# ACSI 2

**Public : DIC1 (Eq. L3)**

**Etablissement : EPT**

**Département : Génie Informatique et Télécommunications**

**Année académique : 2019-2020**

**Responsable du Cours : Dr Abdoulaye GUISSE**

# A Propos de ce cours

- **ACSI 2 ?**
  - Analyse et Conception de Systèmes d'Informations
  - Partie 2 (après la partie 1 sur l'initiation avec MERISE)
  - UML : Concepts de base & (#) Diagrammes & Un exemple d'approche UML
  - *Objectif* : *Aptitude à modéliser les aspects fonctionnels et structurels d'un logiciel avec UML, de bout en bout.*
  - Durée : 28H de (CM+TD+TP) et 12H de TPE
  - Les Diapositives font office de supports de cours

- Un **Elément Constitutif (E.C.) de l'U.E. :**
  - « Génie Logiciel et Bases de Données » de la DIC1 en GIT

- **Prérequis :**
  - ACSI 1 : Méthode MERISE et Initiation aux BD
  - Bases en Algorithmique et Programmation (1)

- **Evaluation : 60% Examen écrit + 40% Projet**

# Plan du cours

- **Introduction**
  - Cycle de vie d'un logiciel
  - Cycle de développement (**Analyse-Conception-Développement**)
  - Différents cycles de développement

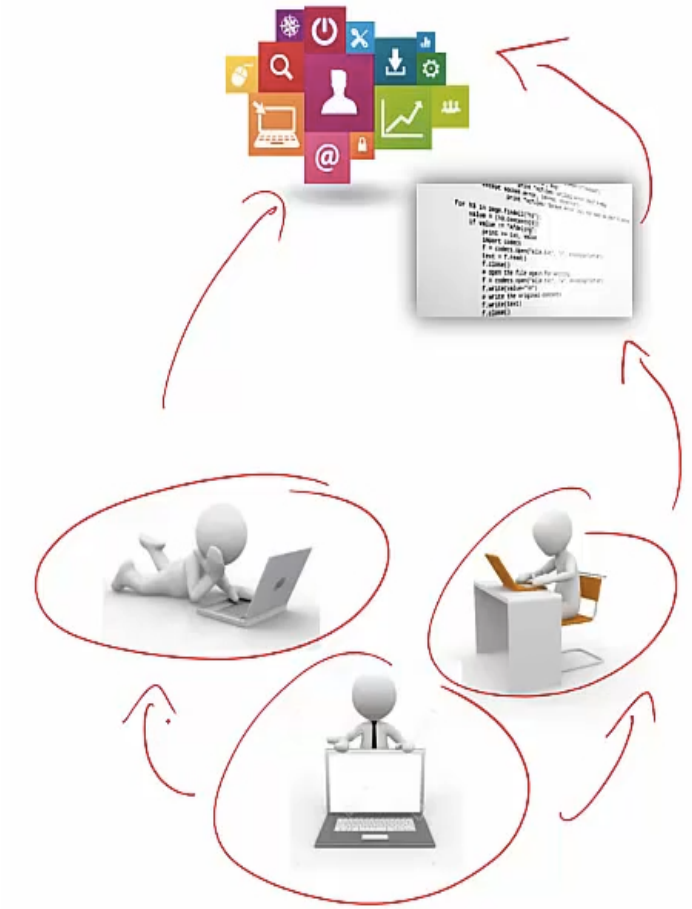
- **Cahier de spécifications avec UML**
  - **Pourquoi** UML?
  - **Conception** structurelle (Classes et Objets)
    - **Vers une base de données relationnelle**
  - **C**onception architecturale (Composants, Déploiement)
  - **C**onception fonctionnelle de comportements (Cas d'Utilisation, activités)
  - **C**onception fonctionnelle d'interactions (séquences, états)
  - **Méthodologie minimale : un exemple d'enchaînement**

- **En pratique**
  - **Passage au code source** Java à partir du UML
  - **OCL** : Restriction d'un modèle
  - **Diagrammes** UML 2.x

# INTRODUCTION

# Cycle de vie d'un logiciel

- Un **logiciel** est :
  - une  Application, un Site Web,
  - un Jeu, un Microcontrôleur, etc.
  - bref un système d'informations (SI)



- **Objectif** :
  - Produire un logiciel
  - => Code Source, Exécutable

- **Acteurs autour de cette production**
  - L'Utilisateur (Client)
  - Le Développeur : Maître d'Œuvre
  - Le propriétaire (l'Entreprise) : Maître d'Ouvrage

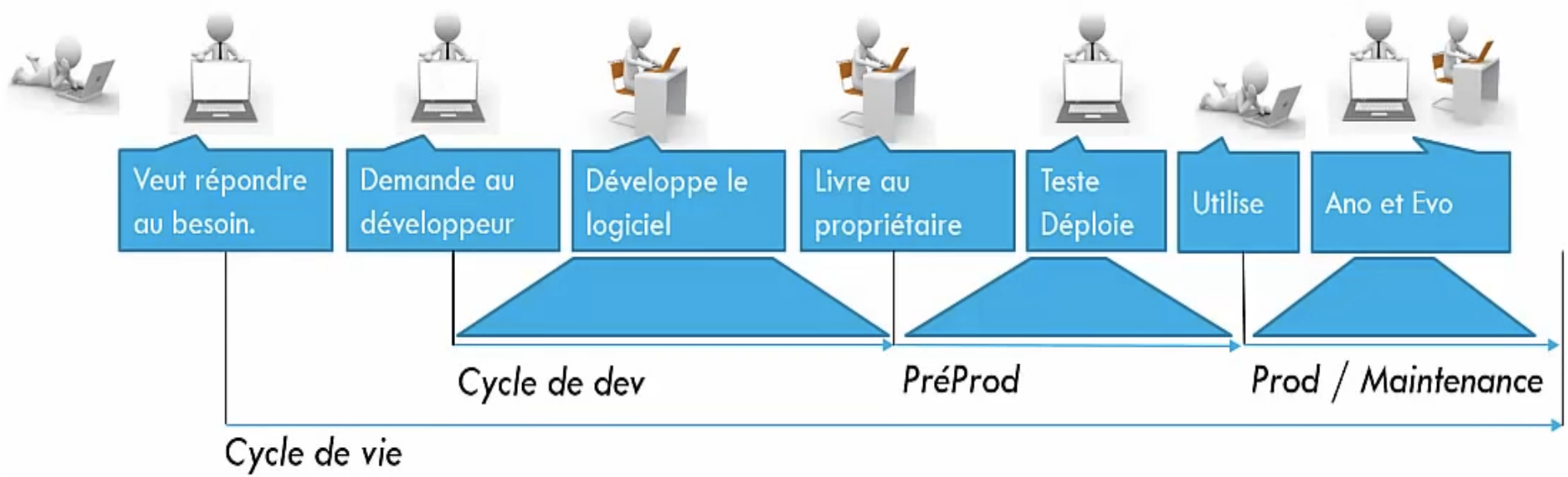
# Cycle de vie d'un logiciel

- **Phase 1 : Développement**
  - L'utilisateur a besoin d'un logiciel
  - Le propriétaire demande au développeur de réaliser le logiciel
  - Le développeur réalise un logiciel fidèle au besoin de l'utilisateur

- **Phase 2 : Exploitation et Maintenance**
  - Le propriétaire déploie le logiciel pour permettre son utilisation
  - L'Utilisateur utilise le logiciel, fait des feedbacks sur les anomalies à corrigées ou des manquements à rattraper
  - Le propriétaire demande au développeur de prendre en compte ces feedbacks
  - Le développeur met à jour et propose une nouvelle version, plus évoluée.

- **Phase 3 : Fin**
  - Plus rien à corriger ou à faire évoluer

# Cycle de vie d'un logiciel



# Cycle de développement

✧ **La phase 1 => Analyse, conception, développement**

- *L'utilisateur a besoin (même s'il le sait pas)*

Analysis

1. **Le propriétaire exprime ce besoin en un Cahier des charges**

Design

2. **Le développeur spécifie ce qu'il est capable de produire en un Cahier des Spécifications techniques**

Dev

3. **Ensuite démarre le développement**
   Codage, et livraison au propriétaire, avant la pré-production.

# Cycle de développement

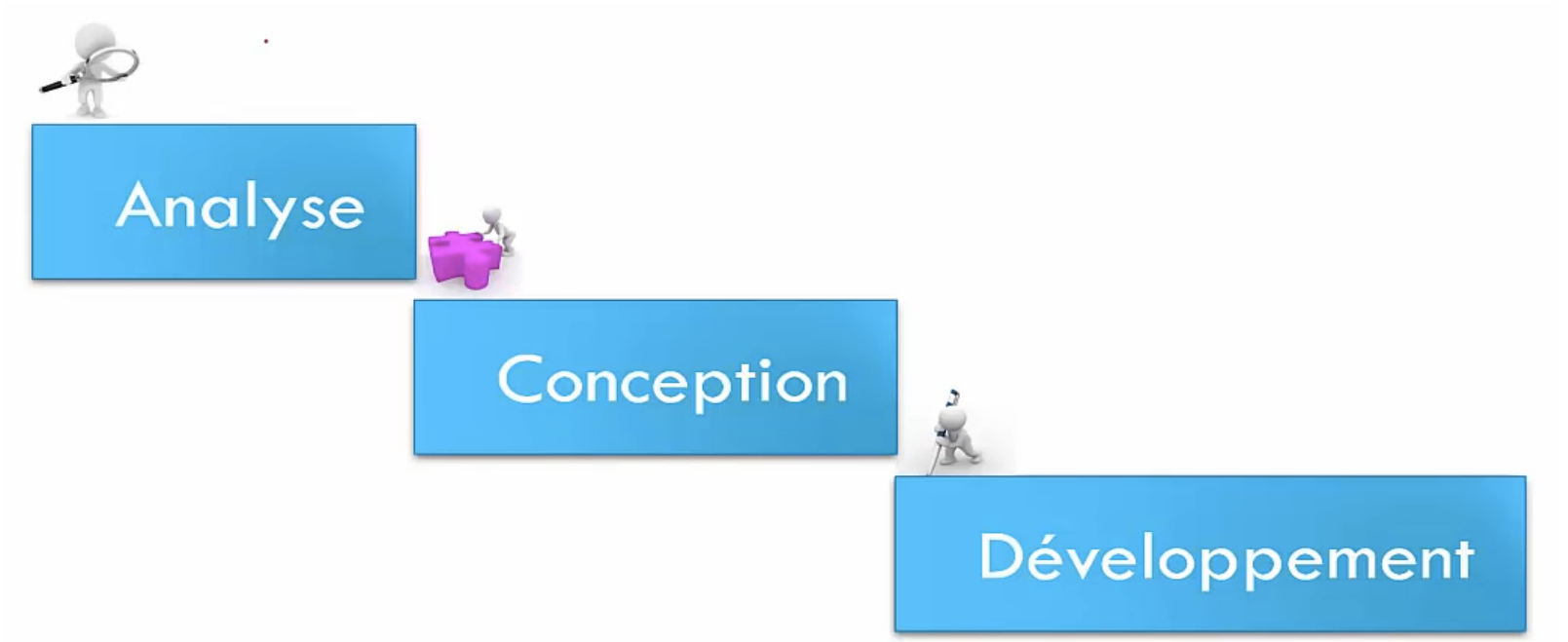
## ✧ La phase 1 : Un exemple

- La **Direction des études** de l'EPT désire disposer un ENT pour coordonnées les activités pédagogiques : gestion notes, des emplois du temps, bibliothèque numérique, cours et cahier de texte numériques, etc.

