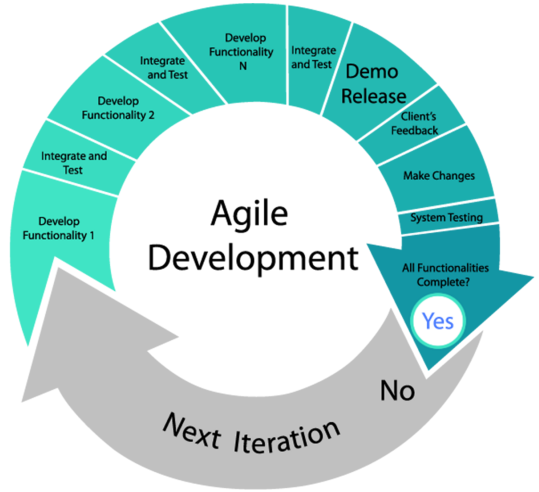
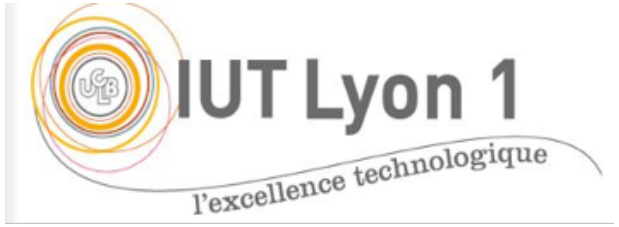
Chp.5

Modélisation en approche agile

V. Deslandres ©
IUT Univ. LYON1
Licence pro DEVOPS





Capture des Besoins

- Peu importe le niveau des experts du domaine ou de l'entreprise. Le point critique du développement logiciel repose sur la capacité des DEV à **comprendre** le Métier.

- Un certain nombre de méthodes existent qui permettent de discuter, d'échanger sur certains points de conception jusqu'à ce que les besoins soient compris et stabilisés
  - **Event Storming Method** (Alberto Brandolini) : concevoir une Event Driven Architecture dès le début du projet
  - Le **DDD** (Domain Driven Development) ou **Interaction Driven Design**
  - **Les méthodes Agiles**

Modélisation en DevOps / approche Agile

- La modélisation répond à différents **objectifs** :
  - **Comprendre les besoins fonctionnels** du Client et mieux répondre à ses attentes
  - **Communiquer avec les autres développeurs**, pour discuter des choix techniques et avoir une vision globale
  - **Communiquer avec les responsables Système** pour les besoins Techniques
- Elle permet de **schématiser**
  - l'architecture globale du logiciel
  - des éléments complexes

Martin Fowler : 'UML as a Sketch'

- 2 utilisations d'UML
  - **avant** d'écrire le code pour accélérer le processus (*forward-engineering*)
  - *Reverse-engineering* à partir du code existant pour aider **à le comprendre**

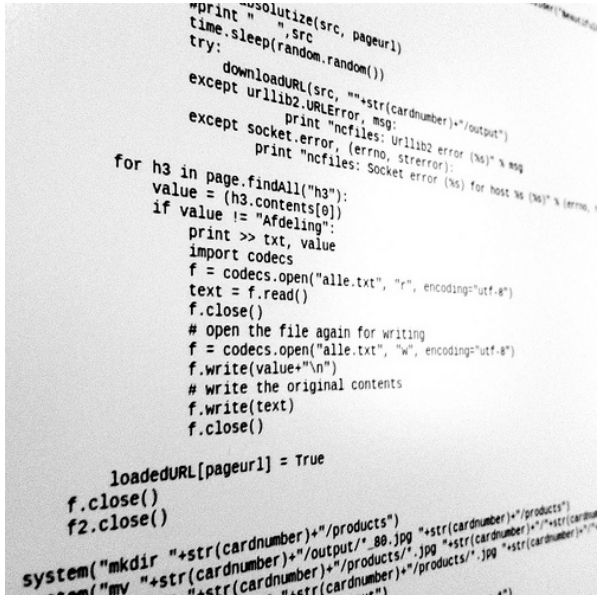
- Utiliser ces 2 moyens sur des **points techniques très limités**
  - Pas sur **tout le code**
  - Pour préciser un aspect, une façon de concevoir un point précis
  - Moins de 10', travail collaboratif, tableau blanc : informel et dynamique (pas d'AGL, *no CASE tool*)

http://martinfowler.com/bliki/UmlAsSketch.html

UML & Scrum

- Utiliser UML pour
  - Expliquer aux autres membres de l'équipe comment vous voyez **qu'une User Story fonctionne**

- **Ne pas passer trop de temps** sur les diagrammes
  - On n'a pas besoin de trop de détails
- Utiliser le moyen **qui parle le plus** pour discuter et choisir sa conception
  - Diagramme UML en dessin libre sur tableau blanc
  - Ce qui compte en Agile, **c'est le code**

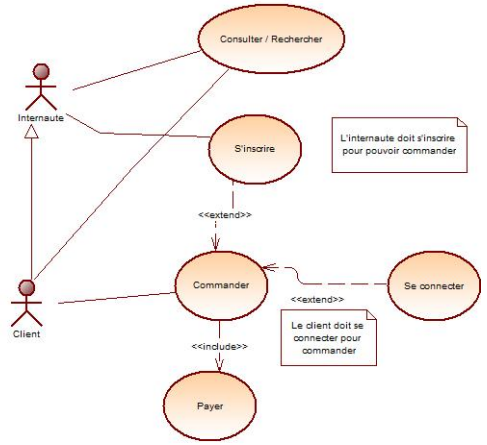


UML & Scrum

- Les diagrammes UML les plus utiles :
  - **Diagramme des Cas d'Utilisation**
    - Pour se mettre d'accord sur les points d'entrée
    - Sur types d'utilisateur à considérer (rôles)
  - **Parfois Diagramme d'activités**
    - Pour fixer le déroulement d'un Cas d'Utilisation (vision Utilisateur)
  - **Souvent Diagramme de Séquences**
    - Mettre au clair un aspect technique (objets)
  - **Diagramme de Classes**
    - Pour la BD, vérifier qu'on a pensé à tout
    - Rétro engineering : comprendre un code, avoir une vue d'ensemble

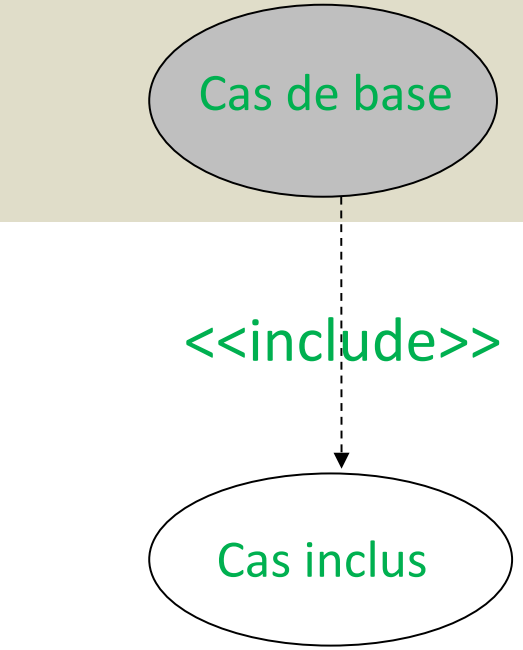
Diagramme des Cas d'Utilisation

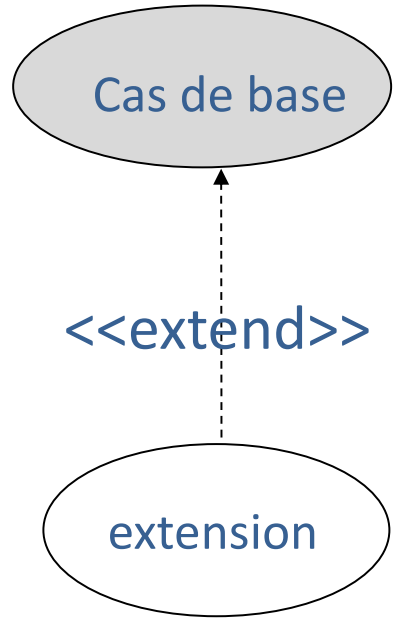
Rester simple, décrire les grandes fonctionnalités

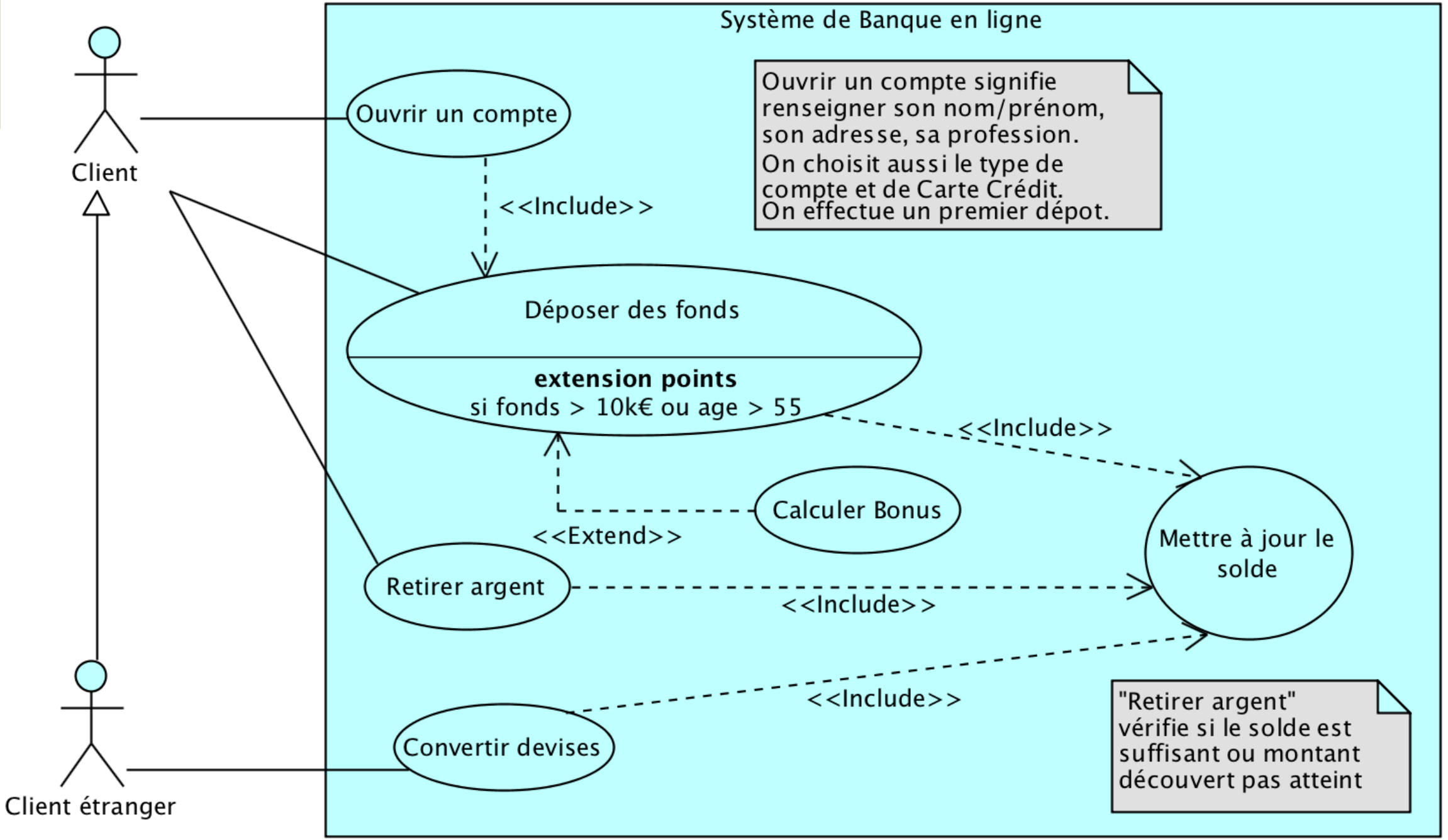


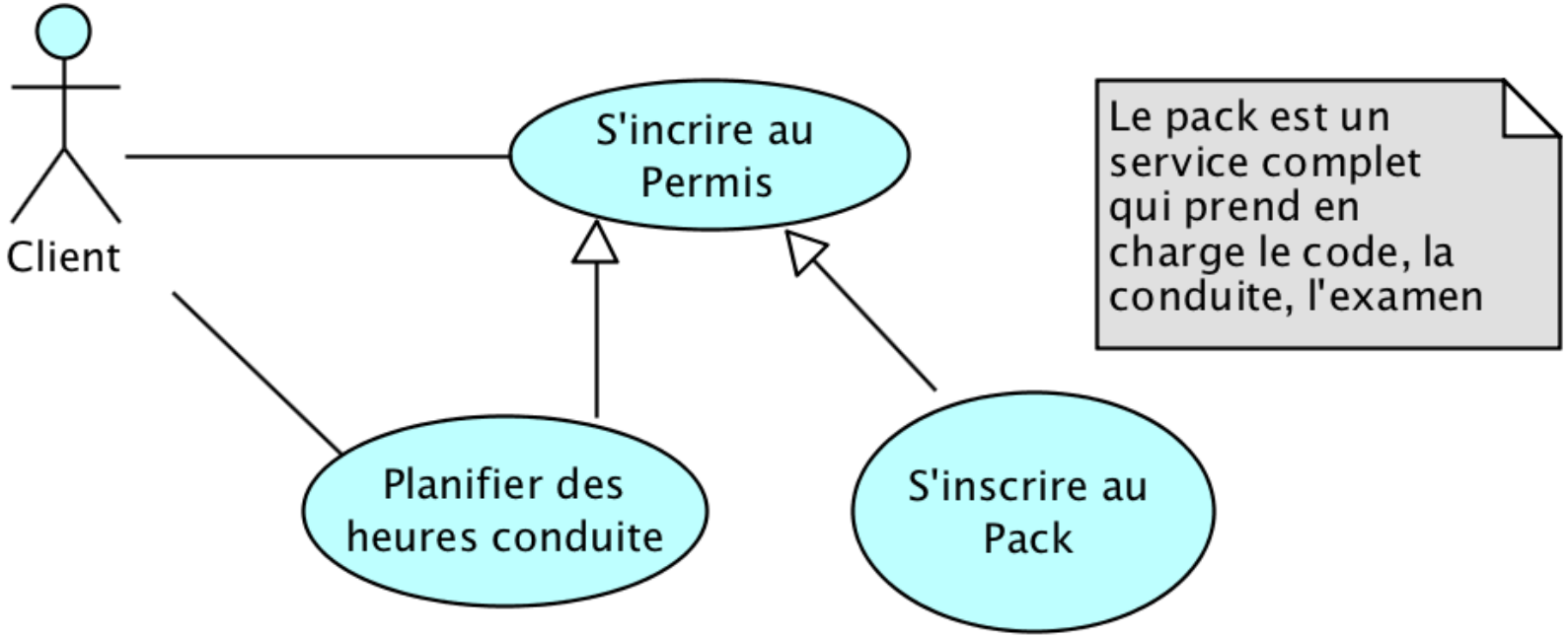
Les bases

- Acteurs **externes**
- **Pas de chronologie** : on liste ce qu'il est possible de faire, avec des dépendances fonctionnelles
  - pas temporelles
- Relations « **include** » et « **extend** » : traduisent les dépendances **fonctionnelles**
  - Un cas inclus traduit une passage **systématique** vers le cas inclus
  - Un cas étendu dépend du cas de base, c'est une extension **possible**
  - Le sens de la flèche d'Extend traduit la phrase : « l'extension étend le Cas de Base »

