UML-2

Associations entre classes

Diagrammes de classes : 2

Diagramme de Classes Métier et BD

Bertrand LIAUDET

USAGE du document :

Les mots / phrases importants sont en bleu

Les exercices sont en vert

SOMMAIRE

SOMMAIRE 1

1 : ASSOCIATIONS ENTRE LES CLASSES 4

0 – Diagramme de classes métier – méthodologie (12 slides) 4

Notions du chapitre (slide 1) 4
Notion de classe métier (slide 2) 5
Exemple : un site de vente (slide 3) 6
DCM d'un site de vente : première approche (slide 4) 7
Beaucoup de sigles ! (slide 5) 8
Modèle de BD : MLD du DCM (slide 6) 9
Convention de nommage : pluriel pour les tables, singulier pour les classes (slide 7) 10
Modèle du code de la BD : MPD du MLD (slide 8) 11
Réalisation des diagrammes (slide 9) 12
Génération automatique de code ou de schéma (slide 10) 13
Simulation du remplissage des tables (slide 11) 14
Code SQL pour créer les tables et les données (slide 12) 15

1 – Association 1 à plusieurs : 1-----* : « one-to-many » (16 slides) 16

Notions du chapitre (slide 13) 16
Exemple de diagramme de classes avec association 1--* (slide 13 bis) 17
Complément du site de vente : des photos pour les produits (slide 14) 18
Mettre des photos dans les produits (slide 15) 19
UML : ajouter une association dans le DCM (slide 16) 20
Cardinalités d'une association et justification des cardinalités (slide 17) (cf. slide 32) 21
Variante UML : notion de composition (slide 18) 22
Variante du modèle : passer les catégories dans une classe (slide 19) 23
Variantes UML (slide 20) 23
Quel modèle réaliser ? (slide 25) 28
MLD : règles de passage du DCM au MLD (slide 26) (cf slide 35) 29
Graphe des tables (slide 27) 30
Génération automatique de code : chatGPT (slide 28) 31
Exercices 1 -- * (slide 29) 32

2 – Association plusieurs à plusieurs : *-----* : « many-to-many » (11 slides) 33
Notions du chapitre (slide 30) 33
Exemple de diagramme de classes avec association *--* (slide 30 bis) 33
Compléments du site de vente (slide 31) 34
Justification des cardinalités (slide 33) (cf. slide 17) 36
Version avec les attributs des associations dans les classes (slide 34) 37
Variante technique : 10 plutôt que * : à éviter dans un DCM ! (slide 35) 38
MLD : règles de passage du DCM au MLD (slide 36) (cf slide 26) 39
Graphe des tables (slide 39) 42
Génération de code SQL : chatGPT (slide 40) 43
Exercices * -- * sans attributs (slide 41) 44
3 – Association plusieurs à plusieurs : *-----* avec attributs (14 slides) 45
Notions du chapitre (slide 42) 45
Exemple de diagramme de classes avec association 1--* et classe-association (slide 42 bis)45
Compléments du site de vente (slide 43) 46
Justification des cardinalités (slide 45) 48
Le problème des attributs de l'association : la classe-association (slide 46) 49
Modèle de BD : MLD du diagramme des classes métier (slide 48) 51
Graphe des tables (slide 51) 54
Génération de code SQL : chatGPT (slide 52) 55
Diagramme de classes avec choix des circulations pour les classes (slide 53) 57
Génération de classes Java (slide 54) 58
Exercices * -- * avec attributs (slide 55) 59
4 – Héritage (xx slides) 60
5 - UML – Exercices Diagramme de classes techniques 61
0 : Qu'est-ce qu'un diagramme de classes techniques 61
1 : Jeu vidéo avec joueur et arme 61
6 - UML – Exercices Diagramme de classes métier 62
Qu'est-ce qu'un diagramme de classes métier 62
Principes 62
0 : Exercices de base 63
1 : Application de gestion des cours d'un institut de formation. 64
2 : Gestion d'une chaîne hôtelière 65
3 : Gestion d'excursions et de randonnées 66
4 : Gestion des emprunts dans une bibliothèque 67
5 : Application de ventes aux enchères 68

2 : COMPLEMENTS THEORIQUES – 1 - LES 4 TYPES DE RELATIONS ENTRE LES CLASSES 69
0 - Présentation des relations 69
1 - Agrégation 70
Rappel : il y a principalement 4 types de relations entre les classes : 70
Exemple d'agrégation : un répertoire de personnes 70
2 - Composition 72
Rappel : il y a principalement 4 types de relations entre les classes : 72
Exemple de composition : la télévision et ses composants (tuner et ampli) 72
3 - Enumération 74
Rappel : il y a principalement 4 types de relations entre les classes : 74
Classe énumération 74
4 – Héritage (retour au I-4) 75

Rappel : il y a principalement 4 types de relations entre les classes : 75
Exemple conduisant à l'héritage 75
Principe de l'héritage : mettre le commun dans une classe de base 76
Inclusion ensembliste, relation « est-un » 77
5 - UML – Exercices de conception de POO UML 78
Équation du second degré 78

3 : COMPLEMENTS THEORIQUES - 2 - CLASSE ABSTRAITES ET INTERFACES 79
1 – Classe et méthode abstraite 79
Classe abstraite 79
Méthode abstraite 80
2 - Interface 81

4 : COMPLEMENTS THEORIQUES - 3 - NOTION DE DEPENDANCE 82
1 – Dépendance 82
Définition 82
3 dépendances marquées par des associations déjà vues : 82
2 nouvelle dépendances qui se traduiront par une flèche orientée en pointillés 82

Edition mai 2025

1 : ASSOCIATIONS ENTRE LES CLASSES

0 – Diagramme de classes métier – méthodologie (12 slides)

Notions du chapitre (slide 1)

- Classes métier
- Table de la BD
- Clé primaire, primary key, pk
- Clé secondaire : attribut « unique »
- Power Designer, DrawIO, chatGPT
- DCM, MCD, MLD graphique, MLD textuel, MPD

Notion de classe métier (slide 2)

Remarque méthodologique

- On aborde le DCM (Diagramme des Classes Métier) qui équivaut au MCD (Modèle Conceptuel des Données) avant d'aborder les associations entre classes. On pourrait aussi le faire après.

Définition d'une classe métier

- Une classe métier est une classe qui représente un concept ou un objet du domaine métier.

Relation en classe métier et table de la BD

- En général, une classe métier correspond à une table de la base de données (BD).

Circulation DCM - BD

- On a ce circuit :

Code Objet (Java) ↔ DCM ↔ MPD ↔ Code SQL

Méthode de construction d'un diagramme de classe métier -> 1ère étape

- Un diagramme de classes métier ne contient que les classes métier.
- Dans un premier temps, on ne met que les attributs. On ne met pas de méthodes.

Exemple : un site de vente (slide 3)

- Dans tous les sites de vente, on trouve des clients qui s'enregistrent sur le site et des produits à vendre.
- Pour commencer, un client a un nom, un prénom, une adresse mail, un mot de passe et une adresse postale (décomposé au minimum en : n° et rue ensemble, code postal et ville). L'adresse mail est unique : chaque client a une adresse mail différente.
- Pour commencer, les produits ont un nom, une catégorie, une description et un prix.
- Le but est de faire un Diagramme des classes métiers.

DCM d'un site de vente : première approche (slide 4)

- Dans cette première approche, on trouve 2 classes : Client et Produit, sans relation entre les classes.
- Dans la suite du cours on complètera le modèle.

Client
id (pk)
nom
prénom
mail (unique)
mot de passe
adresse
code postal
ville

Produit
id (pk)
nom
catégorie
description
prix

Syntaxe du DCM

⇒ On ne présente pas de compartiment des méthodes.

⇒ On ne met pas le « - » sur les attributs : ils sont « - » par défaut.

⇒ Les id sont des clés primaires (noté « primary key » ou « pk » ou souligné).

⇒ mail est une clé secondaire : on met (unique) pour le voir sur le diagramme.

⇒ Ce n'est pas du code : on peut mettre des accents, des espaces, etc.

⇒ On commence le nom des attributs par une minuscule.

⇒ On commence le nom des des classes par une majuscule.

Beaucoup de sigles ! (slide 5)

- Le **DCM** correspond à un modèle de la base de données (Un modèle, c'est une représentation schématique et simplifiée de la réalité).
- Il existe 3 types de modèles de données : **MCD**, **MLD** et **MPD**. Modèle conceptuel, logique ou physique de données.
- Le **DCM** est équivalent au **MCD**. Mais le DCM est toujours fait en UML.
- Pour le moment, **MCD** et **MLD** sont équivalents (car il n'y a pas d'associations donc de clés étrangères).
- Le **MPD**, c'est le schéma qui correspond au code - du SQL - pour créer les tables de la BD.

Bilan :

Ca fait beaucoup de sigles !

