



# 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, il faut explicitement **casser la dépendance** entre les 2 modèles
  - Dupliquer le projet sous l'OS (en dehors de PAMC), le renommer
    - **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 **éléments de conception**
  - Identifiants, accesseurs, etc.
  - On précise les attributs / retours **multiples**
  - On vérifie la **navigabilité des associations**
  - On mentionne les classes persistantes
- Aussi :
  - On optimise les classes

# 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

Propriétés de la classe - employé (employe)

Classificateurs internes | Script | Aperçu | Correspondances | Notes  
Règles | Diagrammes associés | Attributs étendus  
Dépendances | Dépendances étendues | Version  
Général | Détails | Attributs | Identifiants | Opérations | Associations

double-clic

Nom	Code	Type de données	Domain
idEmployé	= idEmploye	int	<Aucun>
nom	nom	java.lang.String	<Aucun>
pre	prenom	java.lang.String	<Aucun>

Hérités... Ajouter...

OK Annuler Appliquer Aide

Propriétés de l'attribut - idEmployé (idEmploye)

Notes | Règles | Attributs étendus | Dépendances | Version  
Général | Détails | Contrôles standard | Contrôles supplémentaires

Valeur initiale :  
Modifiable : Modifiable  
Domaine : <Aucun>  
 Identifiant primaire  
Migré de : <Aucun>

Persistant  
Code :  
Type de données : <UNDEF>  
Longueur : Précision :  
Génération de classe : <Indéfini>

OK Annuler Appliquer Aide

# 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

The image illustrates the steps to create a combined identifier in a software development environment. It consists of three overlapping windows:

- Propriétés de la classe - Demande Approvisionnement (De...)**: The 'Identifiants' tab is selected. A red circle highlights the 'Identifiants' tab, and another red circle highlights the 'idDA' entry in the table below. A large red arrow points from this entry to the next window.
- Propriétés de l'identifiant - idDA (idDA)**: The 'Général' tab is selected. A red circle highlights the 'idDA' entry in the table. A red arrow points from this entry to the third window.
- Sélection (SYSTEME APPROVISIONNEMENT::DA::D...)**: A selection dialog showing a table of attributes. Three attributes are selected: 'numero', 'dateDemande', and 'codeS'. A red arrow points from the 'idDA' entry in the second window to this dialog.

Nom	Code	Classificateur
<input type="checkbox"/> refDA	refDA	Demande Appro...
<input checked="" type="checkbox"/> numero	numDA	Demande Appro...
<input checked="" type="checkbox"/> dateDemande	dateDA	Demande Appro...
<input type="checkbox"/> statut	statutDA	Demande Appro...
<input type="checkbox"/> montantDAA...	montantDAAU	Demande Appro...
<input type="checkbox"/> montantDAA...	montantDAANU	Demande Appro...
<input checked="" type="checkbox"/> codeS	codeS	Demande Appro...
<input type="checkbox"/> liste(DAArticle)	liste_DAArticle_	Demande Appro...