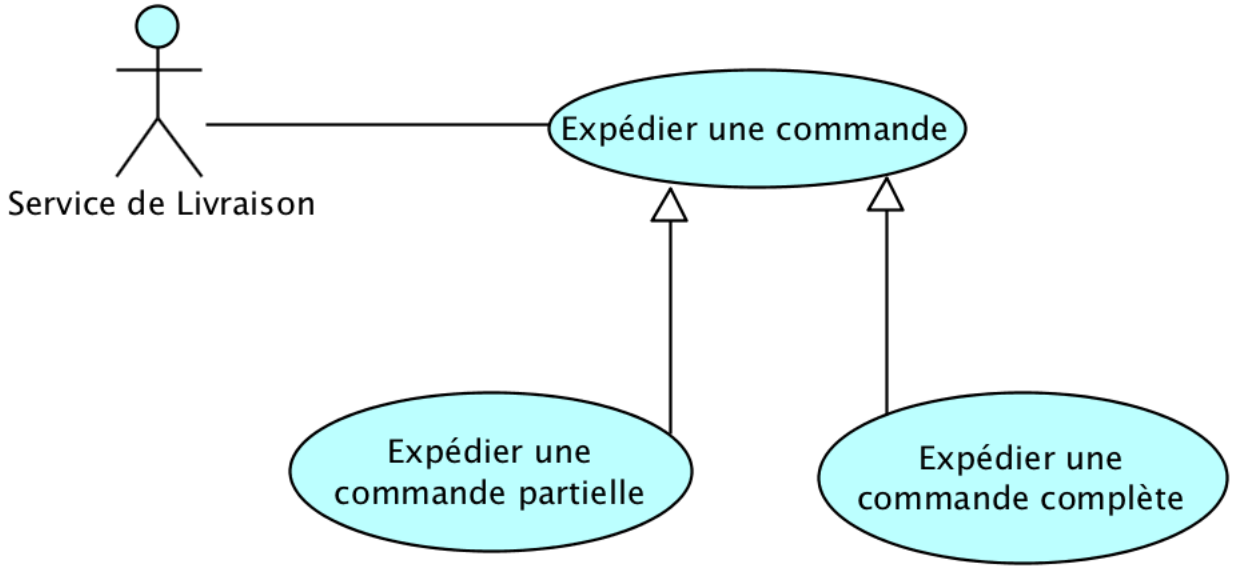


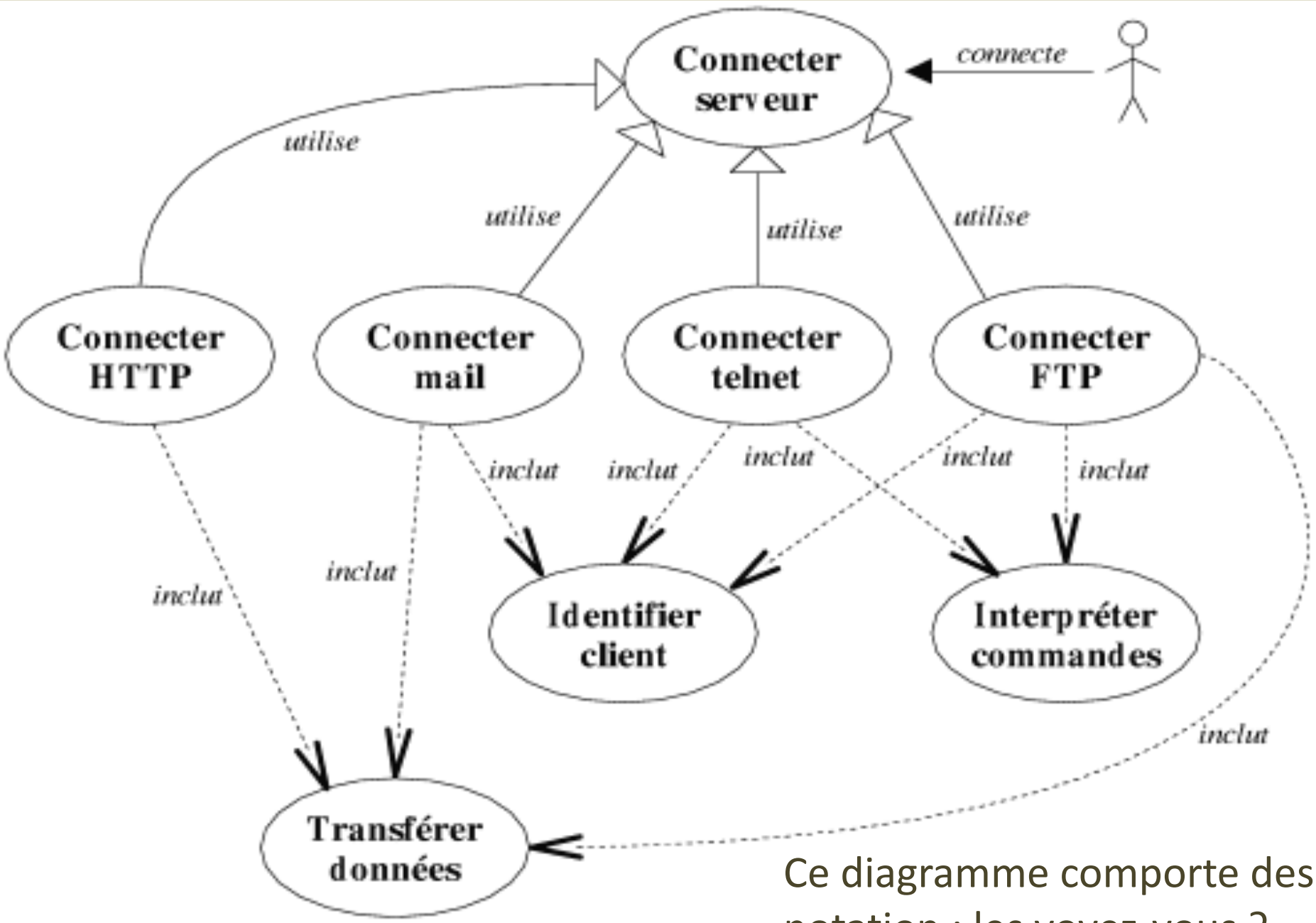



Système de Banque en ligne
Client
Ouvrir un compte
<<Include>>
Ouvrir un compte signifie renseigner son nom/prénom, son adresse, sa profession.
On choisit aussi le type de compte et de Carte Crédit.
On effectue un premier dépot.
Déposer des fonds
extension points
si fonds > 10k€ ou age > 55
<<Include>>
Calculer Bonus
<<Extend>>
Retirer argent
<<Include>>
Mettre à jour le solde
<<Include>>
Convertir devises
"Retirer argent" vérifie si le solde est suffisant ou montant découvert pas atteint
Client étranger



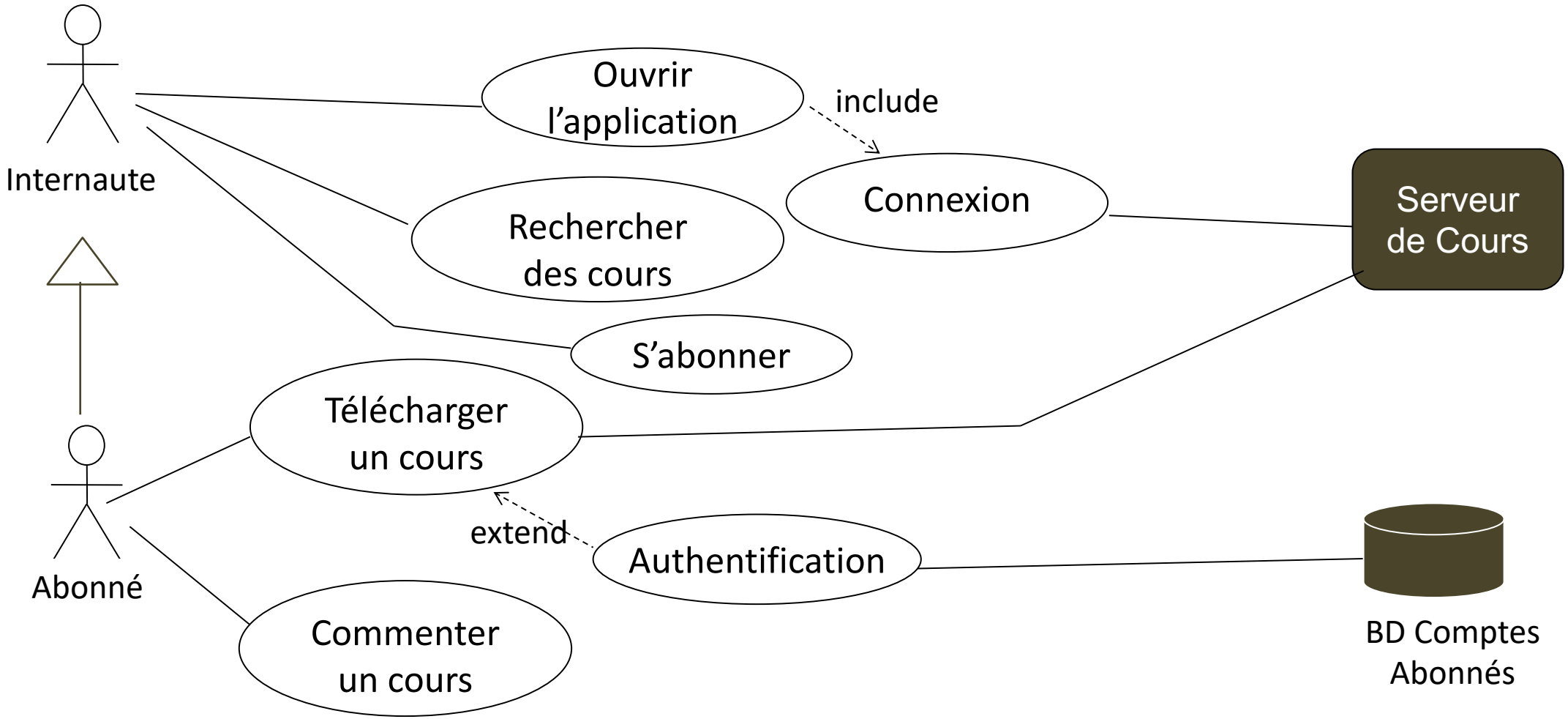
Héritage des cas





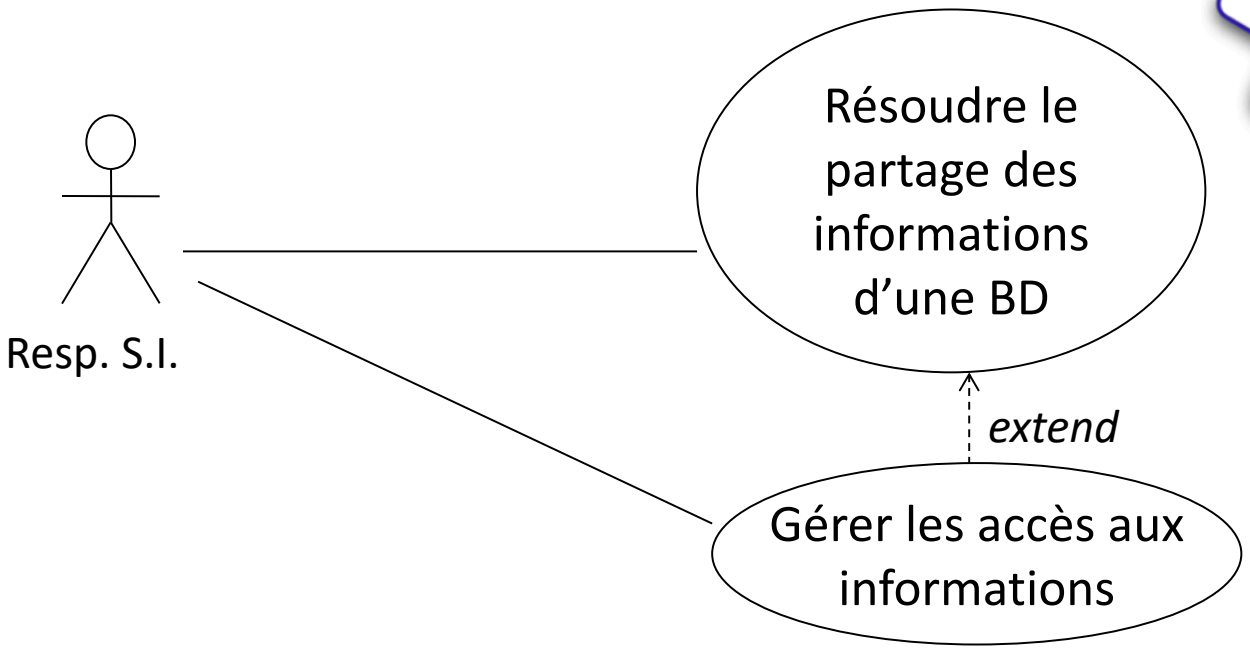
Ce diagramme comporte des erreurs de notation : les voyez-vous ?

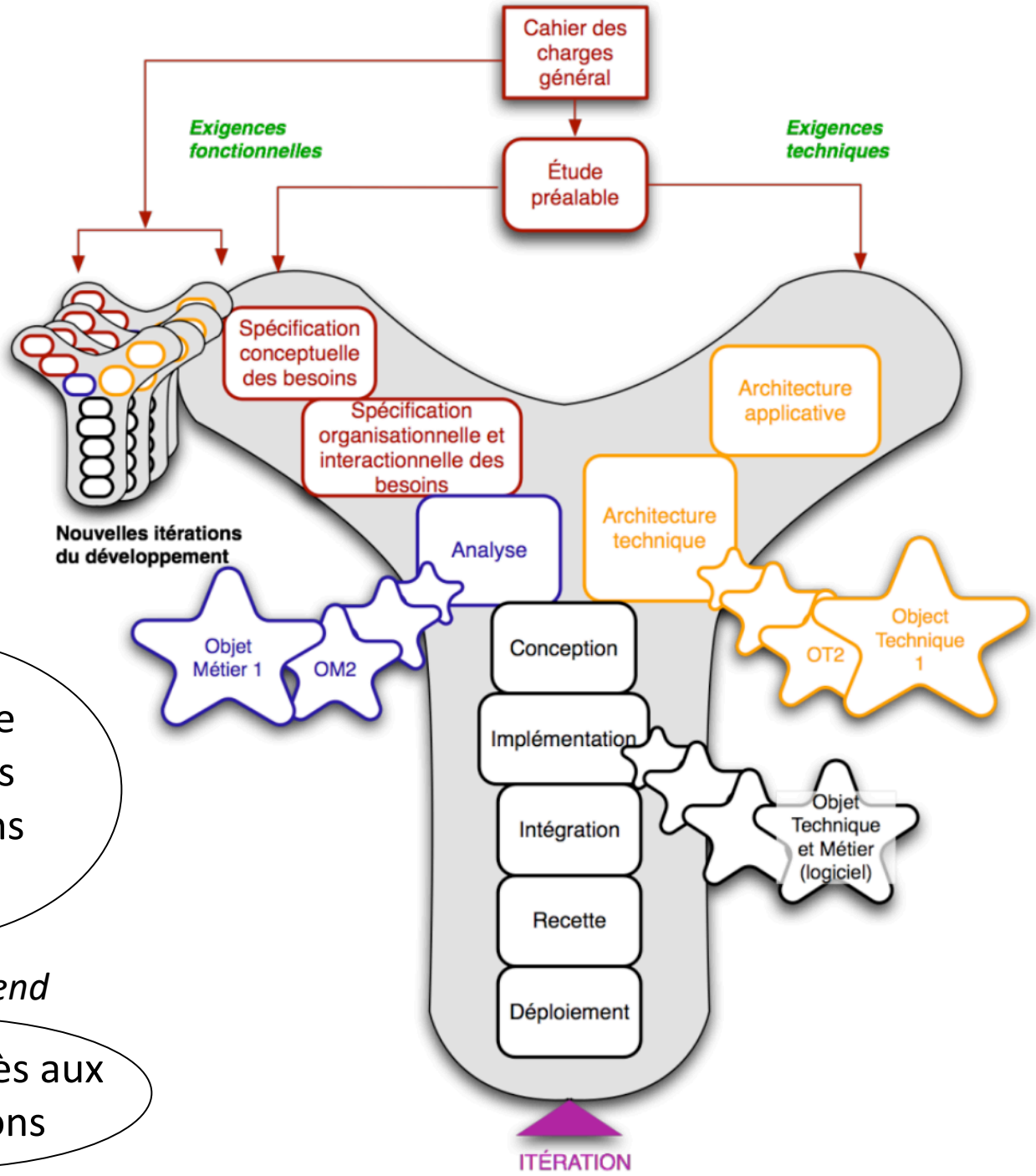
Exercice : Lire et critiquer ce diagramme

