

# Conception

---

Domain-Driven Design, Qualité  
logiciel et UML

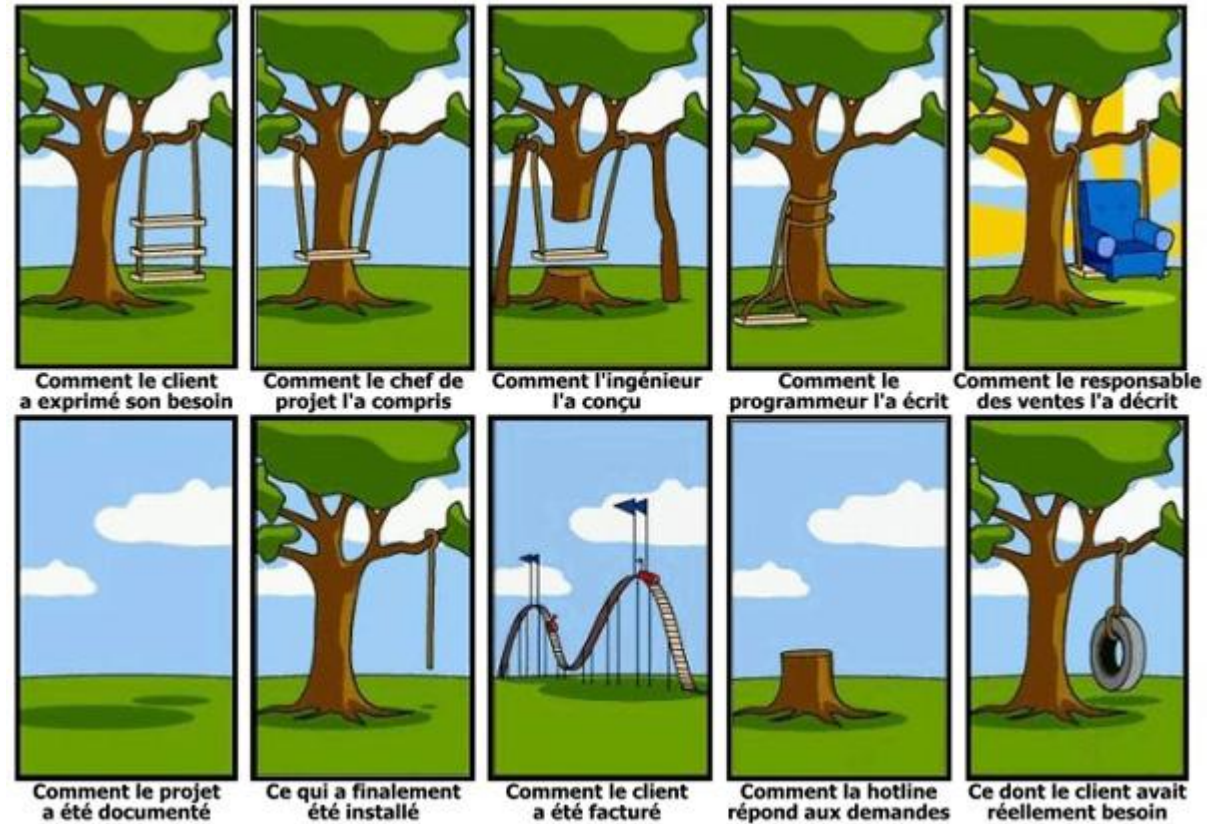
# Introduction

Concevoir un programme qui s'adapte parfaitement aux besoins en évolution permanente des clients.

- Coder sans prendre de la hauteur aboutit à des solutions fragiles et difficiles à modifier.
  - Les clients changent d'avis = anticipation
- votre objectif premier : comment ma solution peut être la plus pratique, utile et ergonomique possible pour ses utilisateurs ?
  - Comprendre les enjeux en gardant à l'esprit les besoins des utilisateurs.
  - Moins de corrections et de modifications = moins de temps et moins d'échanges fastidieux (productivité)
  - Collaboration avec les acteurs impliqués dans la construction du produit = Domain-Driven Design

# L'incontournable

Chacun peut avoir une vision différente et qu'il est judicieux de bien s'assurer que tout le monde parle de la même chose, quitte à répéter, préciser, reformuler et même avoir l'air stupide quelques minutes ..





# Domain-Driven-Design



# Domain-Driven-Design

## Définition

- Le **DDD** (**Domain-Driven Design**) est une approche incontournable pour concevoir des logiciels robustes et évolutifs en plaçant le domaine métier au cœur de la conception.
- Popularisé par **Eric Evans**, ce paradigme permet aux développeurs et aux architectes logiciels de structurer leurs applications autour d'un modèle conceptuel clair, favorisant ainsi la collaboration avec les experts métier.

# Domain-Driven Design

## Modèle de domaine

- Représentation conceptuelle des éléments clés, compris par tous, pour créer la solution souhaitée par le client.
- C'est l'idée qui constitue le fondement même d'un programme, mais qui serait compréhensible à la fois par les parties prenantes et par les développeurs.

## Les éléments clés

- Pour faire simple, si vous en parlez, c'est un élément important, ou élément clé.
- En revanche, si vous le laissez de côté, il est moins important, c'est un élément qu'on appelle mineur.

# Exemple

Prenons l'exemple d'un film. Si nous devons le résumer à quelqu'un, qu'est-ce que vous diriez, et que laisseriez-vous de côté ?

- Le ou les principaux personnages sont importants.
- Les obstacles qu'ils doivent franchir, également.
- Le fait que le personnage doit se rendre sur Tatooine, aussi.
- *Mais qu'il porte une veste couleur blanc cassé ne l'est pas.*
- **Ce regroupement d'éléments clés est votre domaine.**
- Si le film à une suite, certains éléments s'ajouteront aux éléments clés de votre 1<sup>er</sup> domaine. Mais certains éléments devront être à part du fait d'une nouvelle histoire : cela deviendra un domaine à part entière.

# Alors comment Domain-Driven Design peut-il m'aider ?

Le Domain-Driven Design se concentre sur deux questions clés

- Pour qui suis-je en train de concevoir cette application ?
- Que veulent-ils réaliser ?

- **Étape 1** : Identifiez les utilisateurs de l'application.
  - Pour qui et par qui l'application va-t-elle servir ?
- **Étape 2** : Entamez la conversation.
  - Pour chaque utilisateur, quelles sont ses besoins et comment les mettre en œuvres pour l'utilisateur ?
  - Permet de définir les premières fonctionnalités (spécifications fonctionnelles) ...
- **Étape 3** : Posez des questions complémentaires.
  - L'objectif est d'aller au-delà des questions standards pour éventuellement déterminer d'autres fonctionnalités, mais qui peuvent ne pas être retenues.
- **Étape 4** : Appropriiez-vous le vocabulaire du domaine
  - Utiliser le vocabulaire du domaine dit **le langage omniprésent**, ensemble de terme de même signification pour tous les participants.
- **Étape 5** : Ajuster le modèle à la bonne échelle
  - Pour le développement de logiciels, nous devons modéliser les idées importantes, mais nous n'avons pas besoin de tous les détails.



# Exemple : Application bibliothèque

## Les usagers qui empruntent des livres

- Parler aux usagers ?
  - Rechercher des livres !
    - Comment ? Par le titre, par l'auteur, par le sujet du livre etc...
  - Emprunter des livres !
    - Être informé de sa disponibilité ? SMS ? Email ?
    - Comment le réserver ?
    - Comment le retirer ?

## Bilan des usagers ?

- ✓ Ils souhaitent rechercher des livres.
- ✓ Ils souhaitent emprunter des livres.
- ✓ Ils souhaitent réserver des livres.
- ✓ Ils souhaitent être informés des disponibilités.

# Exemple : Application bibliothèque

## Les bibliothécaires qui gèrent la bibliothèque.

- Parler aux bibliothécaires ?
  - Commande des ressources à un fournisseurs
    - Ressources ? Un livre, un magazine, des DVD...
  - Comment procédez-vous ?
    - Pour commander, nous utilisons l'ISBN
    - Ensuite à la réception, il faut les remettre en stock.
  - Pour les emprunts comment procédez-vous ?
    - Nous devons gérer la sortie du stock et son retour en stock et contrôler l'état de l'emprunt.
    - En cas de détérioration, une amende est donner à l'usager et on commande un exemplaire de remplacement.

## Bilan des bibliothécaires ?

- ✓ Ils commandent des ressources (livres, magazines ou DVD)
- ✓ Ils les remettent en stock.
- ✓ Ils contrôlent leur état à leur retour de prêt.
- ✓ Ils infligent des amendes aux usagers.

# Les acteurs, les cas d'usage, les entités du modèle.

Pour être un développeur heureux, il faut être un développeur qui comprend que le changement fait partie intégrante du développement.

Ça vous évitera quelques frustrations à chaque nouvelle demande de modification.

- **Identifiez les acteurs** : les utilisateurs de l'application
  - Tout ce qui utilise votre application est considéré comme un acteur.
  - Cernez les différents domaines grâce aux acteurs.
- **Priorisez les acteurs**
  - les acteurs prioritaires se situent là où vous identifiez les plus grandes retombées pour l'entreprise/le client.
- **Identifiez les cas d'utilisation** ou **use cases** : ce que les acteurs essaient de faire
  - Identifier le ou les objectifs que l'utilisateur (acteur) souhaite atteindre et, ensuite, remonter à l'interaction entre l'utilisateur et le programme, qui leur permet d'atteindre cet objectif.
- **Identifiez les entités** : définissez les concepts de votre programme
  - Rechercher les idées pérennes, c'est-à-dire celles qui vont durer.

# Reprise de l'exemple

Cas d'utilisation  
"rechercher un livre"

- L'**utilisateur** demande à rechercher un livre.
- L'application affiche la page de recherche.
- L'**utilisateur** saisit le nom de l'auteur.
- L'application vérifie que le nom de l'auteur existe.
- L'**utilisateur** saisit le titre du livre.
- L'**utilisateur** demande l'exécution de la recherche.
- L'application lance la recherche.
- L'application affiche les résultats.

# Reprise de l'exemple

Identification des entités

- L'**utilisateur** est une donnée pérenne, ainsi que le **livre**. Ce seront nos deux premières entités.
- L'**auteur** est également une donnée pérenne en lien avec les livres. Cela pourra être une entité (choix à faire).
- La page de recherche est un détail de la mise en œuvre et n'est pas une entité. C'est le programme qui la met en œuvre.



# Qualité d'un logiciel

# Qualité d'un logiciel

## Validité et Adéquation

- Validité
  - La validité (correction, justesse, conformité) est la capacité que possède un produit logiciel à remplir exactement ses fonctions, définies par le cahier des charges et les spécifications.
- Adéquation entre :
  - Le besoin effectif de l'utilisateur
  - Les fonctions offertes par le logiciel

*Même le logiciel le mieux conçu techniquement, s'il ne rend pas les services escomptés, est inutile et son développement aura été du temps perdu.*

# Qualité d'un logiciel

Amélioration

- Pour améliorer la qualité d'un logiciel :
  - En phase sur l'analyse des besoins,
  - Amélioration de la communication (langage commun, démarche participative)
  - Travail rigoureux.



# Qualité d'un logiciel

Coût d'un logiciel

- Le coût d'un logiciel n'est pas seulement le coût de production de sa première version. Il intègre également :
  - La correction de bugs.
  - La maintenance.
- Poids de la maintenance :
  - La maintenance absorbe les 2/3 du coût global d'un logiciel.
- Corriger un problème est d'autant plus coûteux qu'il vient d'une erreur en amont :
  - 4/5 du coût de maintenance est absorbé par la correction de problèmes de définition des besoins.
  - 1% seulement concerne les problèmes de codage.
  - La moitié du coût global d'un logiciel vient donc d'une mauvaise compréhension des besoins (Zeltovitz, De Marco).

# Qualité d'un logiciel

Coté utilisateur.

- **Corrections**
  - Aptitude à répondre aux besoins et à remplir les fonctions définies dans le cahier des charges
- **Robustesse**
  - Aptitude à fonctionner dans les conditions non prévues au cahier des charges, éventuellement anormales
- **Extensibilité**
  - Facilité avec laquelle de nouvelles fonctionnalités peuvent être ajoutées à un logiciel
- **Compatibilité**
  - Facilité avec laquelle un logiciel peut être combiné avec d'autres
- **Efficacité**
  - Utilisation optimale des ressources matérielles
- **Convivialité**
  - Facilité d'apprentissage, de préparation des données, de corrections des erreurs d'utilisation et d'interprétation des résultats.