Objet(s) sélectionné(s) : 3 / 8

# 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)

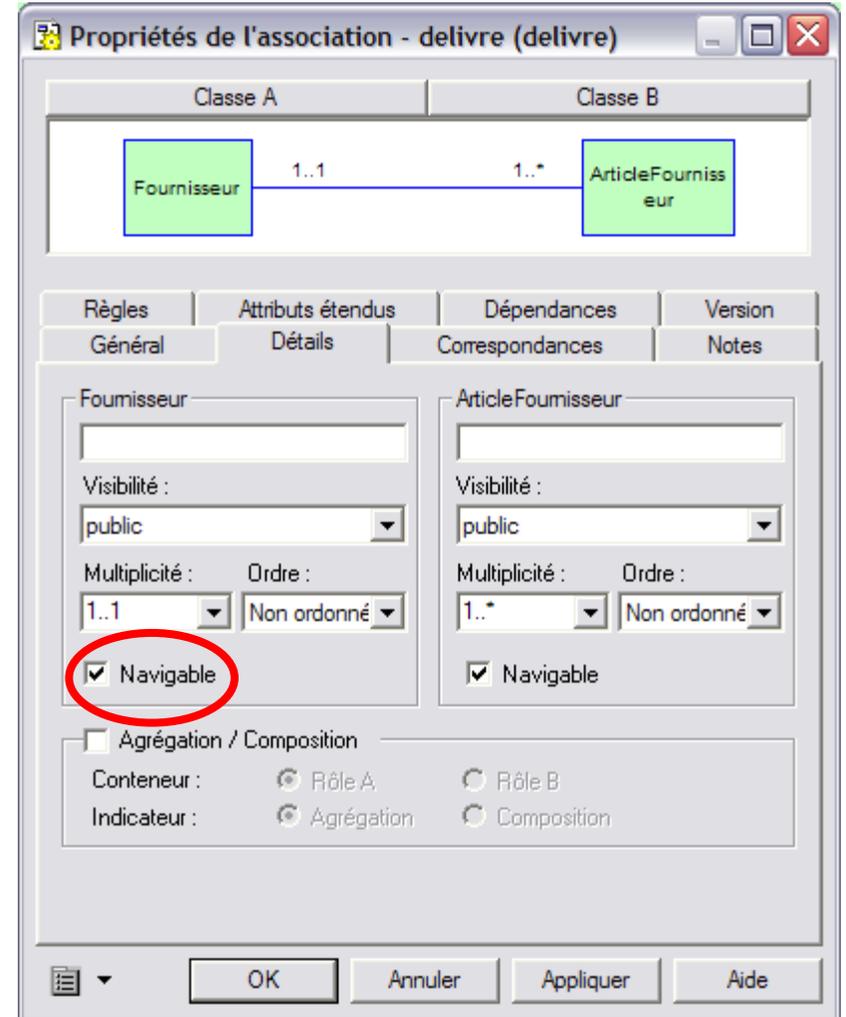


The screenshot shows the 'Propriétés de l'attribut - adresse (ADRESSE)' dialog box. The 'Général' tab is selected. The 'Parent' field is set to 'Client'. The 'Nom' field is 'adresse' and the 'Code' field is 'ADRESSE'. The 'Type de données' is 'ADRESSE'. The 'Multiplicité' field is set to '0..2', which is highlighted by a red arrow. The 'Visibilité' is set to 'Public'. There are checkboxes for 'Dérivé' and 'Statique', both of which are unchecked. The dialog box has buttons for 'OK', 'Annuler', 'Appliquer', and 'Aide'.

# 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

Fournisseur	
+ idFourn	: int
+ nomF	: java.lang.String
+ adrF	: java.lang.String
+ villeF	: java.lang.String
+ codePostalF	: java.lang.String
+ mailF	: java.lang.String

Fournisseur	
+ idFourn	: int
+ nomF	: java.lang.String
+ adrF	: java.lang.String
+ villeF	: java.lang.String
+ codePostalF	: java.lang.String
+ mailF	: java.lang.String
+ <<Getter>> getIdFourn ()	: int
+ <<Getter>> getNomF ()	: java.lang.String
+ <<Getter>> getAdrF ()	: java.lang.String
+ <<Getter>> getVilleF ()	: java.lang.String
+ <<Getter>> getCodePostalF ()	: java.lang.String
+ <<Getter>> getMailF ()	: java.lang.String
+ <<Setter>> setNomF (java.lang.String newNomF)	: void
+ <<Setter>> setAdrF (java.lang.String newAdrF)	: void
+ <<Setter>> setVilleF (java.lang.String newVilleF)	: void
+ <<Setter>> setCodePostalF (java.lang.String newCodePostalF)	: void
+ <<Setter>> setMailF (java.lang.String newMailF)	: void

Ajouter le accesseurs  
get/set

Propriétés de la classe - Fournisseur (Fournisseur)

Classificateurs internes | Script | Aperçu | Correspondances | Notes

Règles | Diagrammes associés | Attributs étendus

Dépendances | Dépendances étendues | Version

Général | Détails | Attributs | Identifiants | Opérations | Associations

	Nom	Code	Type de données	Dom.
1	idFourn	idFourn	int	<Aucun
2	nomF	nomF	java.lang.Strin	<Aucun
3	adrF	adrF	java.lang.Strin	<Aucun
4	villeF	villeF	java.lang.Strin	<Aucun
5	codePostalF	codePostalF	java.lang.Strin	<Aucun
→	mailF	mailF	java.lang.Strin	<Aucun