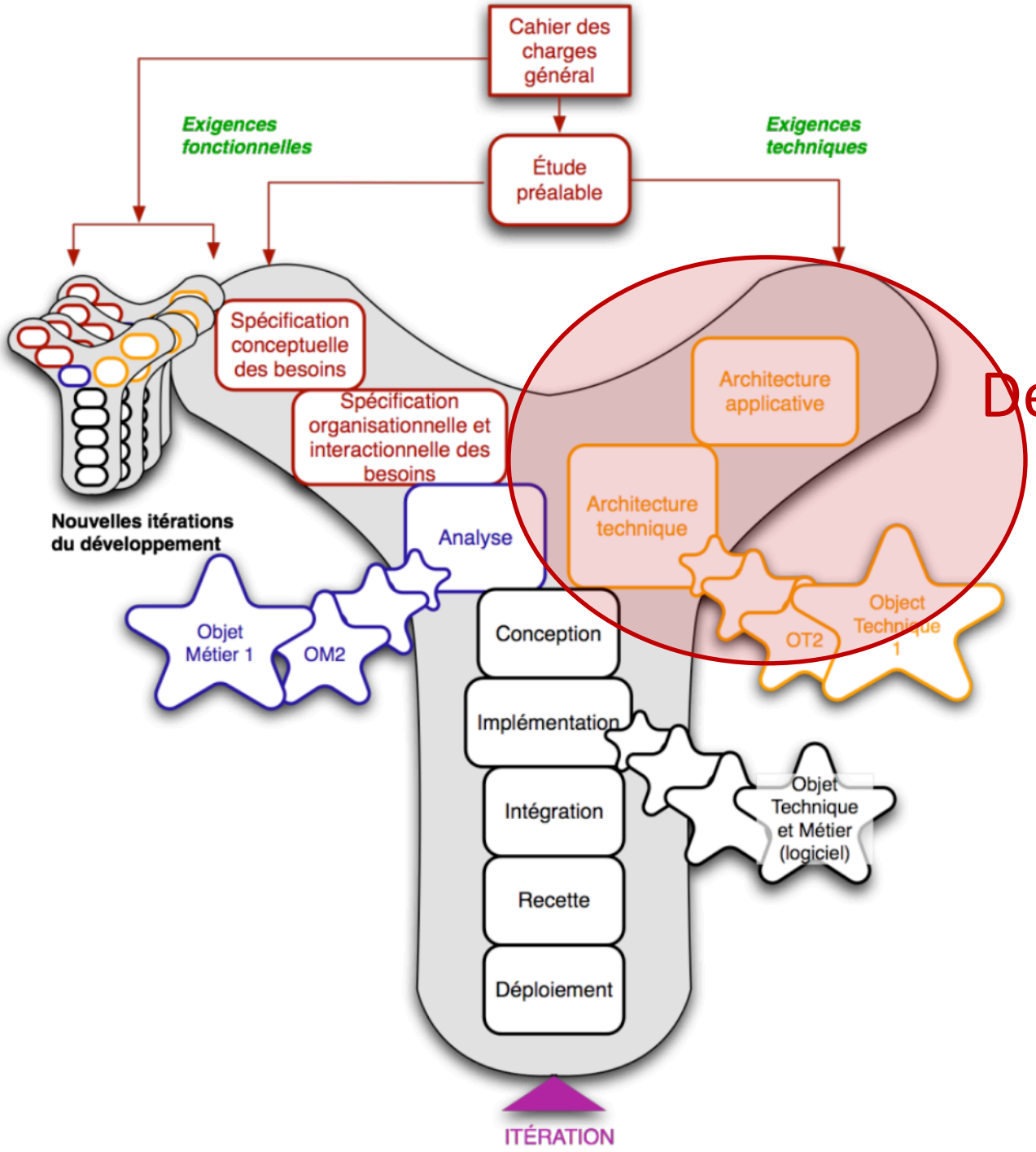


Cycle en Y

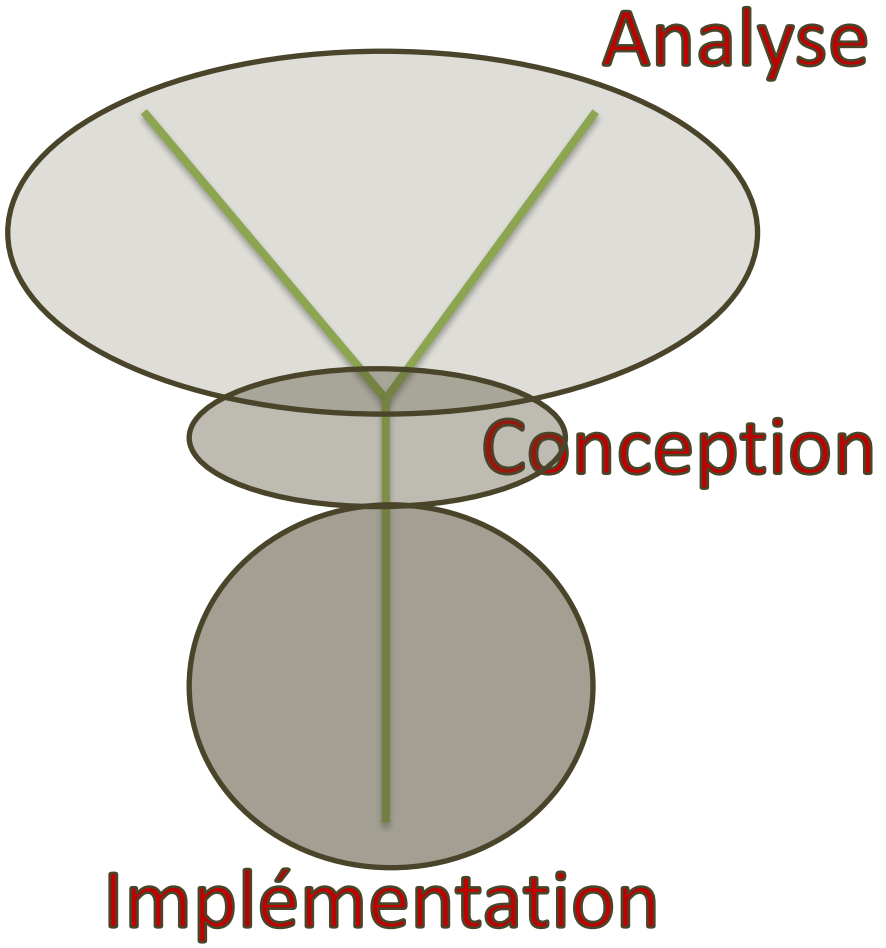
- Pour la capture des besoins, il est intéressant de distinguer les besoins fonctionnels des besoins techniques



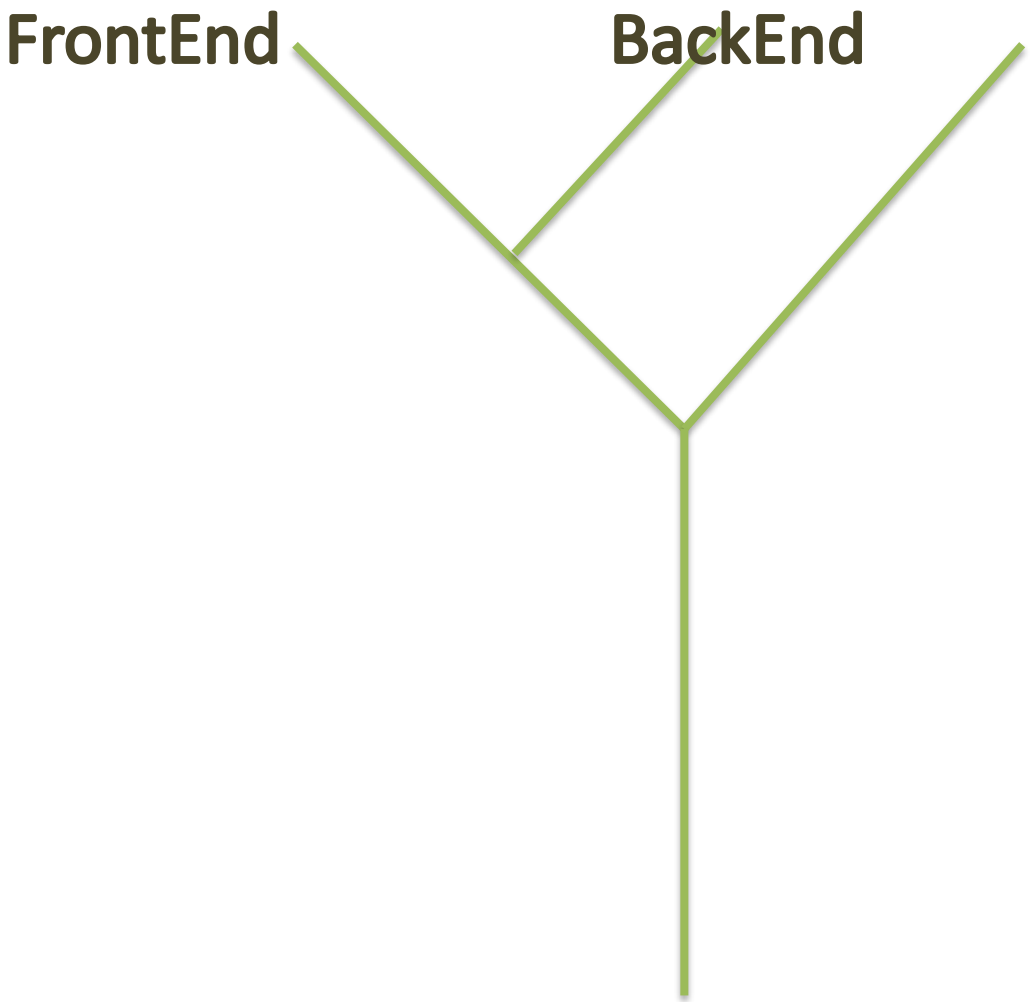



Cahier des charges général
Exigences fonctionnelles
Étude préalable
Exigences techniques
Spécification conceptuelle des besoins
Spécification organisationnelle et interactionnelle des besoins
Architecture applicative
Architecture technique
Nouvelles itérations du développement
Analyse
Objet Métier 1
OM2
OT2
Object Technique 1
Conception
Implémentation
Intégration
Recette
Déploiement
Objet Technique et Métier (logiciel)
ITÉRATION

DevOps


Analyse
Conception
Implémentation

Distinction Front/Back



Analyse

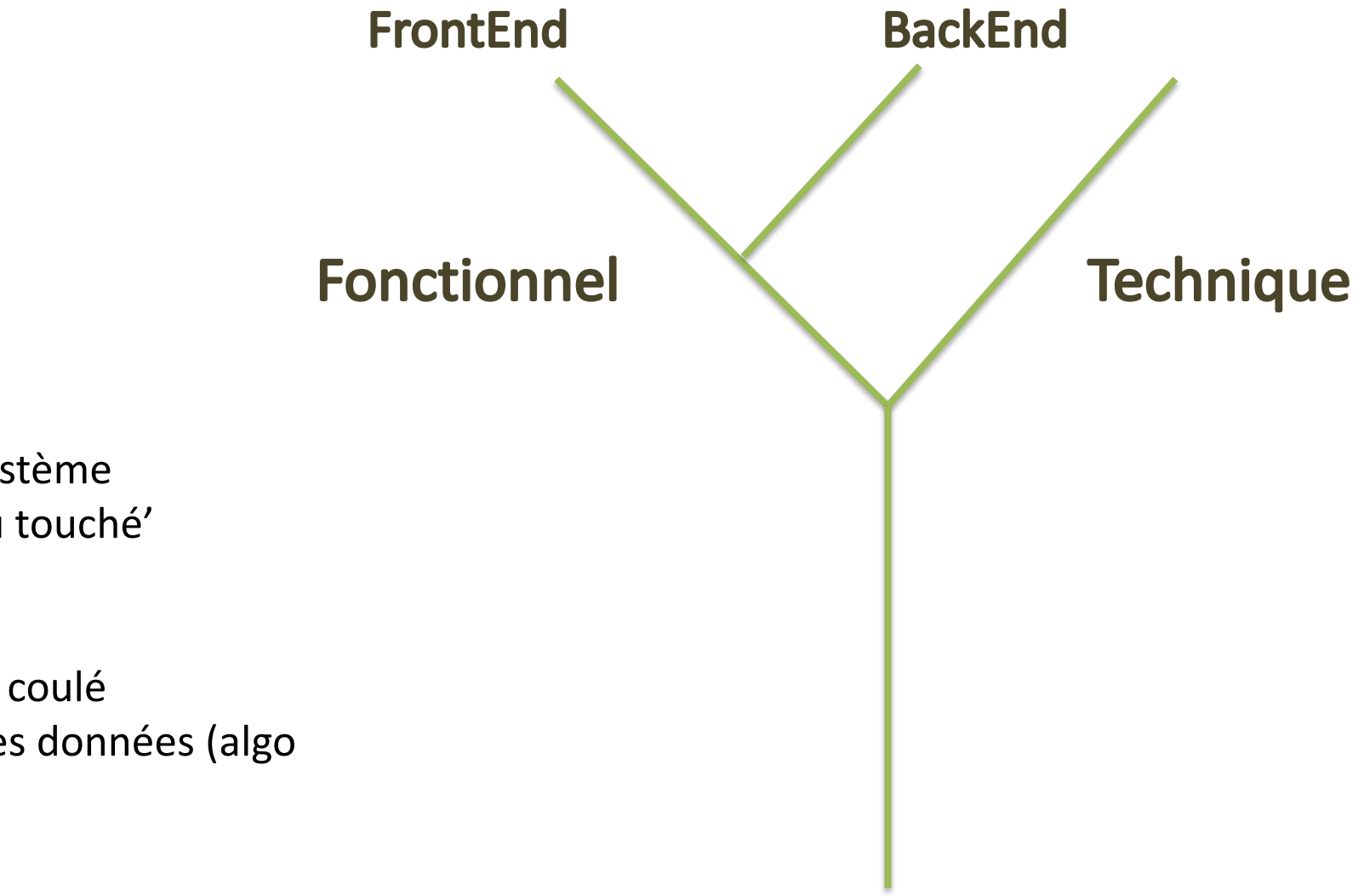
Conception

Implémentation

Exemple : bataille navale

Placer les besoins suivant :

- Placer un bateau
- Voir le résultat d'un tir
- Inscrire un joueur
- Définir le système de persistance
- Définir le nb de bateaux du joueur
- Définir l'accès aux données d'un système
- Définir le rendu d'une case 'bateau touché'
- Tirer(case)
- Répliquer une BD partagée
- Définir la visualisation d'un bateau coulé
- Choisir le système de protection des données (algo de cryptage ? Autre ?)