# Qualité d'un logiciel

Côté concepteur

- **Intégrité**
  - Aptitude d'un logiciel à protéger son code contre des accès non autorisés.
- **Réutilisabilité**
  - Aptitude d'un logiciel à être réutilisé, en tout ou en partie, pour d'autres applications
- **Vérifiabilité (robustesse)**
  - aptitude d'un logiciel à être testé (optimisation de la préparation et de la vérification des jeux d'essai)
- **Portabilité**
  - aptitude d'un logiciel à être transféré dans des environnements logiciels et matériels différents
- **Lisibilité**
- **Modularité**



# Unified Modeling Language - UML

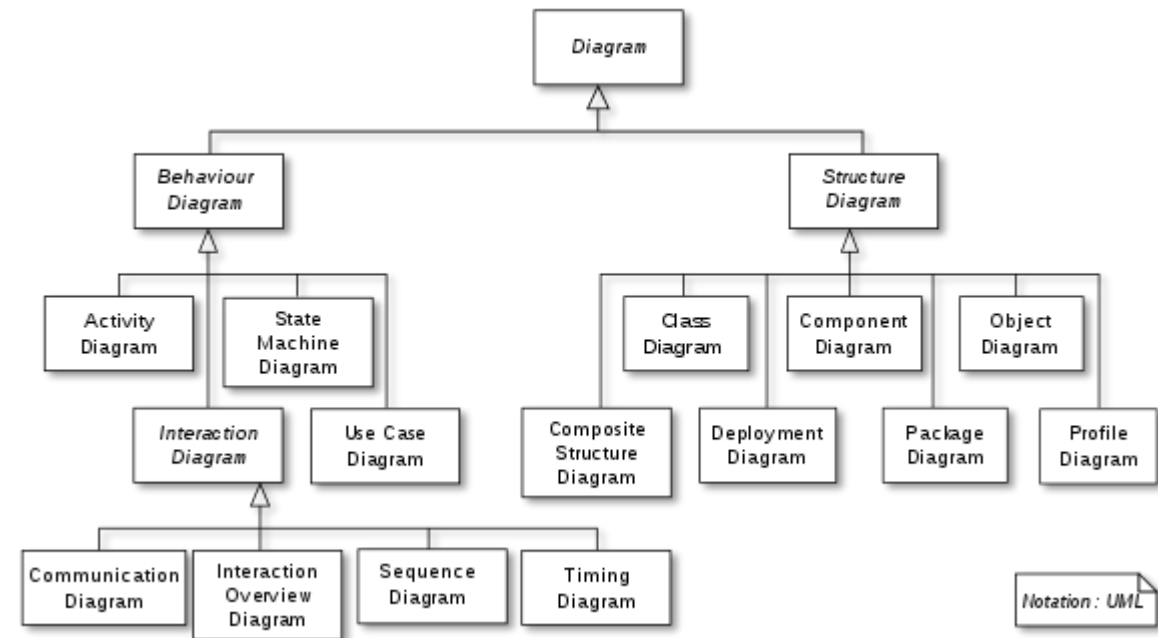
# UML

Unified Modeling  
Language.

- UML signifie **Unified Modeling Language** (langage unifié de modélisation).
- C'est un standard de notation que vous pouvez utiliser pour modéliser ou représenter de manière visuelle une application informatique.
- UML 1.0 a été normalisé en janvier 1997
- UML 2.0 a été adopté par l'OMG en juillet 2005.
- La dernière version de la spécification validée par l'OMG est UML 2.5.1 (2017)
- OMG pour Object Management Group est un consortium international à but non lucratif créé en 1989 pour standardiser et promouvoir le modèle objet.
- Utilisée dans les environnements orientés objet.

# UML

UML regroupe différents diagrammes.





# UML - Diagramme de Cas d'utilisation

## **UML : Diagramme de cas d'utilisation**

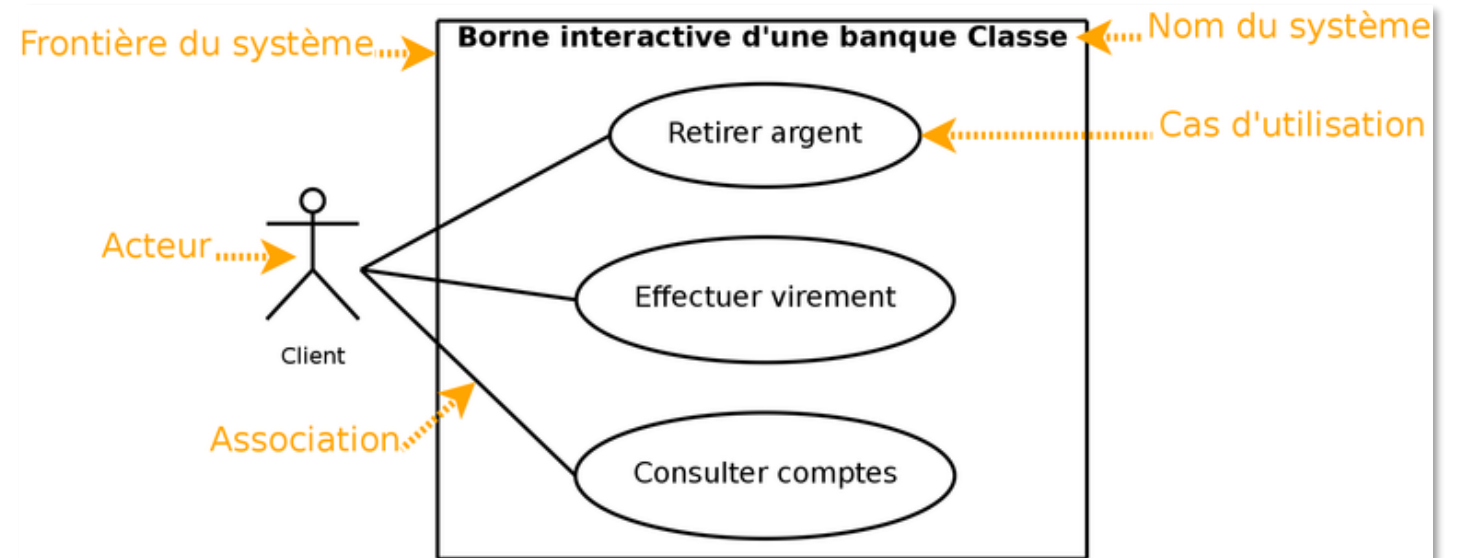
Concevoir un diagramme  
de cas d'utilisation

- Basé sur des diagrammes où des acteurs interagissent avec le système de l'extérieur ou de l'intérieur.
- A partir du cahier des charges ou de notre conception, il faut :
  - Identifier les acteurs de notre système (et uniquement eux)
  - Identifier les événements qui demande une action à notre système
  - Identifier les cas d'utilisations (use-cases)



# UML : Diagramme de cas d'utilisation

Représentation d'un  
diagramme de cas  
d'utilisation



# Les acteurs



## Acteurs

- Un acteur est une entité extérieure au système modélisé, et qui interagit directement avec lui.
- Un acteur correspond à un rôle, pas à une personne physique :
  - Une même personne physique peut être représentée par plusieurs acteurs si elle a plusieurs rôles.
  - Si plusieurs personnes jouent le même rôle vis-à-vis du système, elles seront représentées par un seul acteur.

## Acteurs non humains

- En plus des utilisateurs, les acteurs peuvent être :
  - Des logiciels déjà disponibles à intégrer dans le projet ;
  - Des systèmes informatiques externes au système mais qui interagissent avec lui ;
  - tout élément extérieur au système et avec lequel il interagit
- Pour identifier les acteurs, on se fonde sur les frontières du système.

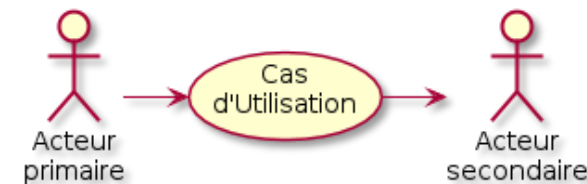
# Cas d'utilisation et associations

## Cas d'utilisation

- Un cas d'utilisation est un service rendu à un acteur : **c'est une fonctionnalité** de son point de vue.

## Associations entre cas et acteurs

- Les acteurs demandant des services aux systèmes, ils sont le plus souvent à l'initiative des échanges avec le système : **Acteurs primaires**
- Lorsqu'ils sont sollicités par le système, ils sont dits **Acteurs secondaires**.



# UML : Diagramme de cas d'utilisation

Relations entre les cas  
d'utilisation



- L'inclusion « include » : B est une partie obligatoire de A et on lit A inclut B (dans le sens de la flèche).



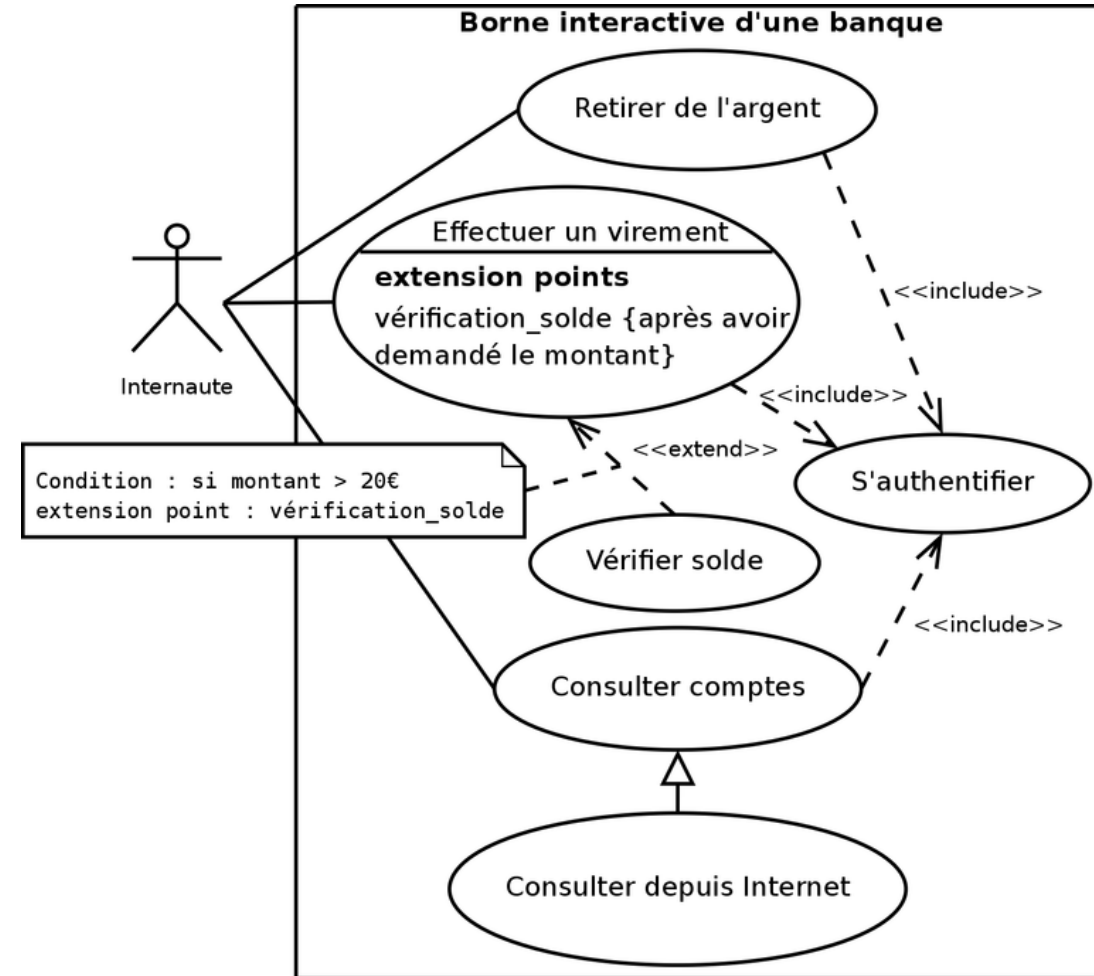
- L'extension « extended » : B est une partie optionnelle de A et on lit B étend A (dans le sens de la flèche).
  - Le point d'extension : Il est possible de préciser exactement à quel moment une extension est appelée.
  - La condition d'extension : Il est possible d'ajouter en note sous quelle condition l'extension doit se produire.



