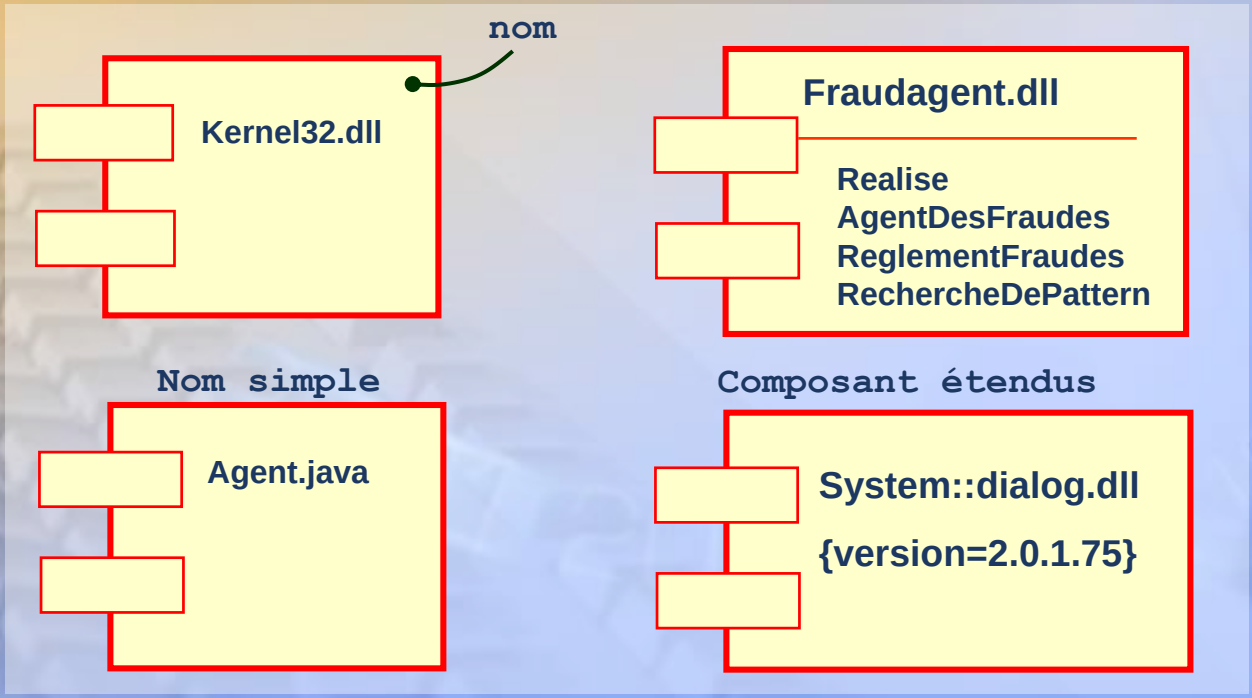
Diagramme de composant

Diagramme de composant

- Les composants vivent dans le monde matériel des bits et constituent des briques de base importantes dans la modélisation des aspects physiques d'un système.
- Un composant est une partie physique remplaçable d'un système qui fournit la réalisation d'un ensemble d'interfaces et s'y conforme.
- On utilise les composants pour modéliser les éléments physiques qui peuvent se trouver sur un nœud, comme
  - les exécutables,
  - les bibliothèques,
  - les tables,
  - les fichiers
  - et les documents.
- Un composant représente en général le regroupement physique d'éléments habituellement logiques, tels que
  - les classes,
  - les interfaces
  - et les collaborations.
- Les bons composants déterminent des abstractions bien délimitées à l'aide d'interfaces bien définies, ce qui permet de remplacer facilement les composants plus anciens par des composants plus récents et compatibles.