Hérités... **Ajouter...** OK Annuler Appliquer Aide

# Intégrer les constructeurs

Fournisseur	
+ <u>idFourn</u>	: int
+ nomF	: java.lang.String
+ adrF	: java.lang.String
+ villeF	: java.lang.String
+ codePostalF	: java.lang.String
+ mailF	: java.lang.String

Fournisseur	
+ <u>idFourn</u>	: int
+ nomF	: java.lang.String
+ adrF	: java.lang.String
+ villeF	: java.lang.String
+ codePostalF	: java.lang.String
+ mailF	: java.lang.String
+ <<Constructor>>	Fournisseur ()
+ <<Copy constructor>>	Fournisseur (Fournisseur oldFournisseur)

Ajouter le constructeur par défaut

Propriétés de la classe - Fournisseur (Fournisseur)

Classificateurs internes | Script | Aperçu | Correspondances | Notes

Règles | Diagrammes associés | Attributs étendus

Dépendances | Dépendances étendues | Version

Général | Détails | Attributs | Identifiants | Opérations | Associations

	Nom	Code	Type de résultat	Visibilité
1	getIdFourn	getIdFourn	int	public
2	getNomF	getNomF	java.lang.St	public
3	getAdrF	getAdrF	java.lang.St	public
4	getVilleF	getVilleF	java.lang.St	public
5	getCodePostalF	getCodePostalF	java.lang.St	public
6	getMailF	getMailF	java.lang.St	public
→	Fournisseur	Fournisseur		public

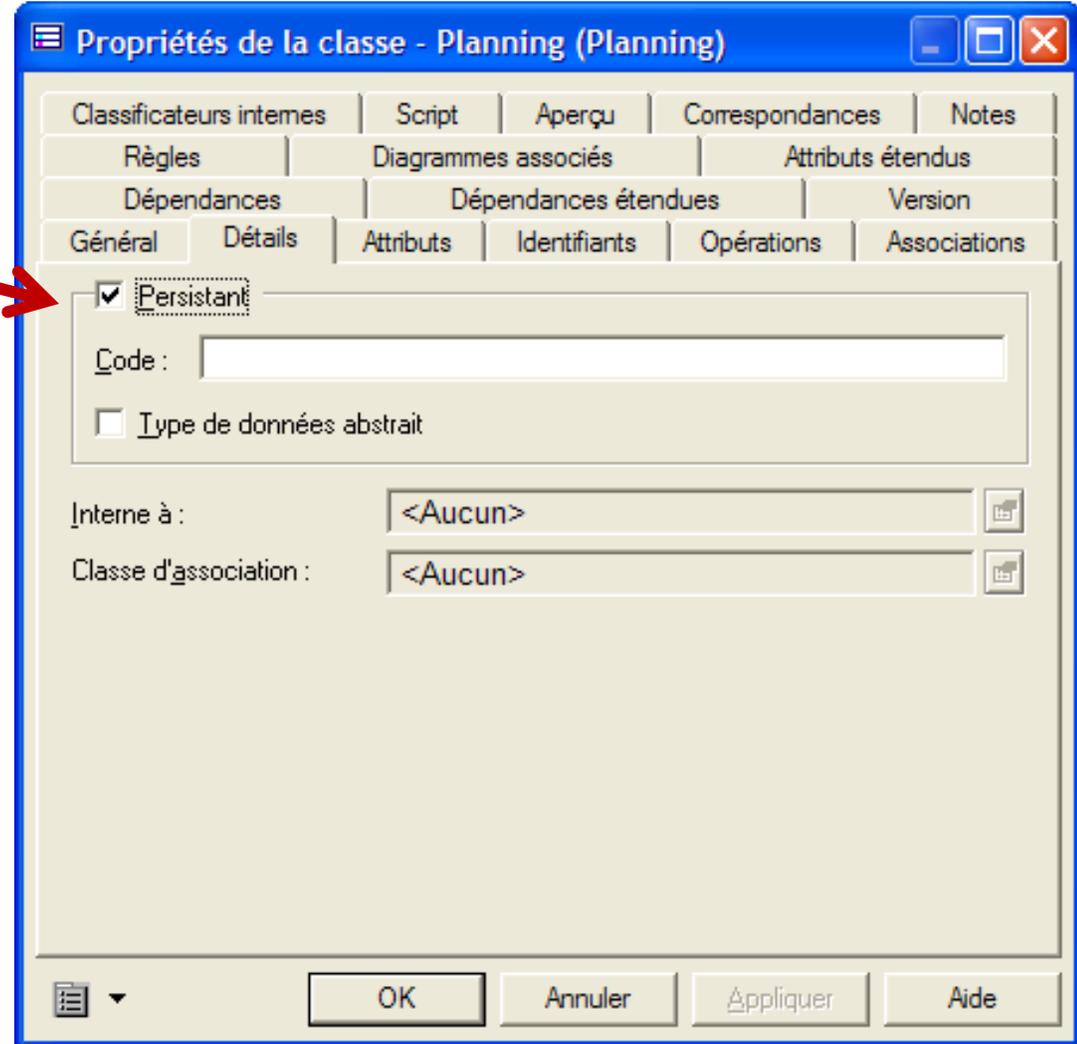
Utilitaires | **Ajouter...** | À réaliser...