- L'héritage : Il permet de définir la spécialisation d'un cas d'utilisation
  - le cas A est une généralisation du cas du cas B et on lit B est une sorte de A.

# UML : Diagramme de cas d'utilisation

Exemple



# Exercices

2023

Conception : Domain-Drive-Design

30



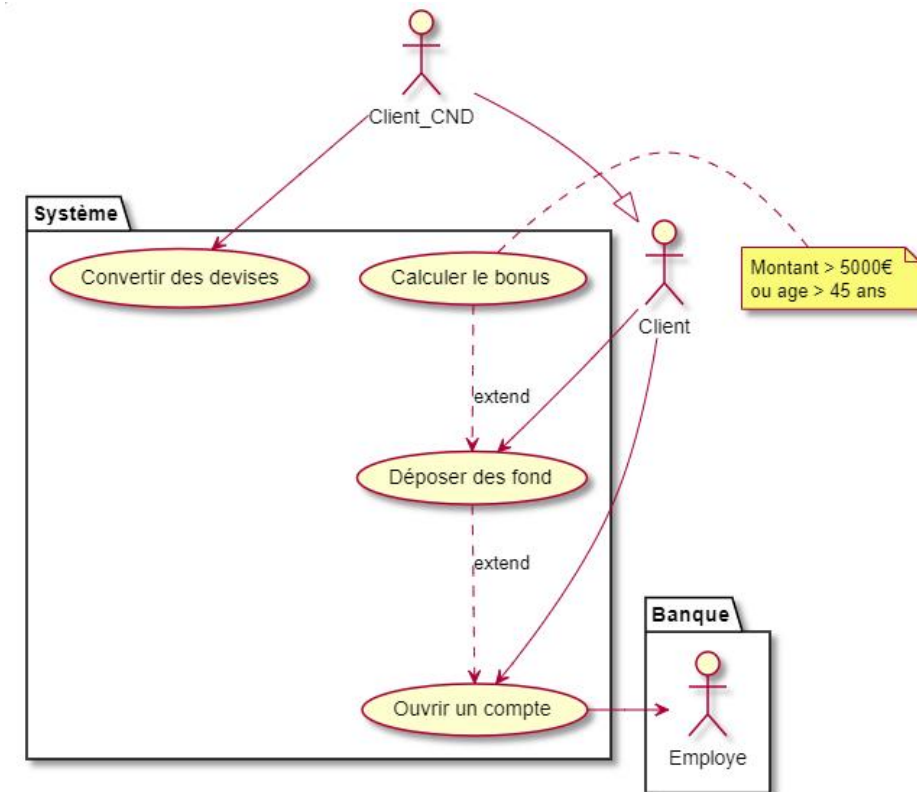
# UML

# Exercices

## Correction

### Exercice 1 : Ouverture d'un compte bancaire

- Modéliser la situation suivante à l'aide de cas d'utilisation :
  - Un client ordinaire peut se présenter à la banque X et demander l'ouverture d'un compte. Il remplit un formulaire et l'employé de la banque valide le formulaire pour ouvrir son compte.
  - Un client peut déposer des fonds, lorsque le montant est supérieur à 5000 € ou que son âge est supérieur à 45 ans, un bonus sera calculé et offert au client.
  - Un client de la CND (Compte Non-résident en Devises) peut également ouvrir un compte, déposer des fonds, mais il peut aussi convertir des devises.





# Exercices

## Correction

### Exercice 2 : Horloge

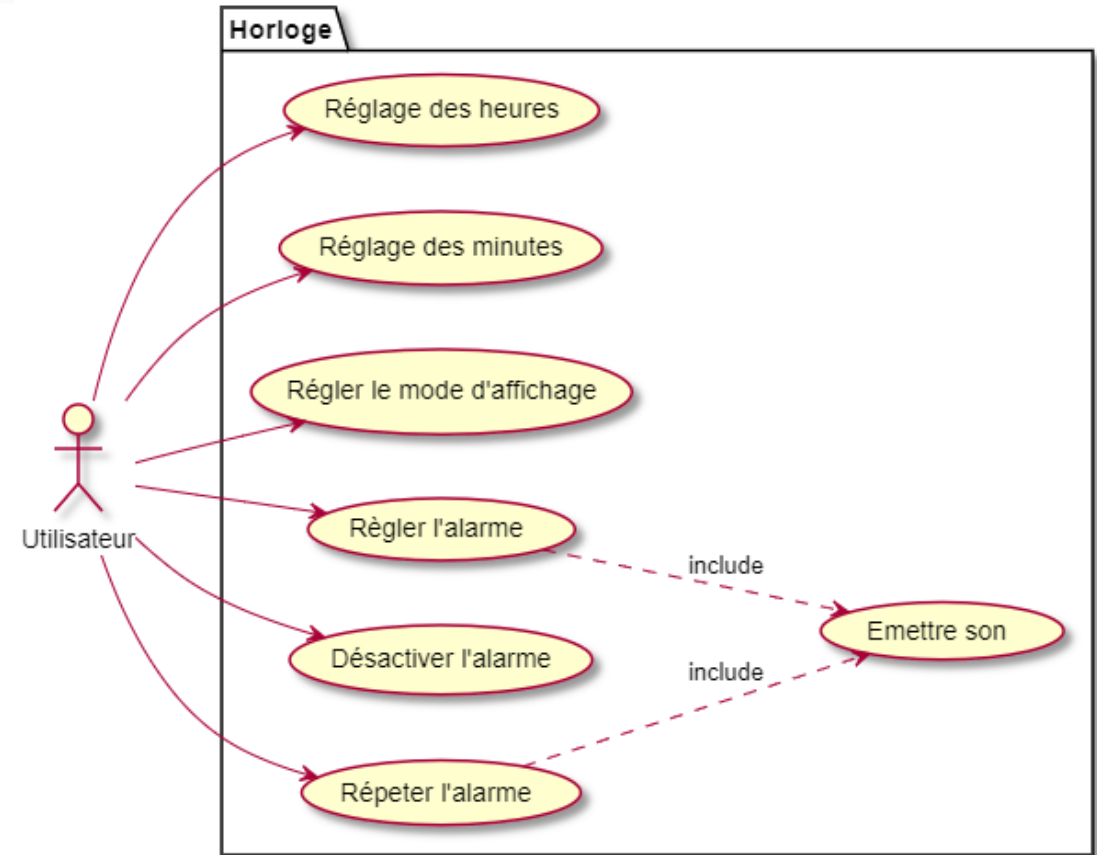
Supposons que nous voulions développer un logiciel pour un réveil.

Le réveil indique l'heure. À l'aide de boutons, l'utilisateur peut régler séparément les champs des heures et des minutes, et choisir entre un affichage sur 12 ou 24 heures.

Il est possible de régler une ou deux alarmes. Lorsqu'une alarme se déclenche, elle émet un son. L'utilisateur peut l'éteindre ou choisir de la « répéter ». Si l'utilisateur ne réagit pas du tout, l'alarme s'éteint d'elle-même au bout de 2 minutes.

La « sieste » consiste à éteindre le son, mais l'alarme se déclenche à nouveau après un délai de quelques minutes. Ce « temps d'arrêt » peut être réglé à l'avance.

- Identifiez l'exigence fonctionnelle pour l'horloge et modélisez-la à l'aide d'un diagramme de cas d'utilisation.







# UML - Diagramme de classes

# UML : diagramme de classes

Concevoir un diagramme  
de classes

- Dans un code orienté objet, on doit déterminer les classes à partir desquelles créer des objets.
- Etape 1 : identifiez les noms.
  - A partir des diagrammes d'un cas d'utilisation, on peut repérer les noms.
  - Nous pouvons en repérer deux sortes : **complexe** ou **simple**. Si le nom est complexe (c'est-à-dire qu'il est impossible à décrire en un mot ou deux), il s'agit probablement d'une classe. Si c'est un nom simple, il s'agira probablement d'un attribut de classe.
- Etape 2 : représentez vos noms en tant que classes ou attributs.

# Reprise de l'exemple

En reprenant notre exemple

- l'utilisateur fait une requête pour chercher un livre ;
- le programme affiche la page de recherche ;
- l'utilisateur saisit le nom de l'auteur ;
- le système vérifie que le nom de l'auteur existe ;
- l'utilisateur saisit le titre du livre ;
- l'utilisateur lance la requête ;
- le système exécute la recherche ;
- le programme affiche les résultats :
  - une série de livres qui correspondent,
  - une image de la couverture de chaque livre,
  - une courte biographie de l'auteur,
  - une image de l'auteur.

# Et si je me trompe ?



*Pas de panique. c'est entièrement normal.*

*Vous pouvez penser à ce que vous faites comme une sorte d'œuvre d'art, mais ce n'est pas une science exacte.*

- Très souvent, vous aurez l'impression qu'un élément est une classe (ou un attribut)
- mais, au moment de l'implémentation, vous vous rendrez compte que c'est plus simple ou plus compliqué que vous ne le pensiez !
- Dans ce cas-là, il faudra simplement adapter votre diagramme.

# UML : diagramme de classes

Représentation

- Un diagramme de classe est un type de diagramme UML qui décrit un système en visualisant les différents types d'objets au sein d'un système et les types de relations statiques qui existent entre eux.
- Il illustre également les opérations et les attributs des classes.
- Ils sont généralement utilisés pour explorer les concepts de domaine, comprendre les exigences logicielles et décrire les conceptions détaillées.

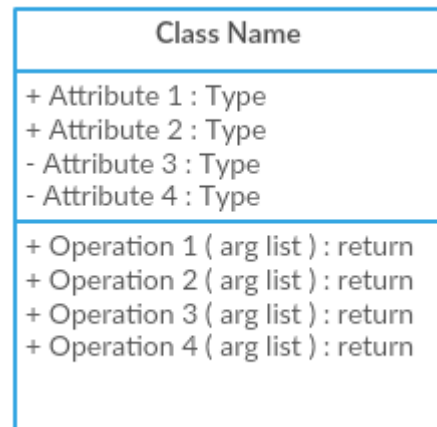
# Objets en UML

Objets du monde réel.

- Un objet est un conteneur, qui possède sa propre existence et incorpore des données et des mécanismes en rapport avec une chose tangible.
  - C'est le concept central de la programmation et de la conception orientée objet.
  - Syntaxe des objets : **nom : Type**
    - Le nom peut être omis, ou le type mais pas les deux.
    - Les : restent dans tous les cas de figure, même sans rien devant ou derrière.
- instance
  - Instanciation d'une classe : un objet est une instance d'une classe avec ses propres attributs et son propre état.
  - Instanciation d'une association : une association entre deux classes.  
C'est un lien concret entre deux objets particuliers.

# Classe en UML

Une classe définit un type d'objet.



- Une classe est composé d'un **nom**, d'**attributs** et d'**opérations**
- Une classe déclare donc des propriétés communes à un ensemble d'objets (des instances).
  - Les attributs correspondant à des variables associées aux objets de la classe
  - Les opérations correspondant à des opérations (fonctions/méthodes) associées aux objets de la classe.

```
{-, #, +, ~} nomAttribut : TypeAttribut {[multiplicité]} {=valeurInitiale}
```

- Syntaxe (entre accolades, les mentions optionnelles) :
  - {-, #, +, ~} : un symbole parmi les quatre pour définir la visibilité de l'attribut : privée, protégée, publique, paquetage
  - la multiplicité définit le nombre de valeurs dans une collection (tableau, liste)
  - Il n'y a pas de visibilité par défaut.

```
{-, #, +, ~} nomOpération ({LISTE_PARAMS}) {:valeurRetour}
```

# UML : Diagramme de classes

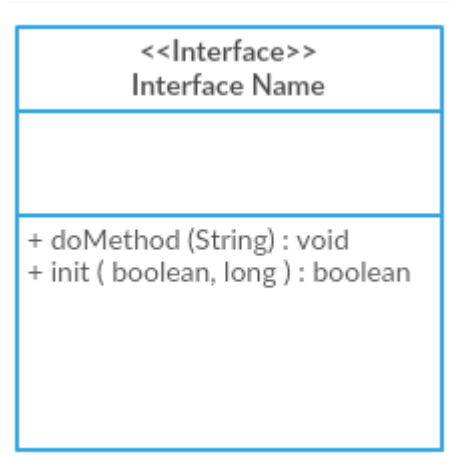
Les visibilités

Niveau de visibilité	Symbole de texte	Description
private	-	Seules les classes dans le même conteneur peuvent voir et utiliser les classes.
protected	#	Seules la classe elle-même et les classes filles (héritage) ont accès à cet attribut.
public	+	Toutes les autres classes ont accès à cet attribut.
package	~	Classe visible uniquement dans le package.