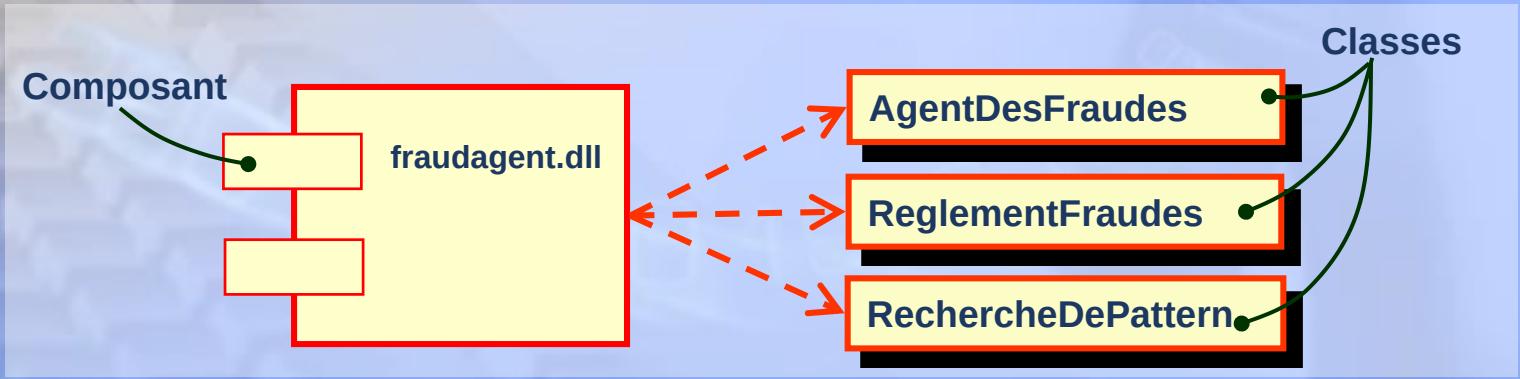
Représentation des composants

- UML fournit une représentation graphique pour les composants.
- Cette notation canonique permet de visualiser un composant en dehors de tout système d'exploitation ou langage de programmation.
- En utilisant les stéréotypes, qui font partie des mécanismes d'extensibilité d'UML, on peut adapter cette notation pour représenter des types de composants spécifiques.



Composants et classes

- Par de nombreux aspects, les composants sont similaires aux classes:
  - ils ont tous deux un nom,
  - peuvent réaliser un ensemble d'interfaces,
  - participer à des relations de dépendance, de généralisation et d'association, et à des interactions,
  - être emboîtés et avoir des instances.
- Il existe cependant des différences importantes entre les composants et les classes:
  - les classes représentent des abstractions logiques alors que les composants représentent des éléments physiques qui existent dans le monde des bits. Pour résumer, cela signifie que les composants peuvent résider sur des nœuds, mais pas les classes.
  - les composants représentent le regroupement physique de ce qu'on pourrait appeler des composants logiques et se situent à un niveau d'abstraction différent.
  - les classes peuvent avoir directement des attributs et des opérations. En général, les composants comportent seulement des opérations que l'on peut atteindre uniquement par leur interface.



Remplaçabilité binaire

- **Un composant est un élément physique remplaçable d'un système qui fournit la réalisation d'un ensemble d'interfaces et s'y conforme.**
  - **Premièrement, un composant est *physique.* Il vit dans le monde des bits et non dans celui des concepts.**
  - **Deuxièmement, un composant est *remplaçable.* Il peut être remplacé par un autre composant qui se conforme aux mêmes interfaces.**
  - **Troisièmement, un composant est un *élément d'un système.***
    - **Un composant existe rarement indépendamment des autres.**
    - **Un composant donné collabore avec d'autres composants et,**
    - **ce faisant, existe dans le contexte d'architecture ou de technologie dans lequel il est supposé être utilisé.**
    - **Un composant est logiquement et physiquement cohérent et dénote donc une partie structurelle et/ou comportementale importante d'un plus grand système.**
    - **Il peut être réutilisé à travers de nombreux systèmes.**
    - **simple composant à un plus haut niveau d'abstraction.**
  - **Quatrièmement, comme cela est expliqué dans la partie précédente, un composant *fournit la réalisation d'un ensemble d'interfaces et s'y conforme.***

Types de composants

- Types de composants
- On distingue trois types de composants.
  - Premièrement, les *composants de déploiement* qui sont nécessaires et suffisants pour former un système exécutable,
    - comme les bibliothèques dynamiques (DLL)
    - les exécutables (EXE). .
- Deuxièmement, les *composants produits par le développement.* Ces composants sont essentiellement le résultat du processus de développement;
  - ils se composent d'éléments tels que
    - des fichiers code source
    - des fichiers de données à partir desquels les composants de déploiement sont créés.
  - Ils ne participent pas directement à un système exécutable mais sont les produits du travail de développement qui servent à créer le système exécutable.
- Troisièmement, les *composants d'exécution.* Ces composants sont créés en tant que produits d'une exécution, comme par exemple un objet COM+ qui est instancié à partir d'une DLL.

