Etape de Conception

à partir d'un modèle UML sous PowerAMC

Véronique Deslandres©, IUT,
Département Informatique
Université de Lyon

MàJ: mai 2019

Introduction

- En Analyse et Conception, il y a 2 étapes bien distinctes
  - L'**Analyse** effectuée avec les clients / utilisateurs
    - But : comprendre les besoins, le problème et le domaine
    - Maturation *progressive* de la connaissance
    - Englobe *plusieurs diagrammes* UML
  - La **Conception**, effectuée par les informaticiens
    - But : proposer une solution qui réponde aux besoins issus de l'analyse
    - Partir des DCL d'analyse pour produire des *DCL « de Conception »*

    + parfois des Diagrammes de Séquences très ciblés

1

Comment procéder avec l'AGL PAMC

Créer le modèle de **conception**

Création d'un DCL de Conception

- Dupliquer les DCL d'analyse pour la Conception
  - Obj.1 : conserver l'analyse **indépendamment** de la plateforme cible
    - Le DCL de Conception repose une technologie cible (java, XML)
  - Obj.2 : garder une modélisation de l'analyse **propre, lisible**
  - Or le DCL de Conception est *rarement* lisible

- 2 solutions :
  - *Générer* un MOO de Conception à partir du MOO d'Analyse via PAMC
    - Auquel cas, casser la dépendance entre les 2
  - Dupliquer le projet sous l'OS (en dehors de PAMC)
    - Plus simple

1.1 – Conception : enrichissement du DCL

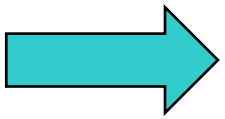
- Changer le langage cible
  - Menu **Langage :** *Changer de langage objet courant*
  - Basculer du langage *Analyse* à la technologie cible (ex. java)
- On ajoute :
  - Les identifiants
    - Vont servir de clef primaire aux tables relationnelles
  - Les types (des paramètres, de retour)
  - Les constructeurs
  - Les accesseurs (get/set)

- Aussi :
  - On précise les attributs / retours **multiples**
  - On vérifie la **navigabilité**
  - On mentionne les classes persistantes

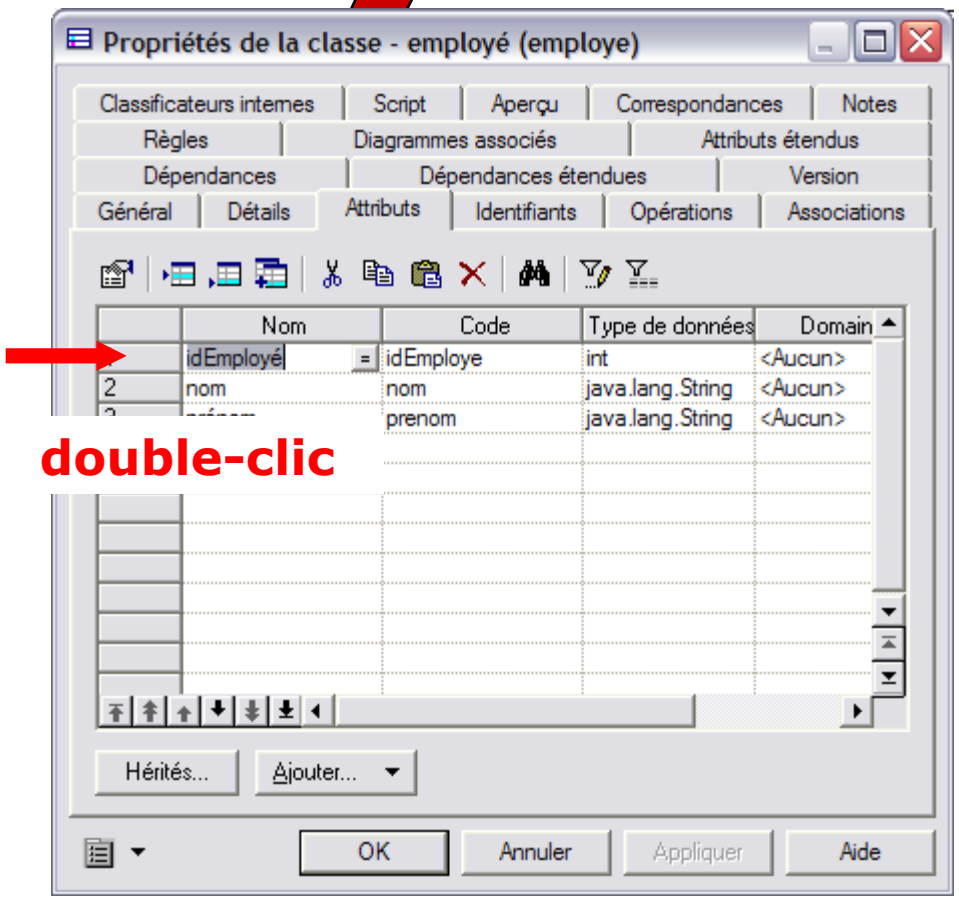
Ajouter un identifiant avec PAMC

- Fenêtre Propriété de l'attribut
- Case à cocher dans l'onglet Détail : « identifiant primaire »

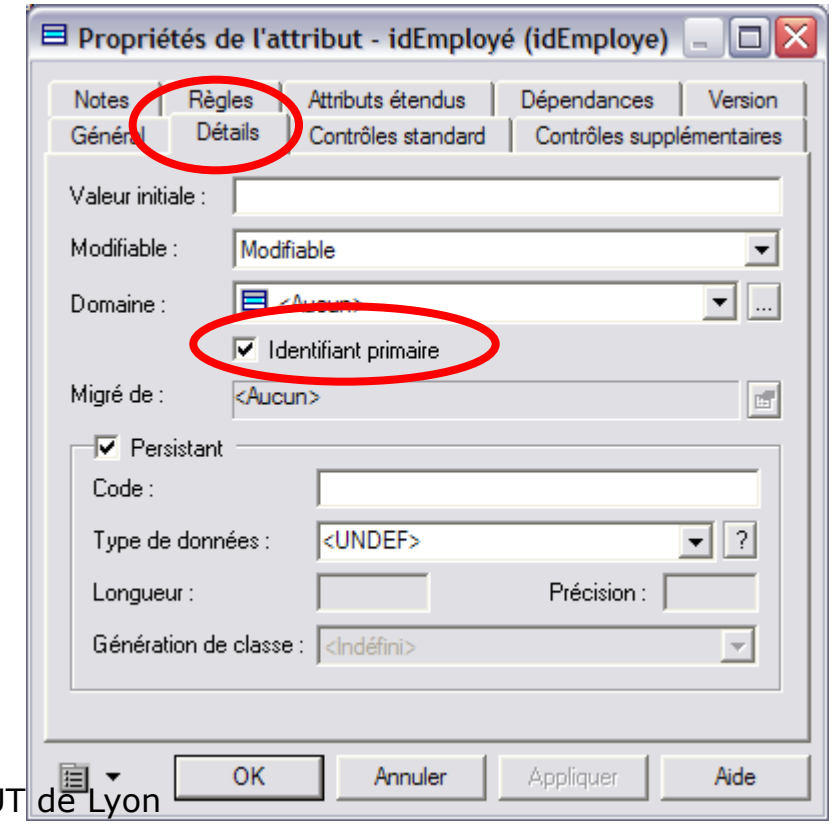
- On obtient la fenêtre Propriété en cliquant sur l'attribut de la classe dans le navigateur d'objet
  - (ou Double clic sur la ligne de l'attribut depuis la fenêtre Propriété de la Classe)