Modèle de BD : MLD du DCM (slide 6)

- Le MLD du diagramme des classes métier consiste à écrire les tables de la BD correspondant aux classes.
- **A ce stade, le MLD est équivalent au diagramme de classes.**
- Les noms des tables sont mis au pluriel : Clients, Produits car une table n'est pas un moule pour un objet mais un « conteneur » d'individus. La table Clients contient les clients. La table produit contient les produits.
- On peut faire un **MLD graphique** :

CLIENTS
id (pk)
nom
prénom
mail (unique)
mot de passe
adresse
code postal
ville

PRODUITS
id (pk)
nom
catégorie
description
prix

- On peut faire une **MLD textuel** :
  - ⇒ **CLIENTS** (id, nom, prenom, mail, motDePasse, adresse, codePostal, ville)
  - ⇒ **PRODUITS** (id, nom, categorie, description, prix)

Convention de nommage : pluriel pour les tables, singulier pour les classes (slide 7)

- Par convention :
  - ⇒ La **classe « Client »** : avec une majuscule au début et au singulier.
    - Au singulier car une classe est un type, un moule, pour un objet.
  - ⇒ La **table « CLIENTS »** : au pluriel et toute en majuscules.
    - Au pluriel car une table est un ensemble d'objets (l'ensemble des clients).
  - ⇒ Les **clés primaires** sont **soulignées** (et/ou repérés par un (**pk**))
  - ⇒ Les **clés étrangères** sont mises **en dernier** (et/ou repérées par un # devant ou un (**fk**))

Modèle du code de la BD : MPD du MLD (slide 8)

- Le MPD du diagramme des classes métier consiste à écrire le code des tables de la BD correspondant aux classes.
- **A ce stade, le MPD est équivalent au MLD, mais, c'est du code, donc :**
  - ⇒ Soit on écrit : « mot de passe », soit on écrit « mot_de_passe » qui correspondra au code.
  - ⇒ Dans le code, on évite les espaces et les accents.
  - ⇒ Souvent on suffixe (ou on préfixe) les attributs avec le nom de la table.
  - ⇒ Souvent le code est en anglais.

- On peut faire un **MPD graphique** :

CLIENTS
id_client (pk)
nom_client
prénom_client
mail_client (unique)
mot_de_passe_client
adresse_client
code_postal_client
ville_client

PRODUITS
id_produit (pk)
nom_produit
categorie_produit
description_produit
prix_produit

- Souvent, on utilise l'écriture MPD : « mot_de_passe_client » directement dans le DCM

Réalisation des diagrammes (slide 9)

Plusieurs outils possibles pour réaliser ces diagrammes :

- **Power Designer** (à charger et installer, ou bien d'autres **modeler UML** : StarUML, Bouml, …)
- **Draw Io** (en ligne)
- **Sur papier** (pas pratique)

- A noter que :

  ⇒ Avec les modeler UML, on a intérêt à suffixer (ou préfixer) le nom des attributs.

Génération automatique de code ou de schéma (slide 10)

- On a ce circuit :

Code Objet (Java) ←→ DCM ←→ MPD ←→ Code SQL

- Avec un **modeler UML** ou avec **chatGPT**, on peut passer automatiquement de l'un à l'autre.

Simulation du remplissage des tables (slide 11)

- Chaque classe correspond à une table dont on peut simuler le remplissage :

Table des CLIENTS :

id	nom	prenom	mail	motDePasse	numeroDeVoie	adresse	code postal	ville
1	Toto	p1	totop1@mail.fr	********	12	rue truc	75001	Paris
2	Tata	p3	tata@mail.fr	********	3	rue chose	75012	Paris
3	Toto	p2	Totop2@mail.fr	********	32	Avenue machin	75010	Paris

Table des PRODUITS :

id	nom	categorie	description	prix
1	n1	c1	Blabla 1	20
2	n2	c1	Blabla 2	10
3	n3	c2	Blabla 3	20

- On peut générer le remplissage plus ou moins proprement en utilisant chatGPT.

Code SQL pour créer les tables et les données (slide 12)

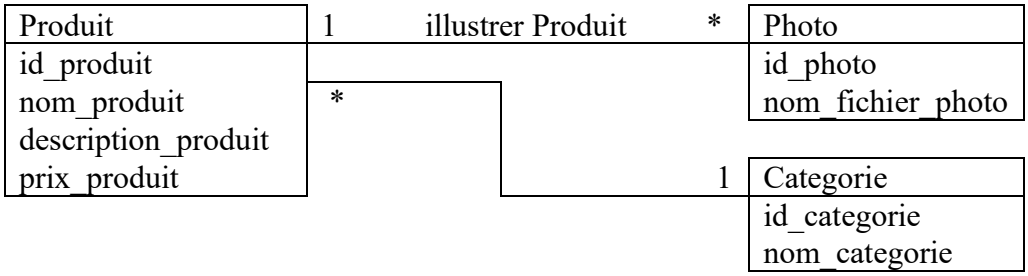
```
DROP DATABASE IF EXISTS BD_SITE_VENTE;
CREATE DATABASE BD_SITE_VENTE;
USE BD_SITE_VENTE;

-- Création de la table CLIENTS
CREATE TABLE CLIENTS (
    id_client INT AUTO_INCREMENT,
    nom_client VARCHAR(50) NOT NULL,
    prenom_client VARCHAR(50) NOT NULL,
    mail_client VARCHAR(100) NOT NULL,
    mot_de_passe_client VARCHAR(255) NOT NULL,
    numero_de_voie_client INT NOT NULL,
    adresse_client VARCHAR(100) NOT NULL,
    code_postal_client VARCHAR(10) NOT NULL,
    ville_client VARCHAR(50) NOT NULL,
    PRIMARY KEY(id_client)
);
-- Création de la table PRODUITS
CREATE TABLE PRODUITS (
    id_produit INT AUTO_INCREMENT,
    nom_produit VARCHAR(50) NOT NULL,
    categorie_produit VARCHAR(50) NOT NULL,
    description_produit TEXT NOT NULL,
    prix_produit DECIMAL(10, 2) NOT NULL,
    PRIMARY KEY(id_produit)
);

-- Insertion des données dans la table CLIENTS
INSERT INTO CLIENTS
VALUES
(1, 'Toto', 'p1', 'totop1@mail.fr', '********', 12, 'rue truc', '75001',
'Paris'),
(2, 'Tata', 'p3', 'tata@mail.fr', '********', 3, 'rue chose', '75012', 'Paris'),
(3, 'Toto', 'p2', 'Totop2@mail.fr', '********', 32, 'Avenue machin', '75010',
'Paris');

-- Insertion des données dans la table PRODUITS
INSERT INTO PRODUITS
VALUES
(1, 'n1', 'c1', 'Blabla 1', 20.00),
(2, 'n2', 'c1', 'Blabla 2', 10.00),
(3, 'n3', 'c2', 'Blabla 3', 20.00);

-- Vérification du contenu des tables :
SELECT * FROM CLIENTS ;
SELECT * FROM PRODUITS ;
```

1 – Association 1 à plusieurs : 1-----* : « one-to-many » (16 slides)

Notions du chapitre (slide 13)

- Association 1 à plusieurs, one-to-many, 1--*
- Variantes UML du diagramme de classes métier
- Nom d'une association : « avoir » par défaut, ou un verbe ou un groupe verbal
- Cardinalité d'une association
- Justification des cardinalités d'une association
- Composition
- Règles de passage du DCM (MCD) au MLD

Exemple de diagramme de classes avec association 1--* (slide 13 bis)



- Justification des cardinalités :
  - ⇒ Un produit a 1 catégorie
  - ⇒ Une catégorie a plusieurs (*) produits
  - ⇒ Une photo illustre 1 produit
  - ⇒ Un produit est illustré par plusieurs(*) photos

Complément du site de vente : des photos pour les produits (slide 14)

- On ajoute que chaque produit contient une liste de photos.
- Une photo est caractérisée par le nom de son fichier image.

La classe photo sera :

Produit
id nom catégorie description prix

Photo
id nom fichier

Mais comment mettre des photos dans les produits ?

Mettre des photos dans les produits (slide 15)

Produit
id nom catégorie description prix photos[] : Photo

Photo
id nom fichier

⇒ On a ajouté un attribut « photos[] » en précisant sont type : il est de classe Photo.

⇒ photos[] : les [] veulent dire que photos est une liste.

⇒ On précise une liste de quoi : d'objet de la classe Photo

⇒ **C'est du code ! Ce n'est pas de l'UML !**

UML : ajouter une association dans le DCM (slide 16)

- En UML, pour dire qu'une classe contient un attribut dont le type est une autre classe : on met une association : un trait entre 2 classes.

Produit	1	*	Photo
id nom categorie description prix			id nomFichier

- Produits 1 ——— * Photo
  - ⇒ * veut dire : plusieurs
  - ⇒ C'est une association 1 à plusieurs
  - ⇒ Comme on a mis l'association, on ne met pas photos[] : Photo
- 1 et plusieurs sont des « cardinalités »

Cardinalités d'une association et justification des cardinalités (slide 17) (cf. slide 32)

Produit	1	*	Photo
id nom categorie description prix			id nomFichier

- Dans notre diagramme il y a 1 association « one to many » : 1--*

  ⇒ « 1 » et « * » s'appellent les « **cardinalités de l'association** ».

- Une association qui relie 2 classes peut se lire comme le verbe « avoir ».

  ⇒ On peut faire une phrase : sujet verbe avoir complément entre les 2 classes, dans les 2 sens.

- Pour faire la phrase, on met toujours le sujet au singulier :

  ⇒ **1 produit a plusieurs photos**.

  ⇒ Dans l'autre sens : **1 photo a 1 produit** (on choisit le fait qu'une photo est pour 1 produit et 1 seul et n'est pas partagée par plusieurs produits).