1. Le **CRI** de l'EPT, via sa division logicielle, décide de répondre à ce besoin
2. Le **CRI** demande à un de ses ingénieurs de développer l'application.
3. **M. X**, ingénieur GIT, réalise l'application, qu'il est actuellement entrain de déployer sur l'intranet de l'EPT.

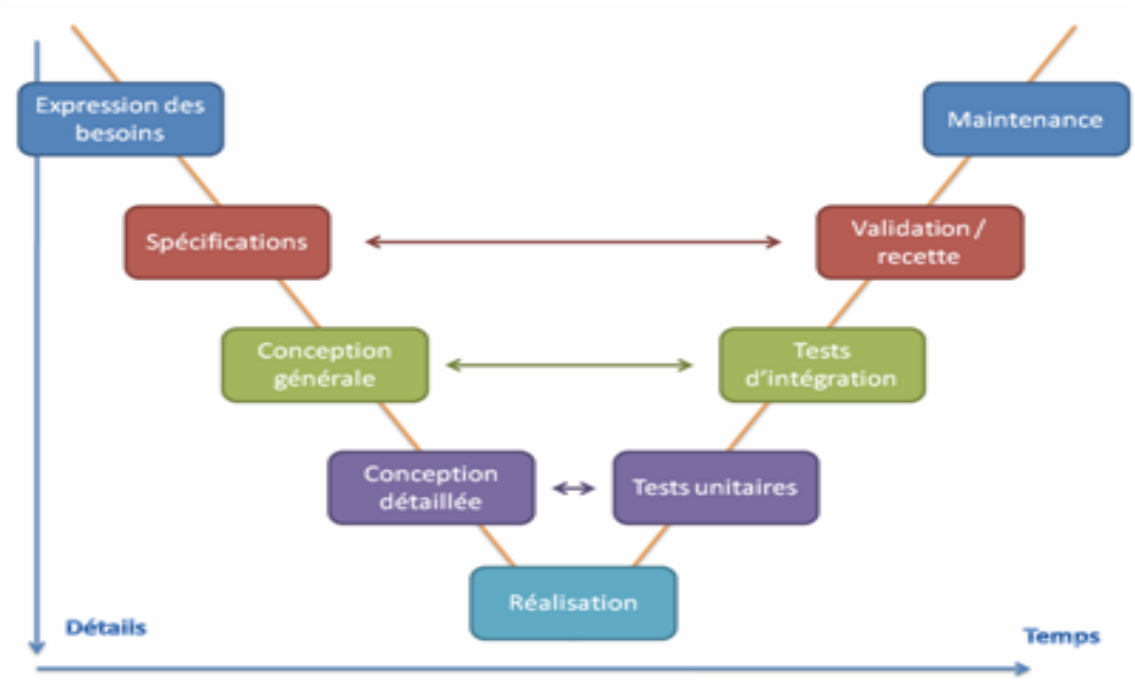
# Différents cycles de développement

- Cycle **en Cascade**
  - Etapes à prendre en charge les unes après les autres
  - « *On ne peut construire la toiture avant la fondation* »
  - Produire des livrables définis au préalable, et jugés satisfaisant dans une étape de validation-vérification



# Différents cycles de développement

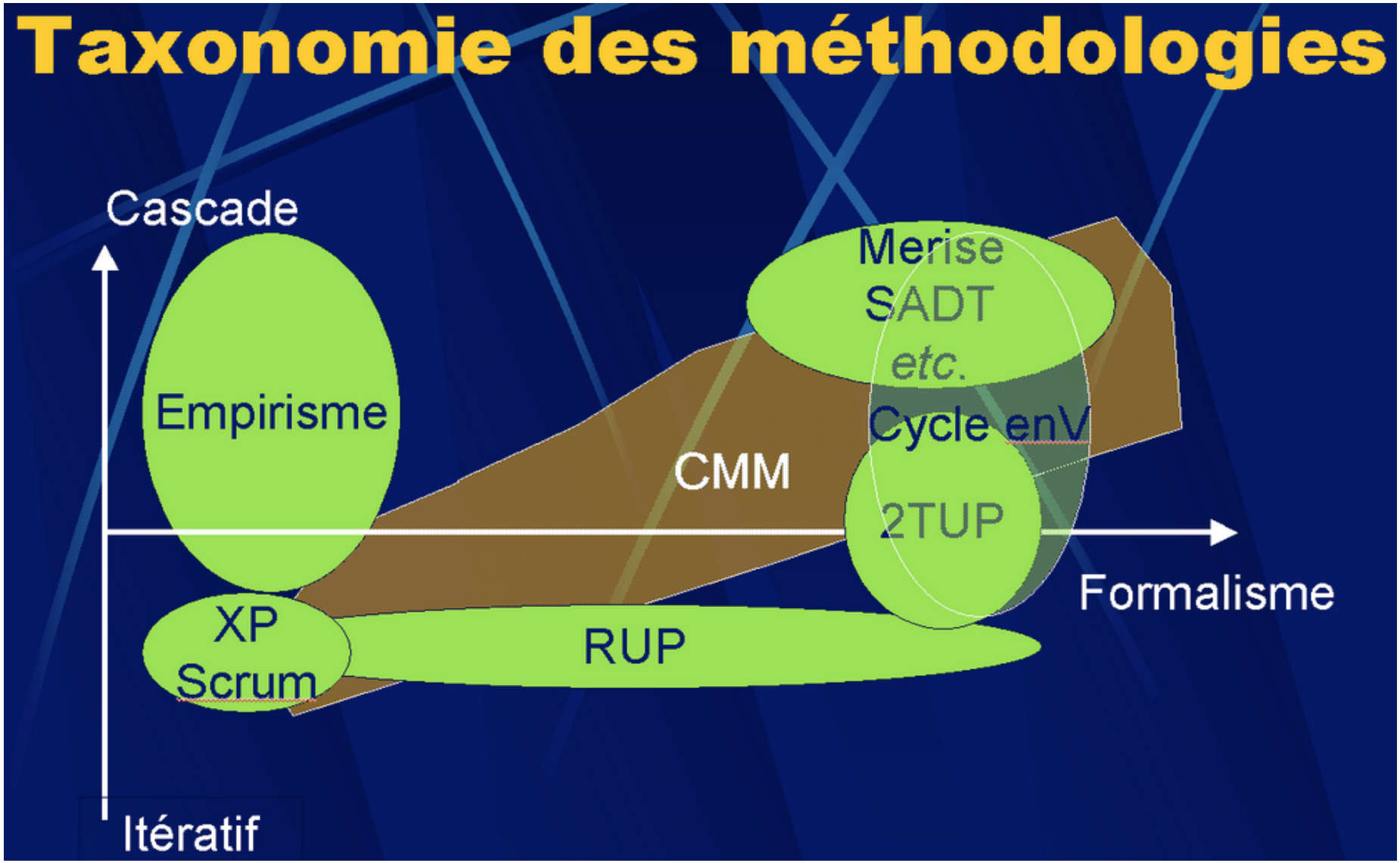
- Cycle **en V**
  - Evolution du cycle en cascade : mieux gérer les erreurs
  - Introduction des tests logiciels à chaque étape
  - Tests de validation (Analyse) - Tests d'intégration (Conception) - Tests unitaires (Développement)

# Différents cycles de développement

- Cycle **en Spirale**
  - Le cahier des charges est décomposé en plusieurs parties qui se complètent pour atteindre le logiciel final.
  - Sur chaque partie est exécuté un cycle en V, appelée **itération**
  - Développement de plusieurs versions du logiciel
    - **Afin d'avoir un produit de plus en plus complet et robuste**

# Différents cycles de développement



# CAHIER DE SPECIFICATIONS AVEC UML

# Le Langage UML : Genèse

- **Depuis 1960 : Les langages de programmation orienté objet** ont vu le jour;
  - tels que Simula, Smalltalk, C++, Java, C#,
  - avec une syntaxe formelle pas lisible par des non-programmeurs,
  - un inconvénients que les humains décrient,
  - ils préfèrent des langages graphiques pour représenter des abstractions afin d'avoir une vue d'ensemble d'un système en un temps court

- **Entre fin des années 1980 et début des années 1990** les notations graphiques se multiplièrent
  - En 1994, les trois Amigos, Jim Rumbaugh (OMT), Grady Booch (méthode Booch) et Yvar Jacobson décidèrent de regrouper leur notation, qu'ils acceptèrent de nommer UML
  - En 1997, la version 1.0 fut lancée et validée par l'Object Management Group (OMG) : avec 9 Diagrammes
  - En 2005, la version 2.0 fut proposée avec des améliorations sur les diagrammes existants, et l'apparition de nouveaux pour atteindre 13 Diagrammes.
  - En 2011, nous atteignons 14 Diagrammes avec la taxonomie des profils UML, dans la version 2.4.1

- **UML (Unified Modeling Language)** est une notation destinée à la modélisation
  - Il ne contient pas de guide méthodologique

# Le Langage UML

- **Besoin de modéliser** pour construire un logiciel
  - Prise en charge des aspects statiques et dynamiques
  - Prise en charge de différents niveaux d'abstraction
  - Indépendant du processus de développement

- **Besoin d'un langage normalisé** pour la modélisation
  - Langage semi-formel : Graphique, compréhensible par l'humain
  - Standard très utilisé

- **Besoin d'une conception orientée objet**
  - Façon efficace de penser un objet
  - Indépendance du langage de programmation (même non objet)

# Le Langage UML : En résumé

| v · m | Unified Modeling Language | [masquer] |
|---|---|---|
| **Organismes** | ISO · Object Management Group · Partenaires UML | |
| **Personnalités** | Grady Booch · Ivar Jacobson · James Rumbaugh | |
| **Concepts** | **Orientation objet** | POO · Méthode d'analyse et de conception d'applications orientées objet · Encapsulation · Héritage · Polymorphisme |
| | **Structure** | Acteur · Artéfact · Attribut · Classe · Composant · Interface · Objet · Package · Propriété |
| | **Comportement** | Activité · Événement · Message · Méthode · État · Cas d'utilisation |
| | **Relation** | Agrégation · Association · Composition · Dépendance · Généralisation · Héritage |
| | **Autres** | Cardinalité · Profile · Stéréotype |
| **Diagrammes** | **Structure** | Classes · Composants · Structure composite · Déploiement · Objets · Paquetages |
| | **Comportement** | Activité · État · Cas d'utilisation |
| | **Interaction** | Communication · Séquence · Global d'interaction · Temps |