OK | Annuler | Appliquer | Aide

# Mentionner une classe Persistante dans PowerAMC

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 d'analyse
  - Souvent en analyse, on prévoit **trop de classes**
    - Les transformer par ex. en attribut si c'est possible (pas d'autres attributs ou relations nécessaires avec la classe). Ex .: des sous-classes *VIP*, attribut *typeVIP* suffit ?
  - On a aussi parfois des **redondances d'association**
    - Vérifier leur utilité en fonction de la **fréquence des traitements**
  - Supprimer des méthodes ?
    - Certaines structures de données **simplifient les choix d'analyse** →

# Importance des structures de données

---

Certaines structures de données réalisent les méthodes initialement envisagées

- Ex. : une classe d'analyse Contact avec un nom, un tél. (=ID) et une **méthode VerifierDoublon()**
  - 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

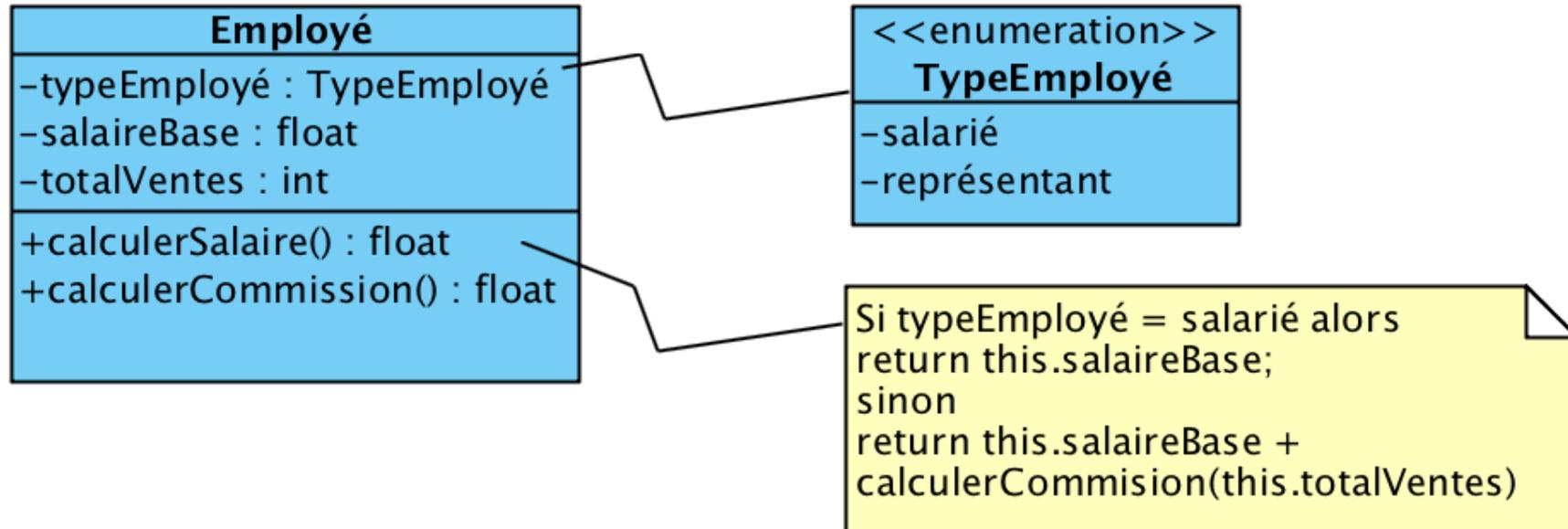
# Concevoir proprement

---

- Objectif : 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** »