- Dire ou écrire toutes ces phrases, c'est **justifier les cardinalités des associations**.

  ⇒ Il est important de le faire pour ne pas se tromper :

  ⇒ Ça a des conséquences sur le code objet et sur le code de base de données.

Variante UML : notion de composition (slide 18)

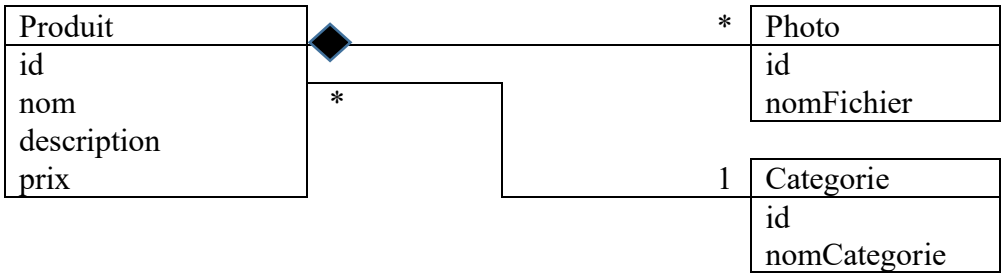
- Le ◆ dit que l'association est une « composition »
  - ⇒ La cardinalité côté ◆ vaut toujours 1.
  - ⇒ La composition veut dire que si on supprime le produit, alors on supprime aussi ses photos.

- Les compositions peuvent être des associations :
  - ⇒ 1 -- * → Justification : 1 produit a plusieurs photos (de 0 à plusieurs)
  - ⇒ 1 -- 1 → Justification : 1 produit a 1 photo et une seule (donc forcément 1)
  - ⇒ 1 -- 0.1 → Justification : 1 produit a 0 ou 1 photo (donc pas forcément de photo)

Variante du modèle : passer les catégories dans une classe (slide 19)

- Les catégories sont un attribut énuméré :

  ⇒ On a une liste prédéfinie et on aimerait pouvoir faire référence à cette liste.



- Produit _*_____1_ Categorie :

  ⇒ C'est une association 1 à plusieurs.

- Justification des cardinalités :

  ⇒ **1 produit a 1 catégorie.**

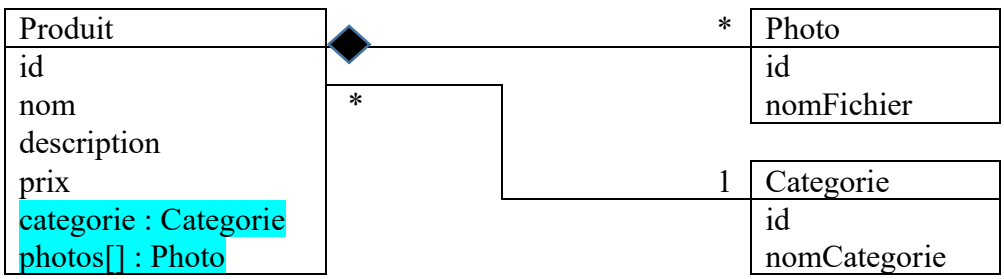
  ⇒ Et dans l'autre sens : **1 catégorie a plusieurs produits**.

Variantes UML (slide 20)

Il y a de nombreuses variantes possibles au schéma précédent : c'est une des difficultés du modèle.

Variante 1 : attributs des associations

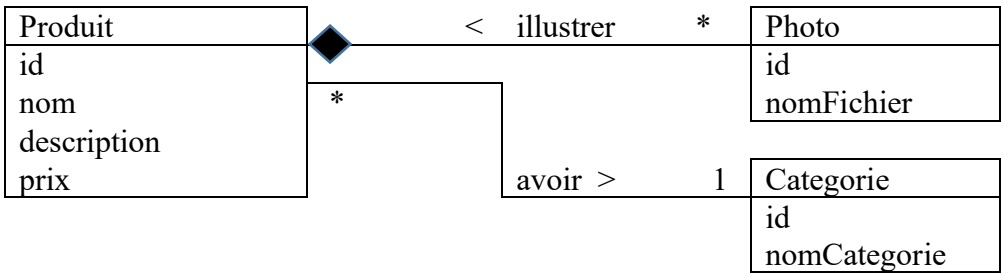
La première variante présentée est une simplification : on met le code des attributs qui correspondent aux associations.



Ce diagramme est le même que le précédent (presque, mais on verra les subtilités plus tard).
Les associations veulent dire qu'il y 2 attributs dans la classe Produit, comme dans le diagramme précédent.

Variante 2 : nommer les associations (slide 21)

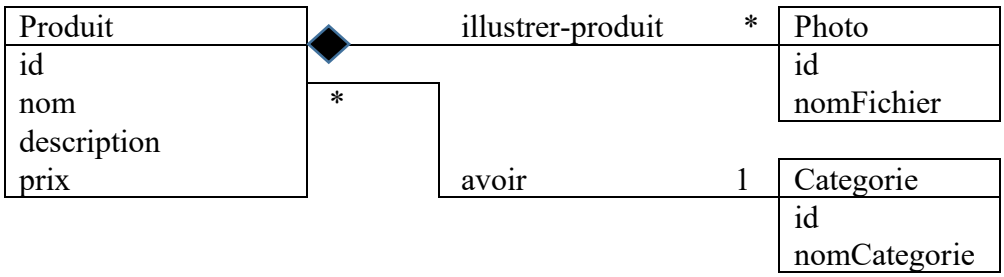
- On peut mettre des noms sur les associations pour clarifier la lecture. Si ça ne clarifie rien, autant ne rien mettre !



- On peut mettre un symbole pour préciser le sens de lecture (ici le « > » va vers le complément)
- Justification des cardinalités :
  - ⇒ 1 photo illustre 1 produit
  - ⇒ 1 produit est illustré par plusieurs photos
  - ⇒ 1 produit a 1 catégorie
  - ⇒ 1 catégories à plusieurs produits

Variante 3 : nommer les associations avec un groupe verbal (slide 22)

- On peut mettre des noms avec un groupe verbal sur les associations pour clarifier la lecture. Si ça ne clarifie rien, autant ne rien mettre !



- Ainsi, on sait dans quel sens lire les associations pour la justification des cardinalité :
  - ⇒ 1 photo illustre 1 produit
  - ⇒ 1 produit est illustré par plusieurs photos
  - ⇒ 1 produit a 1 catégorie
  - ⇒ 1 catégorie a plusieurs produits

Variante 4 : orienter les associations (slide 23)

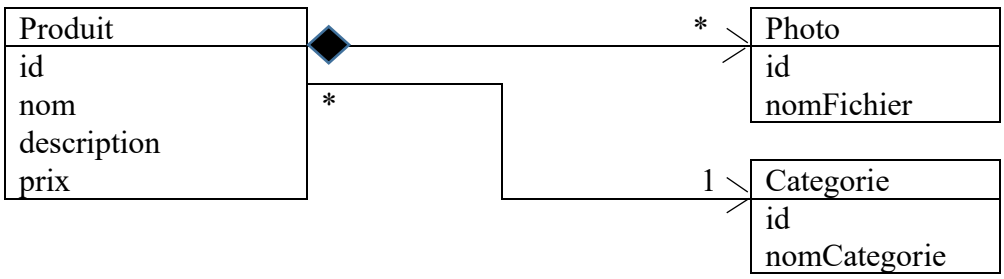
- On peut orienter (flécher) les associations avec une flèche sur l'association :

  ⇒ A ----------> B veut dire :

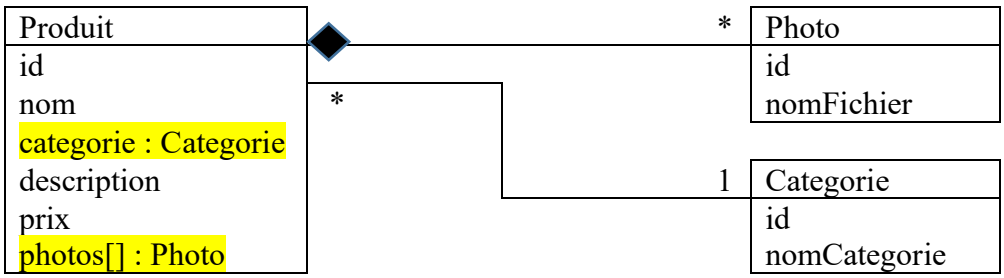
  ⇒ il y a un attribut « b :B » dans A

  ⇒ il n'y a pas d'attribut « a :A » dans B

- Donc :



⇒ Veut dire :

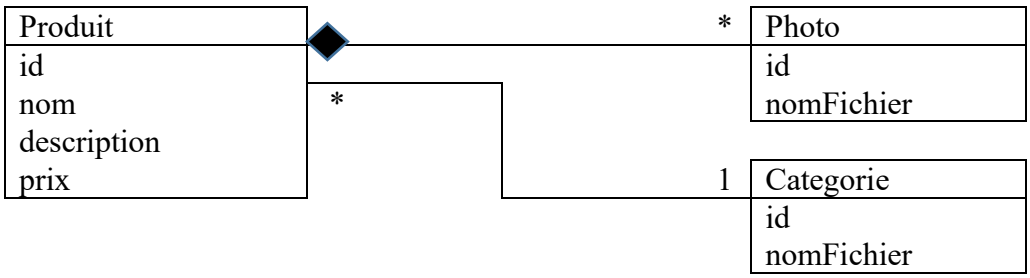


Code d'un modèle non-fléché (slide 24)