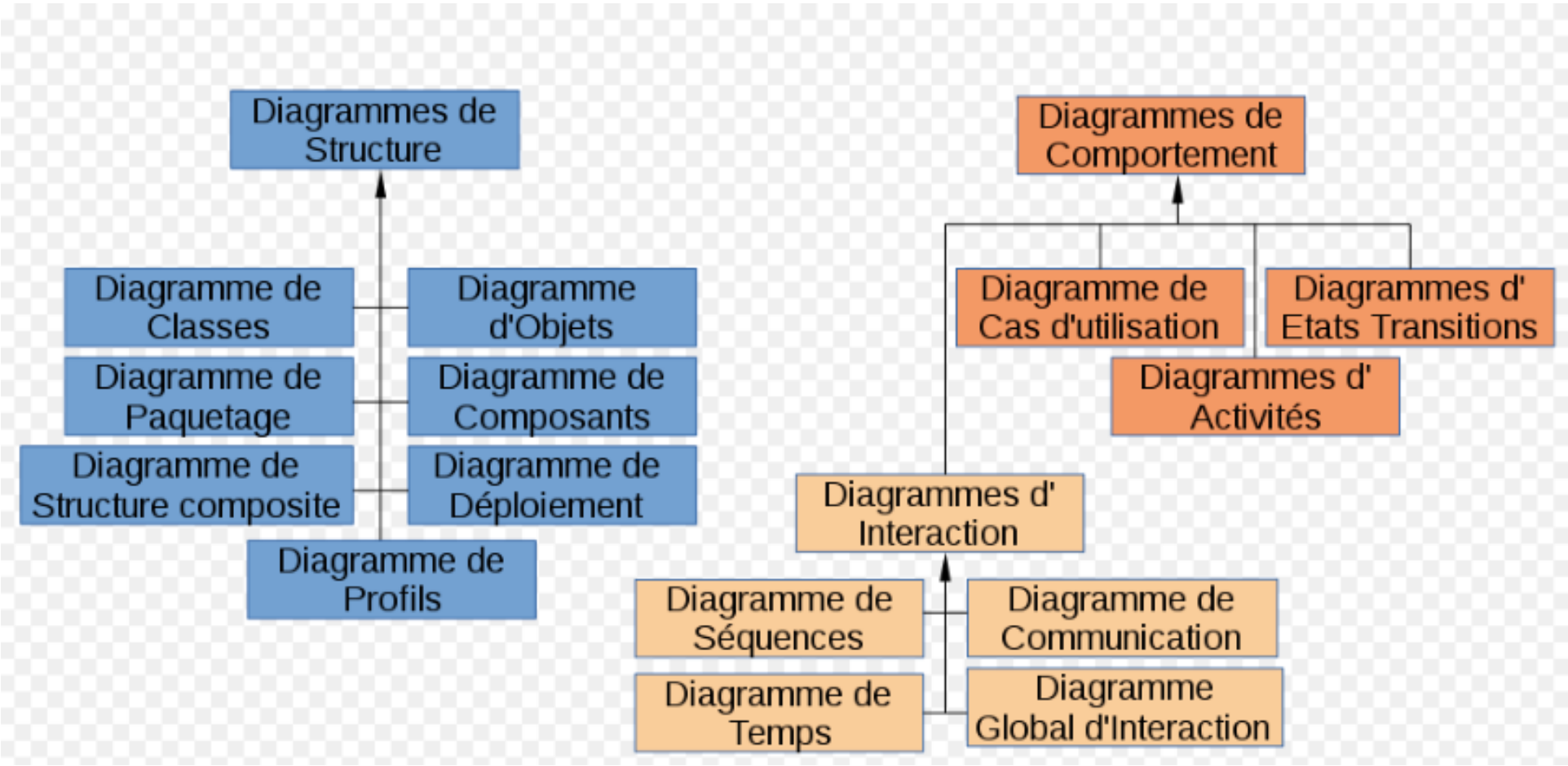
# Le Langage UML

## ✧ Outils de modélisation UML

- PowerAMC
- WinDesign, Dernière version : 2015
- UmlDesigner
- Visio, Microsoft
- Visual Paradigm, Dernière version : 2013
- Oracle Model
- Plugin Eclipse : EMF, Acceleo, papyrus, Dernière version : 2016
- BOUML, Dernière version : 2017
- StarUML, Dernière version : 2015
- ArgoUML, Dernière version : 2011

# Le Langage UML

- **UML 2.0 : 14 Diagrammes**



# Le Langage UML : Tous les diagrammes

- **Les diagrammes de structure ou diagrammes statiques rassemblent :**

  - **Diagramme de classes** : représentation des classes intervenant dans le système.
  - **Diagramme d'objets** : représentation des instances de classes (objets) utilisées dans le système.
  - **Diagramme de composants** : représentation des composants du système d'un point de vue physique, tels qu'ils sont mis en œuvre (fichiers, bibliothèques, bases de données…)
  - **Diagramme de déploiement** : représentation des éléments matériels (ordinateurs, périphériques, réseaux, systèmes de stockage…) et la manière dont les composants du système sont répartis sur ces éléments matériels et interagissent entre eux.
  - **Diagramme des paquets** : représentation des dépendances entre les paquets (un paquet étant un conteneur logique permettant de regrouper et d'organiser les éléments dans le modèle UML), c'est-à-dire entre les ensembles de définitions.
  - **Diagramme de structure composite** : représentation sous forme de boîte blanche les relations entre composants d'une classe (depuis UML 2.x).
  - **Diagramme de profils** : spécialisation et personnalisation pour un domaine particulier d'un méta-modèle de référence d'UML (depuis UML 2.2).

# Le Langage UML : Tous les diagrammes

✧**Les diagrammes de comportement rassemblent :**

- **Diagramme des cas d'utilisation** : représentation des possibilités d'interaction entre le système et les acteurs (intervenants extérieurs au système), c'est-à-dire de toutes les fonctionnalités que doit fournir le système.

- **Diagramme états-transitions** : représentation sous forme de machine à états finis le comportement du système ou de ses composants.

- **Diagramme d'activité** : représentation sous forme de flux ou d'enchaînement d'activités le comportement du système ou de ses composants.

# Le Langage UML : Tous les diagrammes

✧**Les diagrammes d'interaction ou diagrammes dynamiques rassemblent :**

- **Diagramme de séquence** : représentation de façon séquentielle du déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs.

- **Diagramme de communication** : représentation de façon simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets (depuis UML 2.x).

- **Diagramme global d'interaction** : représentation des enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquences (variante du diagramme d'activité) (depuis UML 2.x).

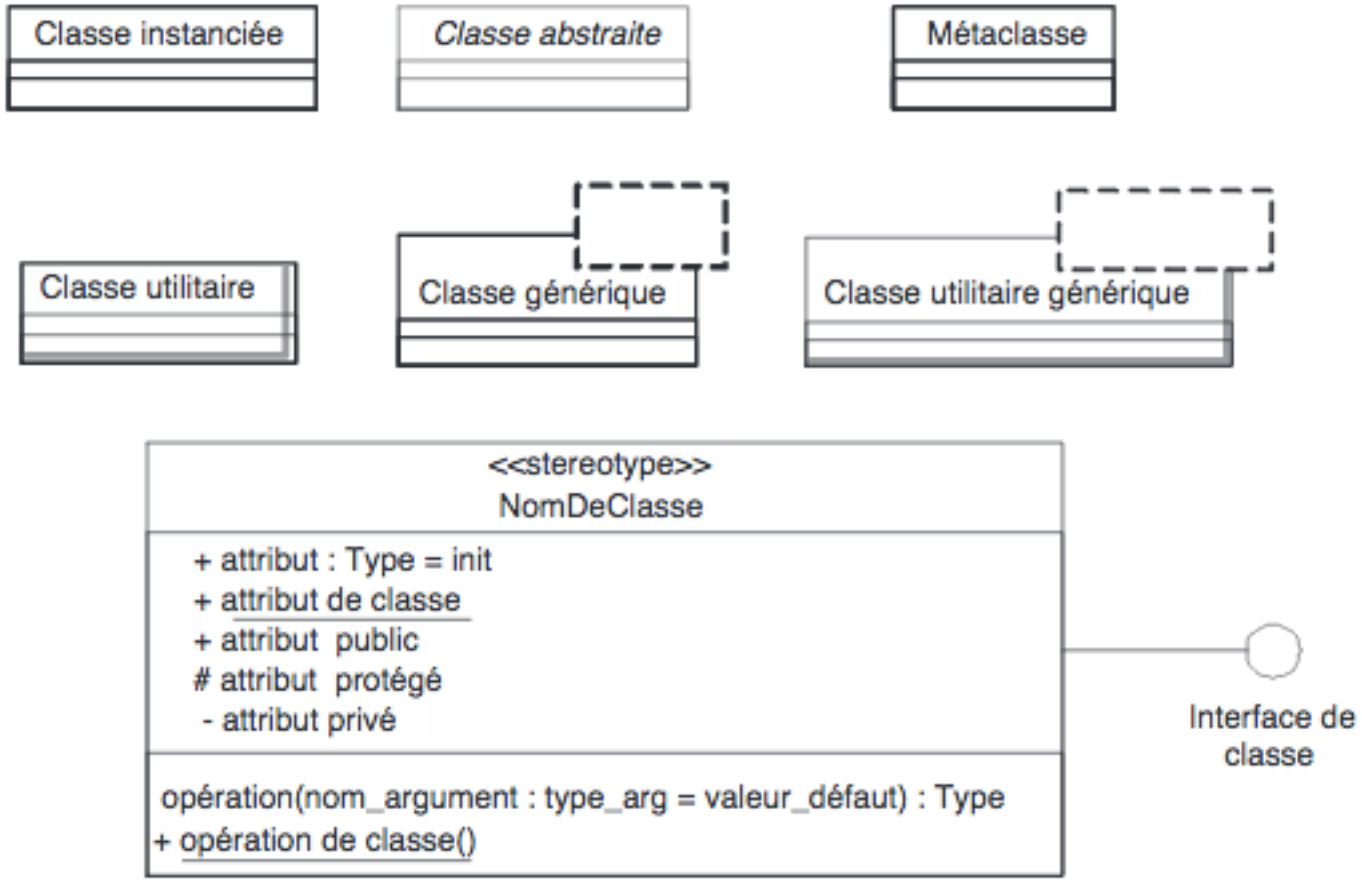
- **Diagramme de temps** : représentation des variations d'une donnée au cours du temps (depuis UML 2.3).