Exemple : Système de co-voiturage

Placer les besoins suivant sur les branches :

- Réserver un trajet
- Découper un tronçon de route en étapes
- Définir un trajet
- Choisir le mode de gestion (manuel ou automatique) de la demande
- Gérer les réservations (en mode manuel)
- Inscrire un abonné
- Définir le système de persistance
- Rechercher un trajet
- Définir le système de paiement
- Répliquer une BD partagée
- Annuler un trajet
- Gérer les informations de son compte abonné
- Choisir le système de protection des données
- Gérer les comptes (suppressions d'abonnés)
- Gérer les abonnés qui annulent régulièrement trop tard

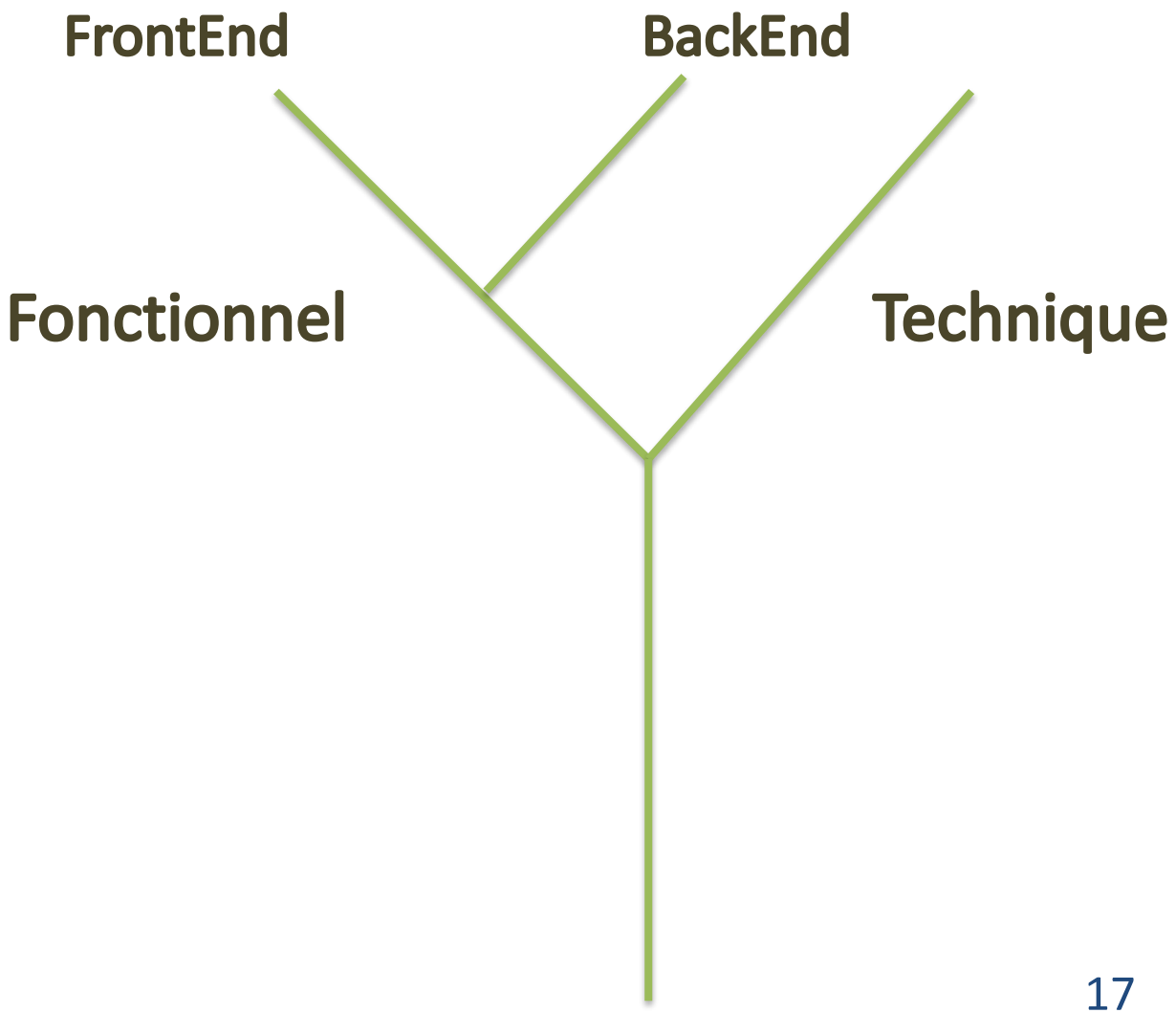


Illustration : salle de sport connectée

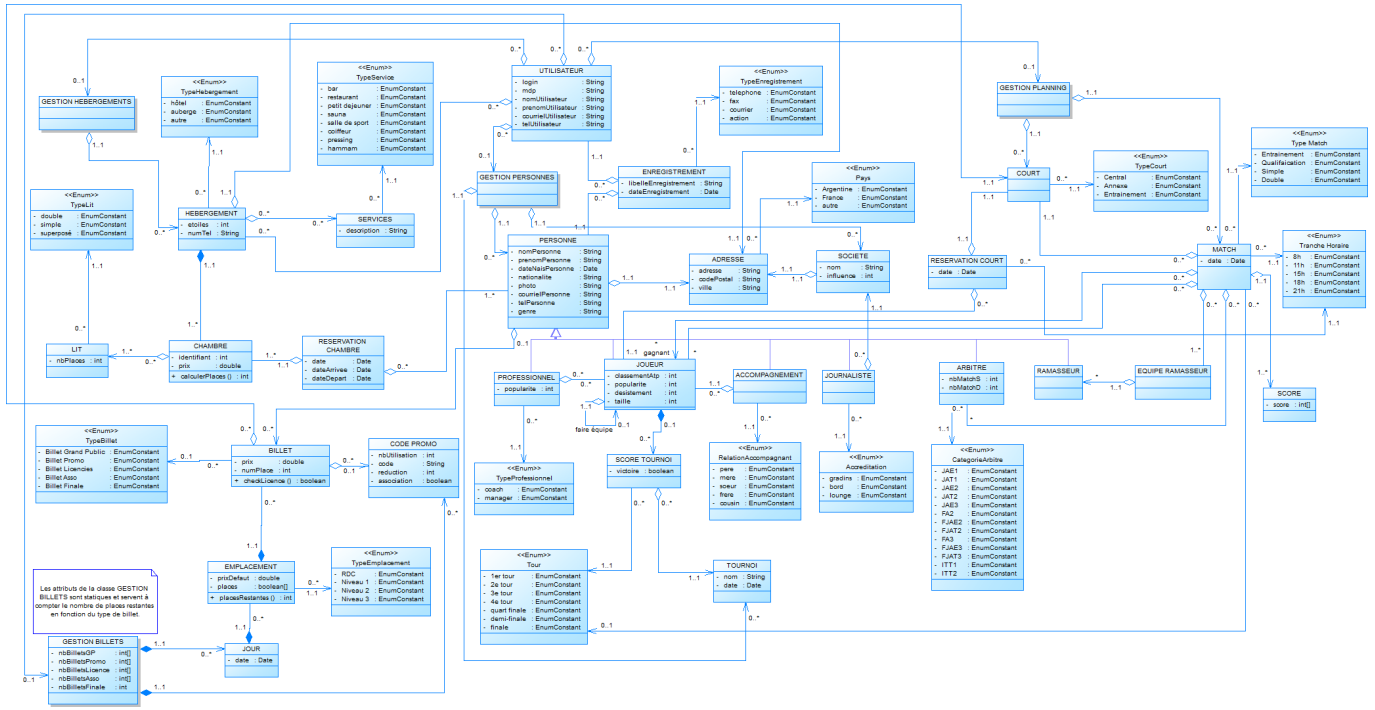
Il s'agit de concevoir le système informatisé permettant la gestion d'une salle de sport avec des machines connectées.

Présentation

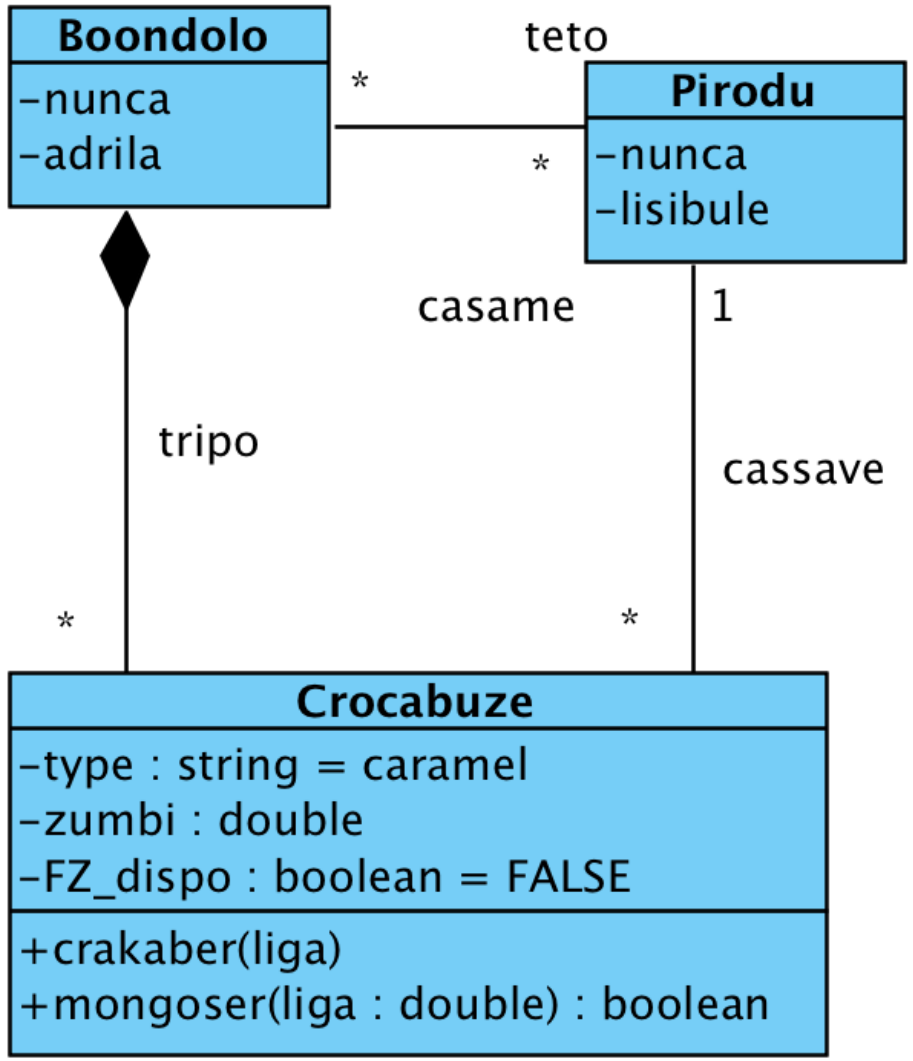
- Dans cette salle, chaque client est enregistré avec son nom, son âge, son adresse, et son téléphone. Ces informations sont inscrites sur la carte à puce que le club lui remet lors de son inscription. Le client souscrit un abonnement (mensuel, 3 mois, 6 mois ou annuel) qui est inscrit sur sa carte et qui lui permet d'accéder à certaines salles (salle de sport collectif, salle de musculation et sauna), mais aussi d'enregistrer des programmes et de décompter les exercices effectués sur les machines.
- Pour la musculation, le client a la possibilité de suivre un programme, défini soit avec un responsable de salle, soit avec un coach. Un programme est une succession d'exercices effectués sur les appareils de musculation, dans un certain ordre. Chaque exercice du programme est un exercice type (biceps, triceps, etc.) ayant une durée de base, auquel on associe un nb de répétitions et un poids défini par le client. L'application devra calculer ainsi la durée de l'exercice envisagé, et la durée du programme en fonction du nb de séries choisi (une série = une succession d'exercices). Un programme peut nécessiter une ou plusieurs séances (1 séance = 40').
- Sans coaching spécifique, le programme est défini par un responsable de la salle avec le client (ex. : « aujourd'hui, je compte faire 30 pectoraux sur la machine X chargée à 9 kg »). Le gérant enregistre le programme sur la carte. L'appareil de musculation possède un lecteur de carte. Ainsi, en autonomie, le client insère sa carte dans l'appareil qui lit les informations, décompte les mouvements effectivement réalisés et informe en temps réel le client de ce qui lui reste à faire.
- La salle de gym propose d'autre part un service de coaching payant pour ses clients (sur une période qui doit être couverte par l'abonnement). Le client qui veut profiter de ce service bénéficie d'un suivi par l'un des coachs de la salle. Il passe aussi des contrôles mensuels où l'on mesure son poids, sa fréquence cardiaque de repos *avant* et *après* l'effort, et le coach définit des programmes en fonction de ses objectifs.

Diagramme de Classes

Rester simple, contraintes minimales de notation



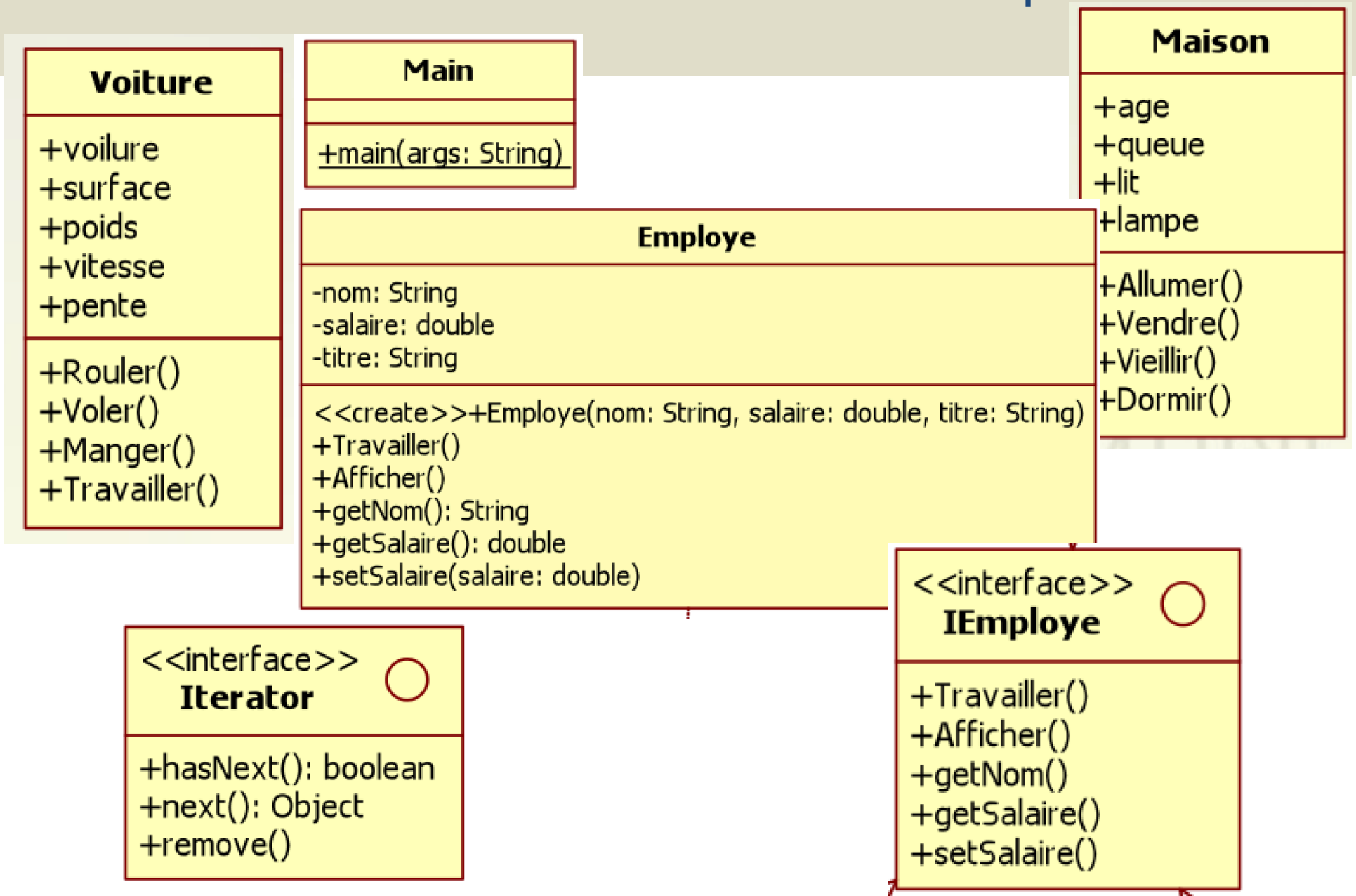
Que raconte ce diagramme ?



Utilisation DCL

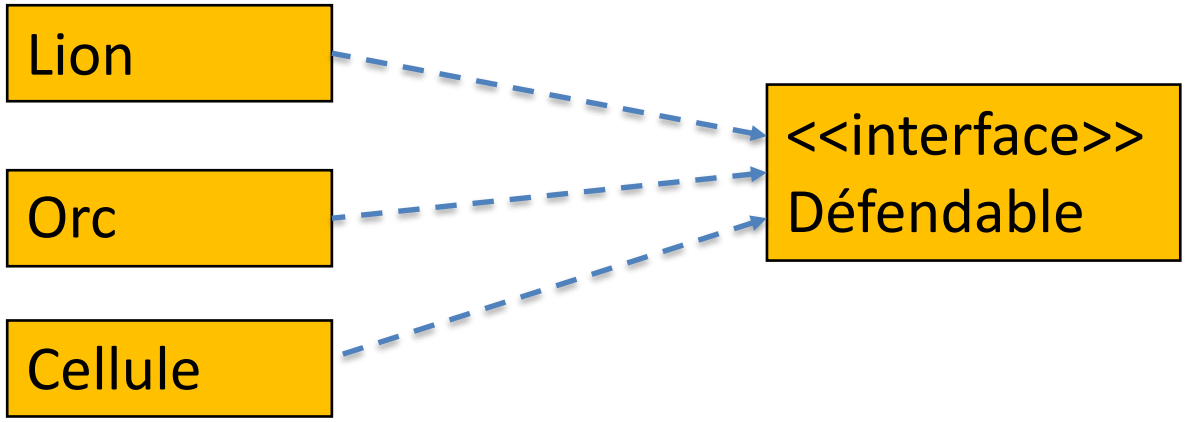
- Très nombreuses utilisations, à différents niveaux :
  - Pendant la capture des besoins :
    - **Modèle du domaine**
    - Classes correspondant à des objets du domaine
  - Pendant la conception / implémentation :
    - **Modèle de conception**
    - Classes correspondant à des objets logiciels

Classes du Domaine vs. Conception ?



Diagrammes de classes

- Les *diagrammes de classes* représentent un ensemble de classes, d'interfaces et de collaborations, ainsi que leurs relations.
- Ils présentent la vue de conception statique d'un système
- Représentation des relations qui existent entre les entités, indépendamment de leur cycle de vie



Kesaco, relations statiques ?