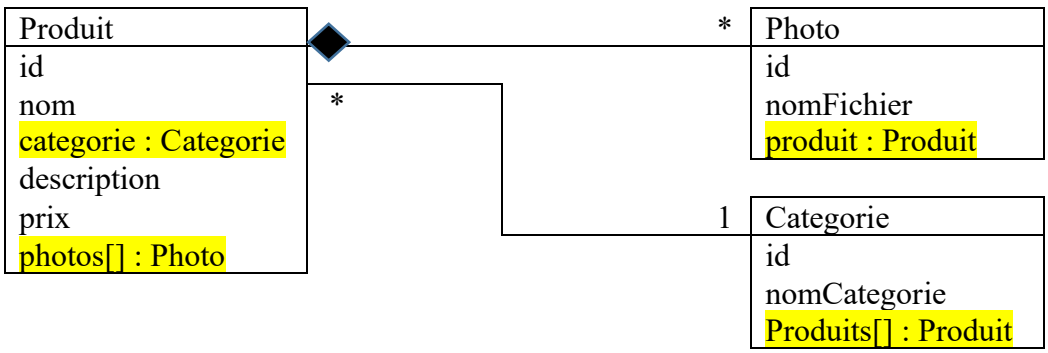
- En théorie, quand on ne met aucune flèche, on peut mettre des attributs de lien partout :

  ⇒ on n'a pas encore décidé !

- Donc :



⇒ Veut dire :

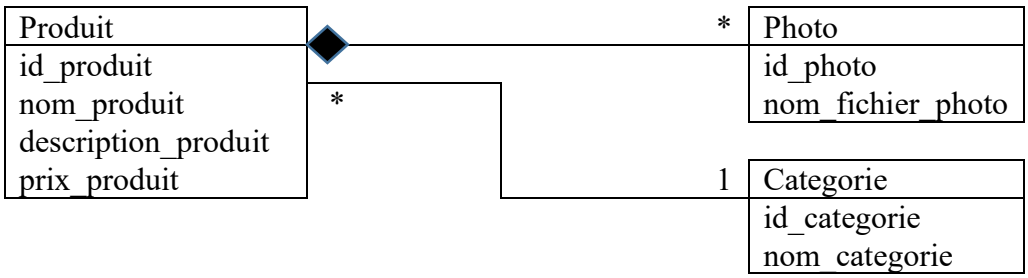


Quel modèle réaliser ? (slide 25)

- Le plus simple ! Et le plus réaliste ! Car on est en conception.

  ⇒ On pourra toujours affiner au fur et à mesure de la conception.

- Le modèle de DCM ci-dessous permettra de coder la BD automatiquement



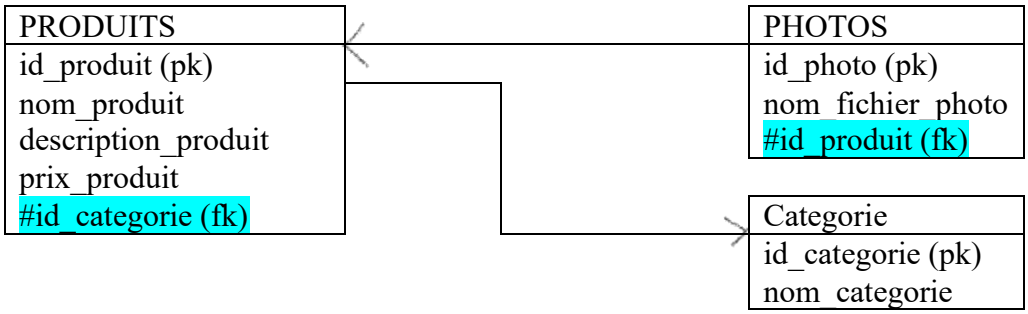
MLD : règles de passage du DCM au MLD (slide 26) (cf slide 35)

Règles pour passer du DCM au MLD :

- **Règle 1** : une classe se transforme en table
- **Règle 2** : une association 1--* se transforme en clé étrangère
  - ⇒ la clé primaire de la classe côté 1 devient un attribut clé étrangère dans la classe côté *.

MLD graphique :

- pour la BD, les flèches sont toujours du côté du 1.



- Les clés étrangères :
  - ⇒ On les met en dernier dans la table.
  - ⇒ On met un # devant le nom pour mieux les repérer (ou un fk, ou les 2)

MLD textuel :

- **Produits** (id, nom, description, prix, # idCategorie)
- **Photos** (id, nomFichier, # idProduit)
- **Catégorie** (id, nom)

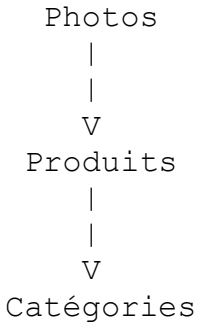
Graphe des tables (slide 27)

- On peut faire un graphe des tables qui montre quelle table est reliée à quelle autre.

  ⇒ Le lien dans le graphe c'est : FK ---> PK

- Quand on fait ce graphe, on représente les flèches vers le bas autant que possible.

  ⇒ Si on ne met pas la flèche, cela veut dire que c'est fléché vers le bas.



Génération automatique de code : chatGPT (slide 28)

- ChatGPT a dit :

  ⇒ Voici le script SQL pour créer la base de données en fonction du modèle relationnel représenté dans l'image :

```
DROP DATABASE if exists Boutique;
CREATE DATABASE Boutique;
USE Boutique;

-- Table Categories
CREATE TABLE Categories (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nomCategorie VARCHAR(255) NOT NULL
);

-- Table Produits
CREATE TABLE Produits (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nom VARCHAR(255) NOT NULL,
    description TEXT,
    prix DECIMAL(10,2),
    id_categorie INT,
    FOREIGN KEY (id_categorie) REFERENCES Categories(id)
);

-- Table Photos
CREATE TABLE Photos (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nomFichier VARCHAR(255) NOT NULL,
    id_produit INT,
    FOREIGN KEY (id_produit) REFERENCES Produits(id)
);
```

Exercices 1 -- * (slide 29)

Le but de ces exercices est de comprendre les principes de la modélisation d'un DCM.
Ce sont des exemples à vocation pédagogiques.

Pour tous les exercices, on traite toutes ces parties :

- Pour la situation décrite : faites un DCM et justifiez les cardinalités pour chaque association.
  - ⇒ Côté SQL : faites un MLD et faites le graphe des tables.
  - ⇒ Côté POO : écrivez tous les attributs de chaque classe (ceux issus des associations)
    - ⇒ Réfléchissez à la présence d'agrégation ou de composition.
    - ⇒ Réfléchissez à orienter vos associations => modifiez la classe côté POO en conséquence.

Employés

Des employés travaillent dans un département (c'est un service de l'entreprise).
Un employé a un numéro d'employé, un nom, un prénom, une date de naissance, une date d'embauche, une fonction, un salaire.
Un département a un numéro de département, un nom et une ville dans laquelle il se trouve.
Un employé a un supérieur hiérarchique lui-même employé.
Le président n'a pas de supérieur hiérarchique.

Livres

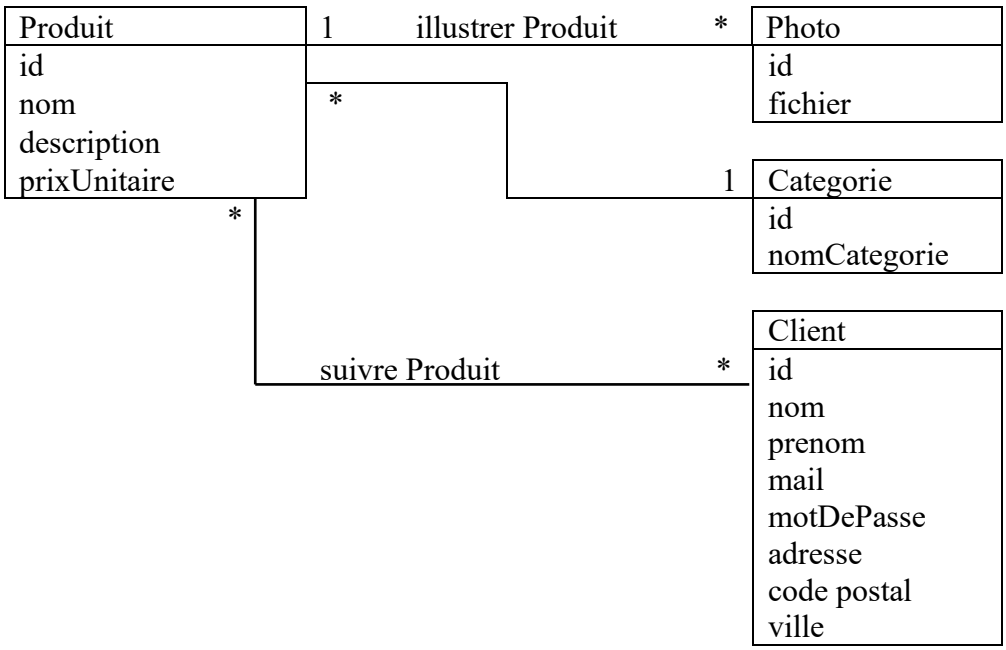
On gère des livres.
Une œuvre a un titre, une année de sortie, une langue de sortie et un auteur (on ne gère pas plusieurs auteurs) et un ISBN qui est un code unique du livre
Un auteur a un nom, un prénom, une date de naissance, une date de décès, une nationalité.
La bibliothèque peut avoir plusieurs exemplaires du même livre, chez le même éditeur ou chez différents éditeurs.
Une édition a une année d'édition, un nom d'éditeur, une langue d'édition.
La bibliothèque donne un identifiant unique à tous ses livres. Elle enregistre aussi la date d'acquisition du livre.