# Conception structurelle

- **Diagramme des classes et Diagramme des Objets:**
  - décrit l'ensemble des classes d'un système ainsi que les associations les reliant.
  - les objets sont identifiés dans le système et portent un nom
  - les classes sont créées en regroupant les objets ayant les mêmes propriétés et les mêmes comportements
  - un objet est une instance d'une classe
- Représentation graphique

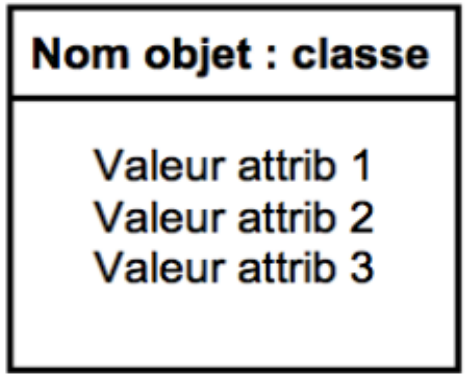
# Conception structurelle

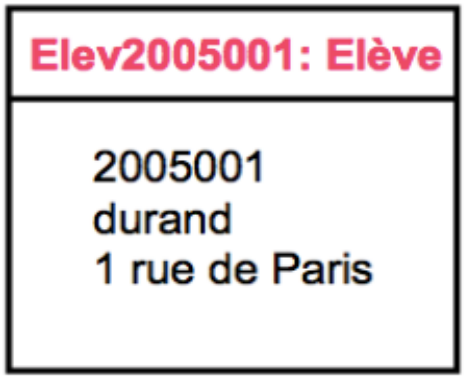
- **Diagramme des classes : même concepts que la POO**



# Conception structurelle

- Représentation UML d'un objet



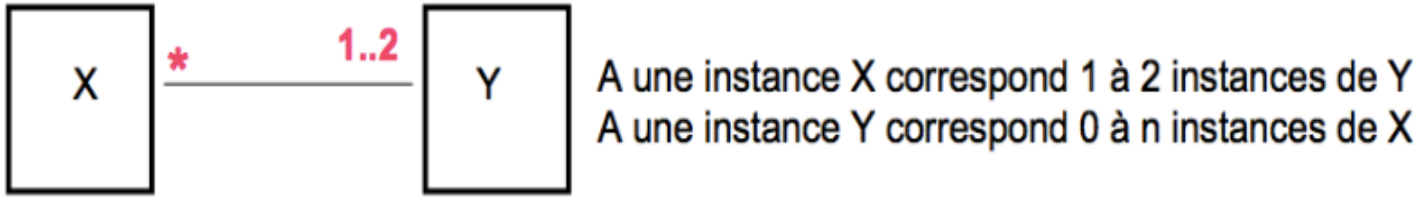


# Conception structurelle

- Association entre classes
  - Liens entre instances
  - Rôle de l'association et son sens

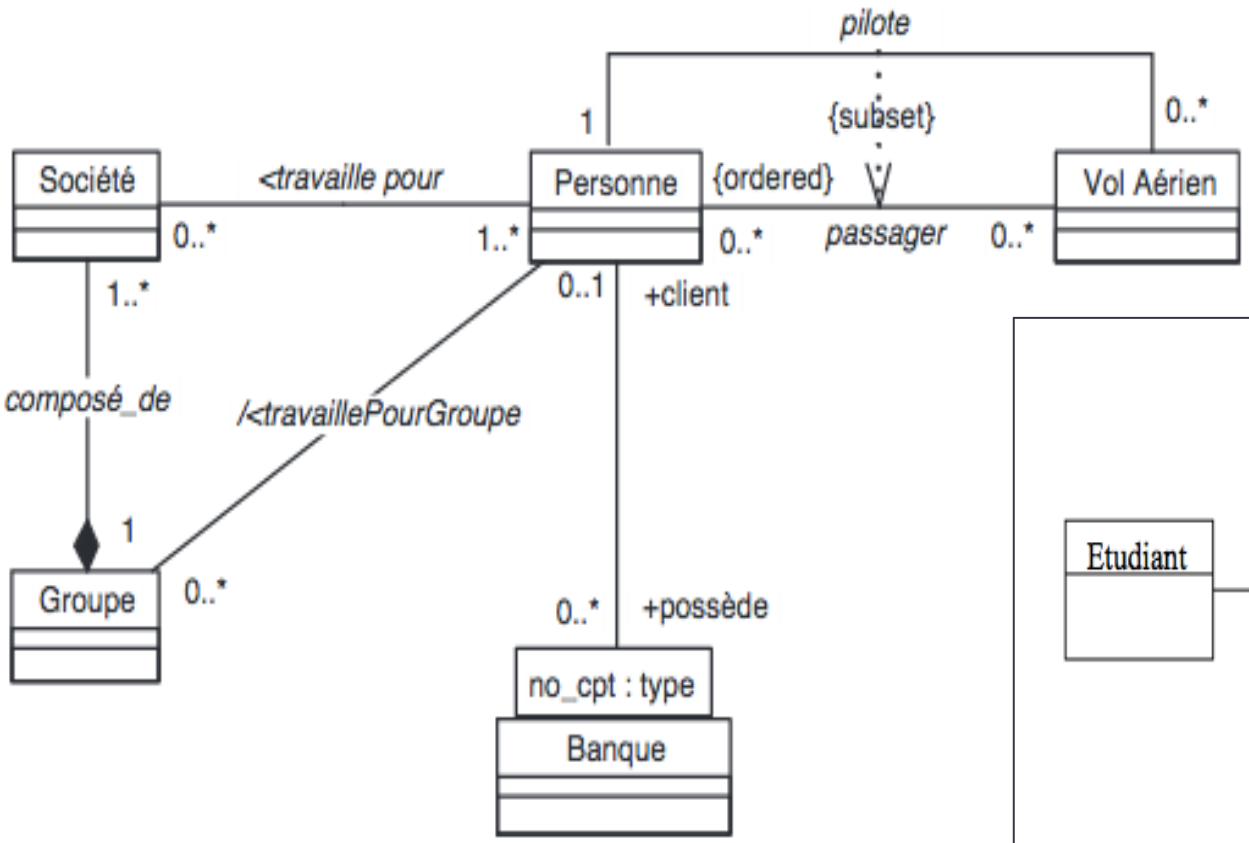
Cardinalités

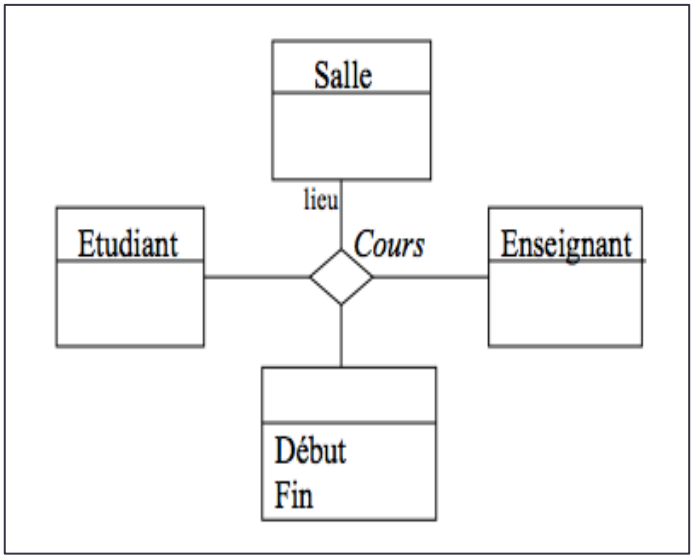
| | |
|---|---|
| **1** | Un et un seul (obligatoire) |
| **0 .. 1** | Zéro ou un (optionnel) |
| **m .. n** | De m à n (entiers) |
| * *ou* **0 .. *** | quelconque |
| **1 .. *** | Au moins 1 |