Créer un identifiant primaire



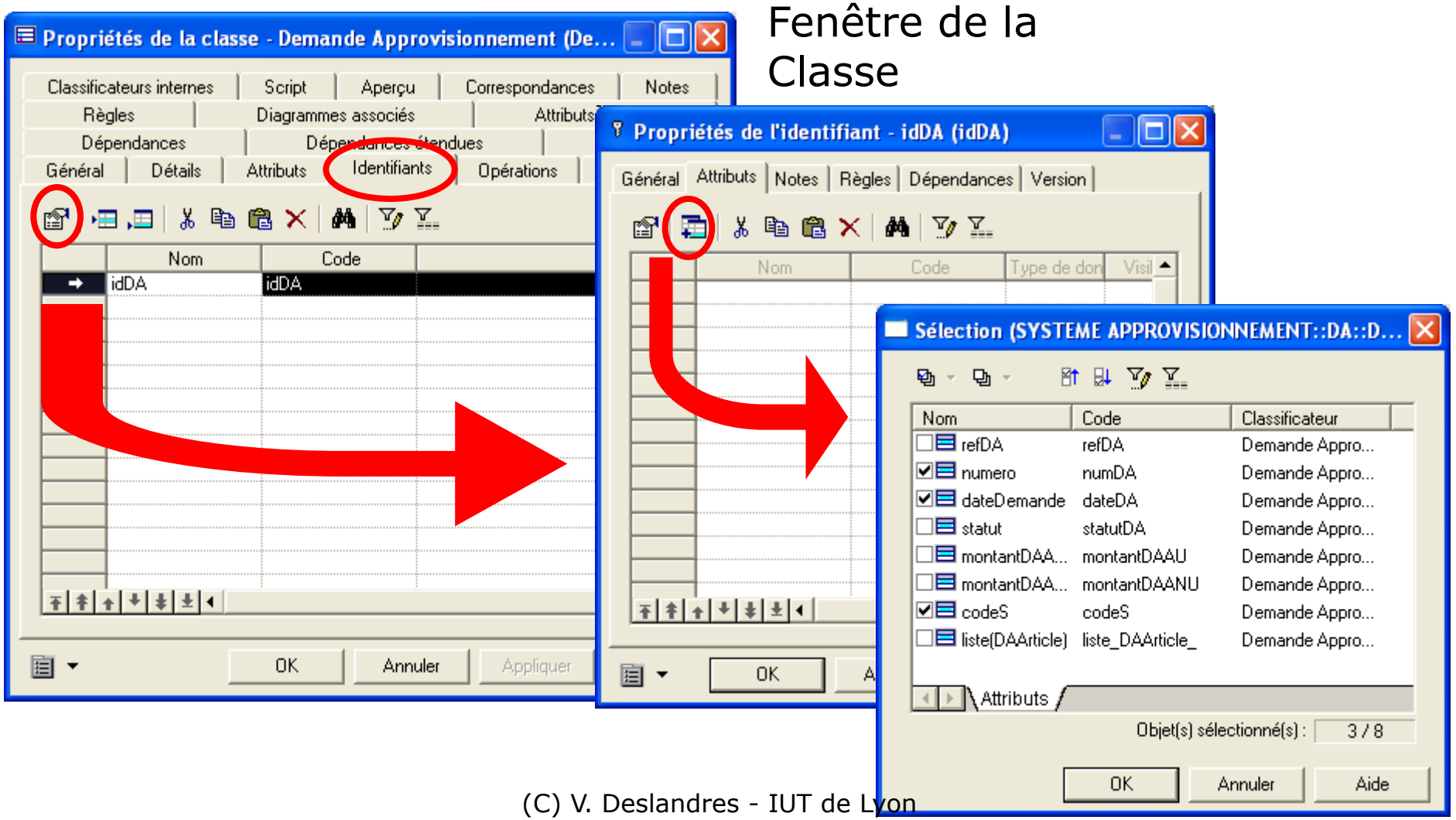
double-clic



Identifiants combinés

- Lorsqu'un identifiant est défini à partir d'une combinaison d'attributs de classe
- Créer un nouvel identifiant pour la classe (onglet Identifiant), puis cliquer sur l'outil `Afficher les propriétés`.
- Cliquer sur l'onglet `Attributs`
- Cliquer sur l'outil **`Ajouter des attributs`** et sélectionner les attributs qui composent l'identifiant

Création d'un identifiant combiné



Fenêtre de la Classe

Mentionner un attribut multiple

C'est un attribut qui peut avoir plusieurs valeurs.

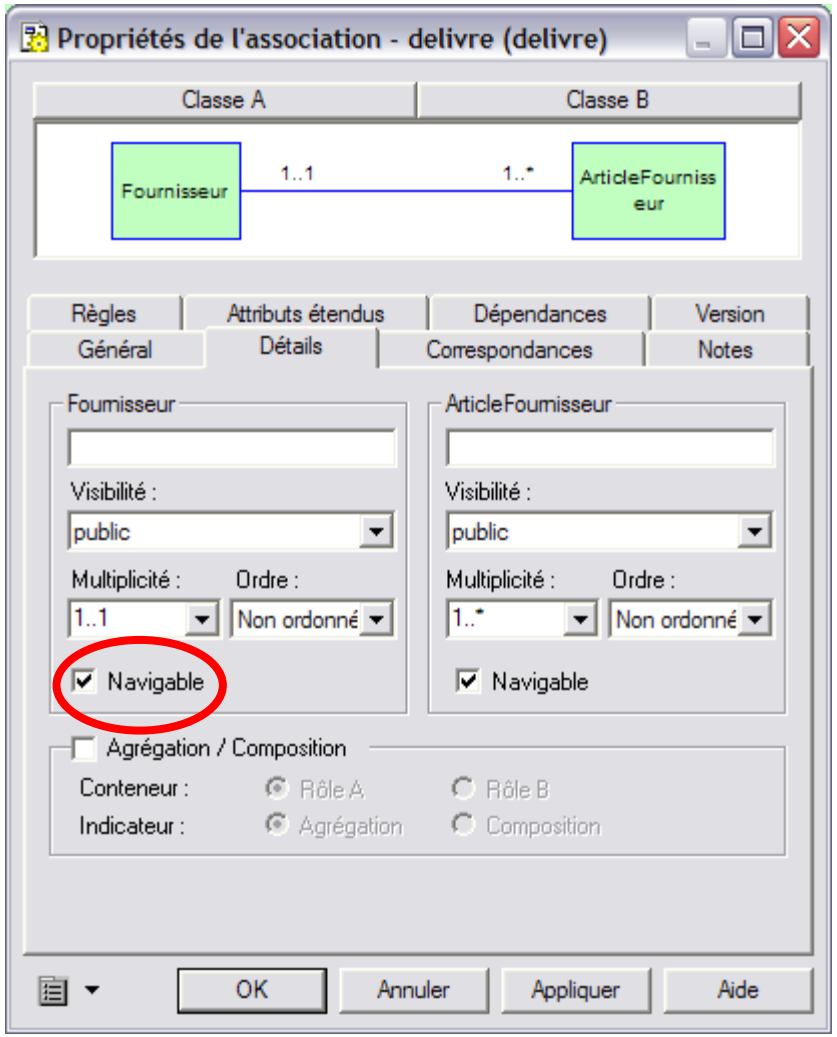
Dans PowerAMC, il faut aller dans la fenêtre de Propriétés de l'attribut, et choisir la « multiplicité » qui convient.