2 – Association plusieurs à plusieurs : *-----* : « many-to-many » (11 slides)

Notions du chapitre (slide 30)

- Association plusieurs à plusieurs
- many-to-many
- *--*
- Table de liaison

Exemple de diagramme de classes avec association *--* (slide 30 bis)

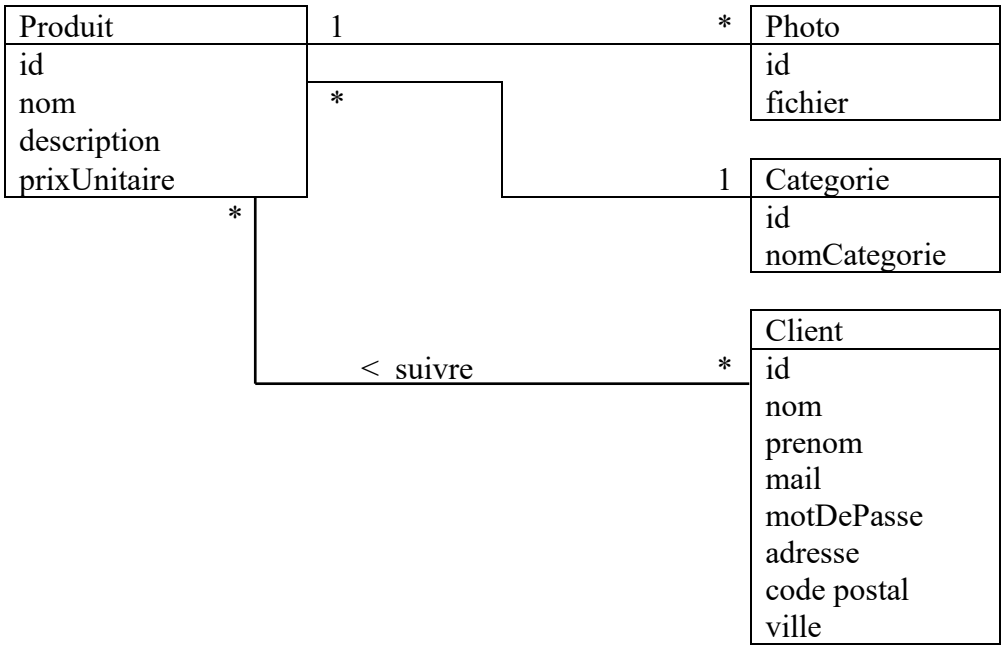


- Justification des cardinalités. On ajoute aux 4 déjà vues :
  - ⇒ Un client suit plusieurs produits
  - ⇒ Un produit est suivi par plusieurs clients

Compléments du site de vente (slide 31)

- Le site gère **des clients** qui ont un id, un nom, un prénom, un mail, un mot de passe et une adresse d'habitation.
- **Un client suit des produits** : il a une liste de produits préférés.

DCM :



Lecture du DCM (slide 32)

- Client * suivre > * Produit :
  - ⇒ * veut dire : plusieurs
  - ⇒ C'est une association plusieurs à plusieurs.
- « plusieurs » (ou * ou 0.N, ou 1.N) sont des « cardinalités »
- Une association qui relie 2 classes peut se lire comme le verbe « avoir »
  - ⇒ On peut aussi mettre un verbe pour mieux comprendre la relation
  - ⇒ et mettre un « > » pour dire que la lecture du verbe est dans le sens : sujet verbe > complément.
  - ⇒ Donc ici : un client suit des produits.

Justification des cardinalités (slide 33) (cf. slide 17)

- Dans ce diagramme il y a 3 associations.
- Chaque association est traduite par 2 phrases.

1 : 1 produit a plusieurs photos
1 photo a un produit (au sens de : correspond à un produit et un seul)

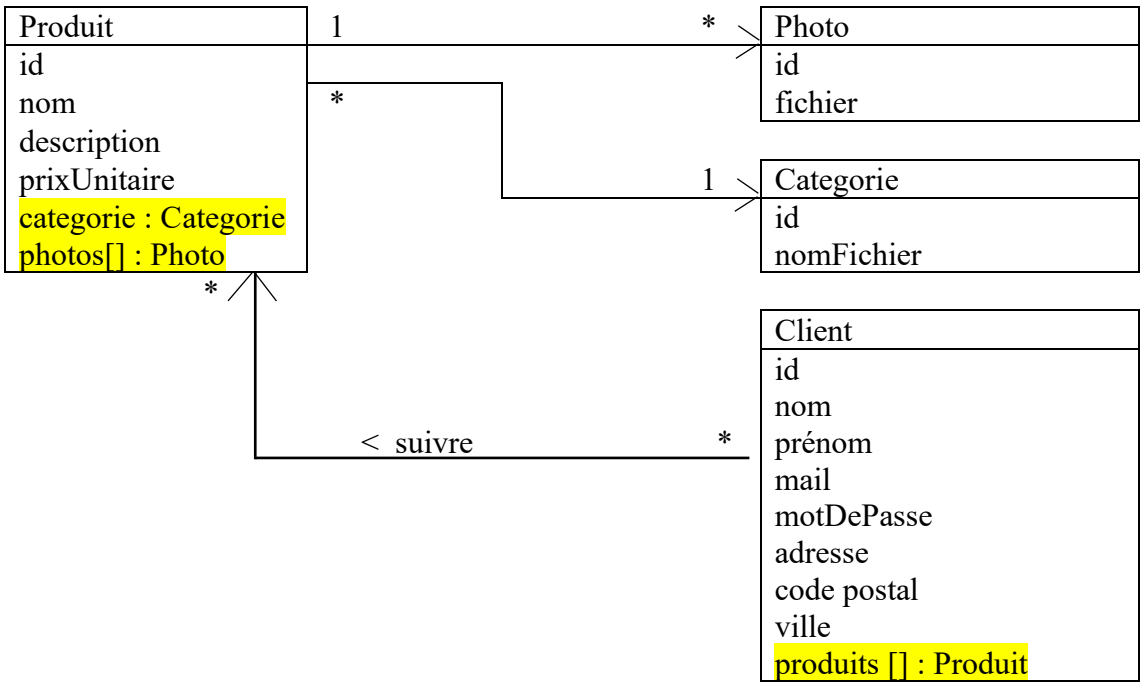
2 : 1 produit à 1 catégorie et une seule
1 catégorie a plusieurs produits (au sens de : correspond à plusieurs produits)

3 : 1 client suit plusieurs produits.
1 produit est suivi par plusieurs clients

- Dire ou écrire toutes ces phrases, c'est justifier les cardinalités des associations.
- Il est important de le faire pour ne pas se tromper : ça a des conséquences sur le code objet et sur le code de base de données.

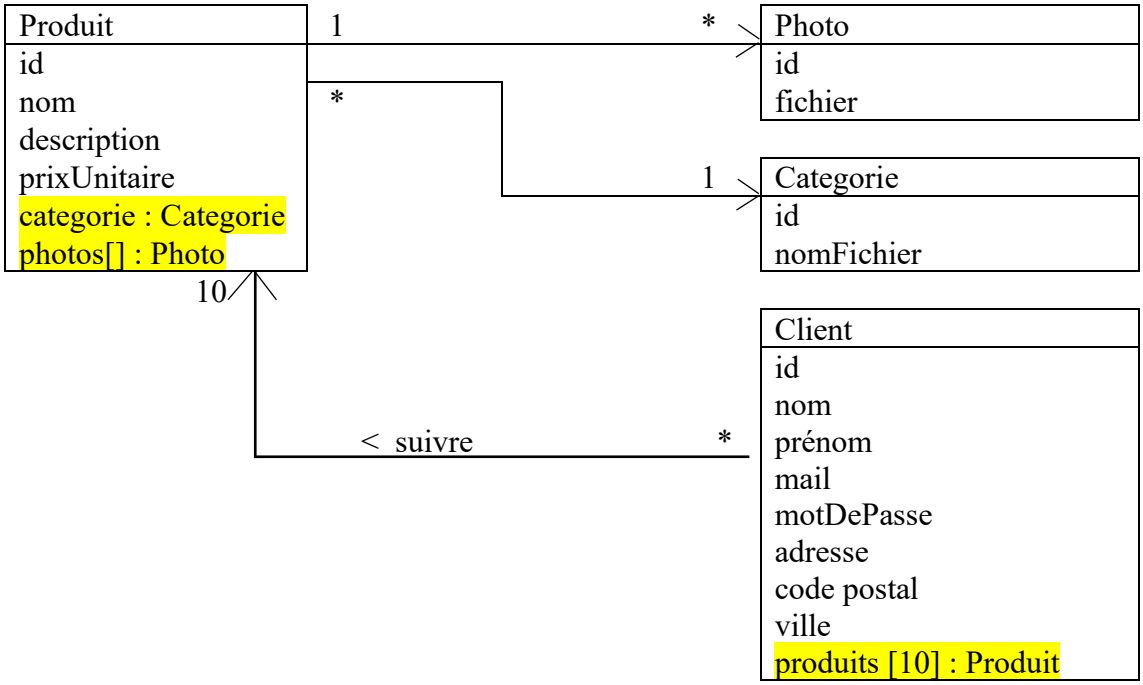
Version avec les attributs des associations dans les classes (slide 34)