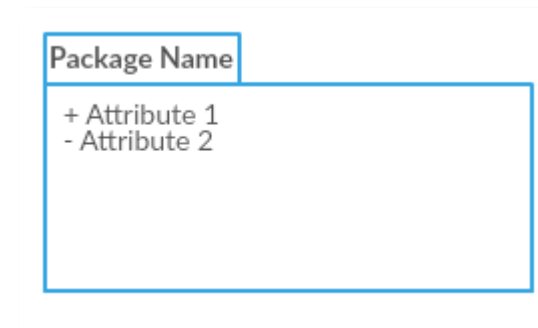


# UML : Diagramme de classes

## Une interface









## Un package



# Les relations entre classes

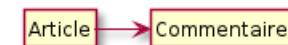
Il existe différentes relations entre les classes.

Class Diagram Relationship Type	Notation
Association	
Inheritance	
Realization/ Implementation	
Dependency	
Aggregation	
Composition	

# Association

Une association est une relation structurelle entre objets.

- Une association est souvent utilisée pour représenter les liens possibles entre objets de classes données
- Elle est représentée par un trait entre classes
- Elle est souvent dirigée par une flèche



- La flèche indique ici que la relation est **uni-directionnelle** : les objets de classe Article connaissent ceux de la classe Commentaire auxquels ils sont liés, mais pas l'inverse.



- Certaines associations sont **bi-directionnelles** mais comme une telle association est plus complexe à implémenter, **on préfère l'éviter autant que possible**.
  - L'absence de flèche indique ici que l'on peut accéder aux catégories à partir des articles qui leur sont liés, et inversement.

# Association : multiplicité

Les multiplicités permettent de contraindre le nombre d'objets intervenant dans les instanciations des associations.

On en place de chaque côté des associations.

- Une multiplicité d'un côté spécifie combien d'objets de la classe du côté considéré sont associés à un objet donné de la classe de l'autre côté.
- Syntaxe : min..max, où min et max sont des nombres représentant respectivement les nombre minimaux et maximaux d'objets concernés par l'association.



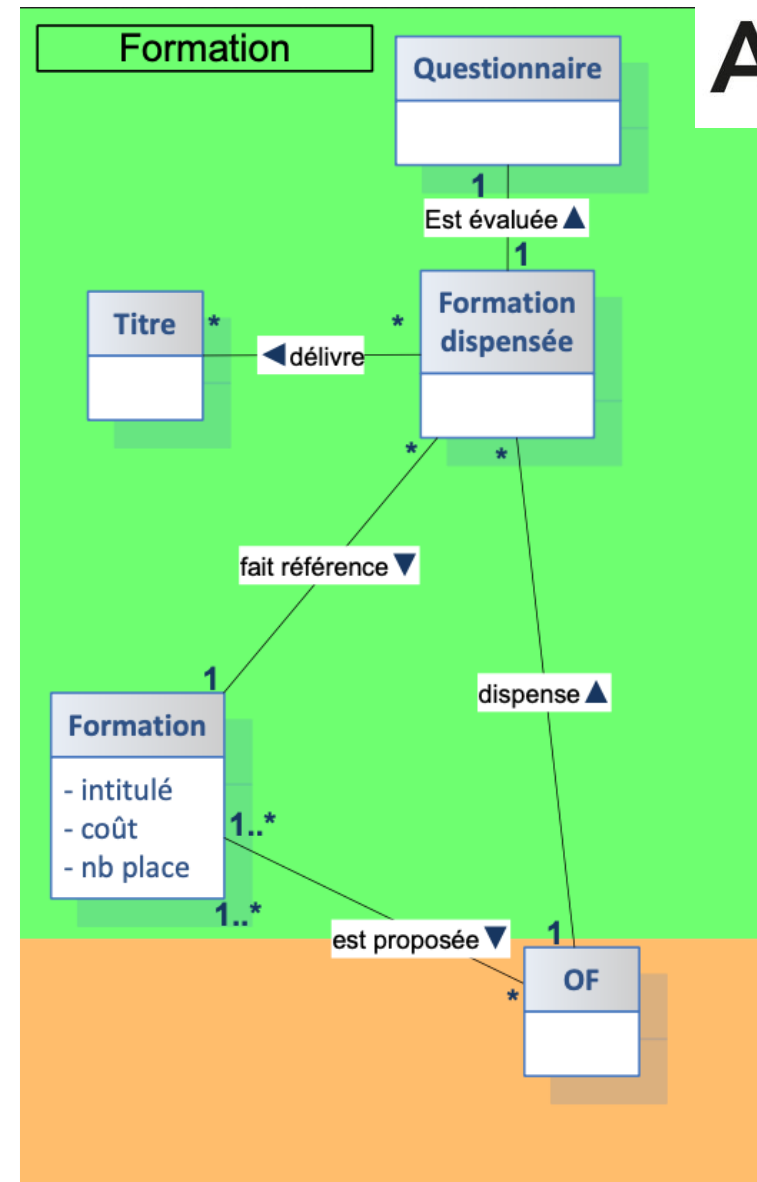
- Ici, le 1..5 s'interprète comme à un objet donné de la classe Article, on doit associer au minimum 1 objet de la classe Catégorie et on peut en associer au maximum 5.
  - \* à la place de max signifie plusieurs sans préciser de nombre.
  - n..n se note aussi n pour exactement n
  - 0..\* se note aussi \*

# Sens de lecture

A 1-----\* B

Un A peut être associé à plusieurs B et un B est associé à un seul A.

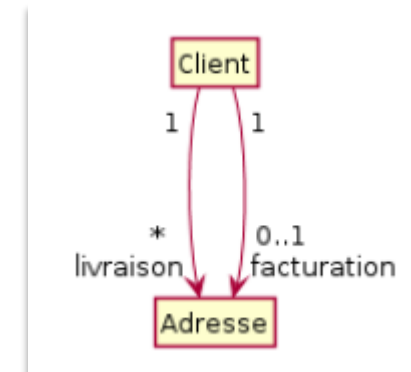
Sur ce diagramme de classe UML, on peut lire que la Formation est proposée par plusieurs Organismes de Formation (OF) grâce à la notation \* en Face de « OF ».



# Association : rôle

On peut donner à une classe un rôle dans une association.

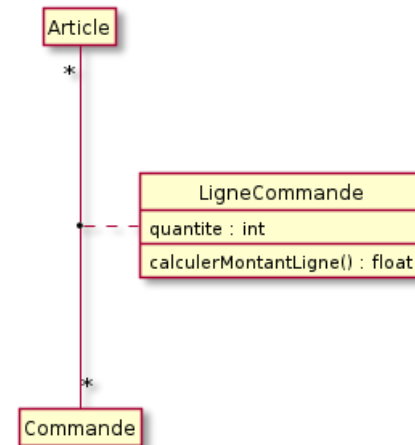
- C'est surtout utile quand plusieurs associations concernent les mêmes classes en qu'en conséquence, de mêmes objets peuvent être liés par des modalités différentes.



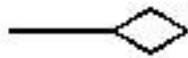
# Classe-association

Pour faire porter des informations par une association, on emploie une classe-association.

- Graphiquement, on la relie à l'association avec des pointillés.



# Agrégation



L'agrégation est une association entre deux objets qui décrit la relation "a une"

L'objet A a une instance de la classe B

L'agrégation est une association avec relation de subordination, représentée par un trait reliant les deux classes et dont l'origine se distingue de l'autre extrémité (la classe subordonnée) par un losange vide.

Une des classes regroupe d'autres classes.



Si l'objet de classe A est détruit, cela n'affectera pas les objets de la classe B. Ces objets existeront toujours.



# Composition



la composition est un type d'agrégation plus spécifique qui implique la propriété..

L'objet A **est composé** de l'objet B.

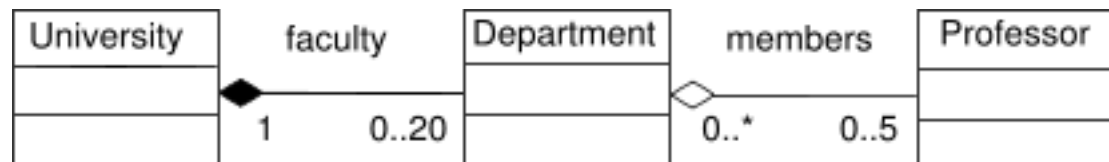
La composition est **une agrégation avec cycle de vie dépendant** : la classe composante est détruite lorsque la classe composée (ou classe composite) disparaît.

L'origine de cette association est représentée par un losange plein.

*Exemple : Une **Université est composée** de **plusieurs Départements**, et chaque département **est un agrégat** de **plusieurs Professeurs**.*

1. *La destruction de l'université implique la destruction des départements qui la composent.*
2. *Alors que la destruction d'un département n'implique pas la destruction des professeurs.*

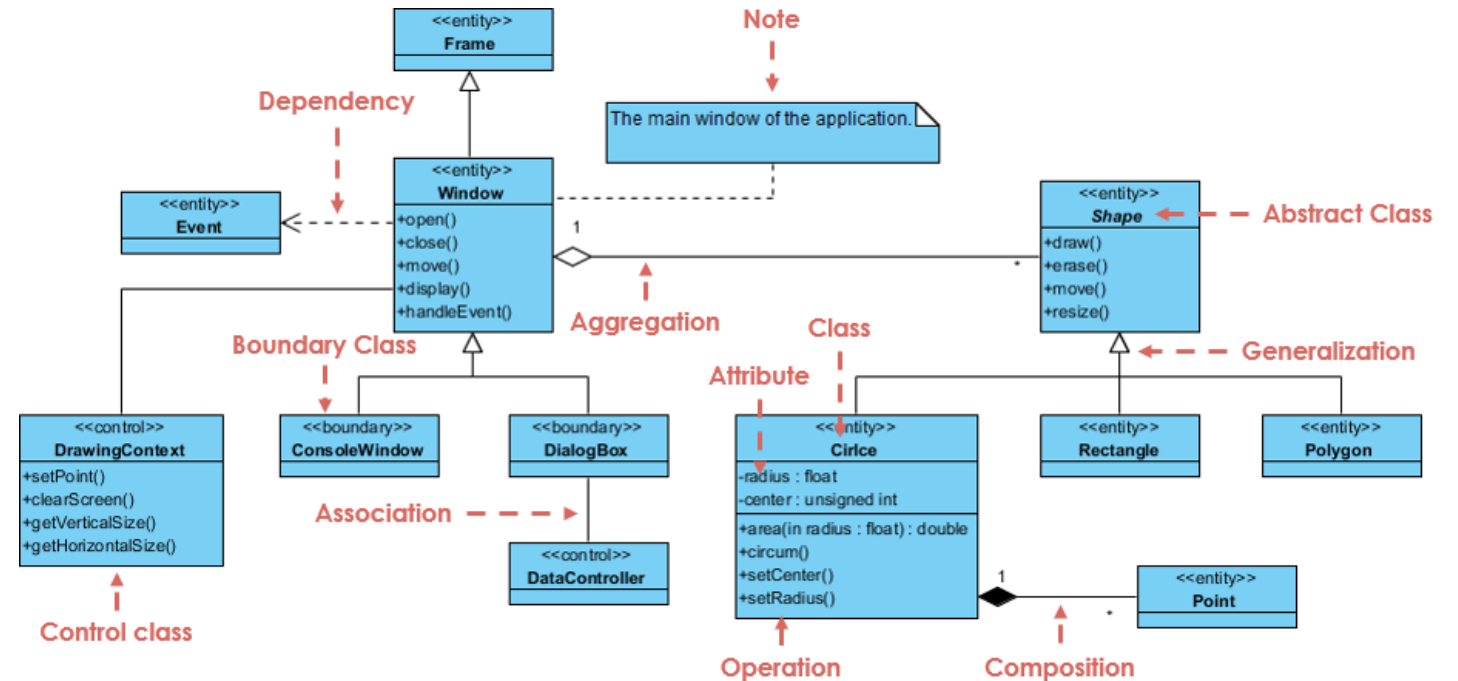
# Représentation de l'exemple



```
public class Professeur;  
  
public class Departement  
{  
    //...  
  
    // Agrégation  
    private Professeur enseignements[50];  
  
    //...  
}  
  
public Universite  
{  
    // ...  
  
    // Composition  
    private Departement facultes[20];  
  
    // ...  
}
```

# Illustration

exemple



# En cas d'ajout de fonctionnalités, de modifications

- Evidemment, en cas d'ajout, de modifications du système, il faut penser à mettre à jour nos diagrammes.
- On repart dans une phase d'analyse de l'existant en rapport à l'ajout ou à la correction.
- Cela permet de maintenir la conception à jour par rapport aux évolutions et aux corrections.
- Les diagrammes évoluent constamment. Cela fait partie du processus.