Ici il peut y avoir 2 adresses (par ex. une privée, une professionnelle)

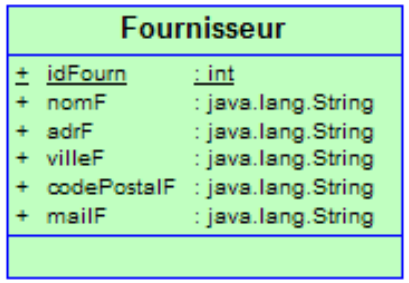
Navigabilité des associations

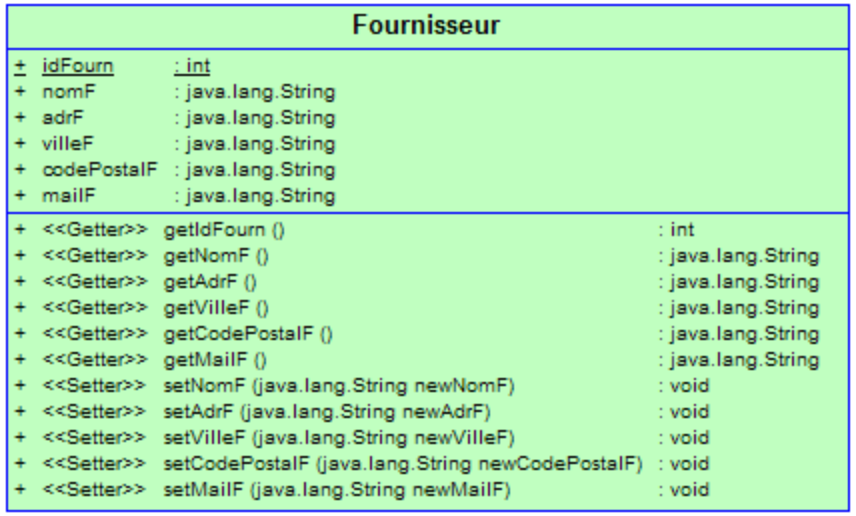
Dans **PowerAMC**, pour qu'une classe récupère les instances avec lesquelles elle est liée, il faut que les associations soient qualifiées de « navigable »

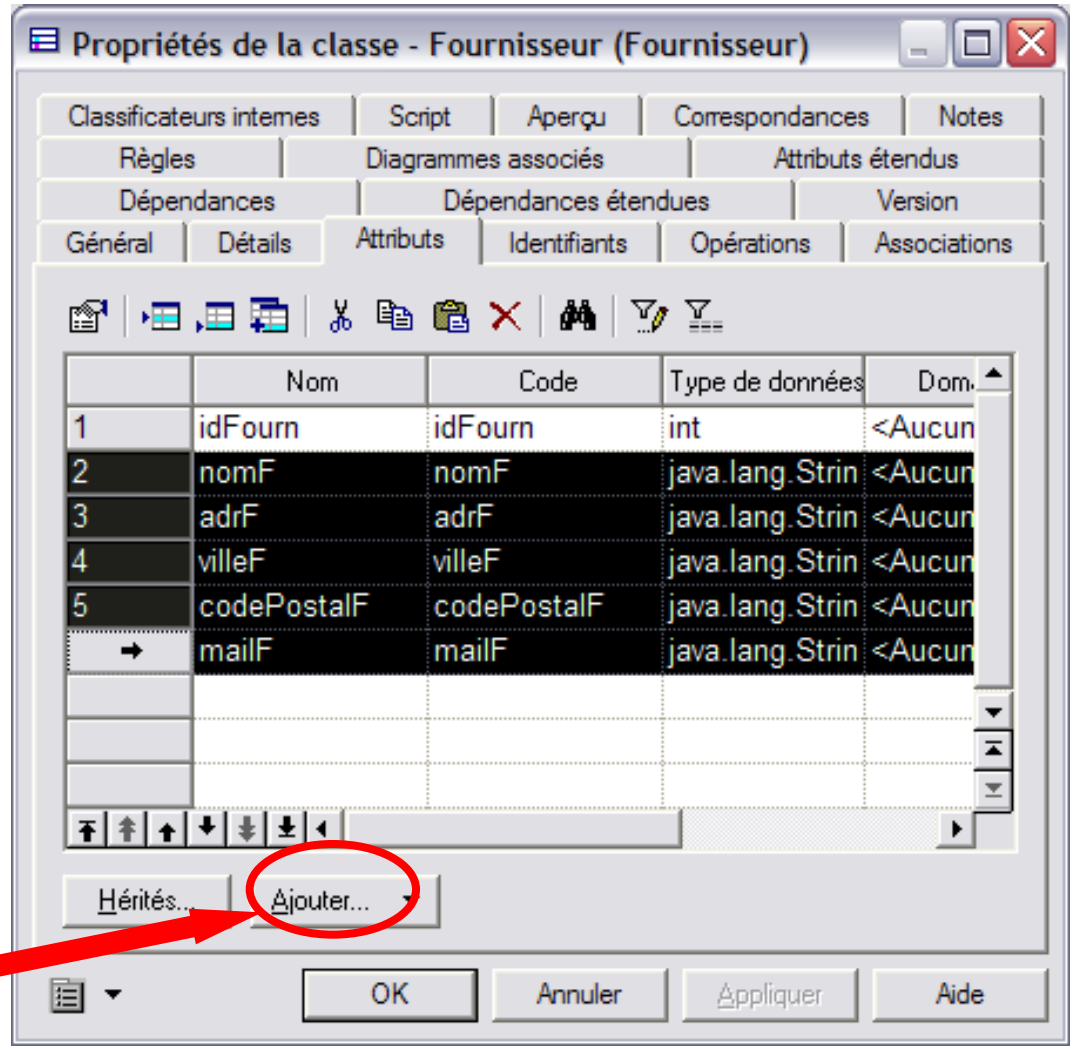
- Les **associations non navigables** sont des associations d'analyse, qui ne se traduisent pas nécessairement par un lien concret au niveau de l'implémentation.