- On choisit le sens des liaisons :
  - ⇒ 1 client a plusieurs commandes
  - ⇒ 1 commande a plusieurs produits
  - ⇒ 1 client suit plusieurs produits



Variante technique : 10 plutôt que * : à éviter dans un DCM ! (slide 35)

- On précise qu'un client peut suivre jusqu'à 10 produits maximum :
- Dans ce cas, on peut mettre 10 à la place de « * »



- Pour un DCM (diagramme de classes métier), on évite de mettre 10 à la place de « * ».
- Pourquoi ? Parce que c'est une considération qui est inutile pour la partie base de données et que de DCM est en rapport direct avec la BD.

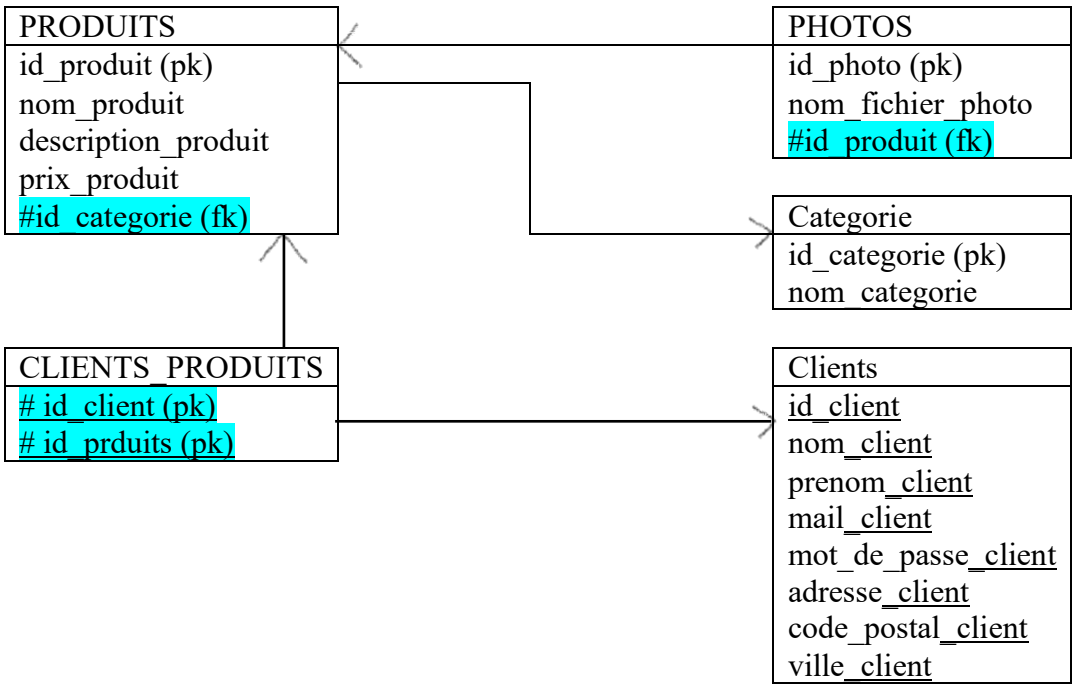
MLD : règles de passage du DCM au MLD (slide 36) (cf slide 26)

- Le MLD du diagramme des classes métier consiste à écrire les tables correspondant aux classes, en mode textuel.

Règles pour passer du DCM au MLD

- **Règle 1** : une classe se transforme en table
- **Règle 2** : une association 1--* se transforme en clé étrangère
  - ⇒ la clé primaire de la classe côté 1 devient un attribut clé étrangère dans la classe côté *.
- **Règle 3** : une association *--* se transforme en table !
  - ⇒ On crée une table qu'on nomme du nom des tables associées : produits_clients
  - ⇒ Dans cette table, on met chaque clé primaire des tables associées comme clé étrangère.
  - ⇒ La clé primaire de cette table est constituée de plusieurs attributs à déterminer. En première hypothèse, c'est la concaténation des clés primaires des tables associées.

MLD graphique : table de liaison (slide 37)



- La table CLIENTS_PRODUITS est une « table de liaison ».

MLD textuel (slide 38)

On simplifie l'écriture :

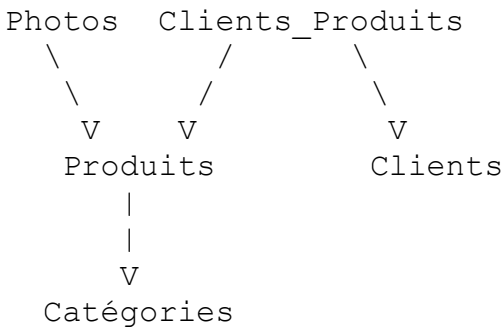
- **Produits** (id, nom, description, prix, # idCategorie)
- **Photos** (id, nomFichier, # idProduit)
- **Catégorie** (id, nomCategorie)
- **Clients** (id, nom, prénom, mail motDePasse, adresse, codePostal, ville)
- **Clients_Produits** (# id_client, # id_produit)

Usages d'écriture

- Les noms des tables sont mis au pluriel.
- Les clés primaires sont soulignées.
- Les clés étrangères :
  - ⇒ idCategorie et idProduit
  - ⇒ Sont précédées d'un « # » pour mieux les repérer.
  - ⇒ Sont nommées d'après l'id de la table à laquelle elles font référence.
  - ⇒ Sont placées en dernier dans la table (sauf si elles font partie de la clé primaire)

Graphe des tables (slide 39)

- On peut faire un graphe des tables qui montre quelle table est reliées à quelle autre.
- Le lien c'est : FK ---> PF (foreign key ---> primary key)
- Quand on fait ce graphe, on représente les flèches vers le bas autant que possible.
- Si on ne met pas la flèche, cela veut dire que c'est fléché vers le bas.



Génération de code SQL : chatGPT (slide 40)

```
DROP DATABASE if exists Boutique;
CREATE DATABASE Boutique;
USE Boutique;

-- Table Categories
CREATE TABLE Categories (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nomCategorie VARCHAR(255) NOT NULL
);

-- Table Produits
CREATE TABLE Produits (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nom VARCHAR(255) NOT NULL,
    description TEXT,
    prix DECIMAL(10,2),
    id_categorie INT,
    FOREIGN KEY (id_categorie) REFERENCES Categories(id)
);

-- Table Photos
CREATE TABLE Photos (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nomFichier VARCHAR(255) NOT NULL,
    id_produit INT,
    FOREIGN KEY (id_produit) REFERENCES Produits(id)
);

-- Table Clients
CREATE TABLE Clients (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nom VARCHAR(255) NOT NULL,
    prenom VARCHAR(255) NOT NULL,
    mail VARCHAR(255) UNIQUE NOT NULL,
    motDePasse VARCHAR(255) NOT NULL,
    adresse VARCHAR(255) NOT NULL,
    code_postal VARCHAR(20) NOT NULL,
    ville VARCHAR(100) NOT NULL
);

-- Table Clients_Produits
CREATE TABLE Clients_Produits (
    id_client INT,
    id_produit INT,
    PRIMArY KEY(id_client, id_produit),
    FOREIGN KEY (id_client) REFERENCES Clients(id),
    FOREIGN KEY (id_produit) REFERENCES Produits(id)
);
```

Exercices * -- * sans attributs (slide 41)

Le but de ces exercices est de comprendre les principes de la modélisation d'un DCM.
Ce sont des exemples à vocation pédagogiques.

- Pour la situation décrite : faites un DCM et justifiez les cardinalités pour chaque association.
  - ⇒ Côté SQL : faites un MLD et faites le graphe des tables.
  - ⇒ Côté POO : écrivez tous les attributs de chaque classe (ceux issus des associations)
    - ⇒ Réfléchissez à la présence d'agrégation ou de composition.
    - ⇒ Réfléchissez à orienter vos associations => modifiez la classe côté POO en conséquence.

Projets

On gère des projets qu'on découpe en étapes successives.
Un projet a un nom, une date de début, une date de fin prévisionnelle, une date de fin réalisée.
Une étape appartient a 1 projet et 1 seul. Elle a un nom, une date de début prévisionnelle, une date de début réalisé une date de fin prévisionnelle, une date de fin réalisée.
A ce stade, les étapes sont numérotées de 1 à N dans le projet.
Les employés participent à des projets. On n'en sais pas plus. Un employé a un nom, un prénom, une fonction, une date d'embauche.
Les projets utilisent des ressources. On n'en sait pas plus. Une ressource a un nom, une description et une photo.