- Ex.: **un Contrat est associé à un Client**
- Synonyme : relations **structurelles**
  - Validité dans la durée, le long terme

- Le fait qu'un Contrat soit signé avec un Client et archivé par le Documentaliste relève de sa « dynamique ». On ne doit pas le modéliser dans le DCL (sinon par les opérations de la classe Contrat)
  - On ne décrit jamais QUI fait QUOI dans un DCL

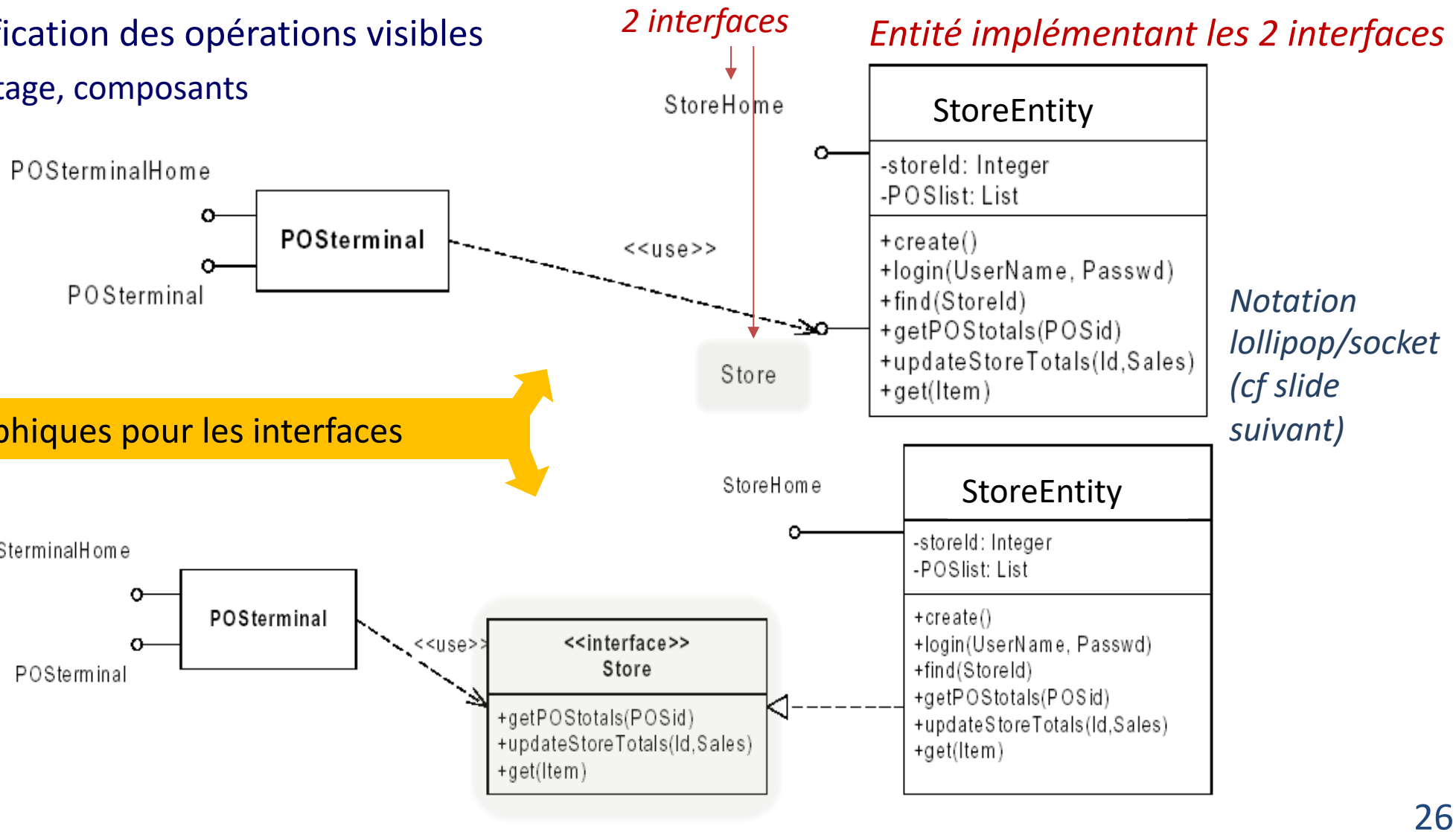
- Le fonctionnement **dynamique** d'une classe pourra être modélisé à travers des diagrammes d'état ou d'activité
  - Relation 'ponctuelle', le temps d'une action (message)

Relation « structurelle » ?

- Exemple avec un Système de Gestion Aéroport :
  - Si on envisage les classes **Avion, Compagnie** et **TourContrôle, Piste, Tarmac** … relations structurelles ?

Variantes de classes / package / composant : les interfaces

- Interfaces : spécification des opérations visibles
  - Classes, paquetage, composants

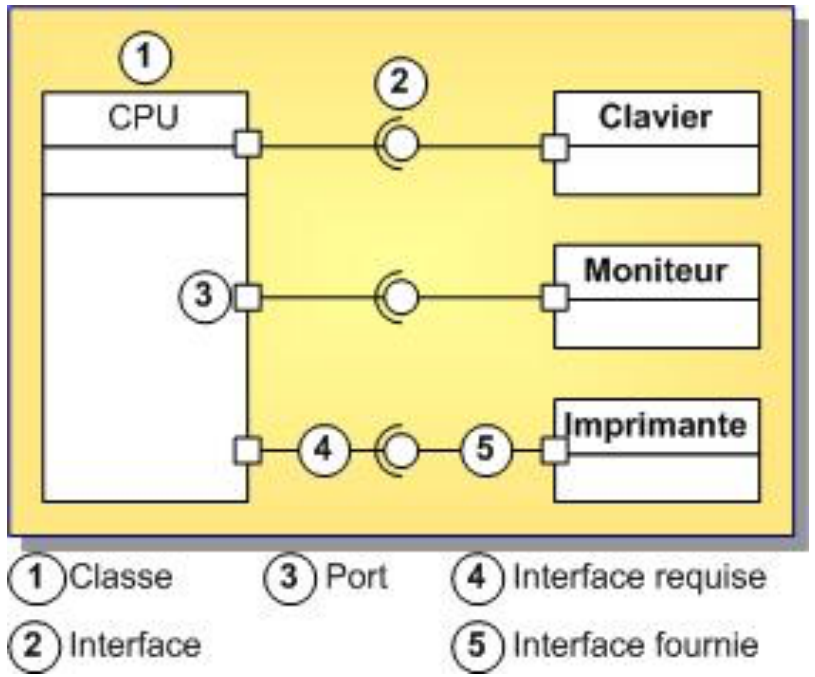


Interface / port (UML2)

- Une classe peut **offrir** un service :
  - (la notation se lit « lollipop »)

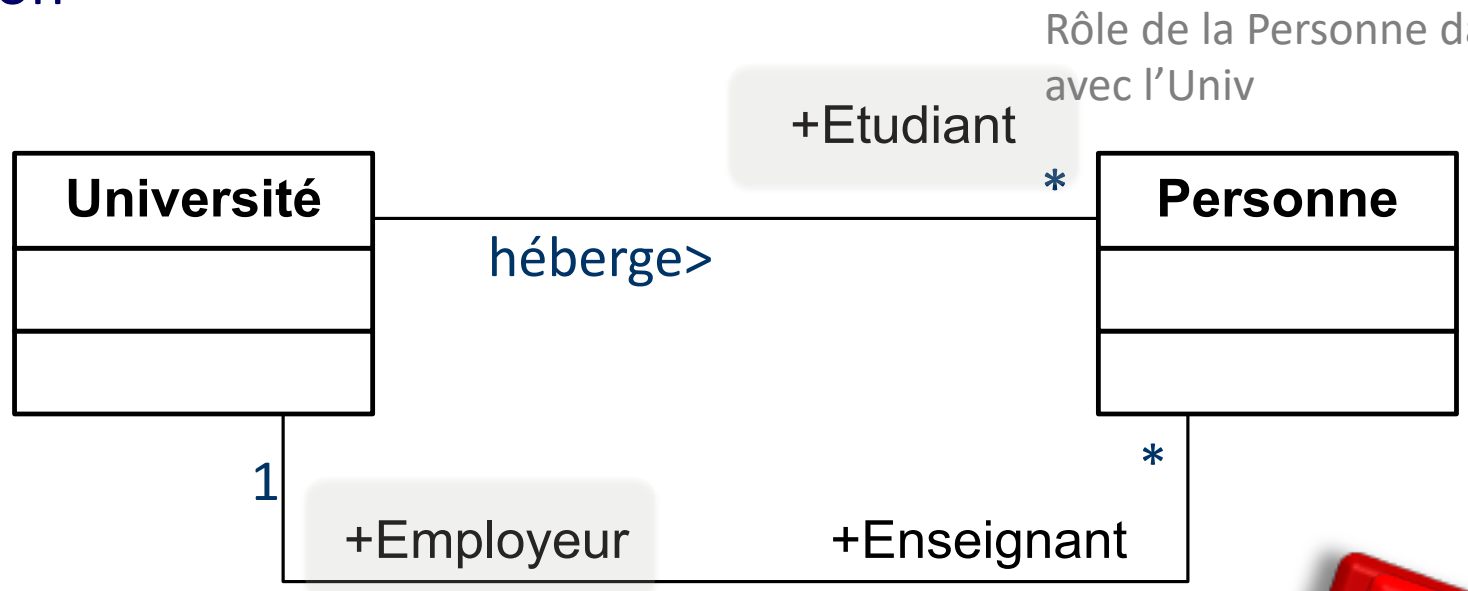
- Une classe peut **avoir besoin** d'un service :

  - (Notation « socket »)
    - Un port spécifie un point d'interaction distinct entre ce classificateur et son environnement
      - ou entre le classificateur et ses parties internes



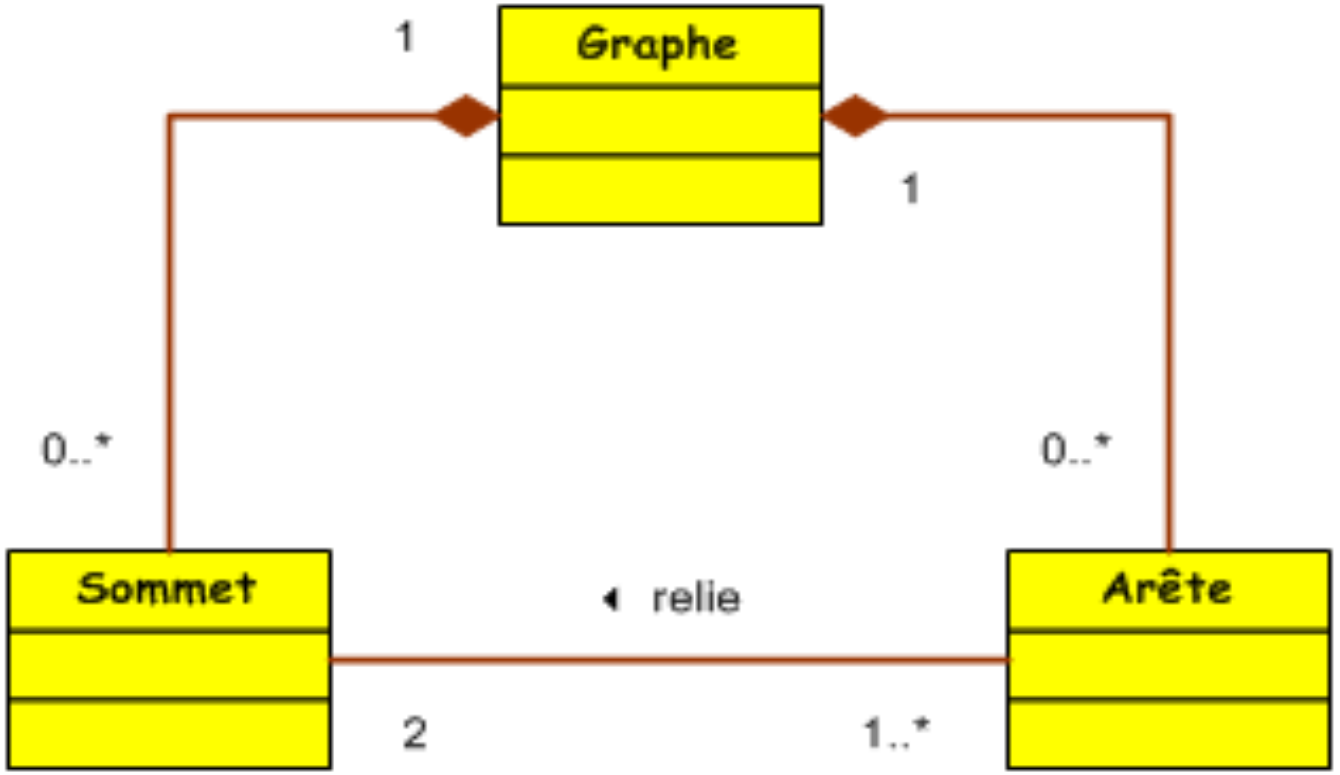
Nommage des rôles

- Le rôle décrit une extrémité d'une association, c'est à dire le rôle que joue la classe dans la relation



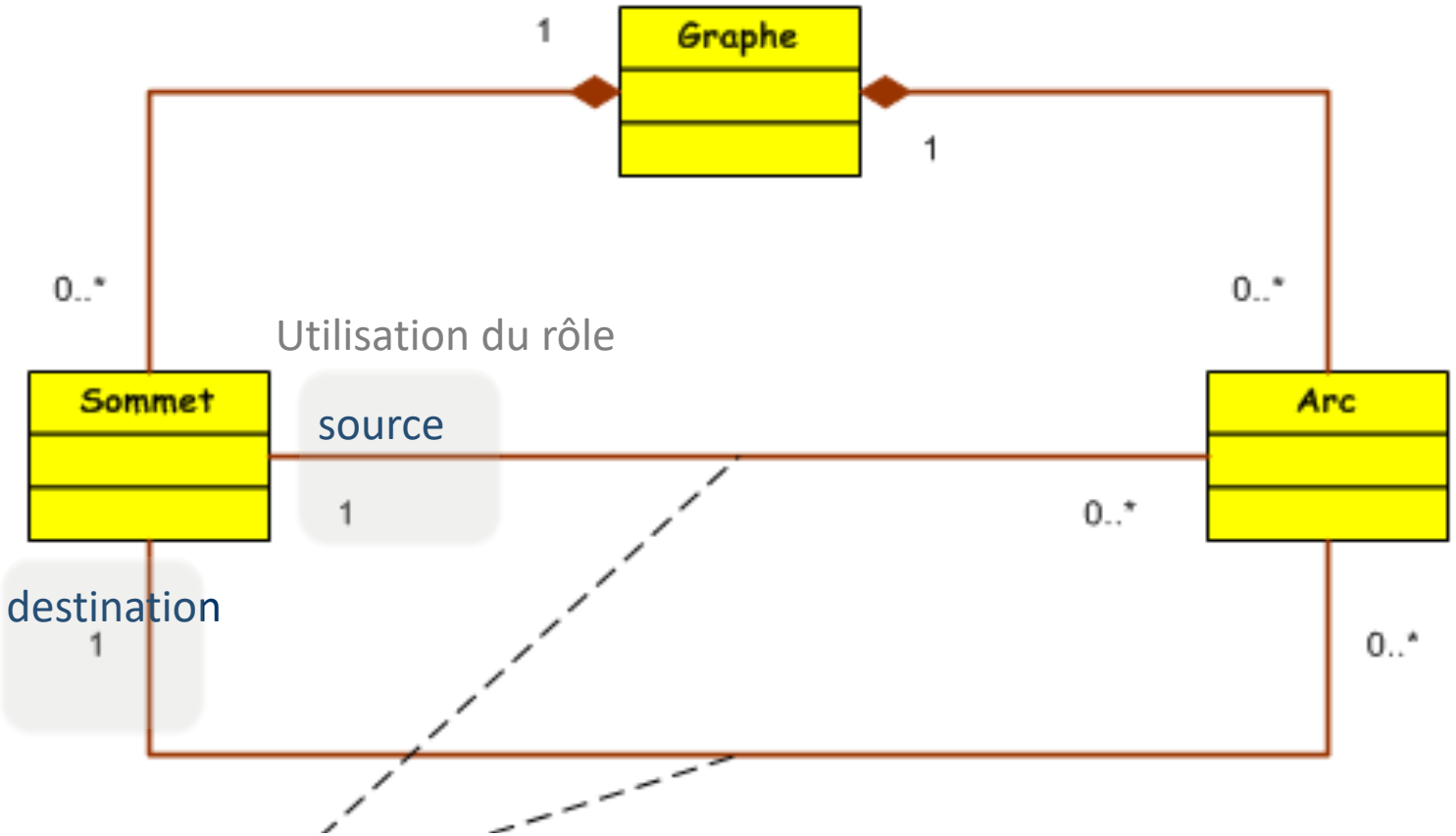
Il peut y avoir plus d'une association entre 2 classes

Exemple : graphe (1) non orienté



Un graphe possède des arêtes et des sommets, quand je crée le graphe je crée aussi ces éléments (idem quand je le supprime).
Une arête relie exactement 2 sommets.