Intégrer les accesseurs get/set

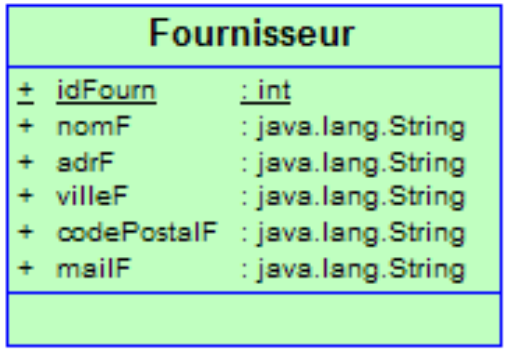


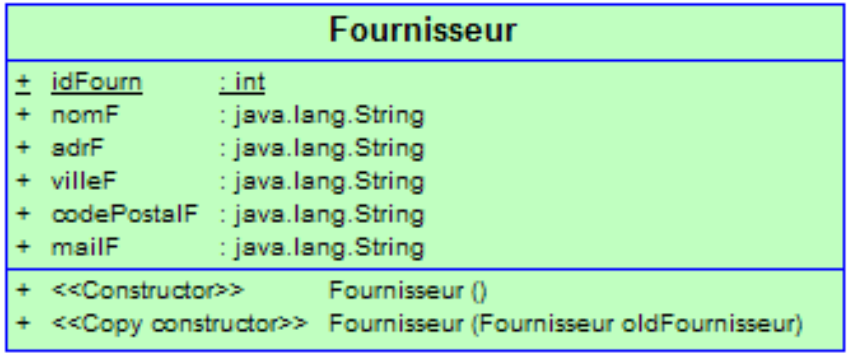




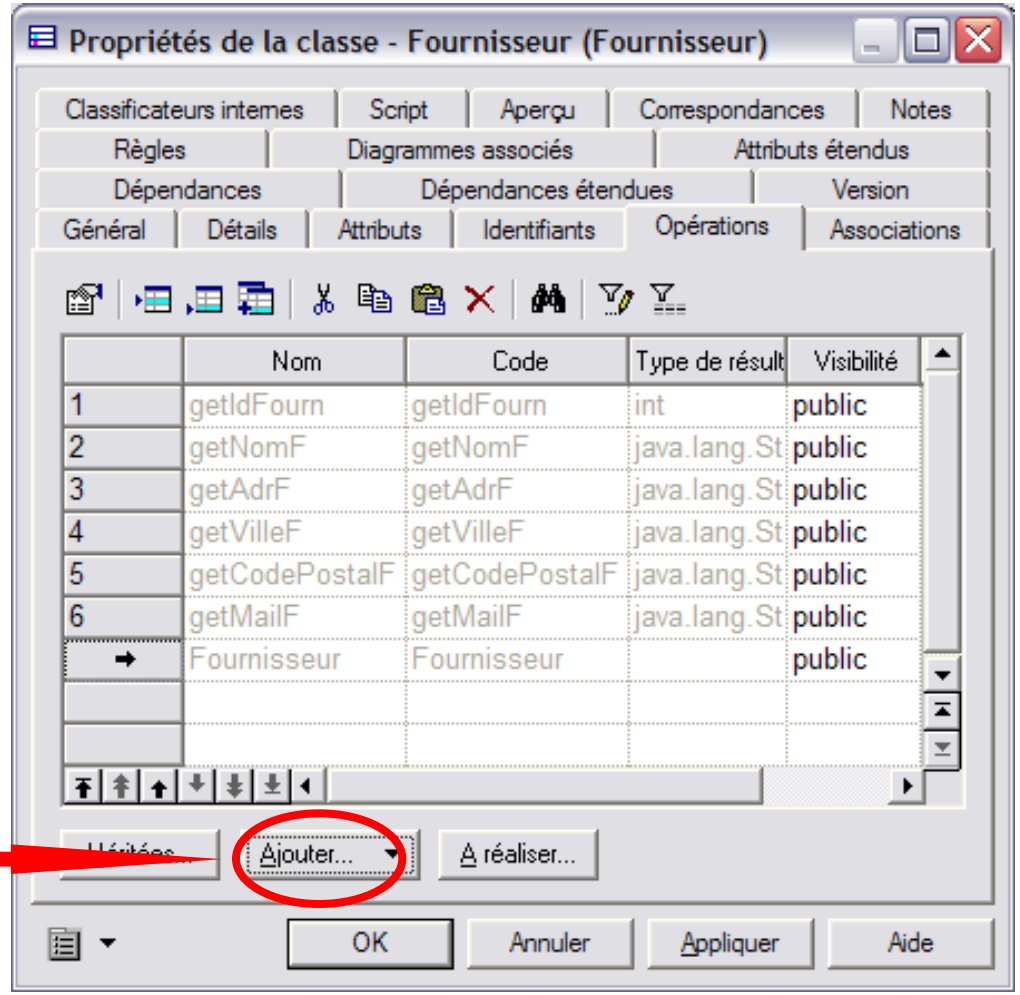
Ajouter le accesseurs get/set

Intégrer les constructeurs





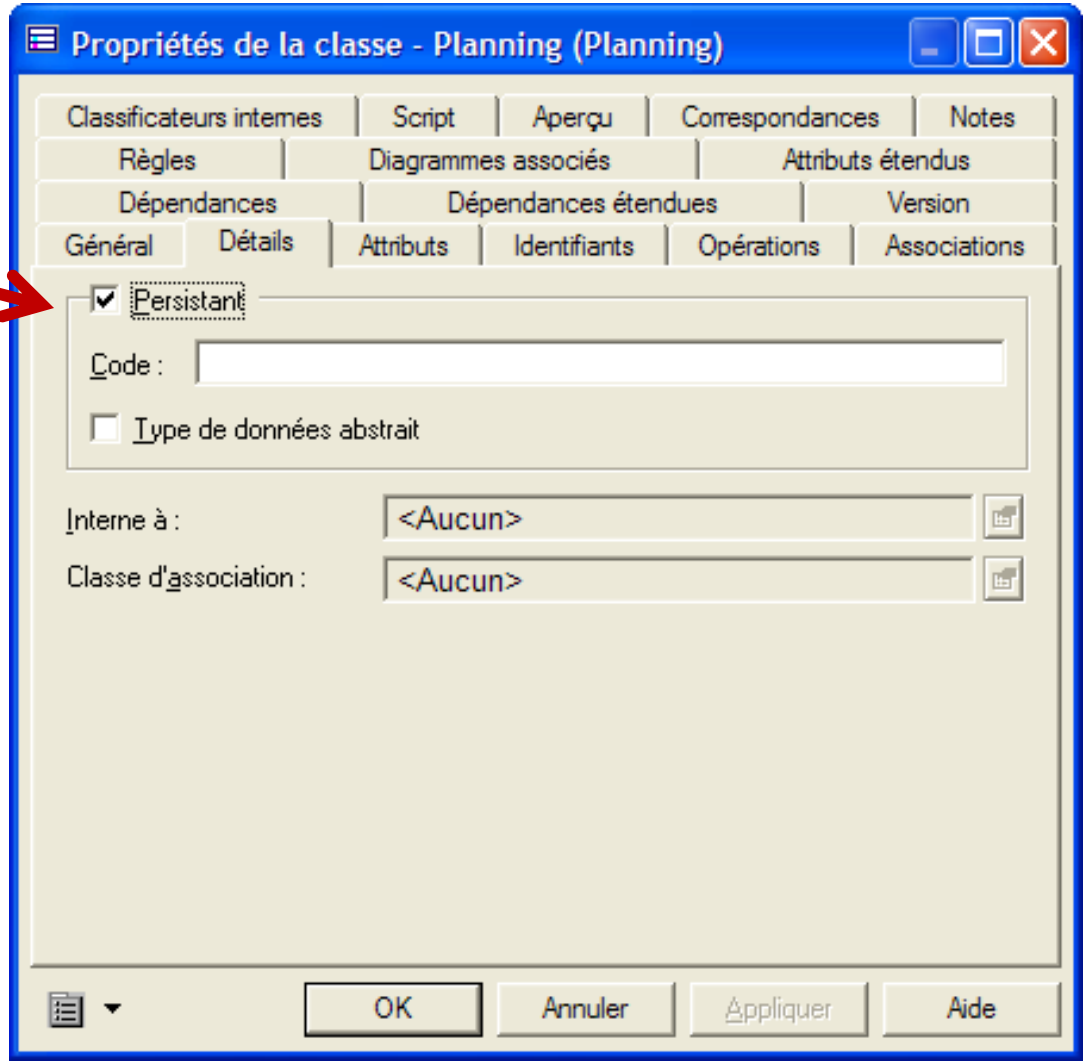
Ajouter le constructeur par défaut



Mentionner une classe Persistante dans PowerAMC : par défaut

Par défaut **toutes les classes** créées dans PowerAMC sont **persistantes**

Il faudra **décocher** la case des classes non persistantes.



1.2 – Ré organiser les classes (1/2)

- **Généraliser** les classes, **spécialiser** des classes existantes par ailleurs
  - Objectif : réutilisation
- **Supprimer** des éléments du DCL issue de l'analyse
  - Souvent en analyse, on prévoit **trop de classes**
    - Les transformer par ex. en attribut (classe *ArbitreChaise*, attribut *typeArbitre*) si ça s'avère possible (pas d'autres attributs nécessaires dans la classe)
  - On a aussi parfois des **redondances d'association**