# Implémenter vos diagrammes

Après avoir réalisé nos différents diagrammes, nous passons à l'étape d'implémentation c'est-à-dire l'écriture de nos classes.

- Après avoir réalisé un modèle de domaine, utile et compréhensible pour toutes personnes liées au projet, l'étape suivante est d'implémenter.
  - **Déterminer les contextes délimités** : dans le système de la bibliothèque, nous avons un ensemble de fonctionnalités pour les bibliothécaires et un pour les usagers. Ce seront nos contextes délimités.
  - **Définir les entités pour représenter des objets uniques** : dans le système de la bibliothèque, les usagers et les livres seront des entités, qui seront persistées.
  - **Identifier les objets valeurs dans votre modèle**, éléments qui changent au fil du temps (valeurs calculées déduites par d'autres valeurs) et qui ne sont pas uniques. Ces données ne sont pas persistées.
  - **Identifier les objets d'agrégation**, éléments qui vont nous permettent de conserver une information en liant des entités entre-elles (liste d'emprunts, liste d'usagers etc...)

# Exercices

2023

Conception : Domain-Drive-Design

54



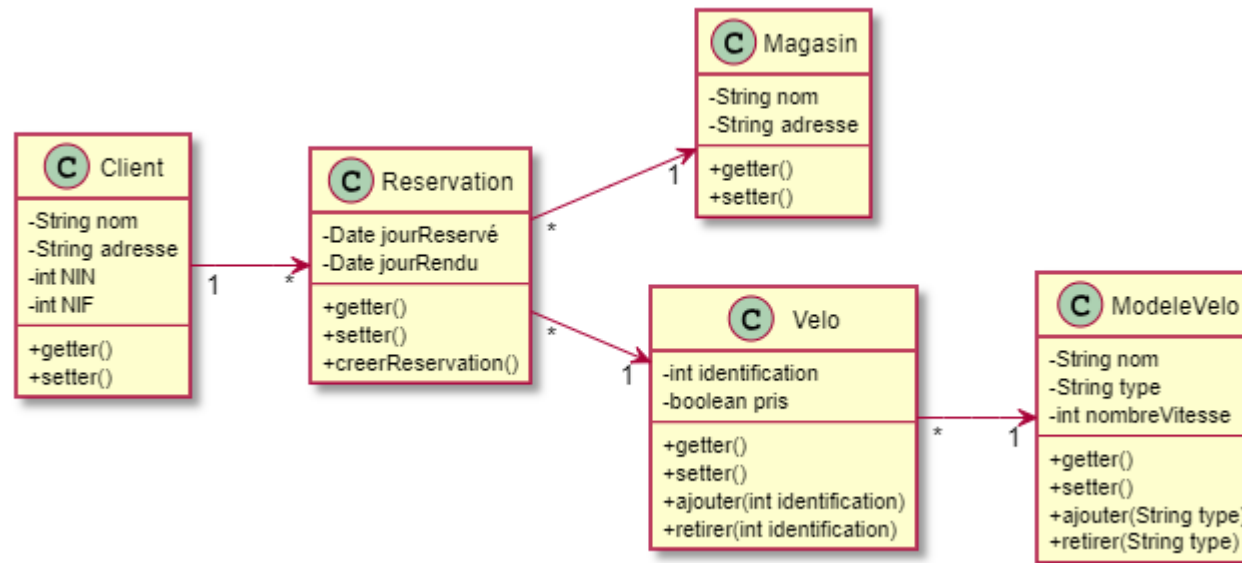
# UML

# Location de vélos

Une société de location de bicyclettes souhaite créer un système d'information qui lui permette de stocker les données relatives à toutes ses réservations et locations.

- Le système doit répondre aux exigences suivantes :
  - Il doit être possible de stocker le numéro d'identification national (NIN), le numéro d'identification fiscale (NIF), le nom et l'adresse de chaque client. Le NIN et le TIN doivent être différents pour chaque client et tous les clients doivent avoir au moins un NIF et un nom.
  - La base de données doit également contenir des informations sur les modèles de vélos qui peuvent être loués – chaque modèle a un nom unique, un type (qui ne peut être que route, montagne, bmx ou hybride) et le nombre de vitesses.
  - Chaque vélo a un numéro d'identification unique et un modèle.
  - L'entreprise dispose de plusieurs magasins où les vélos peuvent être retirés et retournés. Chacun de ces magasins est identifié par un nom unique et possède une adresse (les deux sont obligatoires).
  - Lorsqu'une réservation est effectuée, les données suivantes doivent être connues : quel client a effectué la réservation, quand il viendra chercher le vélo (jour), quel modèle de vélo il souhaite et où il viendra chercher le vélo (magasin).
  - Lorsqu'un vélo est retiré, le vélo qui a été retiré doit être stocké dans la base de données.
  - Lorsqu'un vélo est restitué, la date de restitution doit être enregistrée dans la base de données.

# Correction







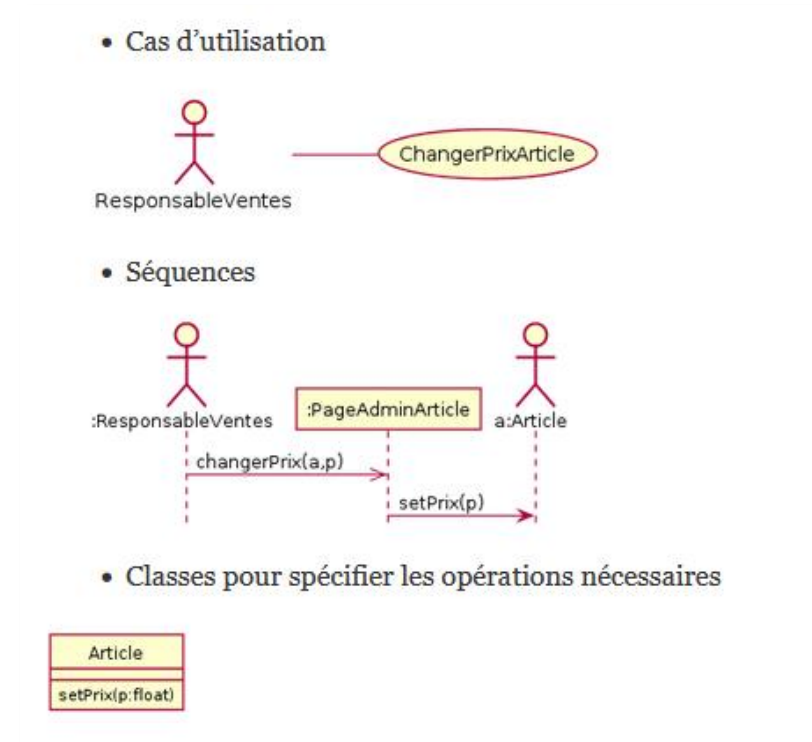
# UML - Diagramme de séquences

# Diagramme de séquence

Ils permettent de décrire COMMENT les éléments du système interagissent entre eux et avec les acteurs :

- Les objets au cœur d'un système interagissent en s'échangeant des messages.
- Les acteurs interagissent avec le système au moyen d'IHM (Interfaces Homme-Machine).

- Interaction
  - Pour être complètement spécifiée, une interaction doit être décrite dans plusieurs diagrammes UML



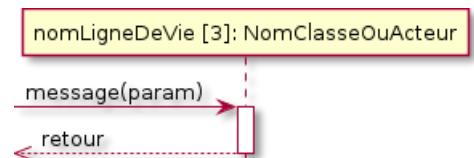
# Les éléments du diagramme

## Ligne de vie

- Une ligne de vie représente un participant à une interaction (objet ou acteur).

```
nomLigneDeVie {[selecteur]}: NomClasseOuActeur
```

- Une ligne de vie est une instance, donc il y a nécessairement les deux points (:) dans son libellé.



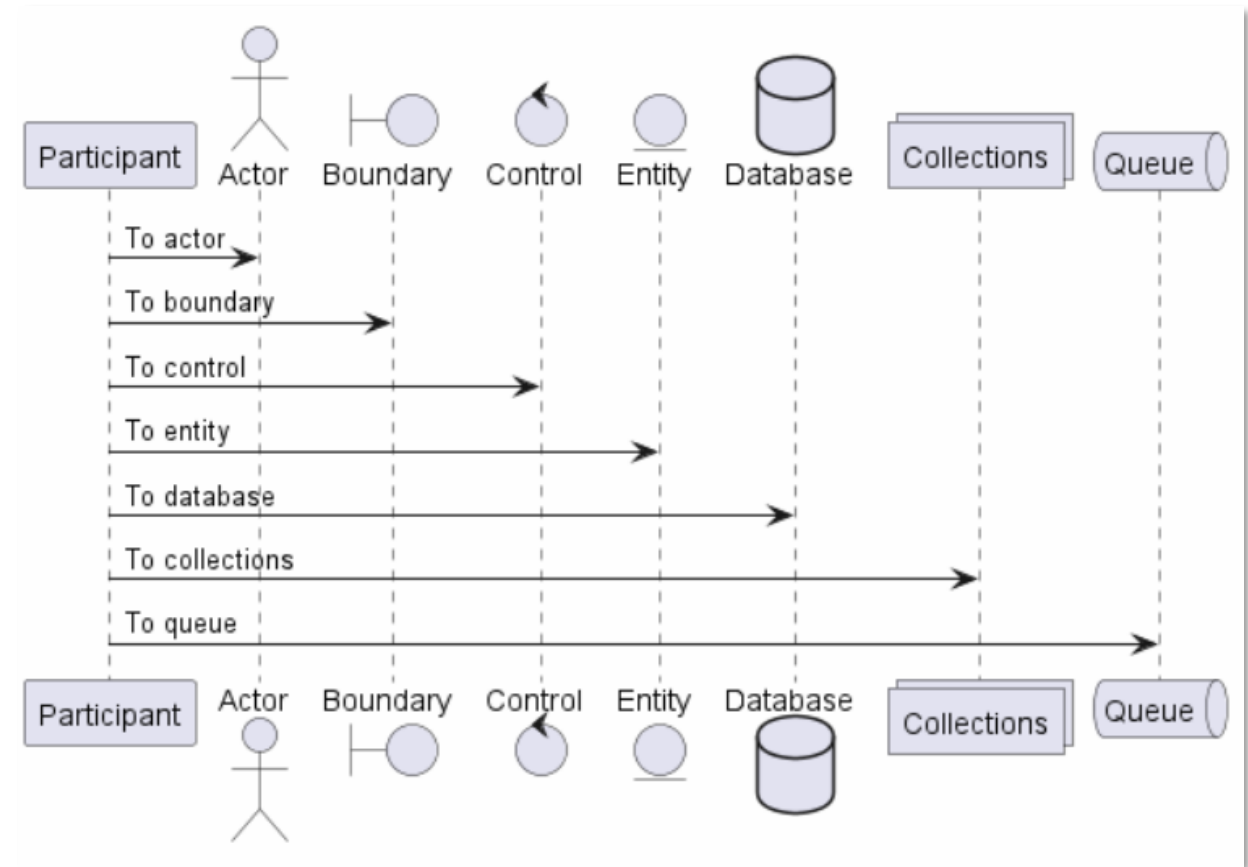
## Messages

- Les principales informations contenues dans un diagramme de séquence sont les messages échangés entre les lignes de vie :
  - Ils sont représentés par des flèches
  - Ils sont présentés du haut vers le bas le long des lignes de vie, dans un ordre chronologique
- Un message définit une communication particulière entre des lignes de vie (objets ou acteurs).

```
nomSignalOuOperation (LISTE_PARAMS)
```

# Type de Participants

- Actor
- database
- Boundary
- collections
- Control
- queue
- entity



# Type de Messages

Plusieurs types de messages existent, dont les plus courants :

- l'envoi d'un signal
- l'invocation d'une opération (appel de méthode)
- la création ou la destruction d'un objet.