Organisation des composants

- On organise les composants en les groupant dans des paquetages de la même façon que l'on organise les classes.
- On peut également organiser des composants en spécifiant leurs relations de dépendance, de généralisation, d'association (y compris d'agrégation) et de réalisation.

Stéréotypes standard

- **UML définit cinq stéréotypes standard qui s'appliquent aux composants:**
  1. **executable précise un composant qui peut être exécuté sur un nœud.**
  2. **library précise une bibliothèque objet statique ou dynamique.**
  3. **table précise un composant qui représente une table de base de données.**
  4. **file précise un composant qui représente un document contenant un code source ou des données.**
  5. **document précise un composant qui représente un document.**
- **Remarque:** UML ne fournit pas d'icônes définies pour ces stéréotypes.

Modélisations

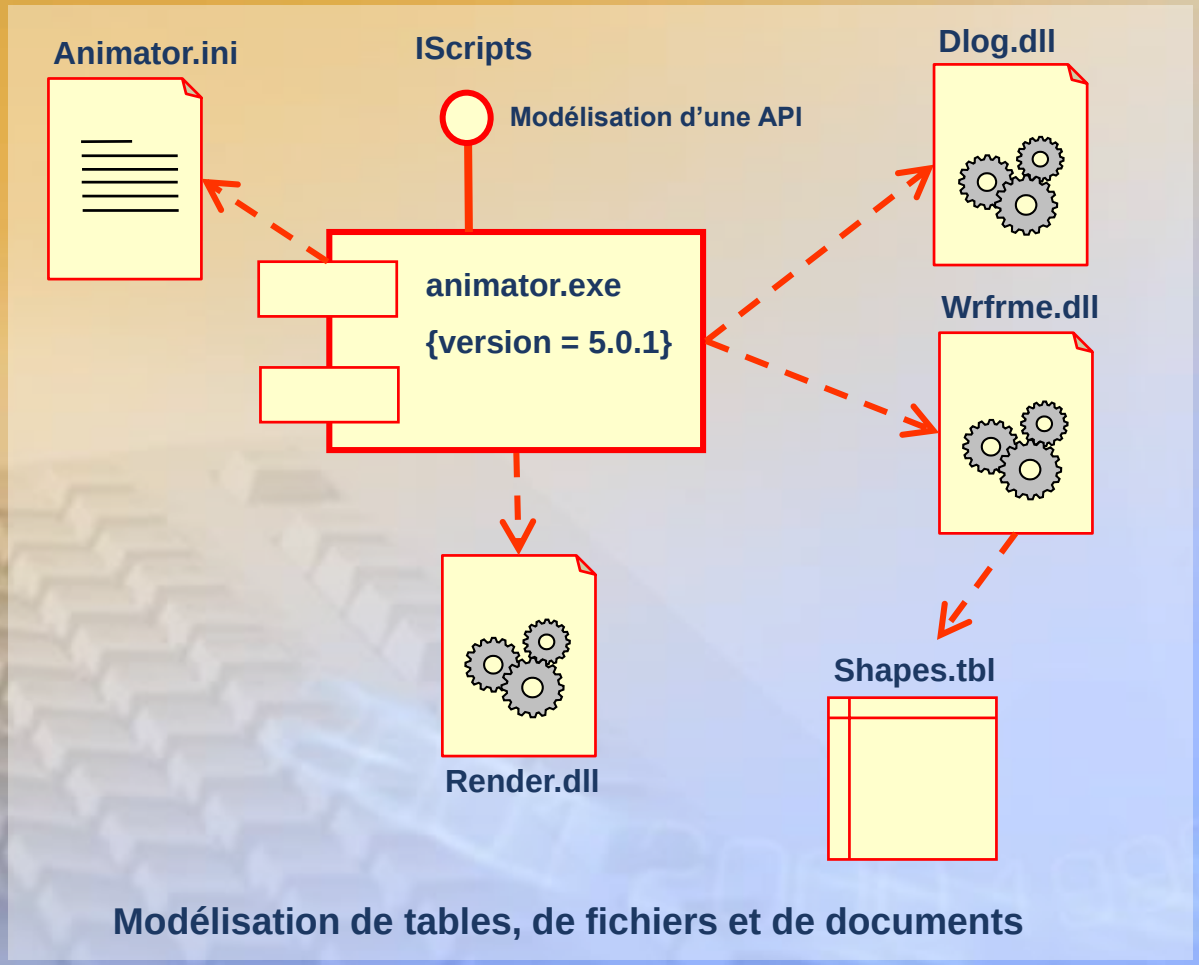


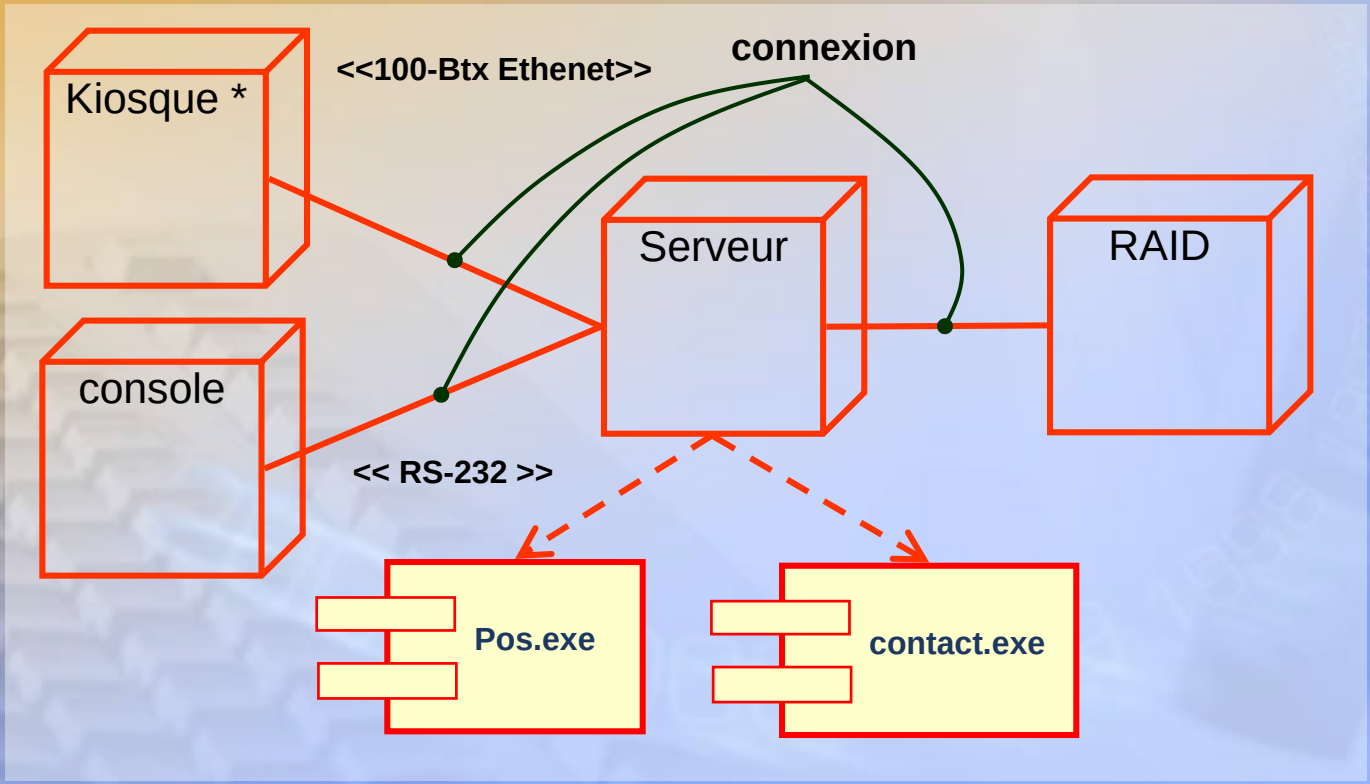
Diagramme de déploiement

Diagramme de déploiement

- Comme les composants, les nœuds vivent dans le monde du matériel et constituent des briques de base importantes pour la modélisation des aspects physiques d'un système.
- Un nœud est un élément physique qui existe au moment de l'exécution et représente une ressource de calcul. En général, il a au moins de la mémoire et souvent, en plus, des capacités de traitement.
- On utilise les nœuds pour modéliser la topologie du matériel sur lequel le système s'exécute.
- La plupart du temps, un nœud représente un processeur ou un périphérique sur lequel les composants peuvent être déployés.
- Les bons nœuds représentent clairement le vocabulaire du matériel dans le domaine de solution concerné.