    - Vérifier leur utilité ne fonction de la **fréquence des traitements**
  - Trop de méthodes ?
    - Non, c'est plutôt que certaines structures de données **simplifient les choix d'analyse**
    - Connaître les structures de données évoluées : les conteneurs Java notamment

Importance des structures de données

- Ex. : une classe d'analyse Contact avec un nom, un tél. (=ID) et une **méthode VeriferDoublon()**
  - Il suffit d'enregistrer la classe Contact dans un conteneur *set* de Java (automatiquement géré, ne tolère pas les doublons)
- Ex.: pour un **contrôle d'accès de salle**, on choisira une *HashTable* avec les numéros de salle (clef) et le code d'accès (valeur).

Ré organiser les classes (2/2)

- Transformer des classes en **interfaces**
  - Certaines classes ont finalement des comportements partagés par d'autres
  - On souhaite réutiliser des « comportements » existants dans d'autres applications
    - Sous la forme d'interfaces

Concevoir proprement

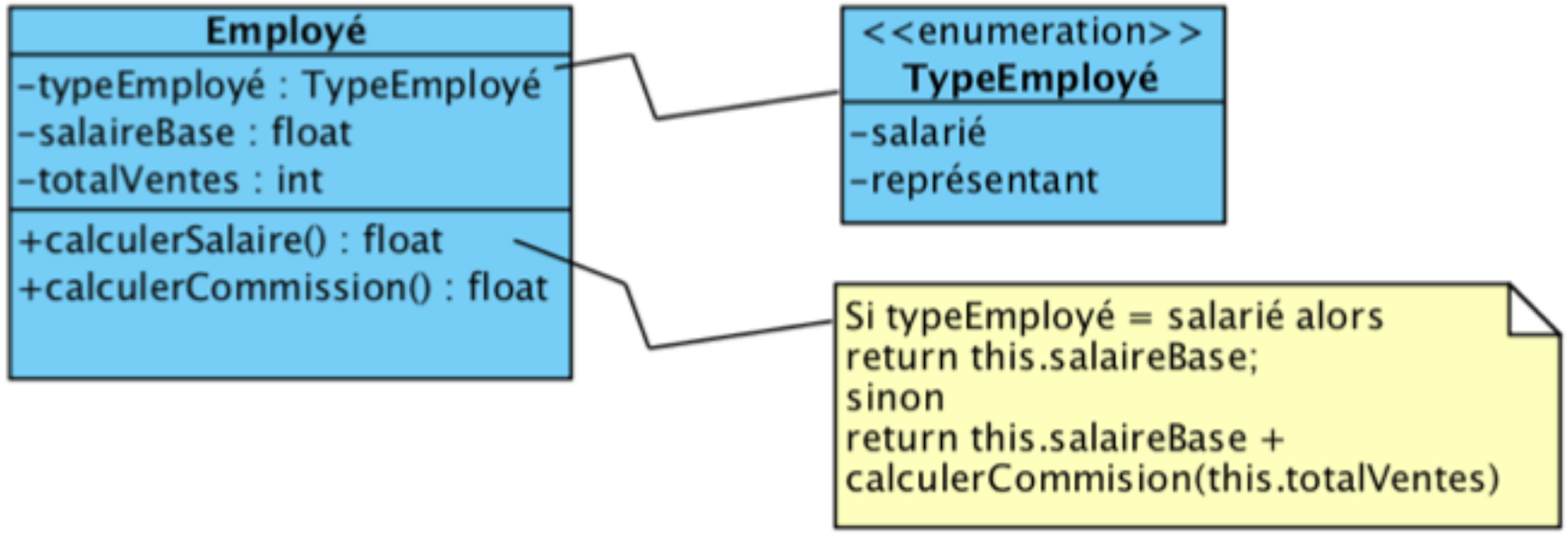
- Obj.: Faciliter la maintenance du logiciel