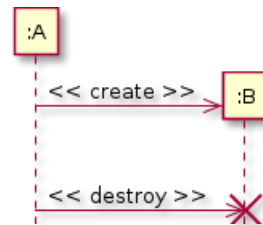
- Messages synchrones et asynchrones
  - Un message synchrone (une opération le plus souvent) bloque l'expéditeur jusqu'à la réponse du destinataire. Le flot de contrôle passe de l'émetteur au récepteur.
    - Si un objet A envoie un message synchrone à un objet B, A reste bloqué tant que B n'a pas terminé.
    - On peut associer aux messages d'appel de méthode un message de retour (en pointillés) marquant la reprise du contrôle par l'objet émetteur du message synchrone.
  - Un message asynchrone n'est pas bloquant pour l'expéditeur. Le message envoyé peut être pris en compte par le récepteur à tout moment ou ignoré.



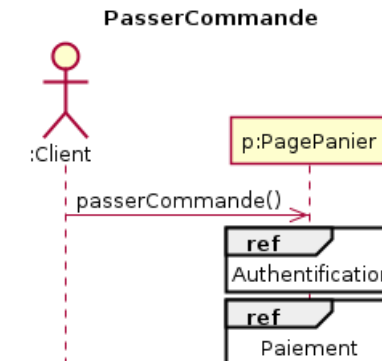
# Messages

## Création et destruction d'objet

- **Création** : message asynchrone stéréotypé << create >> pointant vers le rectangle en tête de la ligne de vie
- **Destruction** : message asynchrone stéréotypé << destroy >> précédant une croix sur la ligne de vie



## référence à un autre diagramme de séquence



# Fragment

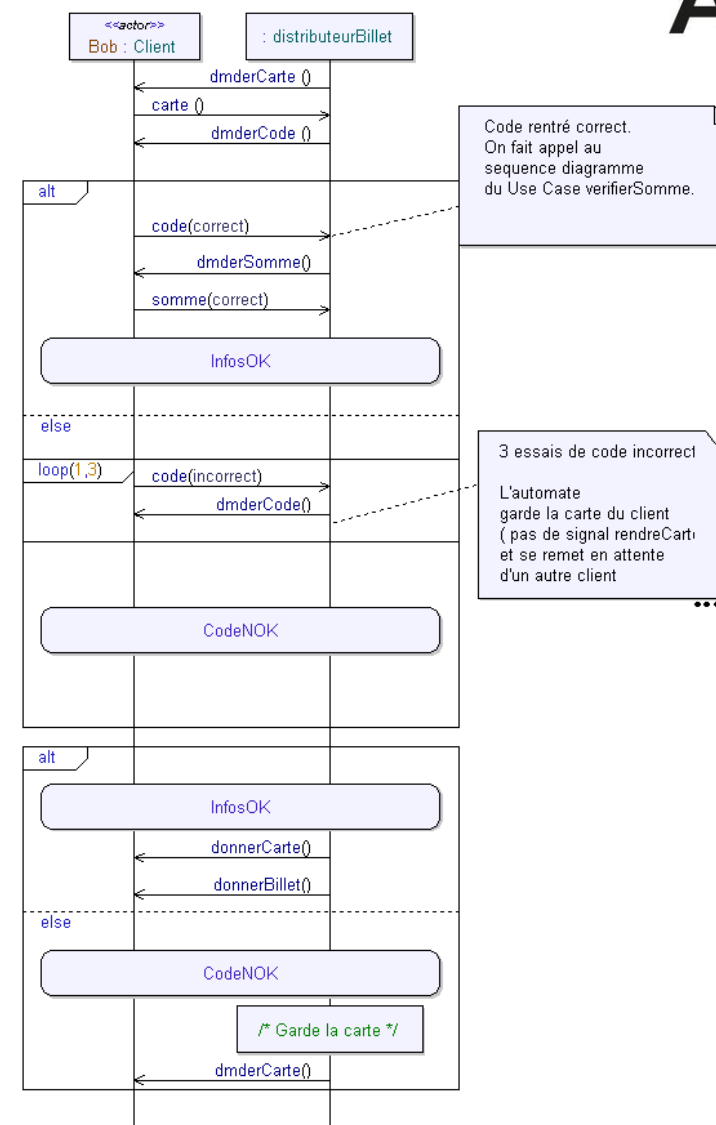
Un fragment combiné permet de décomposer une interaction complexe en fragments suffisamment simples pour être compris.

- Recombiner les fragments restitue la complexité.
- représentation complète de processus avec un langage simple (ex : processus parallèles).

- Un fragment combiné se représente de la même façon qu'une interaction. Il est représenté un rectangle dont le coin supérieur gauche contient un pentagone.
- Dans le pentagone figure le type de la combinaison (appelé *opérateur d'interaction*).
  - **opt** (facultatif\*) : Contient une séquence qui peut ou non se produire.
  - **alt** : Contient une liste des fragments dans lesquels se trouvent d'autres séquences de messages. Une seule séquence peut se produire à la fois.
  - **loop** : Le fragment est répété un certain nombre de fois.
  - **break** : Si ce fragment est exécuté, le reste de la séquence est abandonné.
  - **par** (parallèle) : Les événements des fragments peuvent être entrelacés.
  - **critical** : Utilisé dans un fragment par ou seq. Indique que les messages de fragment ne doivent pas être entrelacés avec d'autres messages.
  - **seq** : Il existe au moins deux fragments d'opérande. Les messages impliquant la même ligne de vie doivent se produire dans l'ordre des fragments. Lorsqu'ils n'impliquent pas les mêmes lignes de vie, les messages des différents fragments peuvent être entrelacés en parallèle.
  - **strict** : Il existe au moins deux fragments d'opérande. Les fragments doivent se produire dans l'ordre donné.

# Illustration

Voici un exemple





# Exercise

2023

Conception : Domain-Drive-Design

65



# UML

# Demande de licence

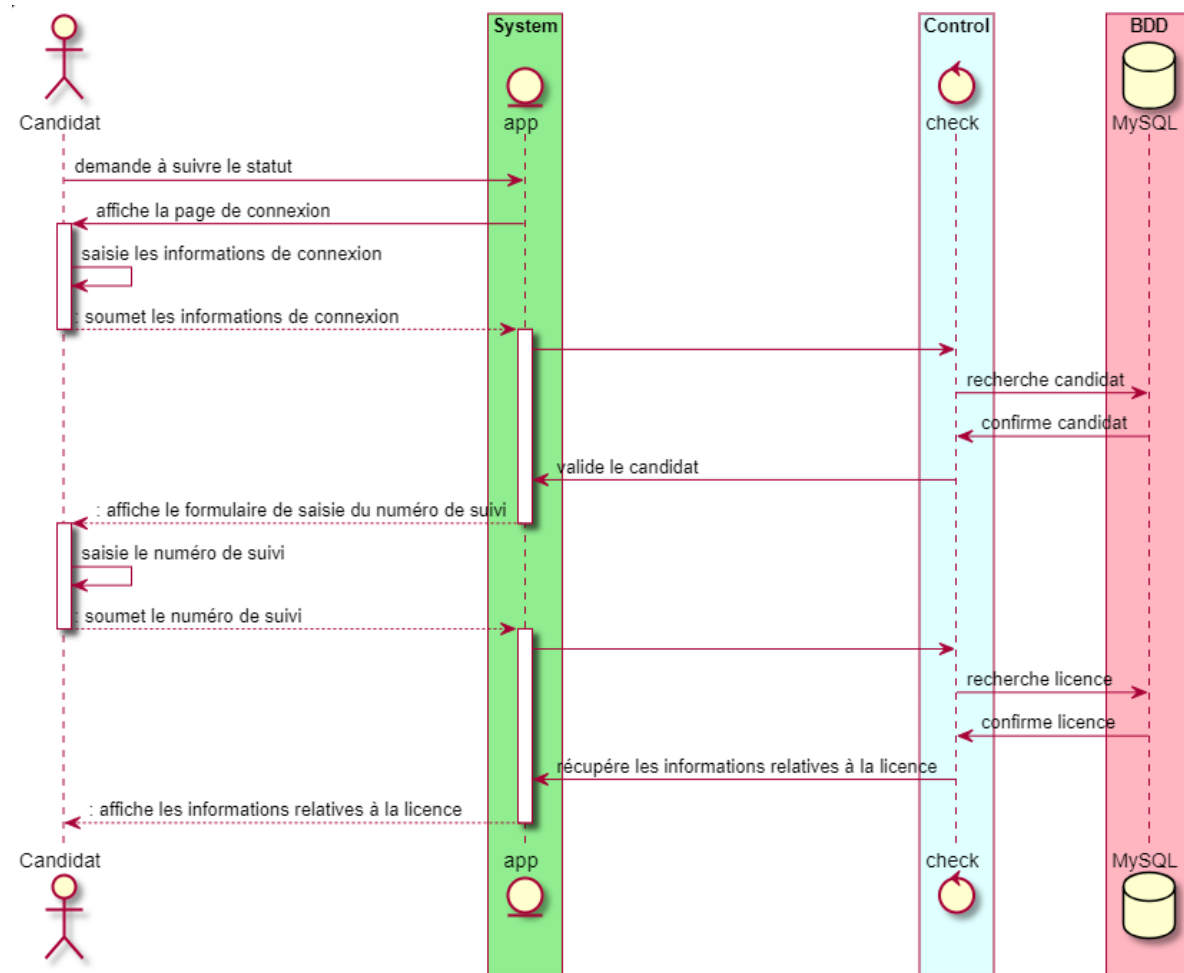
Un candidat suit l'état d'une demande de licence et le système affiche les informations relatives à la licence.

La procédure est la suivante:

1. Le candidat demande à suivre le statut d'une demande de licence.
2. Le système affiche le formulaire de connexion
3. Le candidat saisit les informations de connexion
4. Le candidat soumet les informations de connexion
5. Le système valide le candidat
6. Le système affiche le formulaire de saisie du numéro de suivi
7. Le candidat saisit le numéro de suivi
8. Le candidat soumet le numéro de suivi
9. Le système récupère les informations relatives à la licence
10. Le système affiche les informations relatives à la licence

# Correction

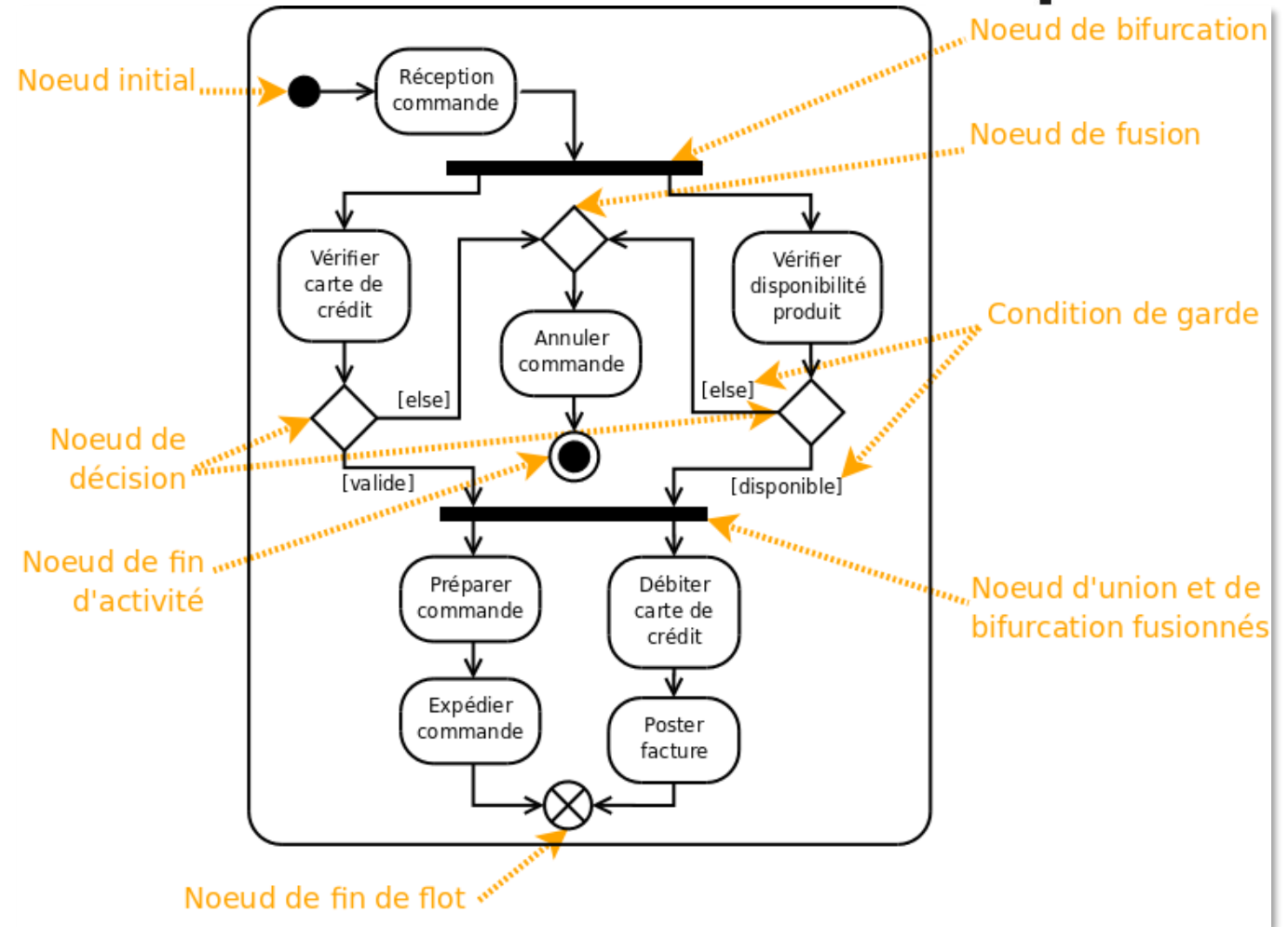
Ici, j'ai ajouté control et BDD qui n'était pas demandé dans l'énoncé.