# Conception structurelle

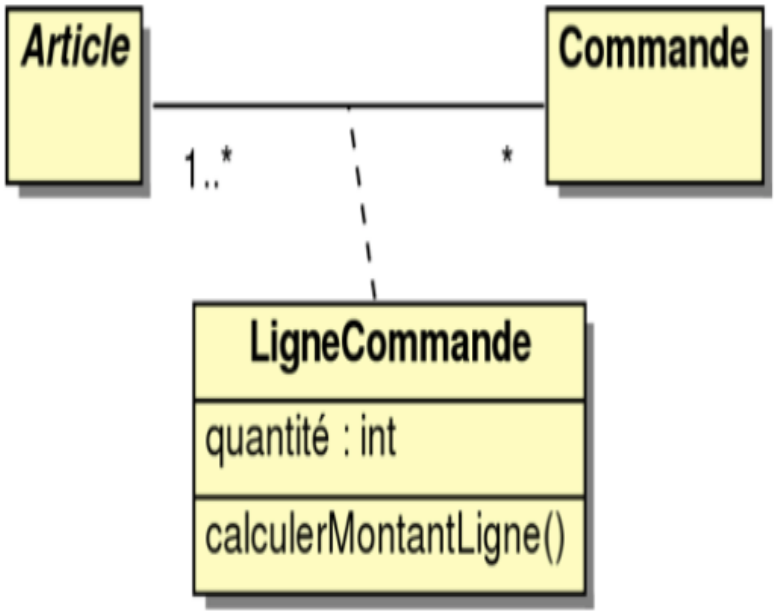
- Association entre classes

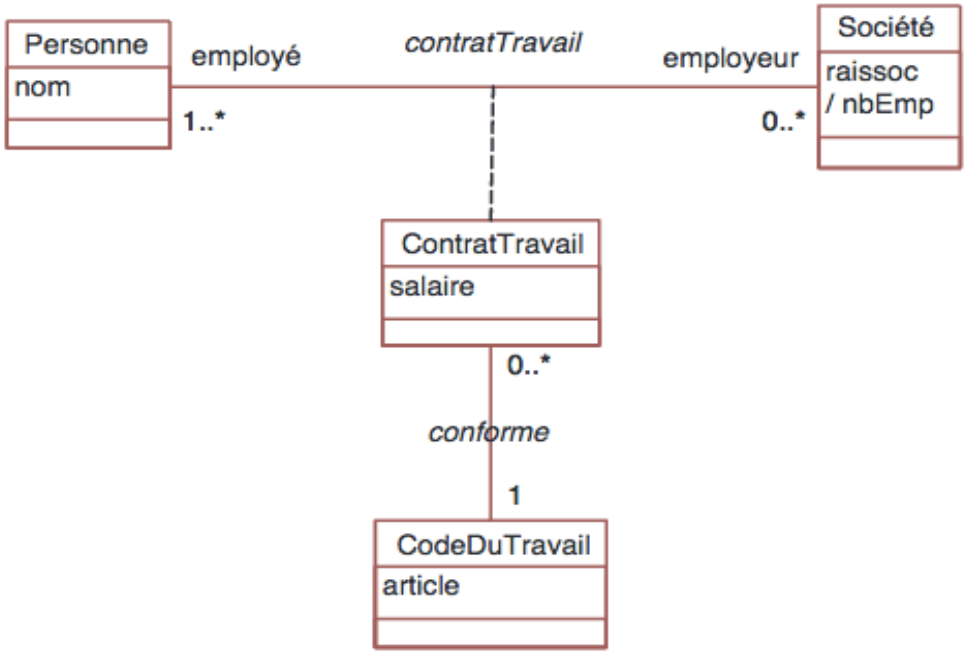


# Conception structurelle

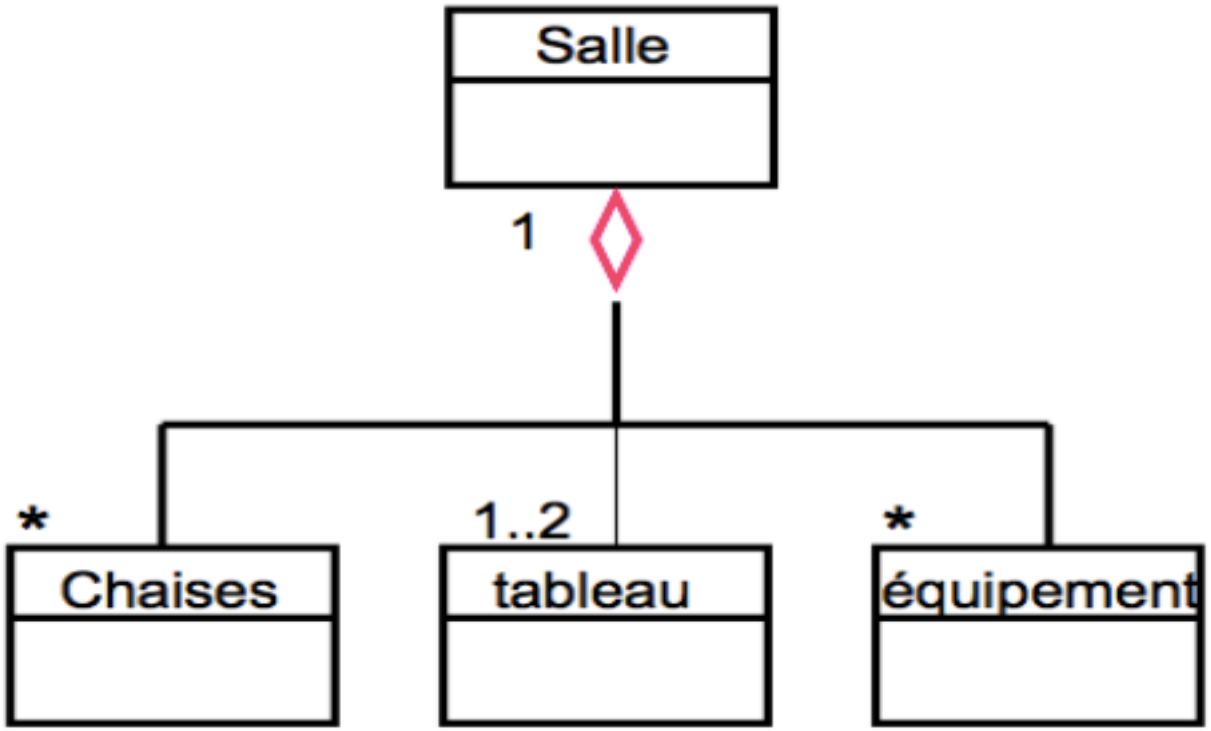
- Association entre classes
  - Peut être raffinée et avoir ses propres attributs, qui ne sont disponibles dans aucune des classes qu'elle lie.
  - Cependant, seules les classes peuvent avoir des attributs, on parle alors de « classe-association »



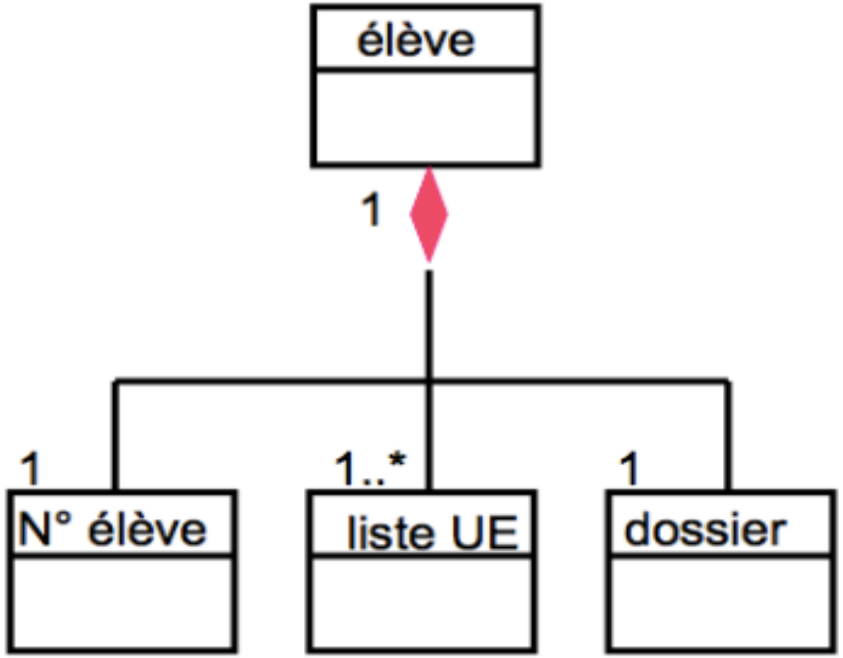
# Conception structurelle

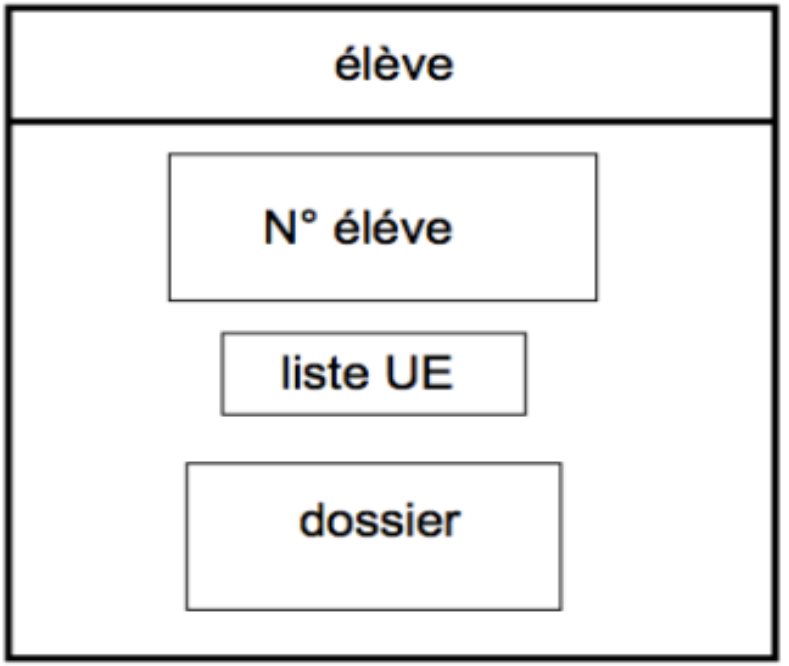
- Agrégation
  - Association entre une classe de type « ensemble » avec plusieurs classes de types « éléments »



# Conception structurelle

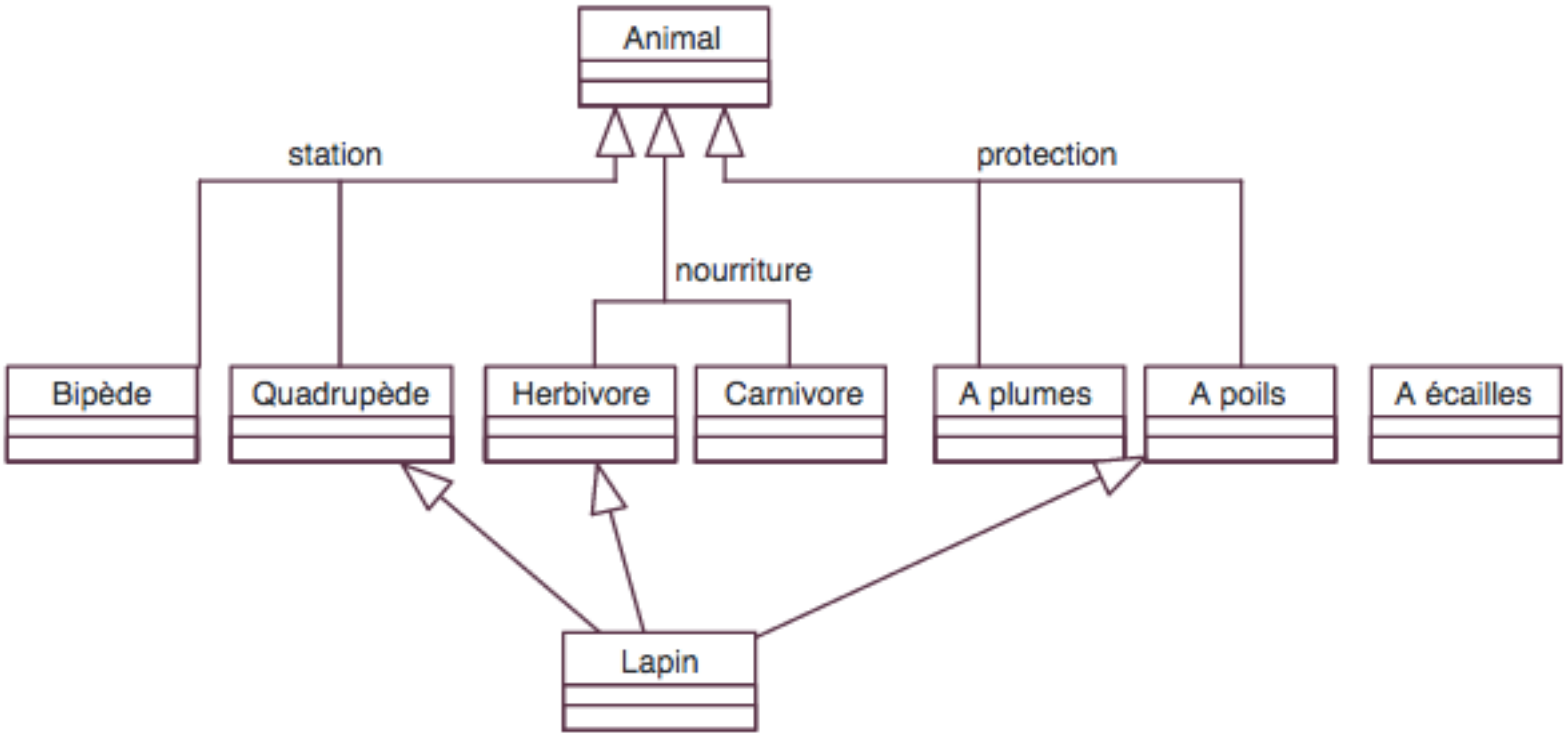
- Composition
  - Agrégation avec une contrainte de durée de vie
  - La suppression de la classe « composée » implique la suppression des classes « composant »