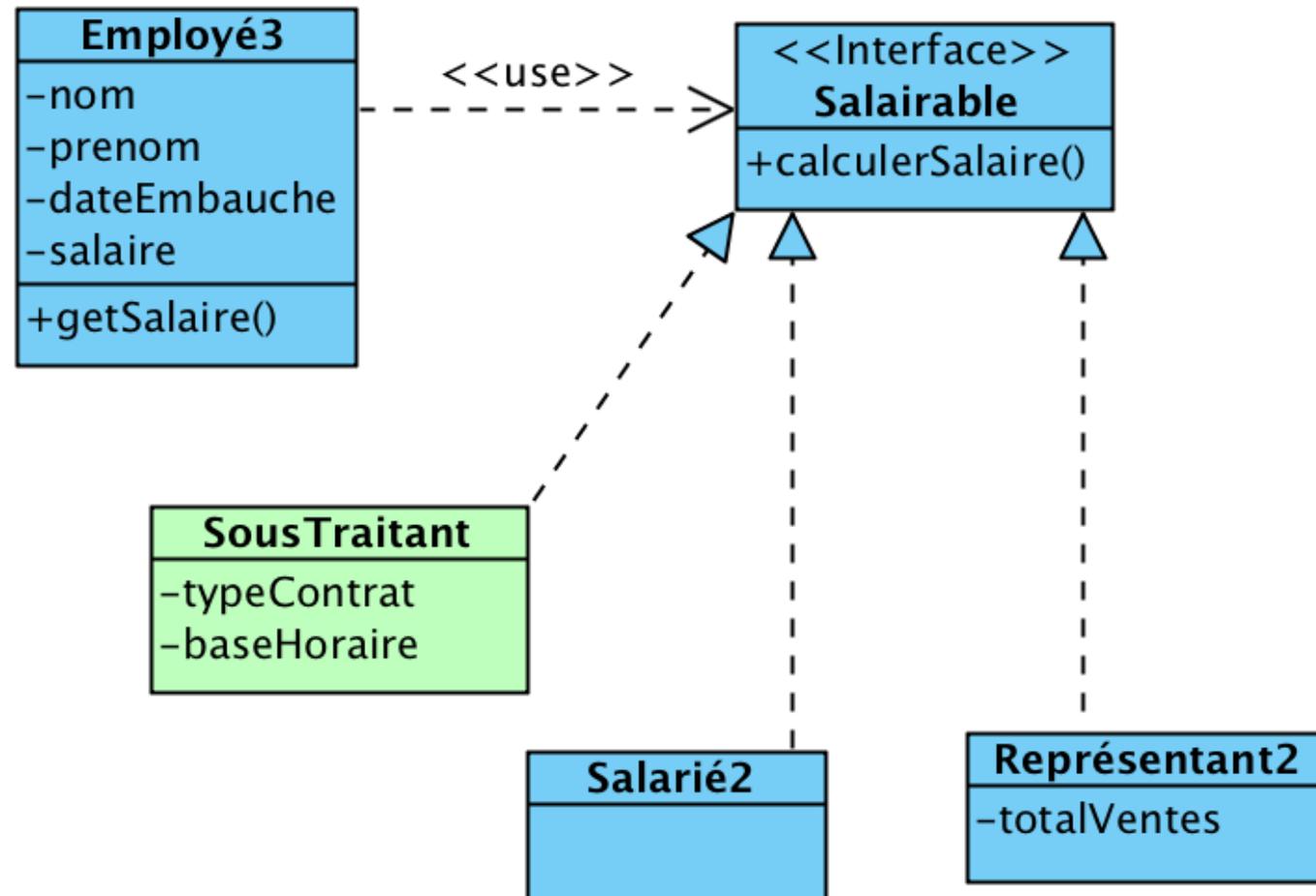
Un des principes SOLID (*Single Responsibility Principle, Open/Closed Principle, Liskov Substitution Principle, Interface Segregation Principle et Dependency Inversion Principle*)

# 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 la conception 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)