- Par ex., respecter le **principe d'Ouverture/Fermeture (OCP)** :

« Les entités logicielles (classes, packages, etc.) doivent être **ouvertes à l'extension** mais **fermées à la modification** »

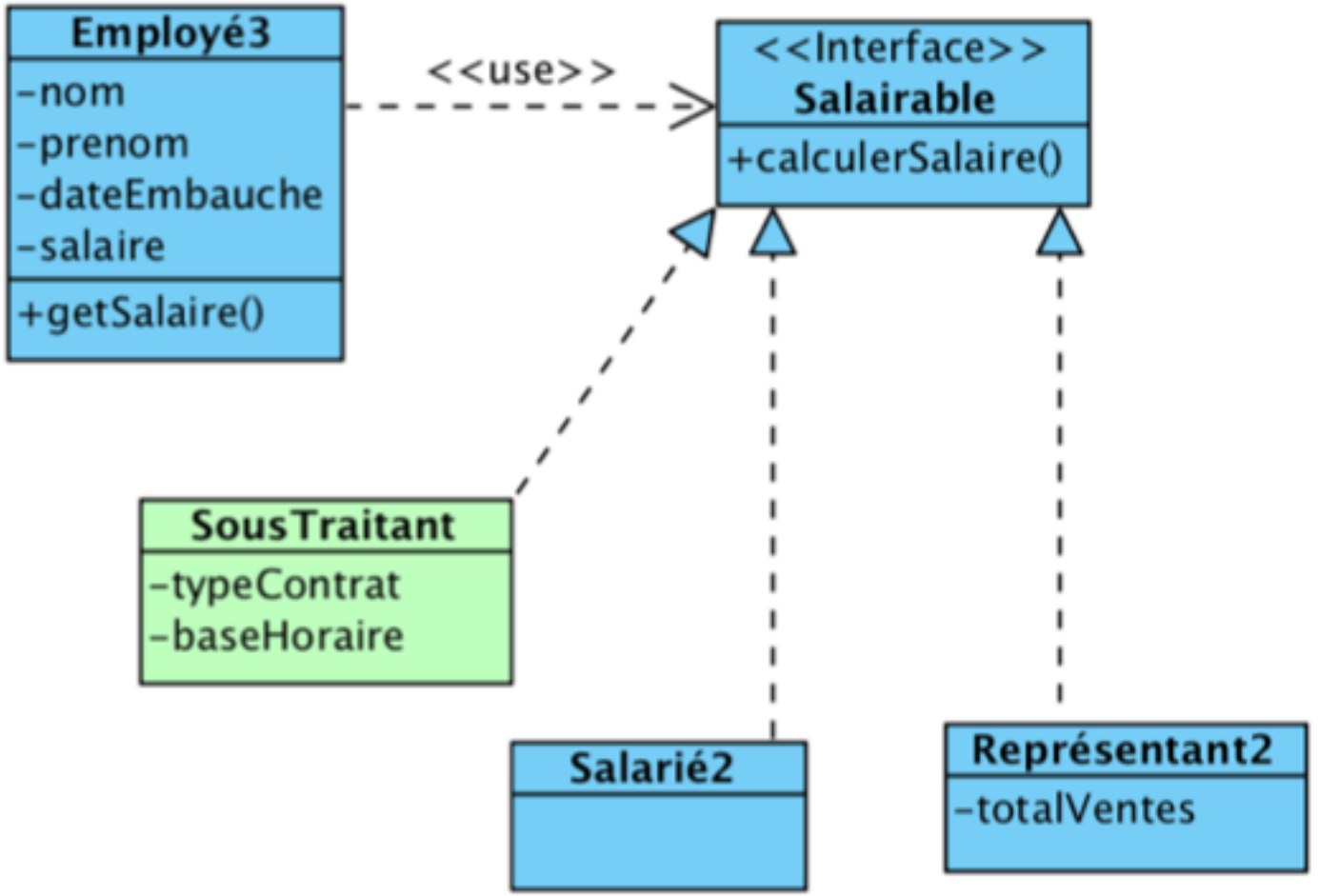
Un des principes SOLID *(Single Responsibility Principle, Open/Closed Principle, Liskov Substitution Principle, Interface Segregation Principle et Dependency Inversion Principle) de* Michael Feathers et Robert C. Martin

Exemple de non respect d'OCP



Si on doit considérer un nouveau type d'employé ?

Avec le principe OCP



Concevoir proprement

- Il existe beaucoup d'autres méthodes d'optimisation de codes que vous apprendrez plus tard
  - notamment, les **Design Patterns** (Strategy, Composite, TemplateMethod, Factory Method, etc.)

- Avec un AGL, une fois le DCL de Conception fini, on peut **générer du code** (cours suivant)

Modèle de Conception…

POUR ALLER PLUS LOIN

1- Générer un nouveau MOO avec PAMC
2- Migrer les associations dans les classes

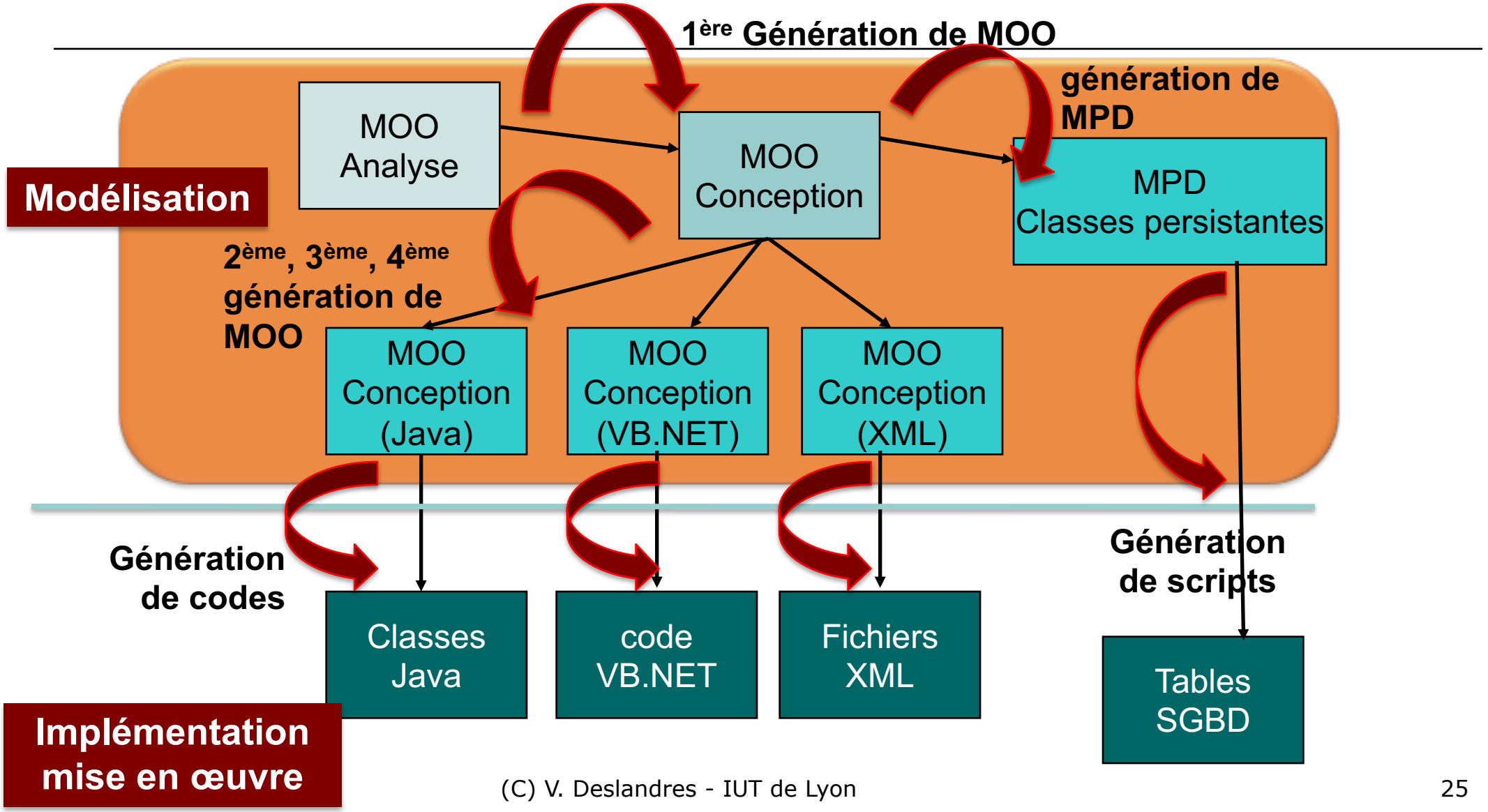
1- Génération d'un Modèle Orienté Objet (MOO) de Conception avec PAMC