Exemple : graphe (2) orienté



Ici avec les rôles, on a une information plus précise qui permet de distinguer les 2 sommets : la source et la destination, puisque l'arc est orienté

Navigabilité : traduit le sens de parcours d'une association

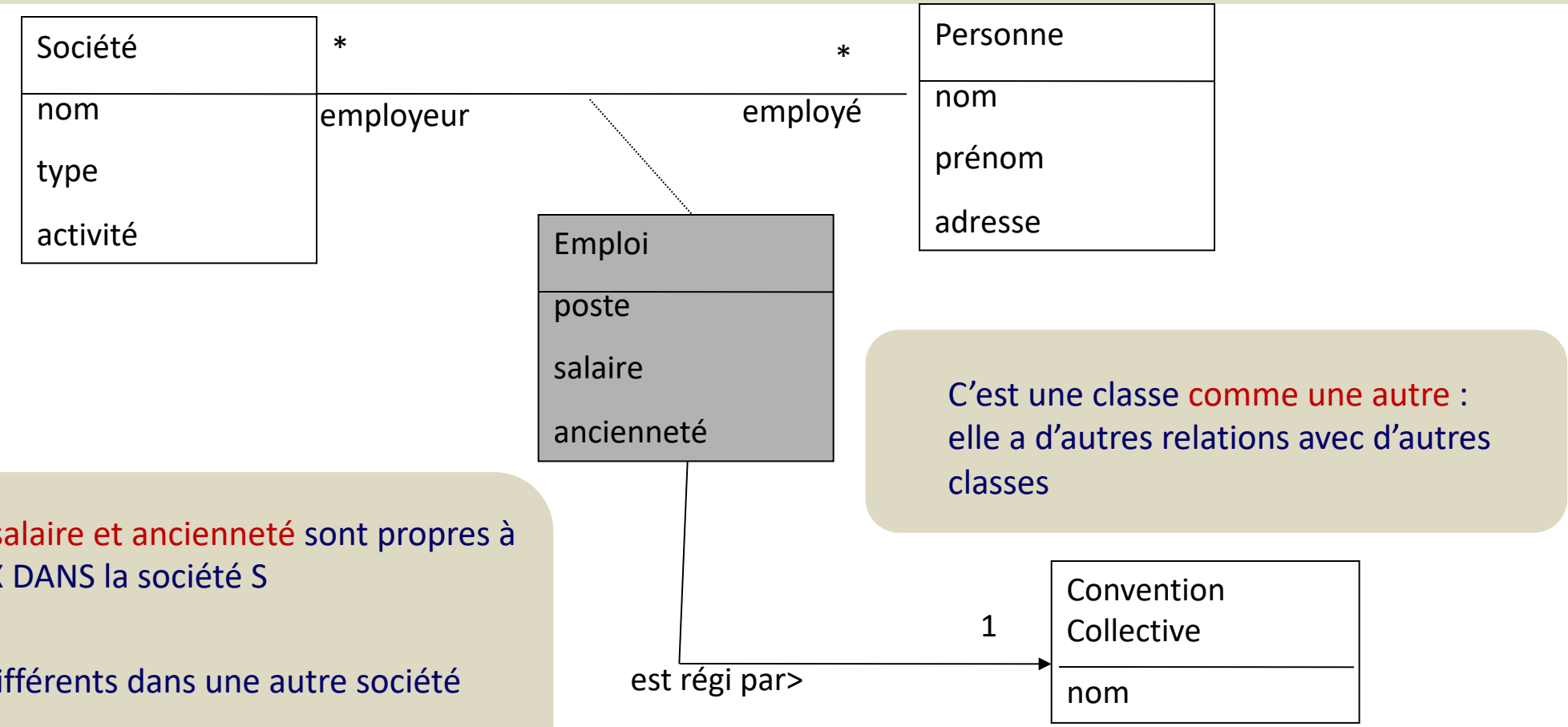
- Par défaut une association est *navigable* dans les deux sens

- L'indication de navigabilité (flèche à une extrémité) suggère que seul un objet peut directement atteindre l'objet relié

- Etant donné un utilisateur, on désire pouvoir accéder à ses mots de passe

- Etant donné un mot de passe, on ne souhaite pas pouvoir accéder à l'utilisateur correspondant

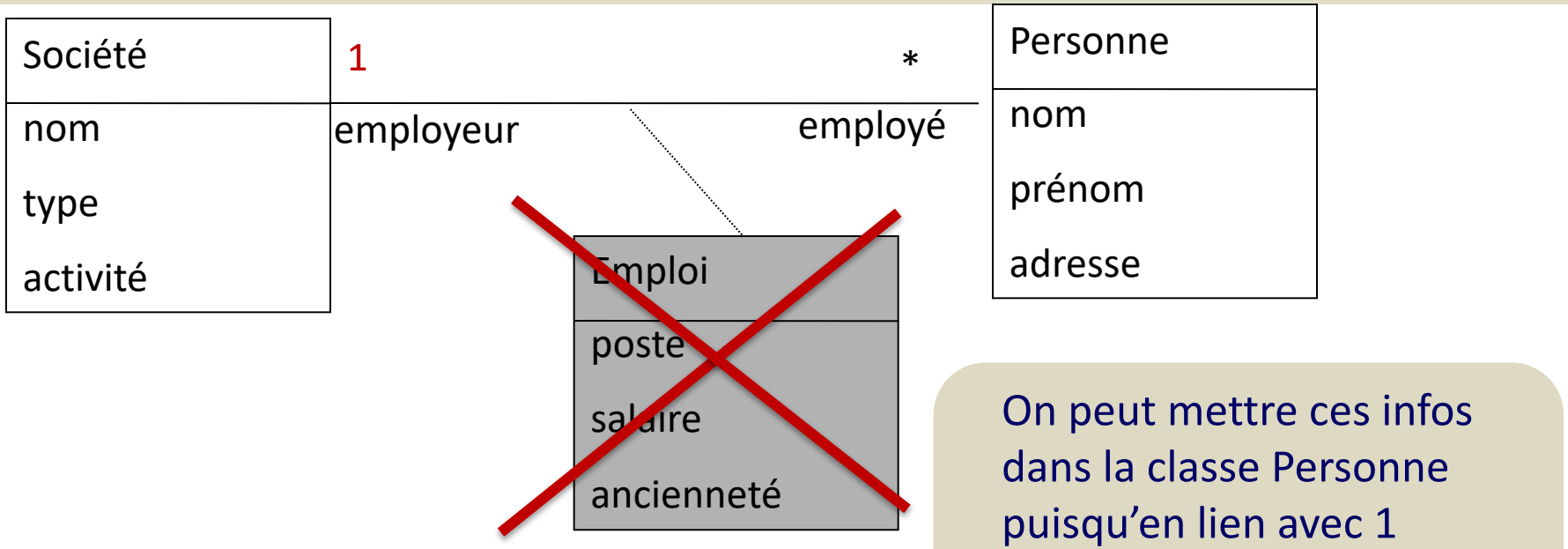
Exemple de Classe d'association



C'est une classe comme une autre : elle a d'autres relations avec d'autres classes

- Les poste, salaire et ancienneté sont propres à l'employé X DANS la société S
- Ils seront différents dans une autre société

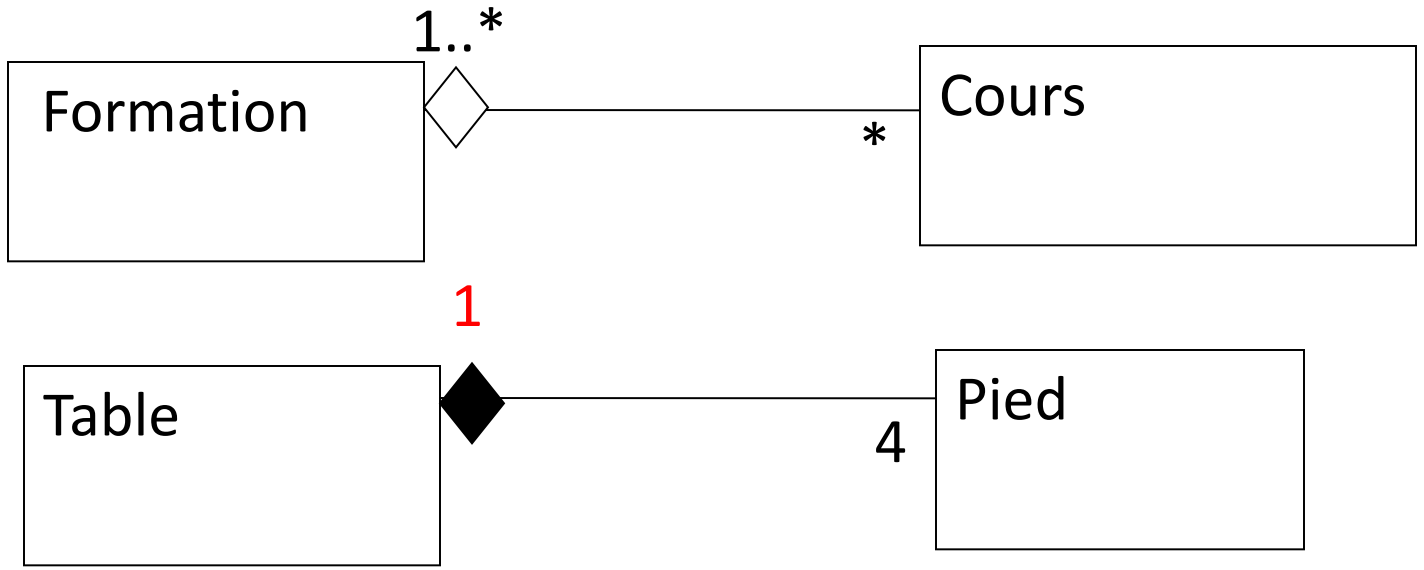
ERREUR de Classe d'association



On peut mettre ces infos dans la classe Personne puisqu'en lien avec 1 employeur

Classe d'association : uniquement sur des relations *n,n*

Agrégation et composition



Relations non symétriques

Agrégation : la partie peut être partagée entre plusieurs entités, les cycles de vie des instances ne sont pas imbriqués

Composition : la partie est uniquement celle du composé, les cycles sont imbriqués (création et suppression)

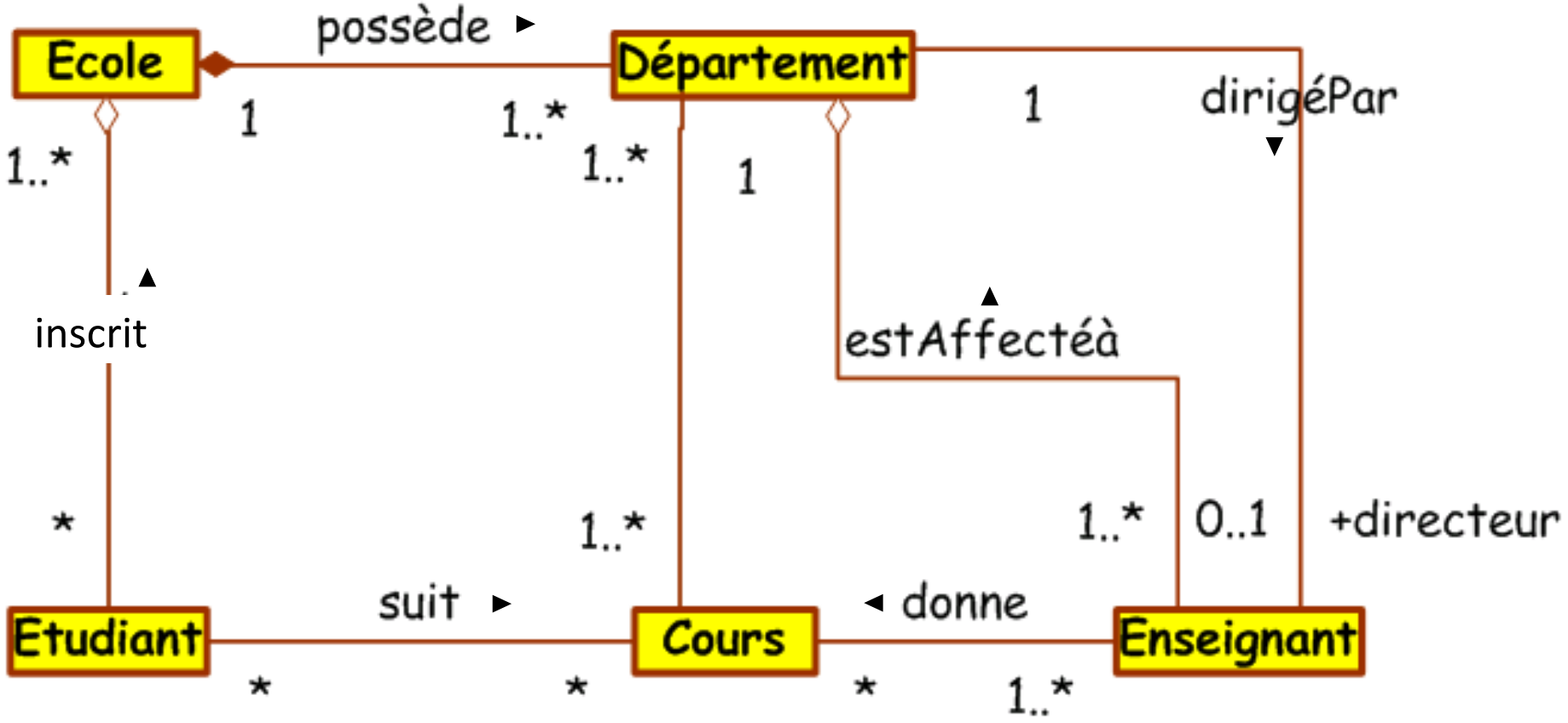
Agrégation / Composition

- Une agrégation exprime qu'il existe une **contrainte d'intégrité** avec des classes dépendantes
  - C'est l'agrégat qui **est responsable** du respect de ces contraintes d'intégrité
  - Ex. une équipe comporte *n* personnes. Les personnes existent à part entière, de manière indépendante. L'équipe sait qu'elle est au complet par ex. quand toutes les personnes lui sont affectées.

Exercice : agrégation ou composition ?