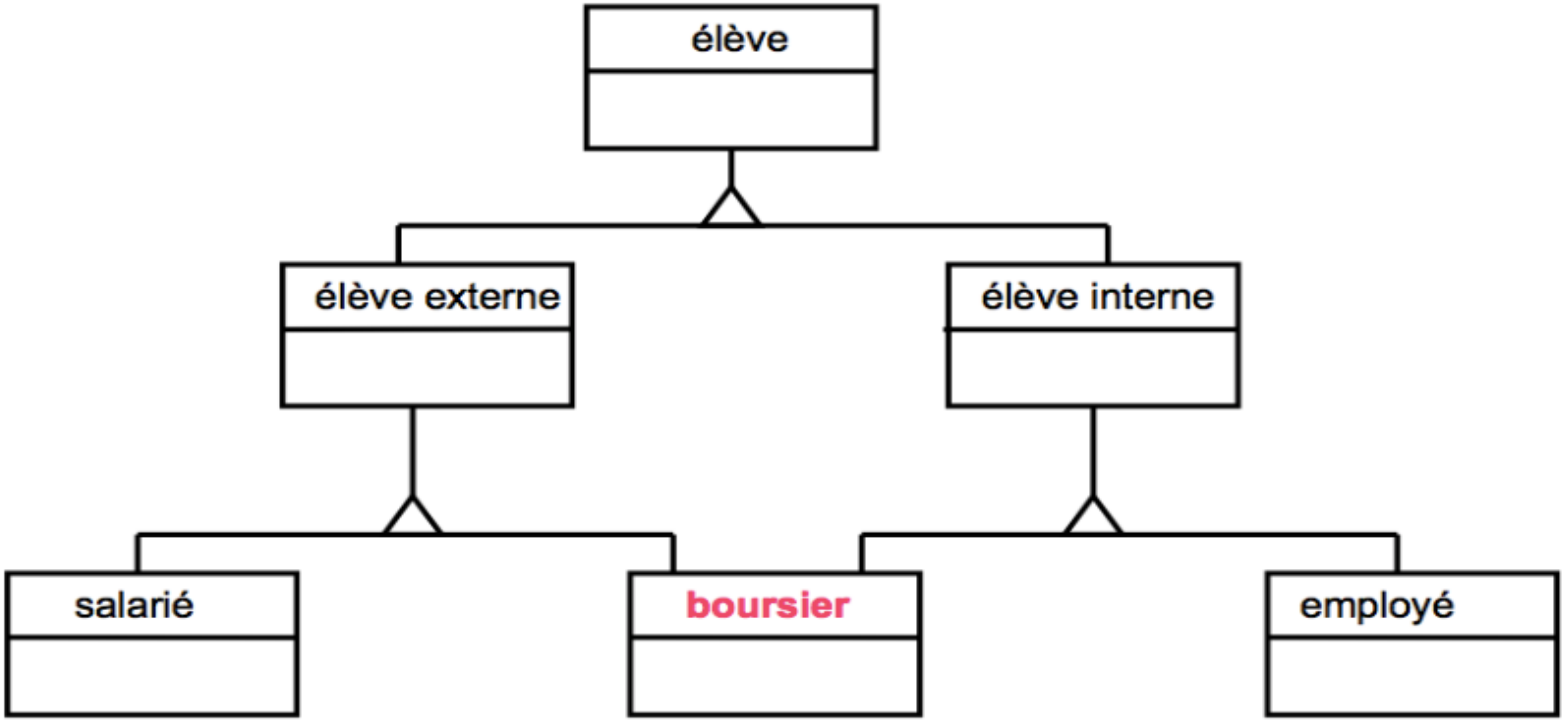
# Conception structurelle

- Généralisation et héritage simple
  - Généralisation : création d'une superclasse à partir de classes
  - **Héritage : création de sous classes à partir d'une classe**

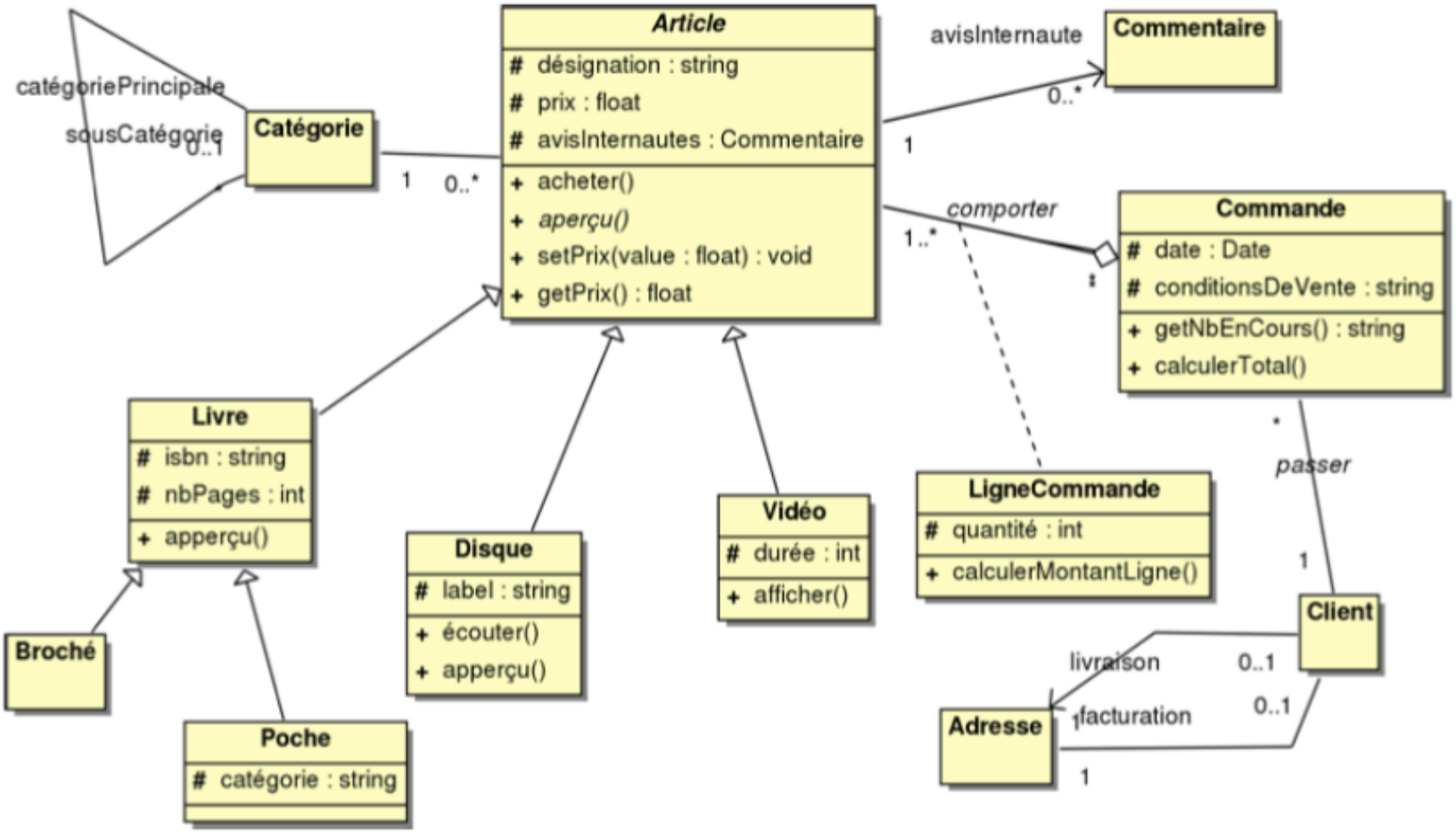


# Conception structurelle

- Héritage multiple
  - Une classe peut hériter de deux classes parentes



# Conception structurelle



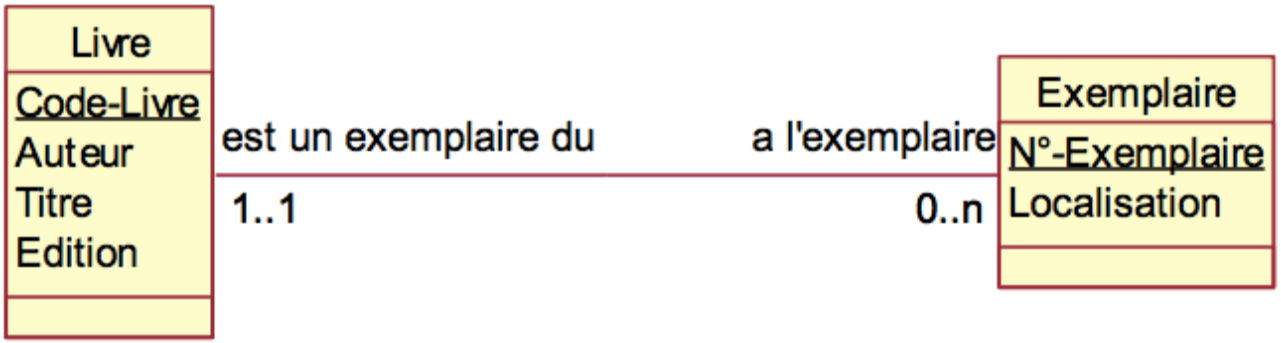
# Du Diagramme de Classes au relationnel

- Pour implémenter une base de données relationnelle, il faut pouvoir traduire le modèle **structurel** conceptuel en modèle logique.

- Cela signifie qu'il faut pouvoir convertir un diagramme de classes en modèle relationnel sous forme de tables relationnelles.

- Le DC est suffisamment formel pour que ce passage soit systématisé à travers des règles de passages.