- On génère un MOO depuis le MOO d'analyse:
  - *Outils / Génération d'un modèle Orienté objet*, et choisir le langage cible (ici Java)
  - *Dans Paramètres du modèle,* on peut spécifier : le type des données par défaut (int), le conteneur par défaut, afficher les classes comme type de données, etc.
- Puis on casse la dépendance entre les 2
  - Sinon toute modification faite en Conception sera répercutée sur le diagramme source d'analyse
  - Dans les *Options de génération d'un MOO, Onglet Détails*, décocher *Enregistrer les dépendances de génération*

Génération de plusieurs MOO

- Si on veut implémenter le modèle d'analyse avec différents langages (.NET, XML, JAVA) :
  - On va **générer** un MOO par type d'implémentation
  - Puis faire la génération de code depuis chaque MOO obtenu

n générations de modèles

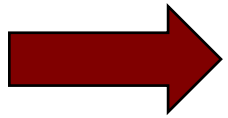


Sur la génération de modèles à partir du MOO…

- On peut donc générer à partir d'un MOO :
  - des MPD, d'autres MOO, mais aussi des MCD !
  - Intérêt du MCD : fournir aux responsables de BD qui ont une culture Entité / Association, un modèle connu.

Deux possibilités depuis le MOO :

- Créer un **nouveau modèle** (option par défaut) contenant les objets convertis
- MàJ un **modèle existant**, issu d'une ancienne génération



MàJ un modèle généré précédemment

- En fait l'AGL va créer un modèle par défaut contenant les objets convertis depuis le MOO
  - puis les fusionner ensuite dans un modèle existant.

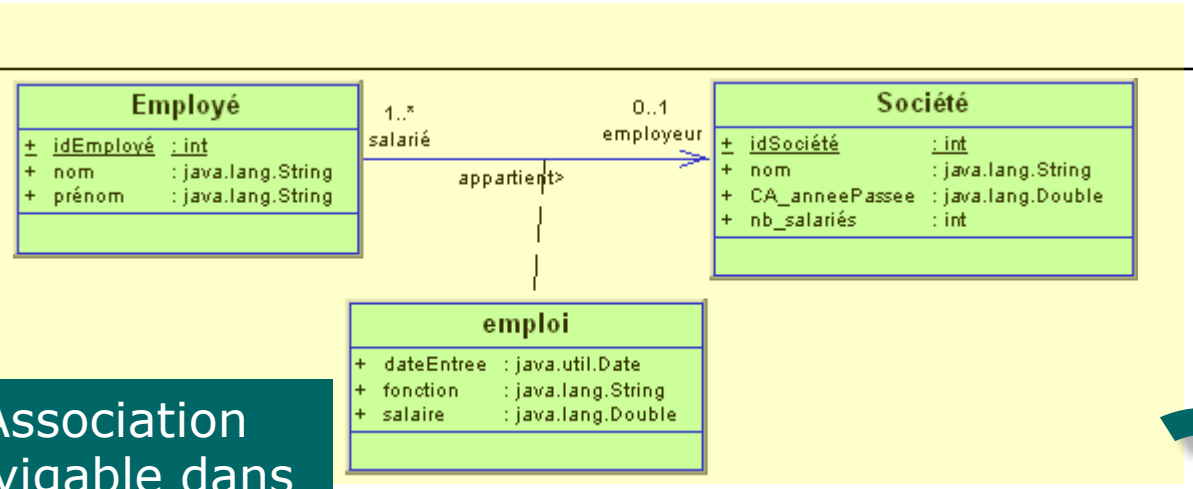
- Des options sont disponibles pour **paramétrer la fusion** :
  - vous pouvez ainsi choisir de mettre à jour, supprimer ou ajouter des objets dans le modèle existant (modèle à fusionner, dans le volet droit) en fonction des modifications apportées dans le modèle par défaut (dans le volet gauche)

2- Transformer les associations en attributs

- Il est possible de migrer des rôles d'association et de créer des attributs **avant** la génération
  - Clic droit sur l'association : Migrer
    - Soit les rôles navigables, soit les 2 rôles