Classes

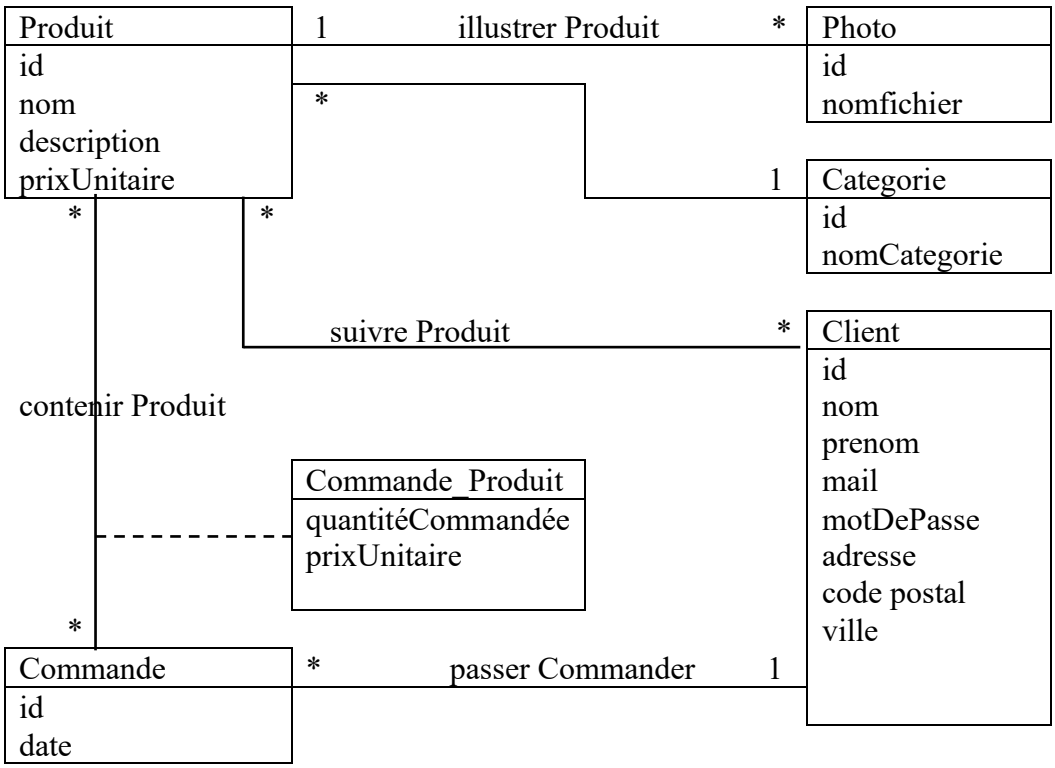
On gère des classes de collège.
Une classe à un niveau (6è, 5è, etc.), un nom (A, B, etc.) et une année (23-24, 24-25, etc.)
Dans l'école, il y a des élèves. Les élèves sont inscrits dans une classe.

3 – Association plusieurs à plusieurs : *-----* avec attributs (14 slides)

Notions du chapitre (slide 42)

- Association plusieurs à plusieurs, many-to-many, *--*
- Classe-Association
- Attributs dans une classe-association
- Attributs dans une table de liaison

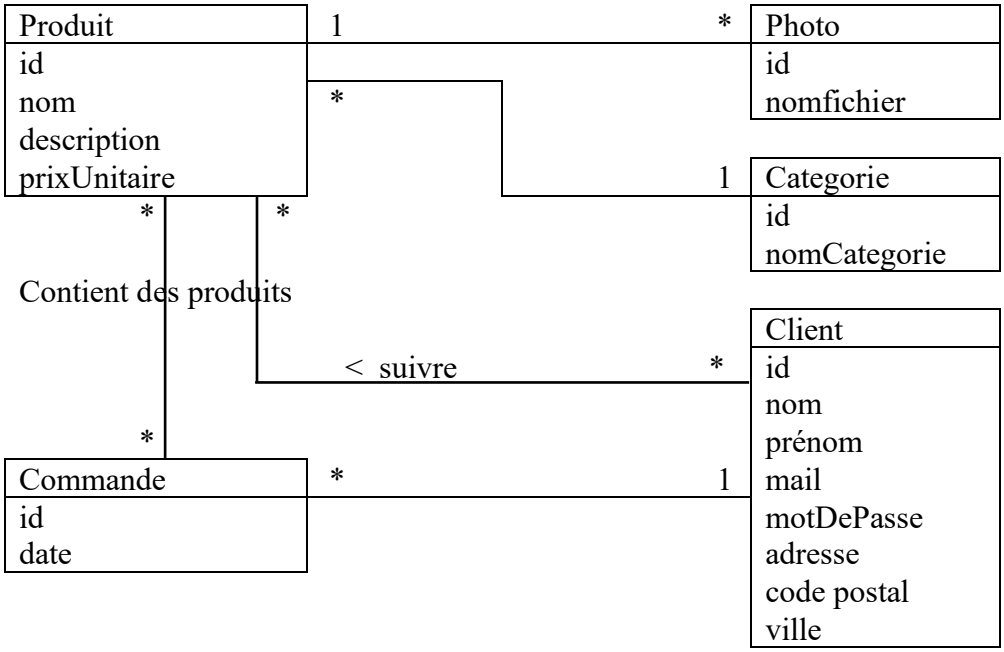
Exemple de diagramme de classes avec association 1--* et classe-association (slide 42 bis)



- Justification des cardinalités. On ajoute aux 6 déjà vues :
  - ⇒ Un client passe plusieurs commandes
  - ⇒ Une commande est passée par un client
  - ⇒ Une commande contient plusieurs produits
  - ⇒ Un produit est contenu dans plusieurs commandes

Compléments du site de vente (slide 43)

- Un client passe une commande.
- Une commande a un numéro identifiant et une date de commande. Elle est faite par un client est un seul.
- Elle contient des produits. Pour chaque produit commandé, on sait combien on en a commandé et le prix Unitaire du produit pour la commande.



Lecture du DCM (slide 44)

- Commande * contient des produits * Produit : c'est une association.
  - ⇒ * veut dire : plusieurs
  - ⇒ C'est une association plusieurs à plusieurs.
  - ⇒ « plusieurs » (ou * ou 0.N, ou 1.N) sont des « cardinalités »
- Une association qui relie 2 classes peut se lire comme le verbe « avoir »
  - ⇒ On peut aussi mettre un verbe pour mieux comprendre la relation
  - ⇒ et mettre un groupe verbal, ici : « contient des produits » pour dire le sens de la lecture (produit c'est le nom de la table « complément » de la lecture table_1-sujet verbe complément.
  - ⇒ Donc ici : une commande contient plusieurs produits.

Justification des cardinalités (slide 45)

- Dans ce diagramme il y a 5 associations.
- Chaque association est traduite par 2 phrases.

1 : Produit 1--* Photo :
1 produit a plusieurs photos
1 photo a un produit (au sens de : correspond à un produit et un seul)

2 : Produit *--1 Catégorie :
1 produit à 1 catégorie et une seule
1 catégorie a plusieurs produits (au sens de : correspond à plusieurs produits)

3 : Client *--* Produit
1 client suit plusieurs produits.
1 produit est suivi par plusieurs clients

4 : Facture *--1 Client
1 facture est pour 1 client et un seul.
1 client a plusieurs factures.

5 : Produit *--* Commande
1 produit est contenu par plusieurs commandes.
1 commande contient plusieurs produits.

- Dire ou écrire toutes ces phrases, c'est justifier les cardinalités des associations.
- Il est important de le faire pour ne pas se tromper : ça a des conséquences sur le code objet et sur le code de base de données.

Le problème des attributs de l'association : la classe-association (slide 46)

Le problème

- Pour chaque produit commandé, on veut connaitre la quantité commandée et le prix auquel on a commandé le produit.

  ⇒ Autrement dit, l'association doit porter des attributs.

- On a vu qu'une association *--* se transforme en table dans la BD.

  ⇒ On aura une table Commandes_Produits

  ⇒ On peut ajouter les attributs : quantité, prixUnitaire

- En BD, cette table sera :

  ⇒ Commandes_Produits(# id_commande, #id_produite, quantité, prixUnitaire)

  ⇒ Comment mettre ces attributs dans notre DCM UML ?

Classe-association

- On va « accrocher » une classe sur l'association *--* : c'est une classe-association.