# UML - Diagramme d'activités

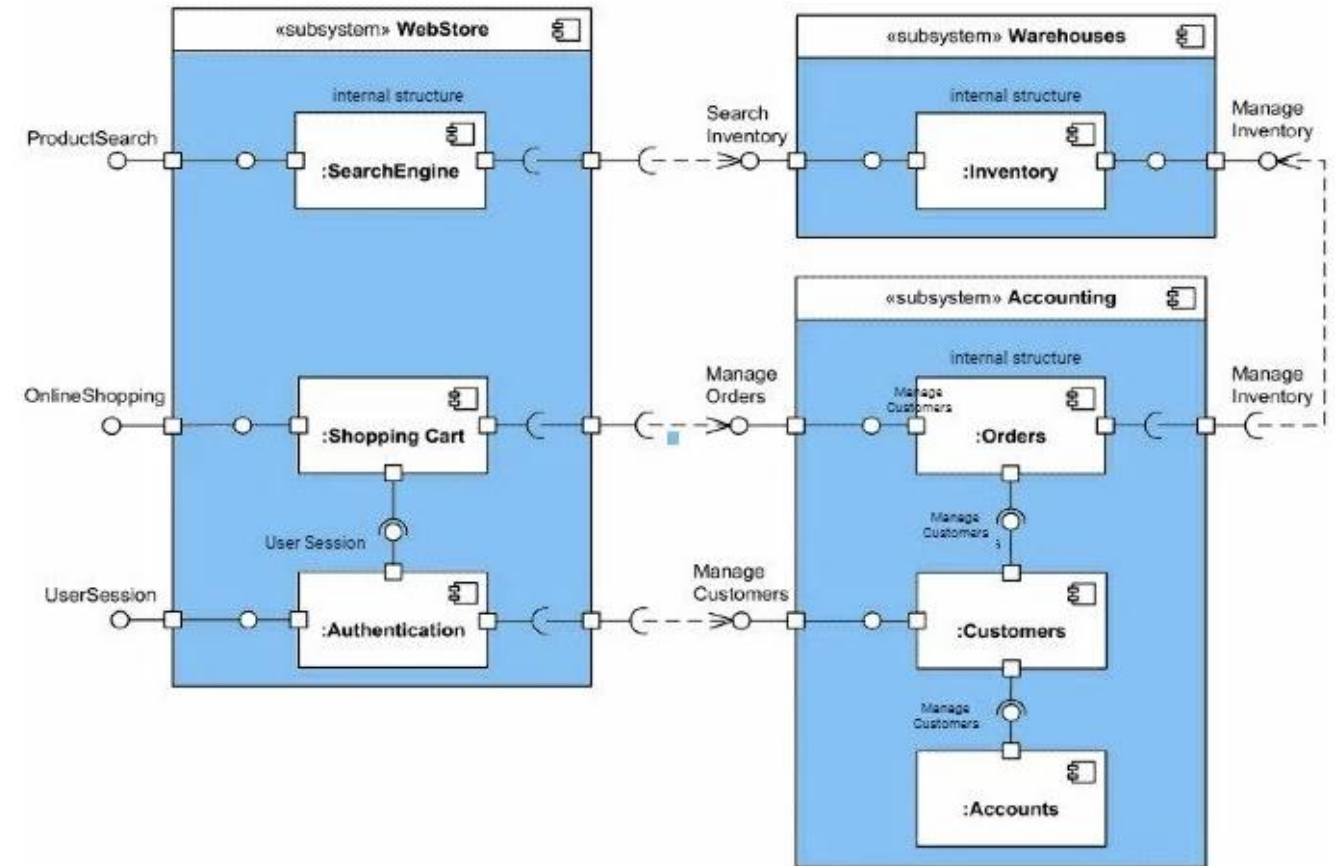
# Diagramme d'activités





# UML - Diagramme des composants

# Diagramme des composants

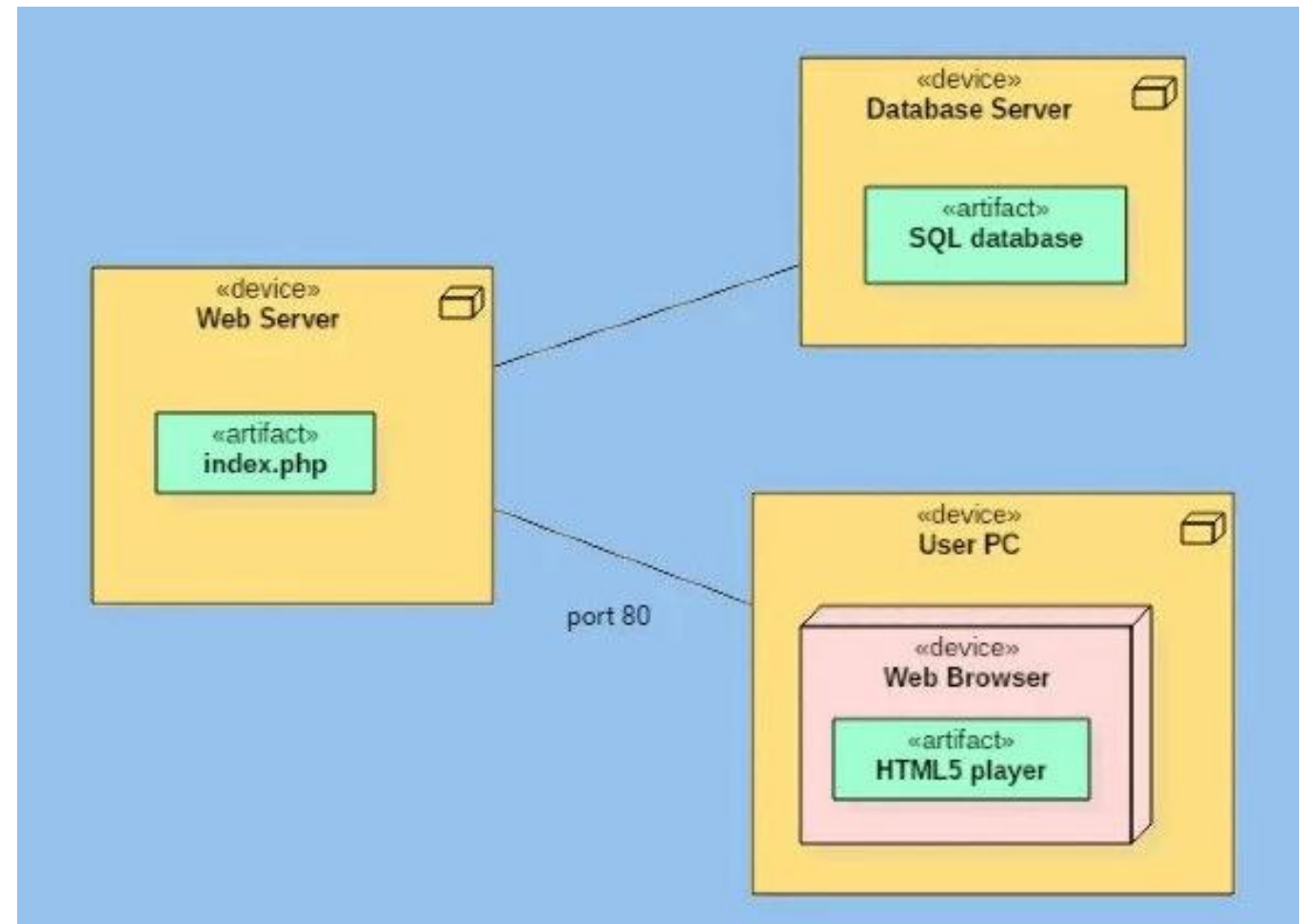




# UML - Diagramme de déploiement



# Diagramme de déploiement





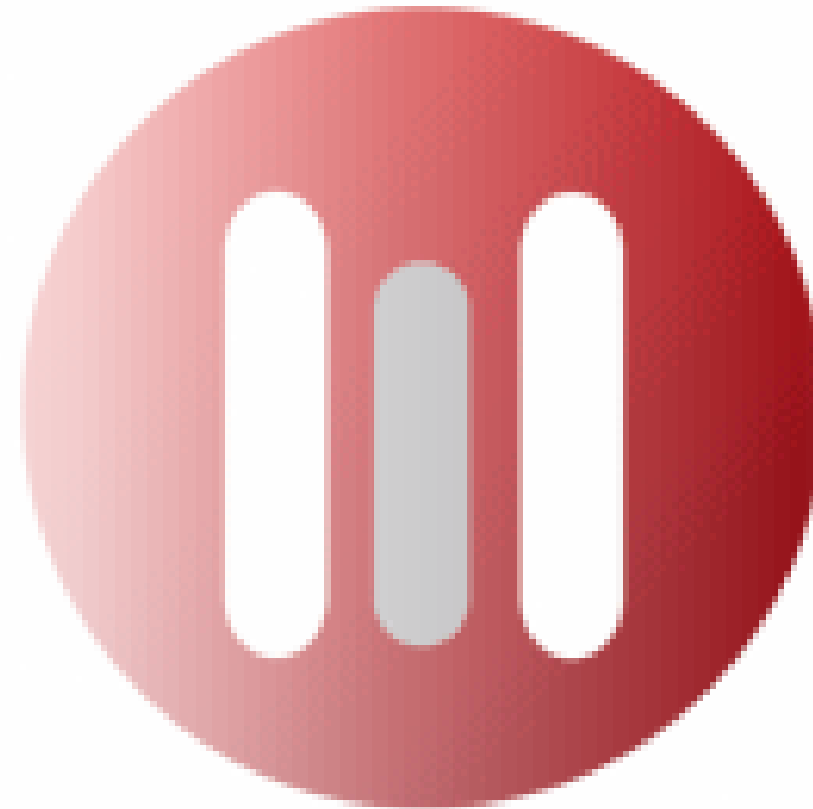


# UML - outils

# MODELIO

---

- Logiciel OpenSource
- Gratuit 
- En français 
- UI proche d'Eclipse





<https://github.com/ModelioOpenSource/Modelio/releases/tag/v5.3.1>

MODELIO SOFT  
SOFTEAM group

# PowerDesigner





---

- Logiciel de modélisation de données
- Outils tout-en-un de modélisation d'entreprise
- Professionnel 
- Payant 

<https://powerdesigner.biz/FR/>

# PlantUML


Outil open-source pour dessiner des diagrammes UML.