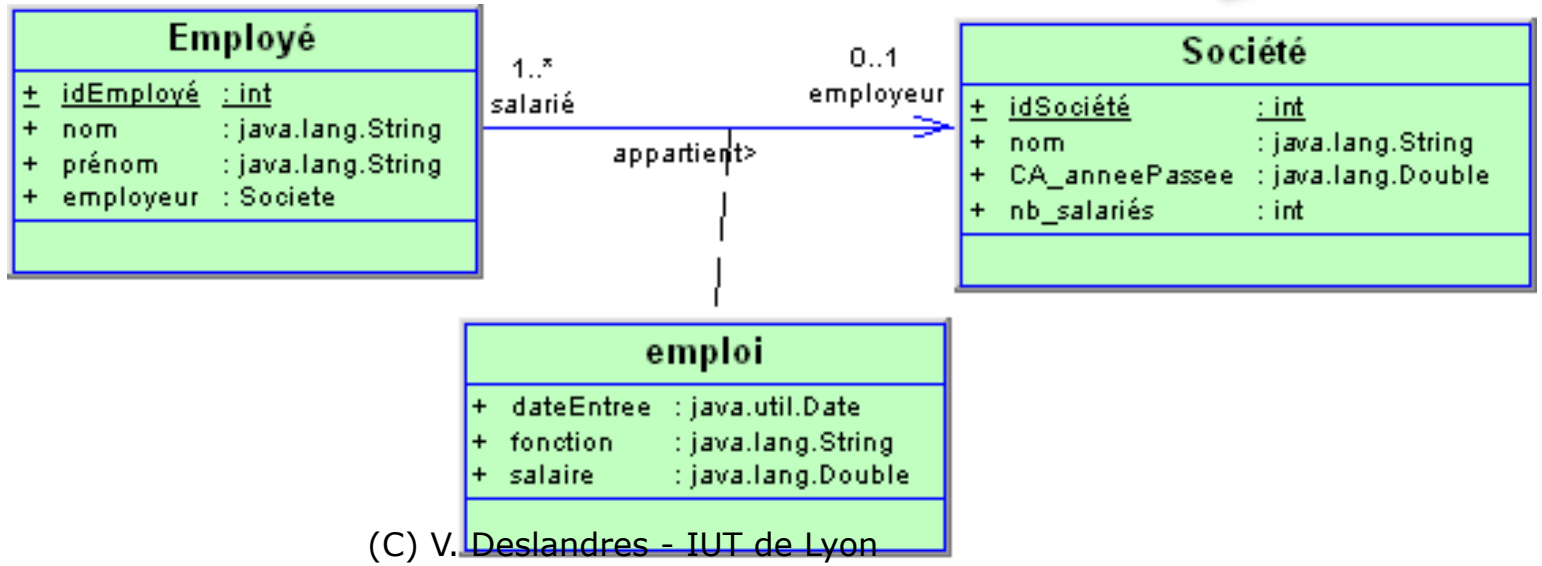
- Cette fonctionnalité permet entre autres de **personnaliser les types de données** et de modifier l'ordre des attributs dans les listes d'attributs
  - ce qui peut se révéler particulièrement **utile pour XML**

Exemple de migration de rôles (1/2)

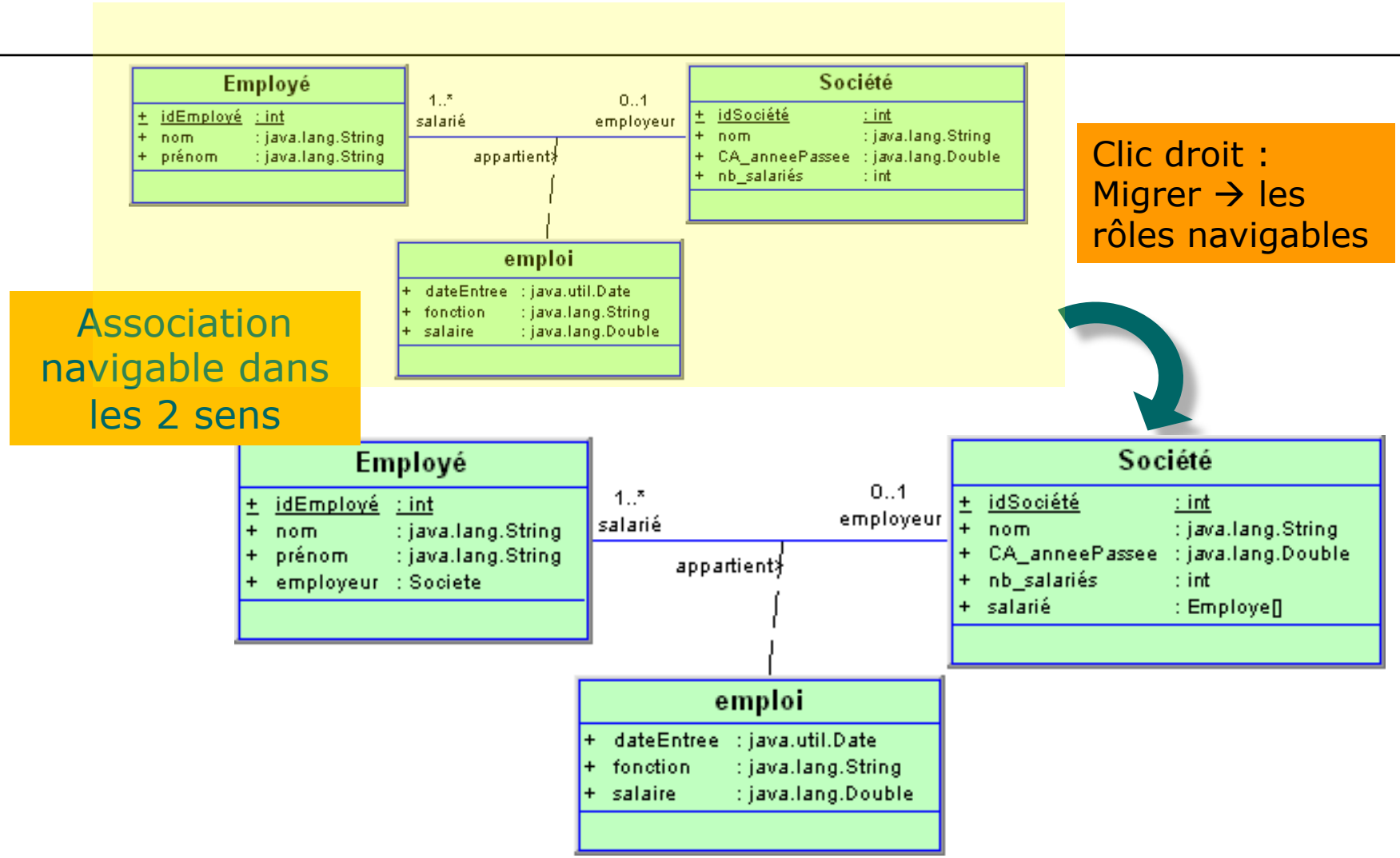


Association navigable dans un sens

Clic droit : Migrer → les rôles navigables



Exemple de migration de rôles 1/2)



Règles de migration de rôles

- Quelle que soit la navigabilité, la migration crée un attribut et définit ses propriétés comme suit :
  - Nom et code de l'attribut : le rôle de l'association s'il est déjà défini, dans le cas contraire, utilise le nom de l'association
  - Type de données : le code du classificateur lié par l'association
  - Multiplicité, visibilité : celles du rôle (classificateur)

Migration de rôles : synchronisation

- Toute modification de l'association (chgt de nom, de multiplicité, destruction) sera répercutée automatiquement sous l'AGL au niveau de l'attribut

- Par contre, toute modification **manuelle** de l'attribut migré cassera la dépendance.