- Une **voiture** possède 2 **essieux**, chaque **essieu** possède 2 **roues**
- Une **société** a des **salariés**
- Un **système de fichiers** possède des **répertoires;** un **répertoire** contient des **fichiers**
- Un **échiquier** est composé de 64 **cases**
- Toute **fenêtre** possède un bouton **Valider** et un bouton **Annuler**
- Une **page web** possède une **feuille de styles**

Exemple : Ecole d'ingénieurs



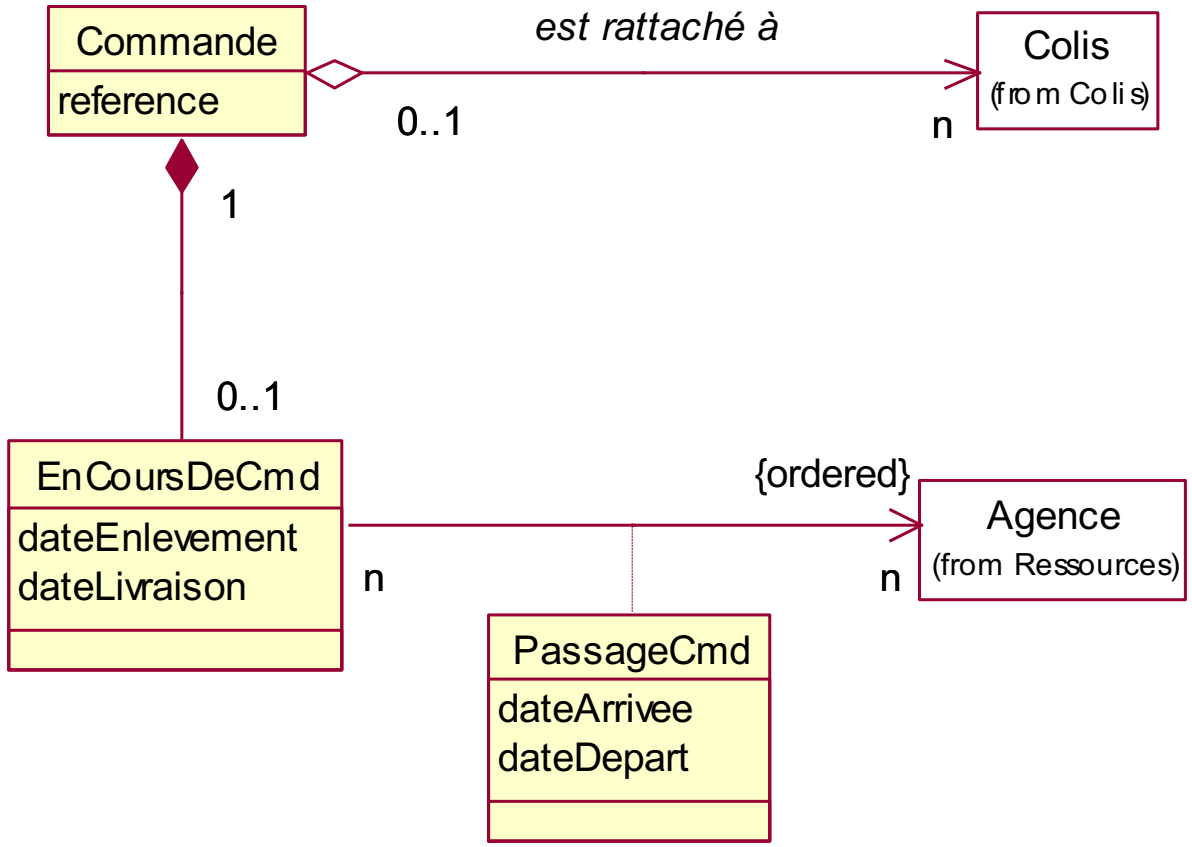
Ecole : implémentation d'une agrégation et composition

```
public class Ecole {
   private List<Departement> listDepartements;
   private List<Etudiant> listEtudiants;

   Ecole() {
   /* création et initialisation des départements
      dans le constructeur de l'école */
   /* + création du conteneur d'étudiants, vide */ ...
    }
    private setListEtudiants(List<Etudiant> uneListeEtu) {
        /* copie d'une liste dans une autre, les étudiants
           ont été créés par ailleurs */
    }
    private addEtudiant(Etudiant unEtu) {
        /* ajout de l'étudiant unEtu à la liste actuelle */
    }

}
```

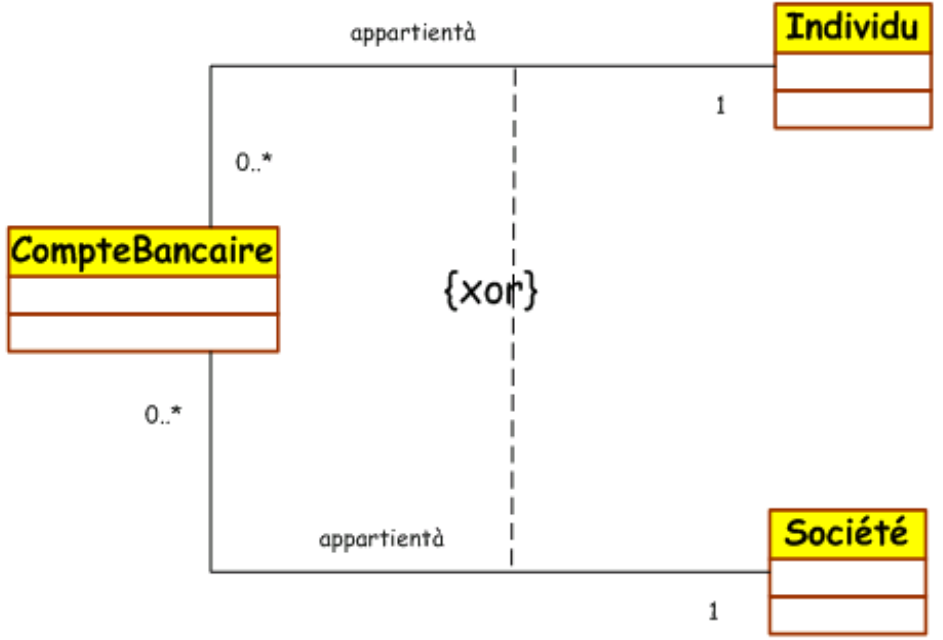
composition

agrégation

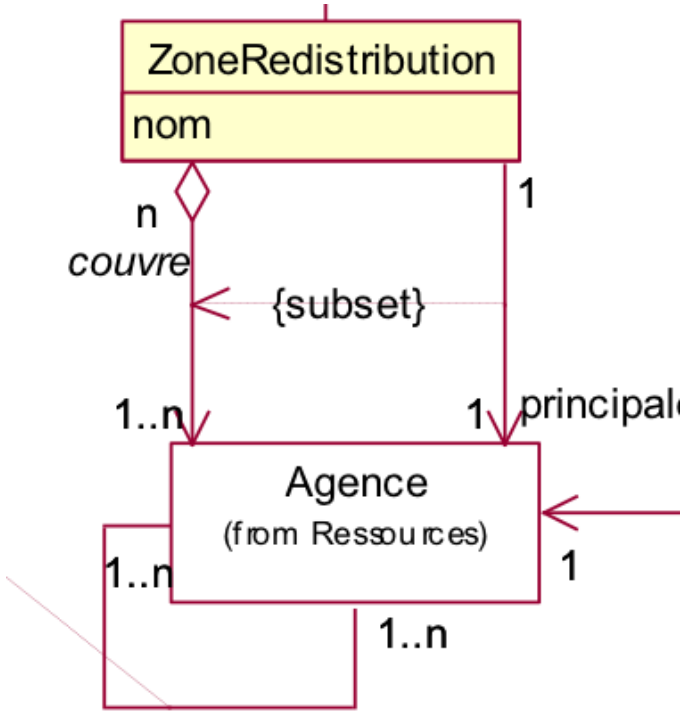
Contraintes sur les associations



{ordered}



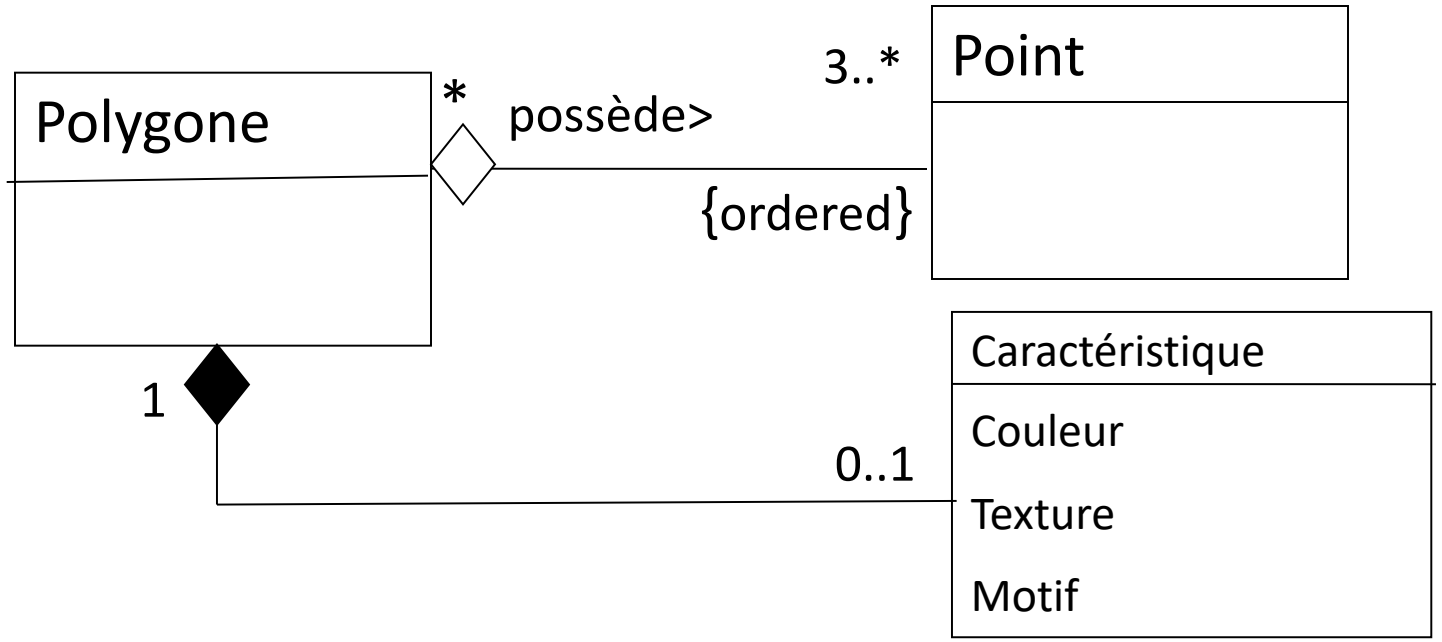
{XOR}



{subset}

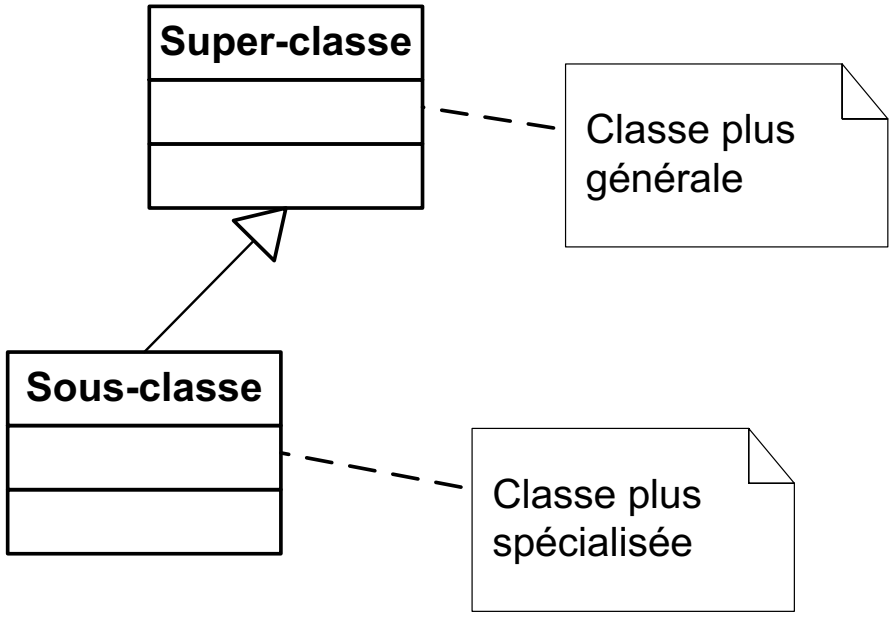
Exemple de contrainte

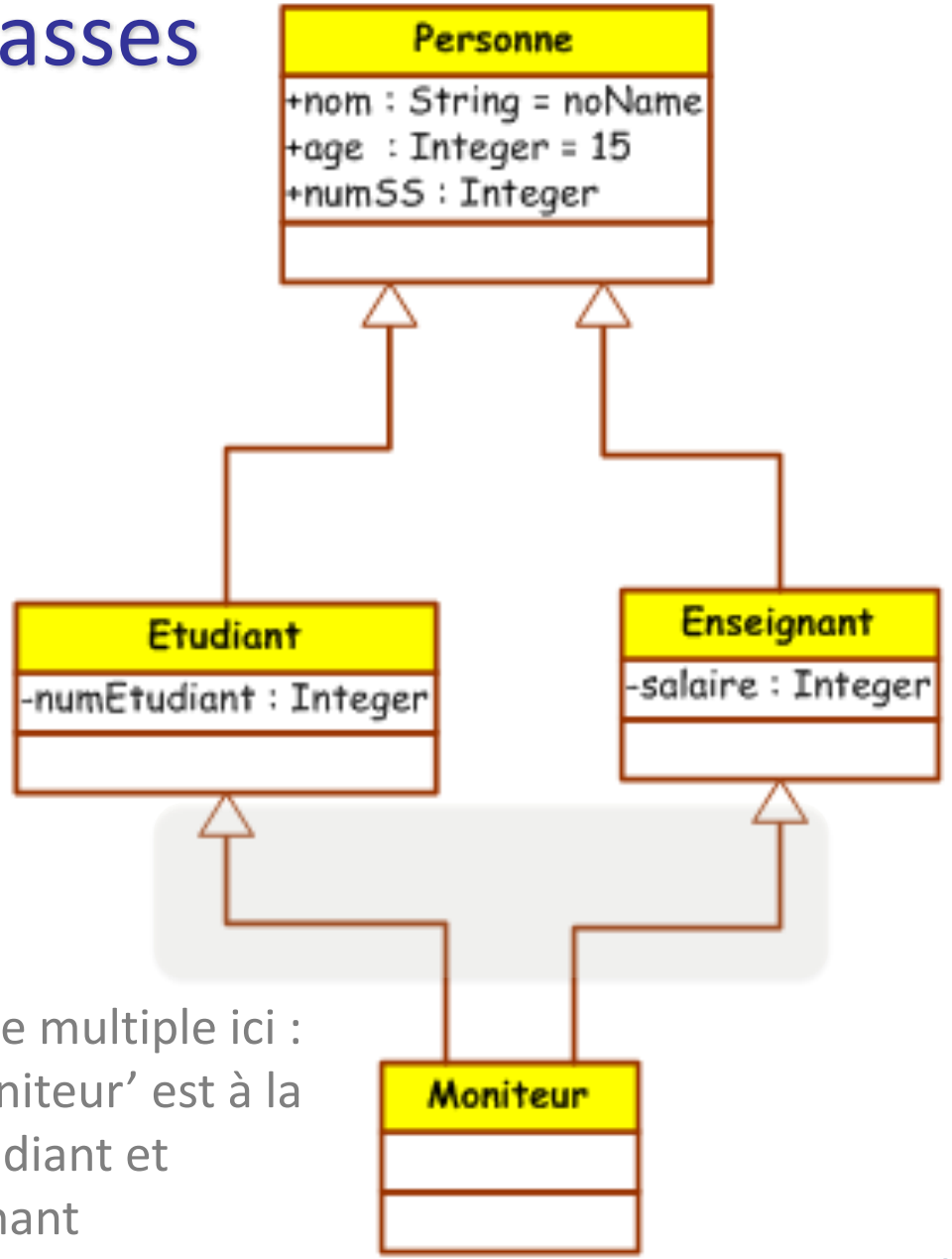
- Un **polygone** est défini à partir d'au moins 3 points
- C'est un **agrégat** de points ordonnés
- Il peut être composé d'une **caractéristique**



Relation d'héritage : hiérarchies de classes

- Gérer la complexité
  - Arborescences de classes d'abstraction croissante





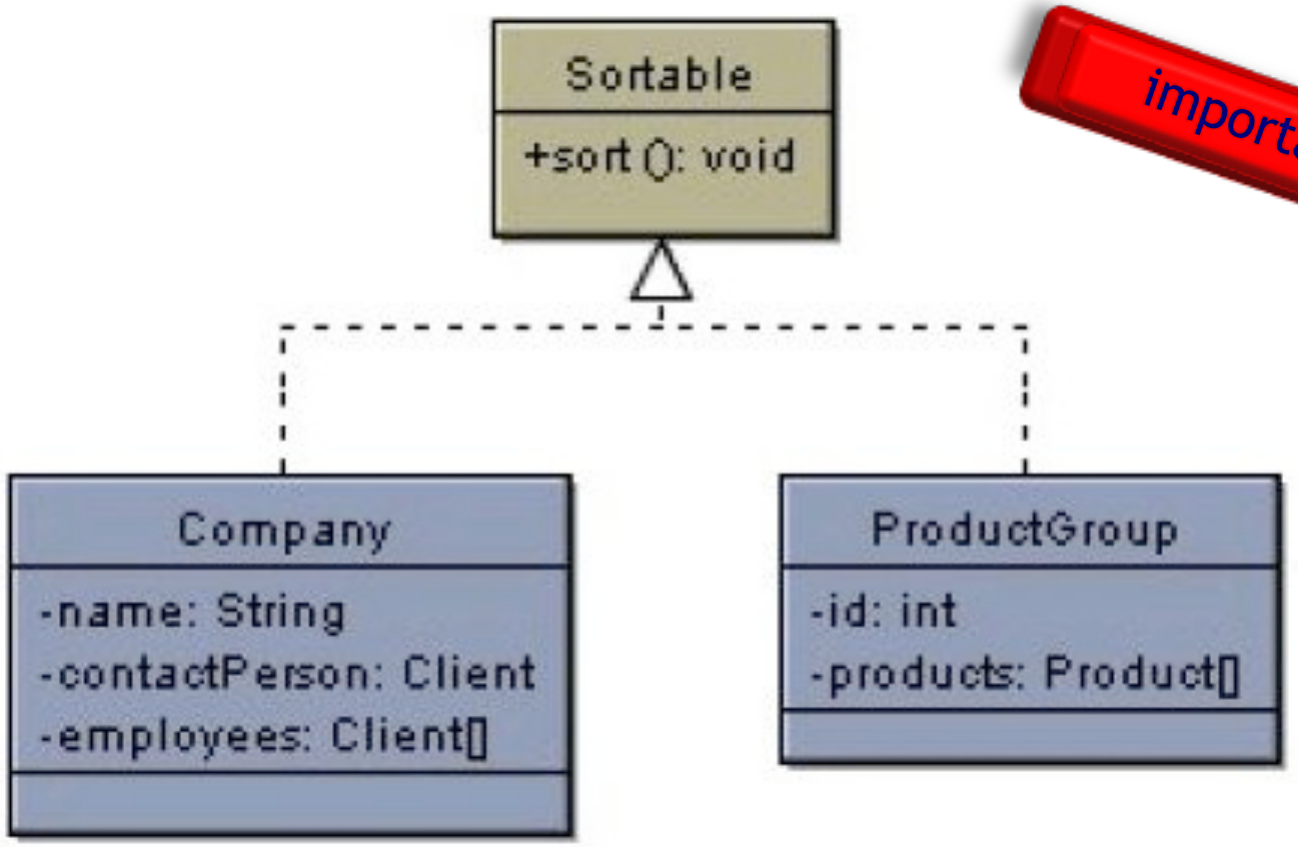
Héritage multiple ici : un 'moniteur' est à la fois étudiant et enseignant

Relation d'implémentation

Ici les classes Company et ProductGroup implémentent l'interface **Sortable**, ie définissent un code d'implémentation pour la méthode **sort**().