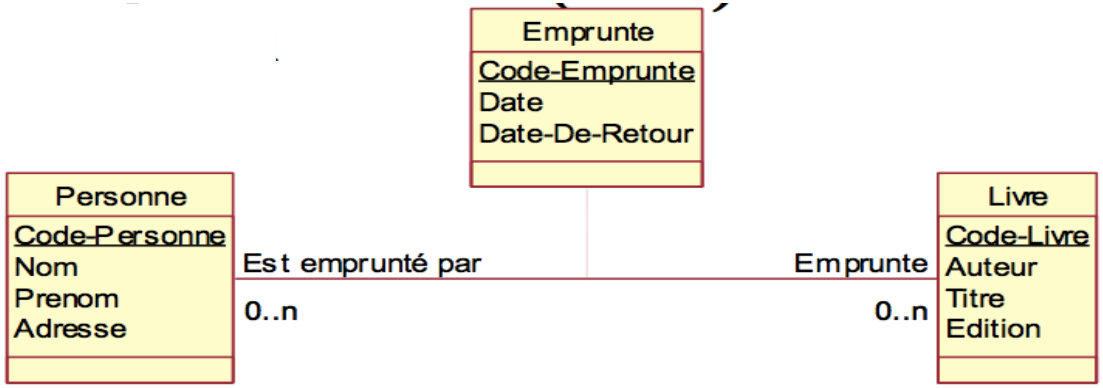
# Règle 1: présence de la cardinalité (?..1) d'un côté de l'association




- Chaque classe se transforme en une table
- Chaque attribut de classe se transforme en un champs de table
- L'identifiant de la classe qui est associée à la cardinalité (?..1) (ex: Livre) devient le clé étrangère de l'autre classe (ex: Exemplaire)


**Contrainte d'intégrité référentielle**:
CléEtrangère ⊆ CléPrimaire
Ex: Exemplaire.Code-Livre ⊆ Livre.Code-Livre

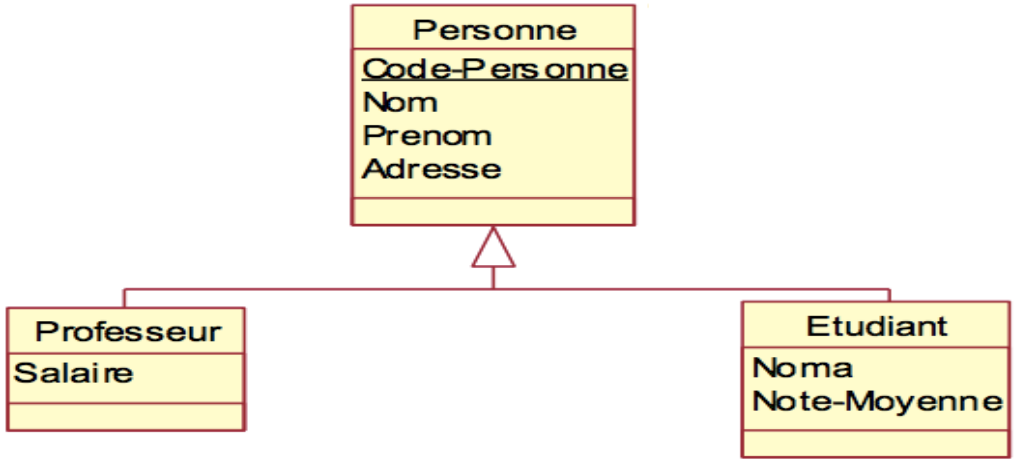
# Règle 2 : présence de (?..N) des deux côtés de l'association



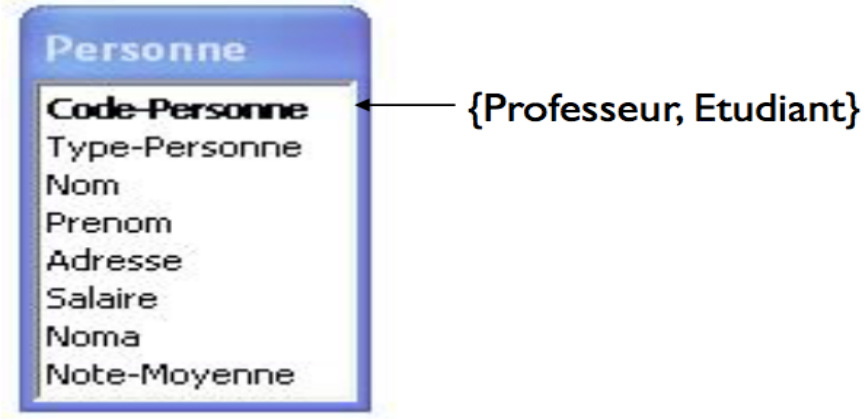

- Chaque classe se transforme en une table
- Chaque attribut de classe se transforme en un champs de table
- L'association se transforme en une table. Cette table a comme champs l'identifiant de chacune des deux classes, plus d'éventuels autres attributs.


Emprunte.Code-Personne ⊆ Personne.Code-Personne Emprunte.Code-Livre ⊆ Livre.Code-Livre

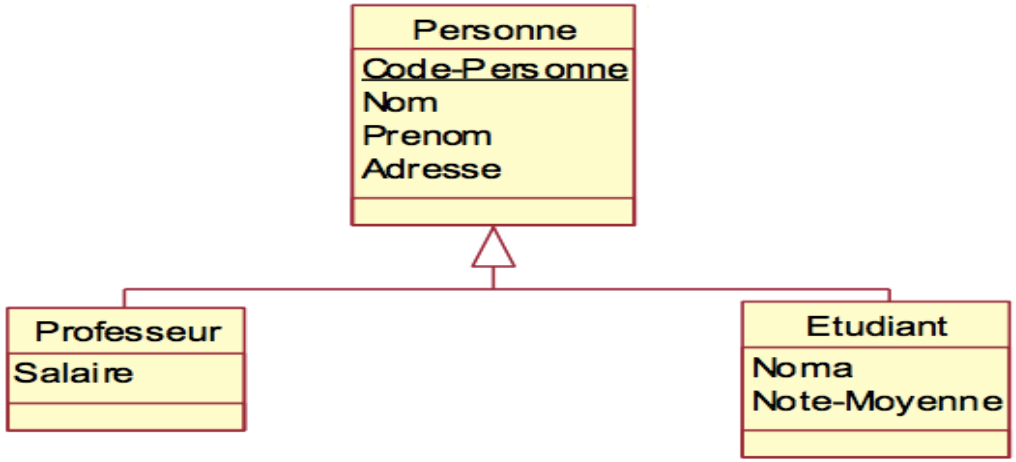
# Règle 3 : présence d'une généralisation




- **Méthode 1** :
  - Créer une table avec tous les attributs des classes
  - Ajouter un attribut pour distinguer les types des objets

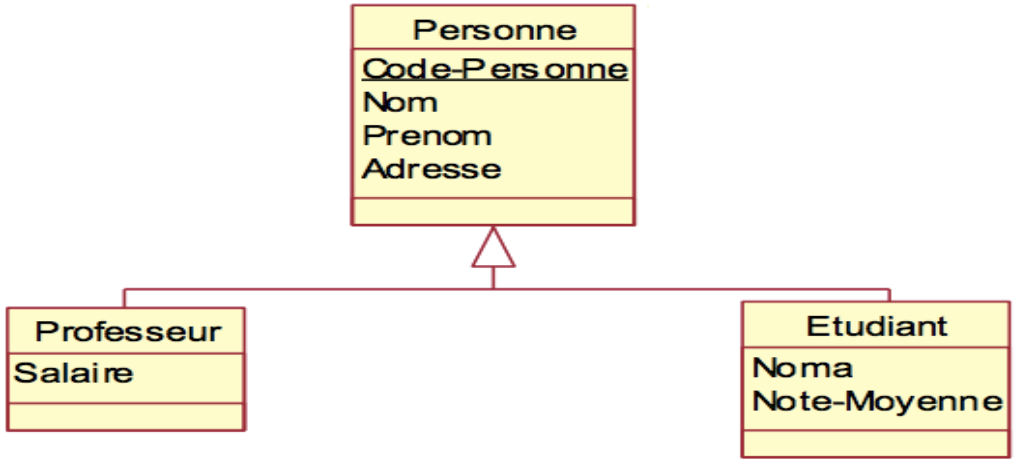


# Règle 3 : présence d'une généralisation

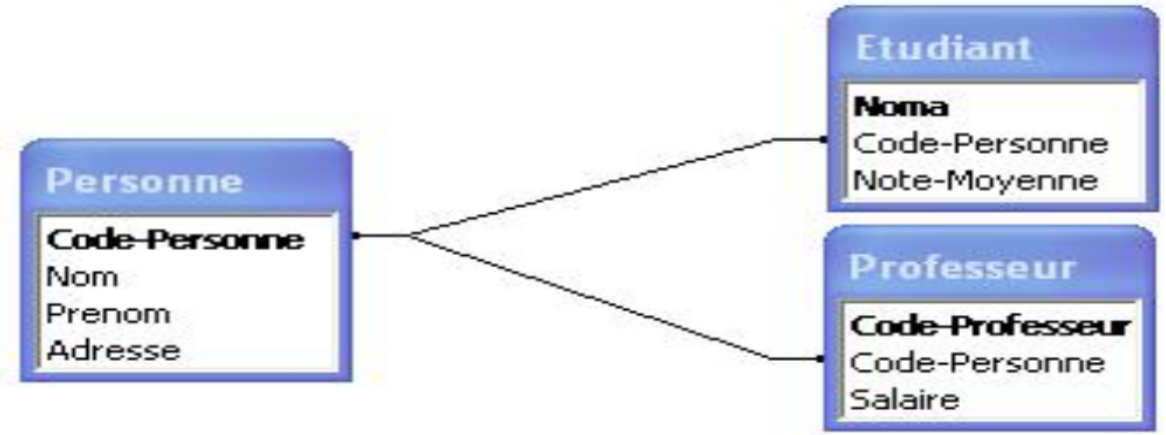



- **Méthode 2** :
  - Créer une table pour chaque sous type, chaque table se compose des attributs génériques et d'attributs spécifiques