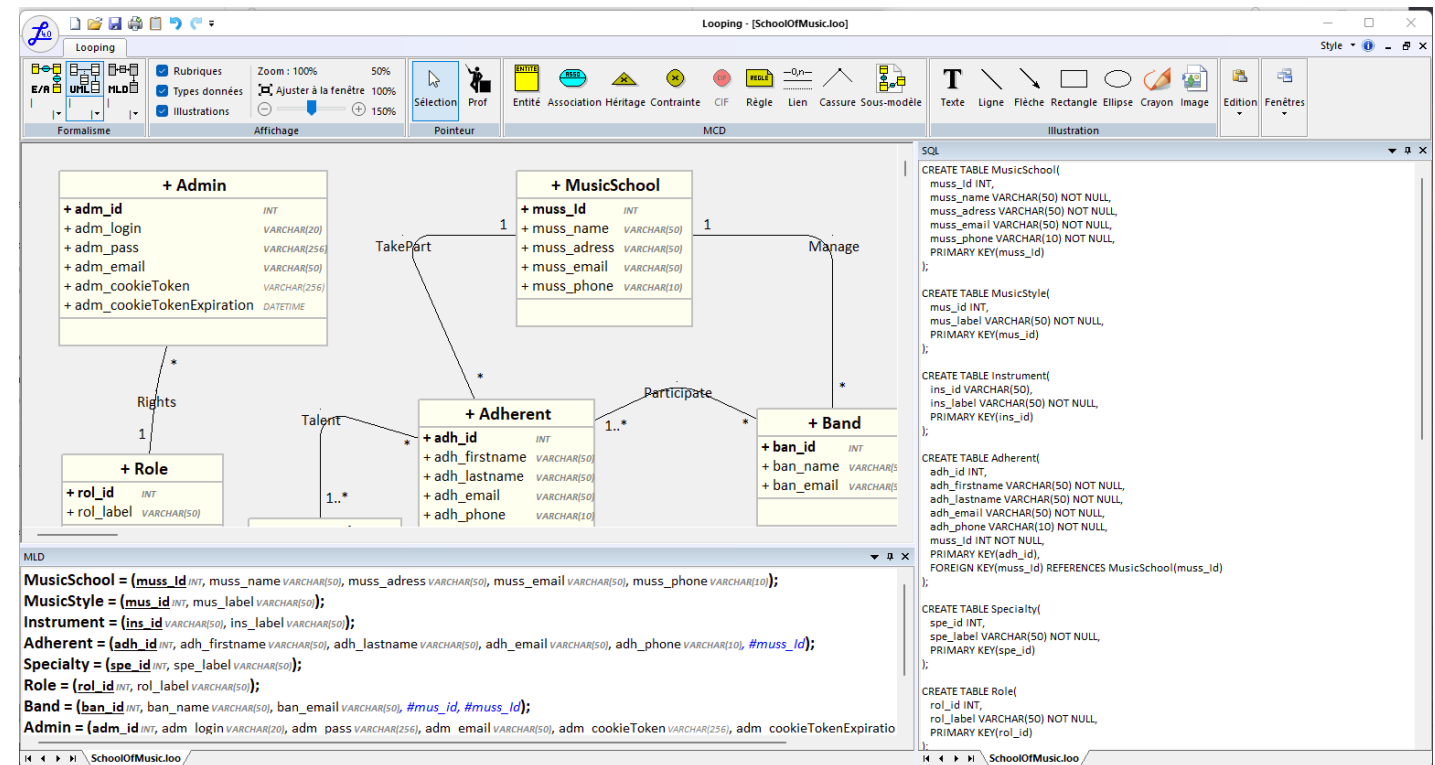
- Gratuit 
- Intégrable dans les IDE 
- Documentation en français 
- Syntaxe à prendre en main 



<https://plantuml.com/fr/>

# Looping

Logiciel gratuit simple mais il ne fait pas vraiment de l'UML. Il se base sur un MCD pour produire un UML 



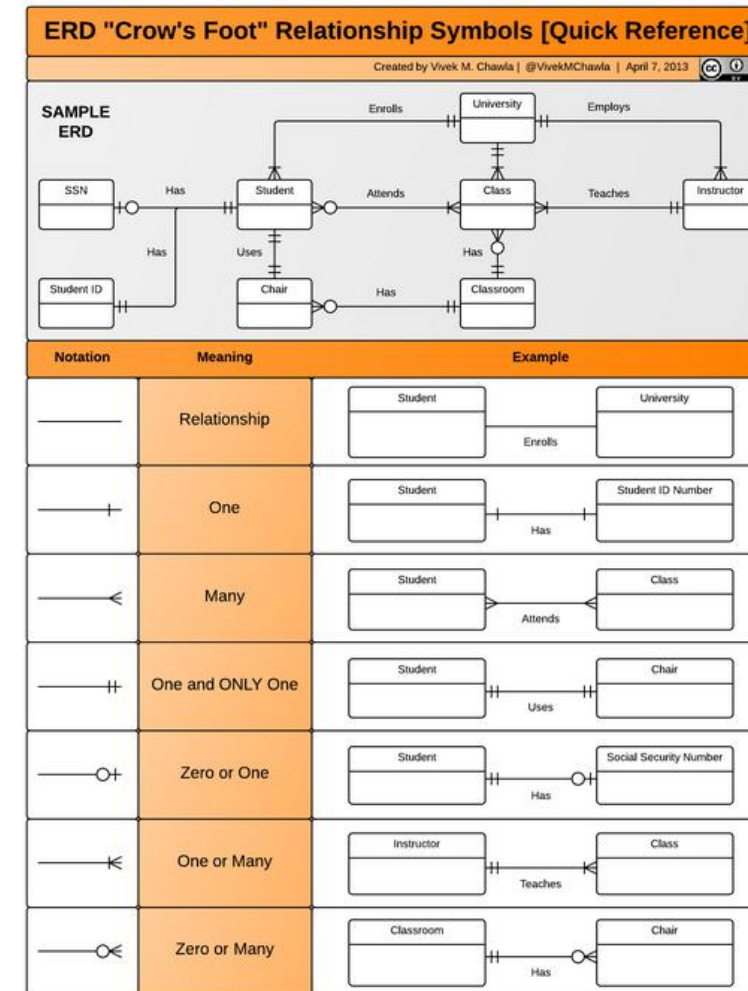


# Autres types de Notation ERD (*pour infos*)



# ERD Crow's Foot

- ERD Crow's Foot (pattes de corbeau) se lit comme UML (cardinalité en face de l'entité concernée).
- Très utilisée dans les milieux professionnels et outils comme MySQL Workbench, Oracle, etc.

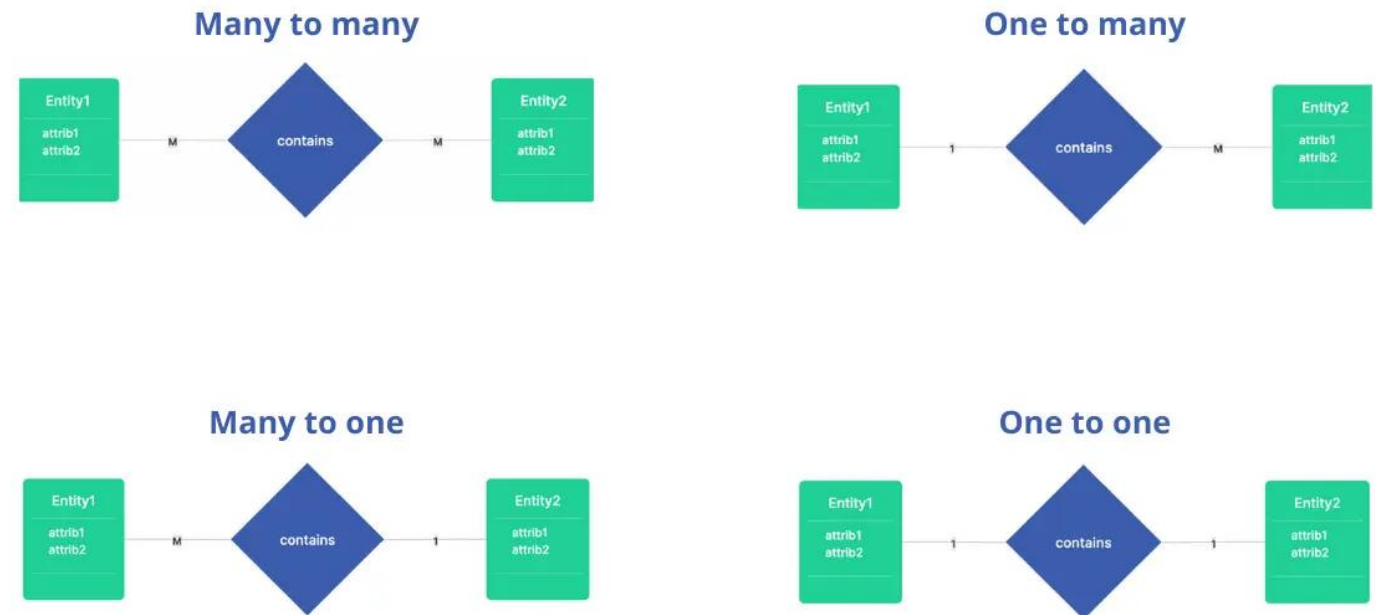




# ERD Chen Notation

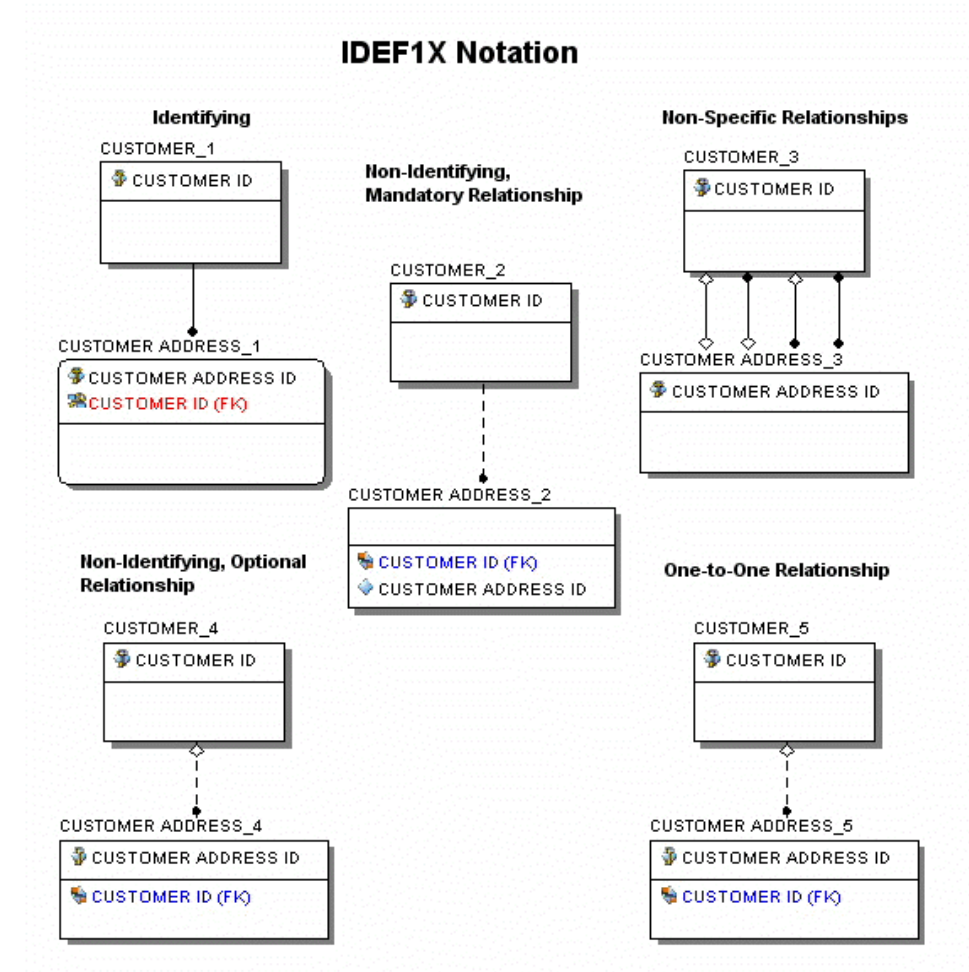
- Utilisée en contexte académique, pour l'enseignement.
- ERD de type Chen se lit comme Merise.

## Chen notation: Cardinality



# ERD IDEF1X Notation

- Moins connue du grand public
- Très utilisée dans l'ingénierie des systèmes et les grandes organisations, notamment gouvernementales ou industrielles.



# MERCI !



Jérôme BOEBION  
Concepteur Développeur d'Applications  
Version 1 - révision 2024