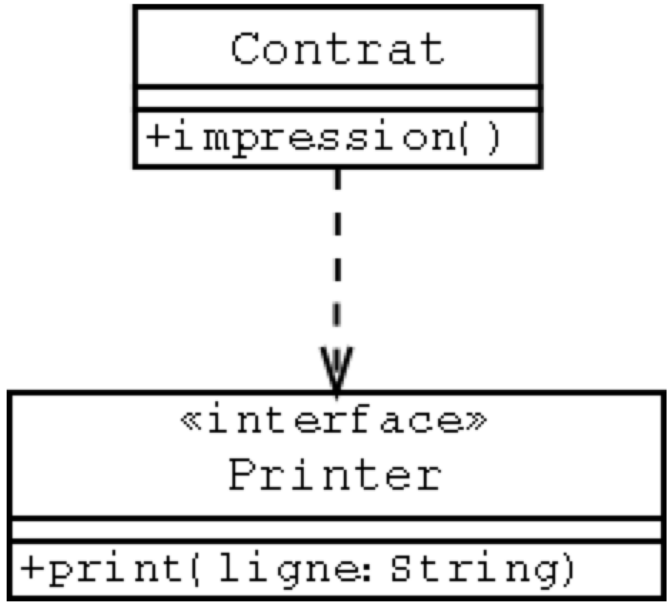
Trier des employés

Trier des produits



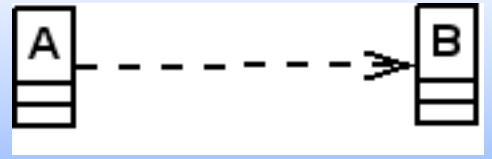
Relation de dépendance

- Relation entre deux éléments de modélisation
  - Toute modification effectuée sur un élément de modélisation (*influent*) affecte l'autre élément (*dépendant*)
- Dépendance : différent d'une relation *structurelle*
  - L'objet peut avoir besoin à un moment donné, des services d'un autre objet
  - Ex.: un contrat dispose d'un service d'impression (méthode `impression()`), qui utilise une méthode (`print()`), dont la spécification est déclarée par l'interface Printer.




```
class Contrat {

  ...
  public void impression() {
     Printer imprimante = PrinterFactory.getInstance();
     ...
     imprimante.print(client.getName());
    ...
  }
}
```

Relation de dépendance



Une classe A dépend d'une autre classe B quand celle-ci utilise, à un moment dans son comportement, au moins un élément de B.

Il sera donc nécessaire de redéfinir une partie de A si cette partie de B est modifiée.

- Relation la plus générique du diagramme de classe.
- Relation non transitive : si A dépend de B et B dépend de C, A ne dépend pas nécessairement de C

S'applique à:

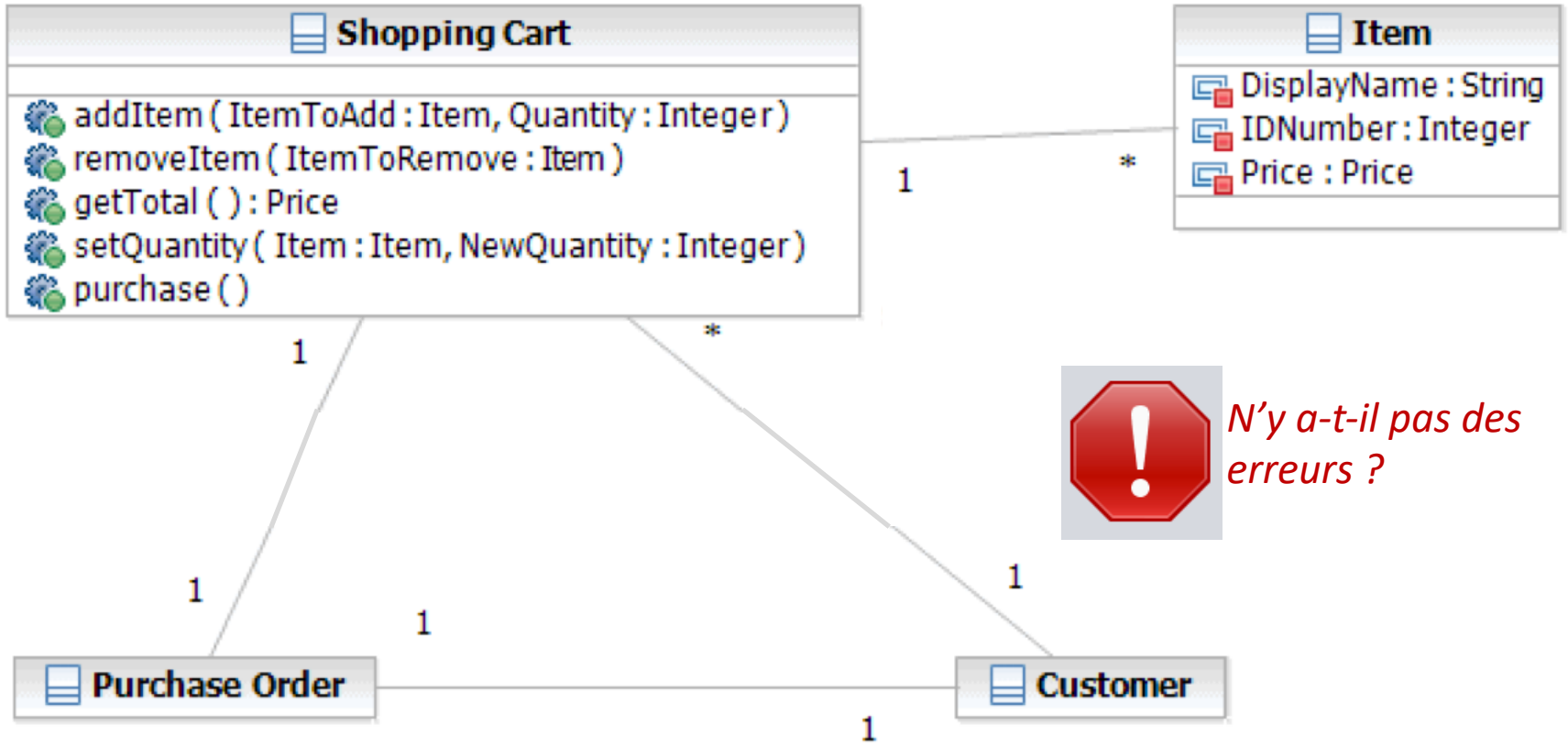
- Diagramme de cas d'utilisation
- Digramme de classes
- Diagramme d'objets
- Diagramme de composants
- Diagramme de déploiement

On cherchera à limiter ces dépendances, pour éviter le plat de spaghettis

DCL : Règles de nommage

- Modéliser pour exploiter la génération automatique de code
- Génération automatique de code ➔ respecter au maximum les **chartes de nommage** ainsi que les bases de syntaxe des langages
  - ✅ Nom de classe commence par une **Maj** ; tout le reste en minuscule (sauf les constantes en maj.);
  - ✅ Noms de classe au **singulier** ;
  - ✅ Séparer les mots composés par des majuscules ;
  - ✅ Ecrire soit en FR soit en Anglais, pas de mélange

Que raconte ce diagramme (trouvé sur le cloud) ?



N'y a-t-il pas des erreurs ?

http://www-01.ibm.com/support/knowledgecenter/SS8PJ7_9.1.1/com.ibm.xtools.modeler.doc/topics/cclassd.html?lang=fr

DCL : conclusion

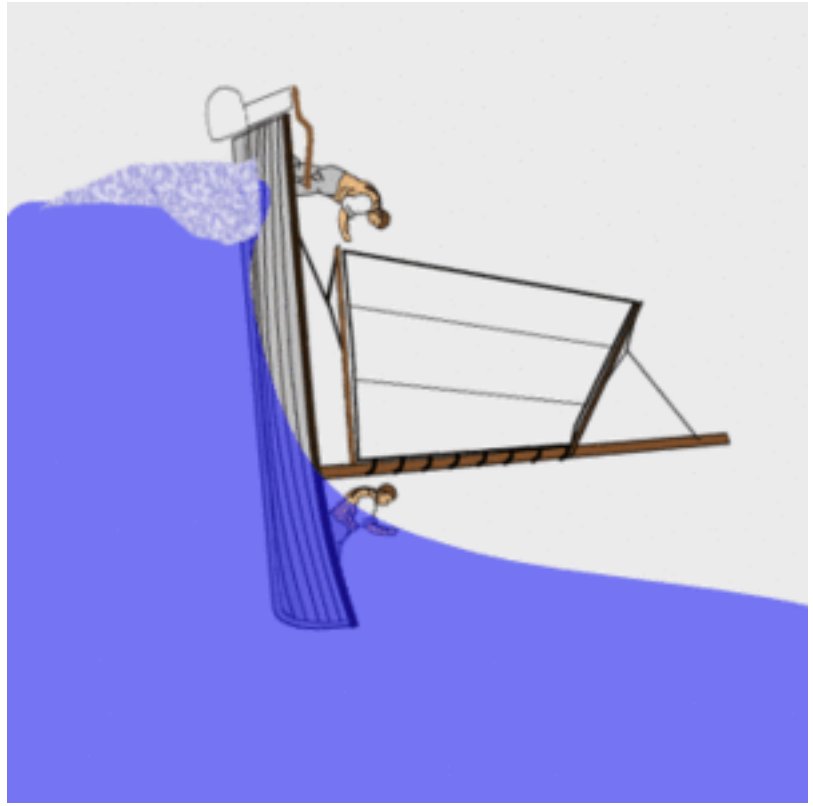
- Un DCL n'est pas forcément une représentation **exhaustive** d'un modèle
- Un diagramme **bien structuré** doit :
  - Être centré sur la communication d'une *vue* du système
  - Ne contenir que les éléments *essentiels à la compréhension* de cet aspect. Un diagramme peut cacher des parties du modèle si cela sert la communication

- *Les modèles ne sont pas **justes** ou **faux**; ils sont seulement plus ou moins utiles - Martin FOWLER*
- *« Un bon modèle n'est pas un modèle auquel on ne peut plus rien **ajouter**, mais un modèle auquel on ne peut plus rien **ENLEVER** » St Exupéry*

DCL : je retiens

- Les classes et les relations **structurelles**
- Les **différents types** de relations
- Importance des **cardinalités**, comment on les détermine
- La notion de **rôle d'une classe**
- Les **classes d'association** sur les relations n,n
- Les **propriétés** *ordered*, *XOR* et *subset* des associations

Atelier Naufrage



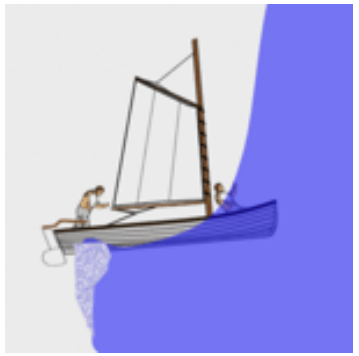
Hiérarchiser les besoins

Enoncé

- Vous êtes sur un bateau dans l'océan Pacifique, à 1800 km de toute côte. Le feu est déclaré à bord. D'ici 1h, le bateau aura coulé. Il y a des barques de sauvetage à rames en nbre juste suffisant pour sauver tout le monde.

- Peut-être pourrait-on emmener quelques objets à bord ? L'équipage rapporte une liste d'objets susceptibles d'être emporté, il y en a 15.

- Il faut en choisir 5. Comment procédez-vous ?



15 objets possibles

1. Sextant + tables de calcul
2. Miroir pour se raser
3. Bidon de 15l. d'eau
4. Moustiquaire
5. Paquet de ration de survie de l'armée
6. Carte du Pacifique
7. Coussin flottant
8. Bidon 10l. mélange essence gazoil
9. Petite radio réceptrice
10. Produit anti-requin
11. 3 m² de toile plastique
12. Bouteille rhum
13. 5 m. corde nylon
14. 2 boites tablettes choc.
15. Équipement de pêche

Si on ne devait emporter que 5 objets, lesquels choisir ?

LA SOLUTION ?

- Proposée par les spécialistes de la Marine
- Il faut **identifier les objectifs** poursuivis et les **hiérarchiser**

- 5 objectifs sont identifiés par la Marine :
  - Soutenir le moral des naufragés
  - Survivre
  - Se diriger vers la côte
  - Se faire repérer
  - Repêcher et soigner ceux qui tombent à l'eau

- A votre avis, comment les hiérarchiser ?

LA SOLUTION

- Proposée par les spécialistes de la Marine
- Il faut **identifier les objectifs** poursuivis et les **hiérarchiser**

- 5 objectifs **hiérarchisés** sont identifiés par la Marine :
  - Se faire repérer
  - Survivre
  - Repêcher et soigner ceux qui tombent à l'eau
  - Soutenir le moral des naufragés
  - Se diriger vers la côte

Liste des priorités

Se faire repérer

1. Miroir pour se raser
2. Bidon 10l. mélange essence gazoil

Survivre

3. Bidon de 15l. d'eau
4. Paquet de ration de survie de l'armée (nourriture concentrée)
5. 2 boites tablettes choc.
6. 3 m² de toile plastique (recueillir eau pluie : suppose qu'il pleuve)
7. Équipement de pêche (suppose qu'il y ait du poisson)

Repêcher et soigner ceux qui tombent à l'eau

8. 5 m. corde nylon (lancer à celui qui est tombé)
9. Coussin flottant (bouée sauvetage)
10. Produit anti-requin
11. Bouteille rhum (soigner blessures)

Soutenir le moral des naufragés

12. Petite radio réceptrice
13. Moustiquaire (protection insectes)

Se diriger vers la côte

14. Carte du Pacifique
15. Sextant + tables de calcul (se diriger)

Parallèle avec le développement Logiciel

- Revenir aux **objectifs fondamentaux** du Client
- Les **hiérarchiser**
- Démarrer le développement sur les **objectifs prioritaires**

- Problème : **le Client ne sait pas toujours quels sont les objectifs prioritaires**
  - Un travail est nécessaire, mené avec l'analyste
- L'analyste *n'est pas du domaine* et *ne peut pas* identifier les besoins prioritaires
  - Même si pour nous, le bon sens voudrait que…
  - La preuve avec cet exercice : seule la connaissance Métier compte