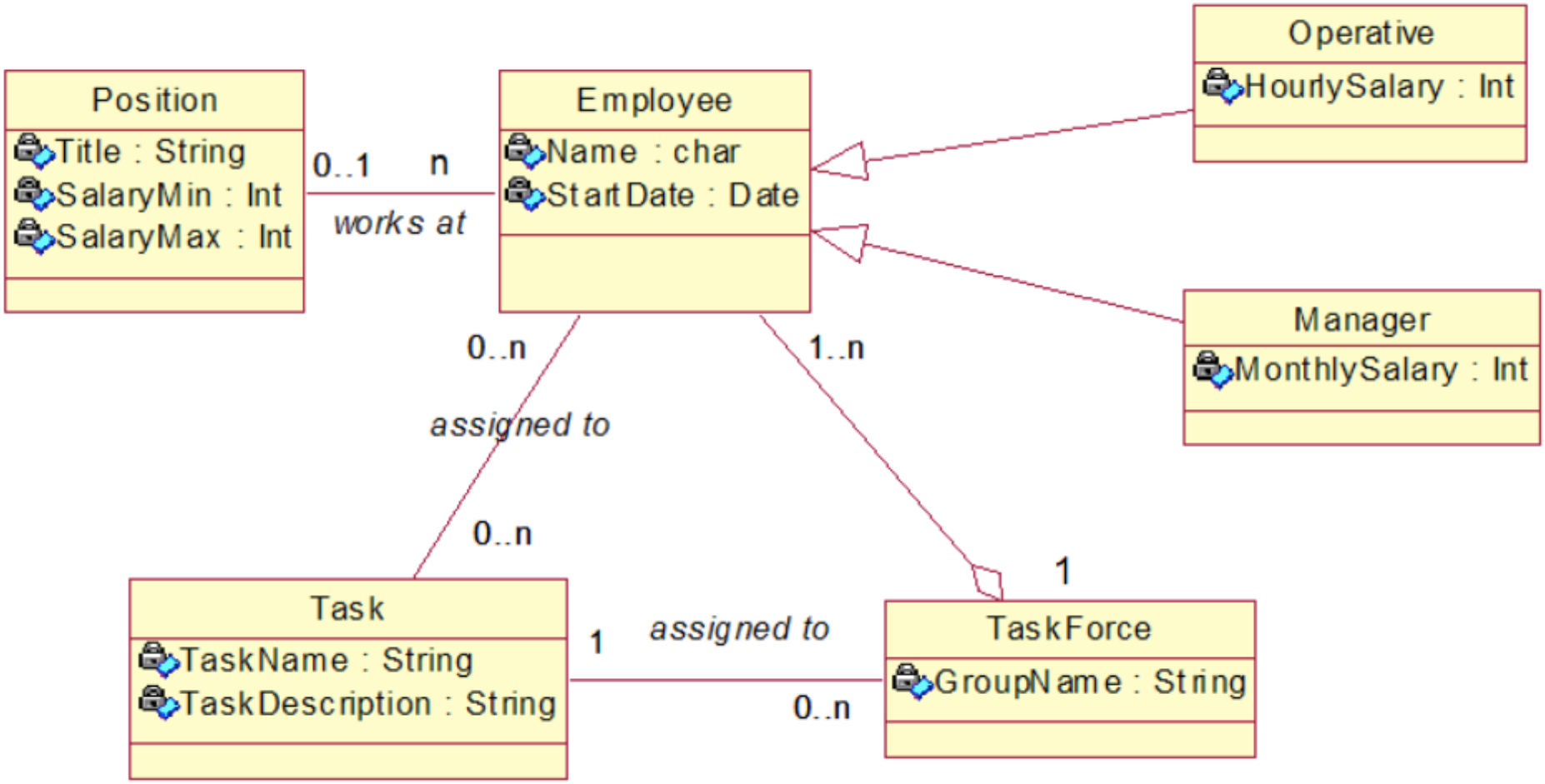

# Règle 3 : présence d'une généralisation



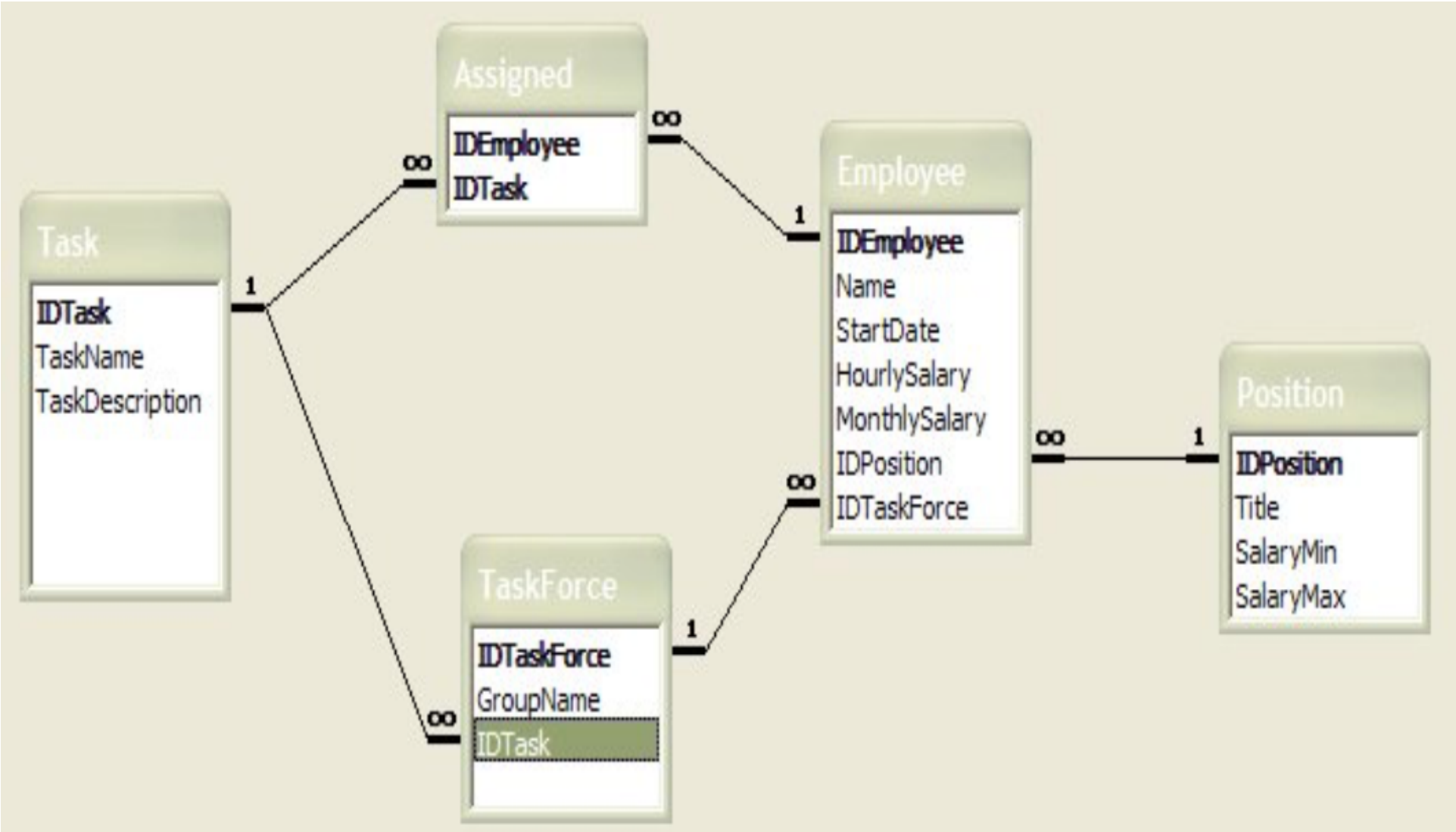

- **Méthode 3** :
  - Créer une table par classe et des associations



# Exemple de passage d'un DC à un modèle logique d'une BDR

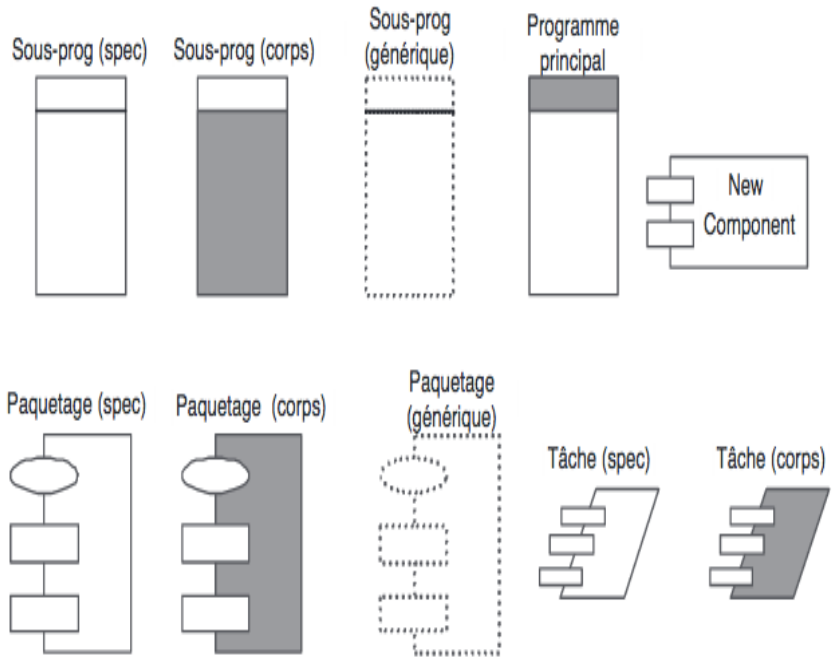


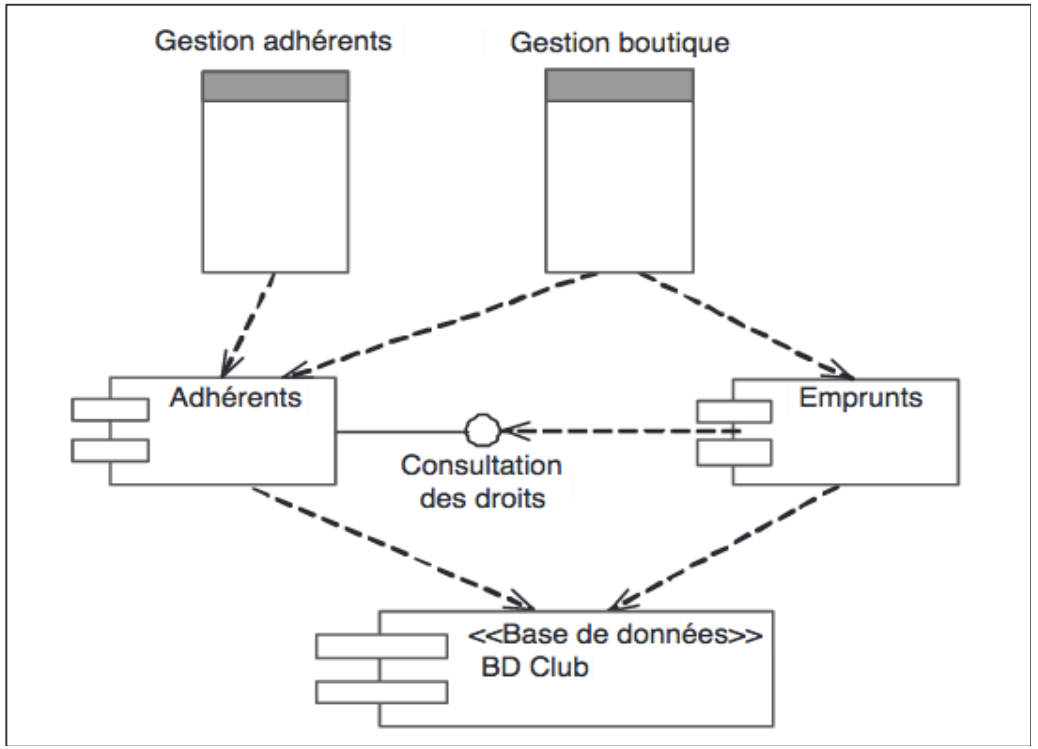
# Exemple de passage d'un DC à un modèle logique d'une BDR



# Conception architecturale

- **Diagramme de composants**
  - Description des composants du système
  - Description en composants, processus, applications, bibliothèques
  - Prise en compte des dépendances



# Conception architecturale

## ✧Diagramme de Déploiement

- Description de l'architecture physique
- Répartition des composants sur les nœuds physiques