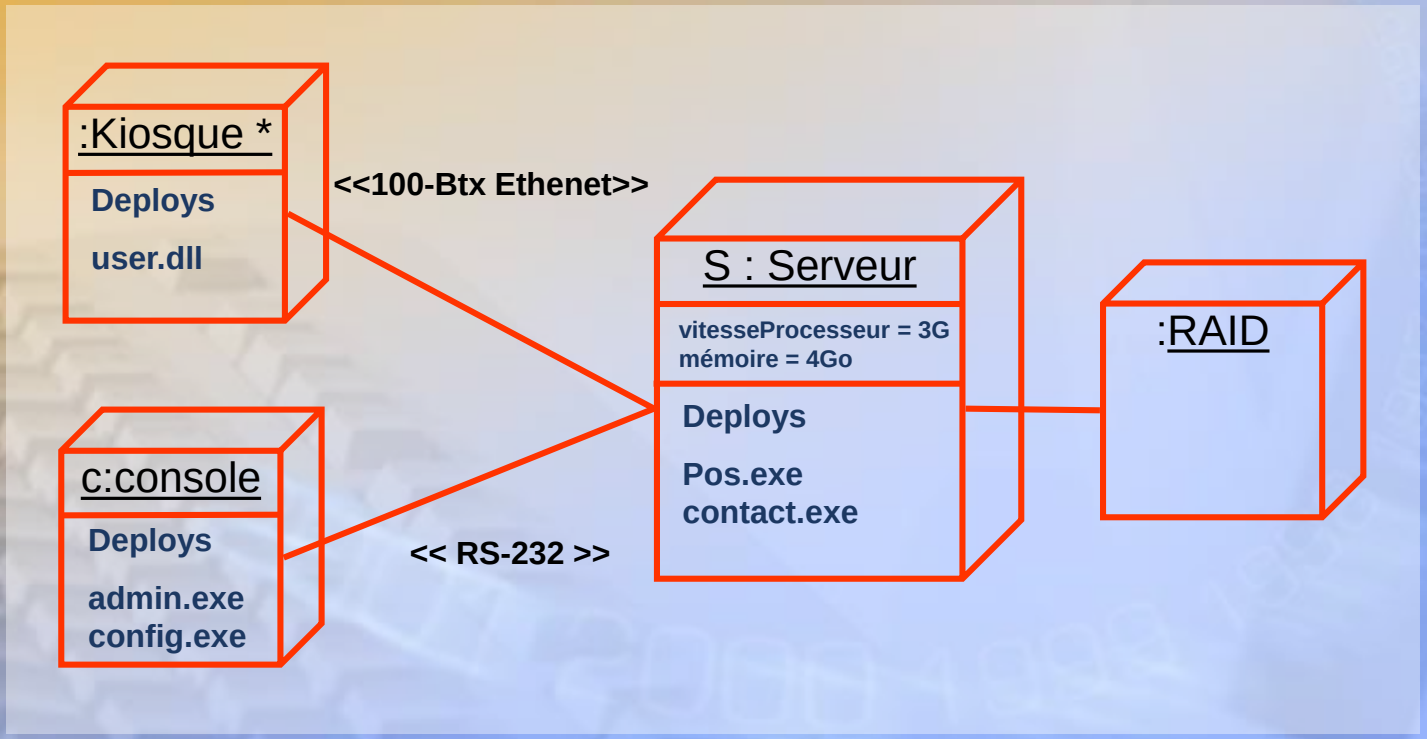
Représentation

- **UML propose une représentation graphique des nœuds. Cette notation canonique permet de visualiser un nœud en dehors de tout matériel spécifique.**



Modélisation de la répartition de composant

- **Visualise l'instance spécifique de chaque noeud**



A vous de jouer (échauffement)

Exercice 1 :

Réaliser les diagrammes de classe des expressions suivantes:

1. Un pays possède une capitale.
2. Une personne dîne avec une fourchette.
3. Un chemin peut représenter un fichier ou un répertoire.
4. Un chemin est un répertoire avec éventuellement un nom de fichier.
5. Un fichier contient des enregistrements.
6. Un fichier est accessible par un utilisateur selon des droits d'accès.
7. Un dessin est soit du texte, soit une forme géométrique, soit un groupe de dessins.
8. Des personnes utilisent un langage pour un projet.
9. Une personne joue dans une équipe pour une certaine durée.
10. Une équipe est composée de plusieurs personnes.
11. Une route connecte deux villes.

Corrections exercices 1

Plus d’information …

- http://www.rational.com/products/rup/
- http://www.therationaledge.com
- http://www.ambysoft.com/
- http://www.ronin-intl.com/publications/unifiedProcess.html

Dialogue