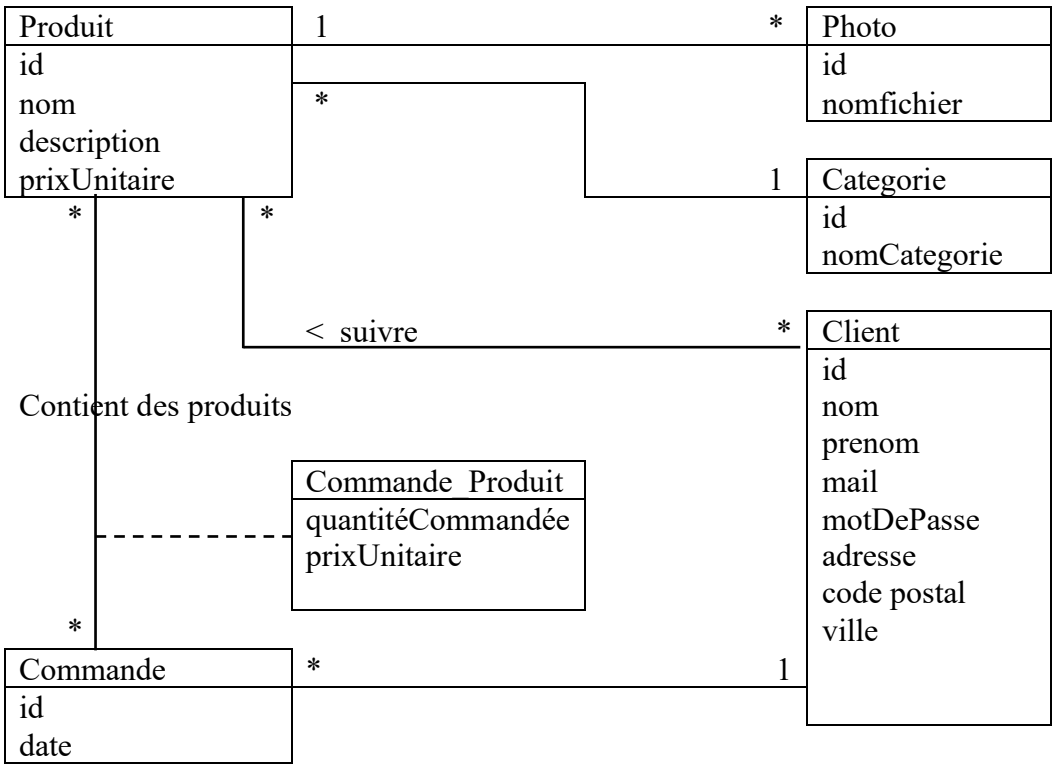
  ⇒ Dans cette classe, on peut mettre des attributs.

  ⇒ On la nomme du nom des classes reliées : Commande_Produit

Règle :

- Seules les associations *--* peuvent porter des classes-associations.

DCM UML avec classe association (slide 47)



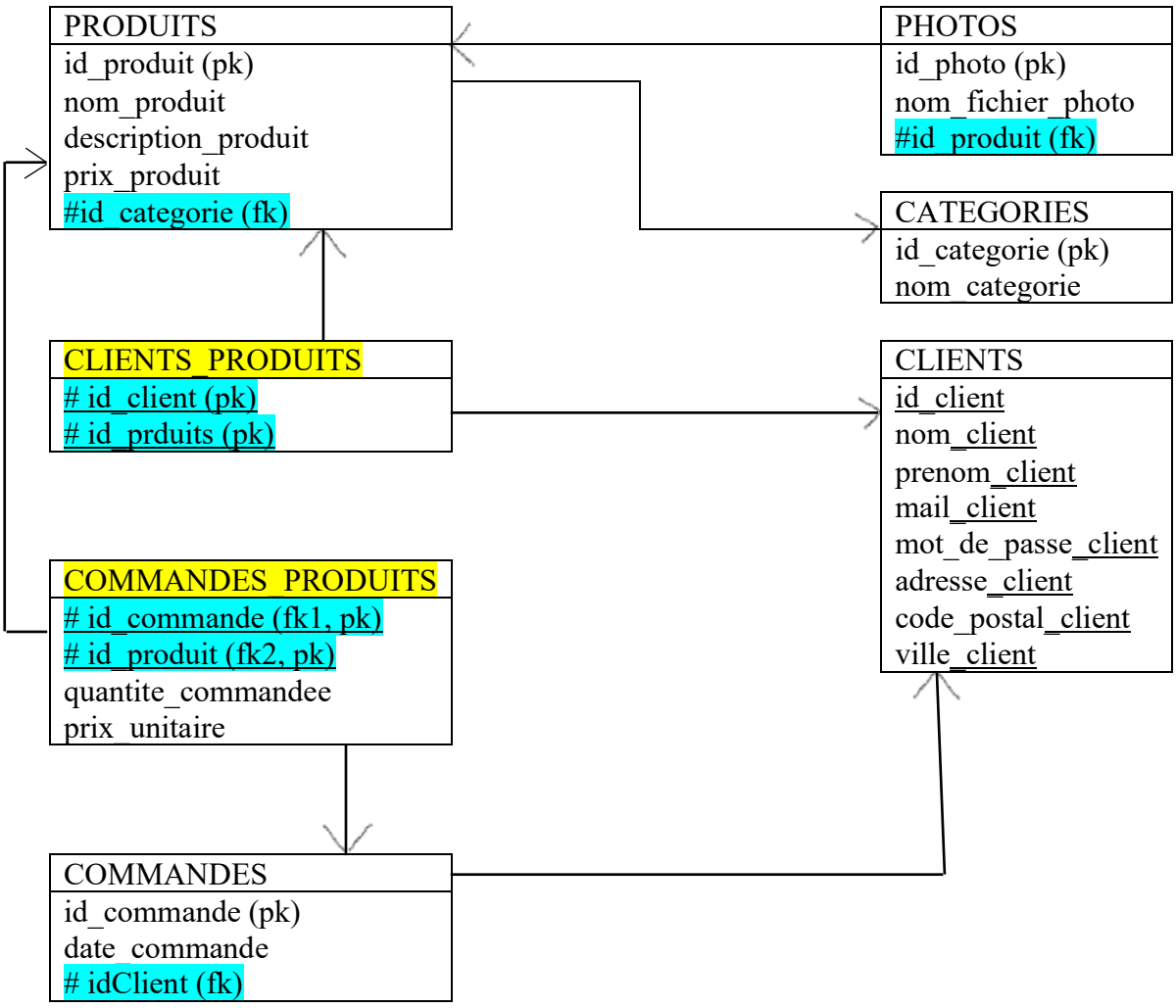
Modèle de BD : MLD du diagramme des classes métier (slide 48)

- Le MLD du diagramme des classes métier consiste à écrire les tables correspondant aux classes, en mode textuel.

Règles pour passer du DCM au MLD :

- **Règle 1** : une classe se transforme en table
- **Règle 2** : une association 1--* se transforme en clé étrangère (ou 1—ce qu'on veut)
  - ⇒ la clé primaire de la classe côté 1 devient un attribut clé étrangère dans la classe côté *.
- **Règle 3** : une association *--* se transforme en table !
  - ⇒ On crée une table qu'on nomme du nom des tables associées : produits_clients
  - ⇒ Dans cette table, on met chaque clé primaire des tables associées comme clé étrangère.
  - ⇒ La clé primaire de cette table est constituée de plusieurs attributs à déterminer.
    - En 1ère hypothèse, c'est la concaténation des clés primaires des tables associées.
  - ⇒ Les attributs d'une classe-association accrochée à une association *--* rentrent comme attributs dans la table issue de l'association *--*.

MLD graphique (slide 49)



- Les table Clients_Produits et Commandes_Produits sont des « tables de liaison ».

MLD textuel (slide 50)

On simplifie l'écriture :

- **Produits** (id, nom, description, prix, **# idCategorie**)
- **Photos** (id, nomFichier, **# idProduit**)
- **Catégorie** (id, nomCategorie)
- **Clients** (id, nom, prénom, mail motDePasse, adresse, codePostal, ville)
- **Clients_Produits (# id_client, # id_produit**)
- **Commande** (id, dateCommande, **# idClient**)
- **Commandes_Produits (# idCommande, # idProduit**, quantité, prixUnitaire)

Usages d'écriture :

- Les noms des tables sont mis au pluriel.
- Les clés primaires sont soulignées.
- Les clés étrangères :
  - ⇒ Sont précédées d'un « # » pour mieux les repérer.
  - ⇒ Sont nommées d'après l'id de la table à laquelle elles font référence.
  - ⇒ Sont placées en dernier dans la table (sauf si elles font partie de la clé primaire)

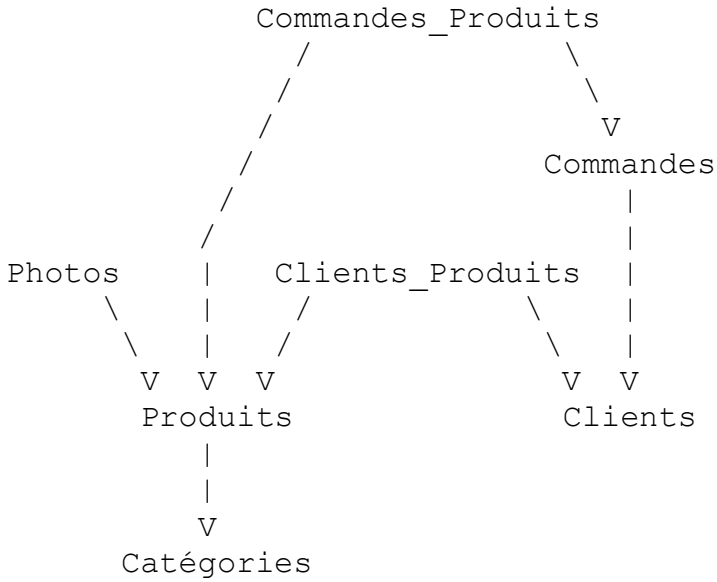
Graphe des tables (slide 51)

- On peut faire un graphe des tables qui montre quelle table est reliées à quelle autre.

  ⇒ Le lien c'est : FK ---> PF (foreign key ---> primary key)

- Quand on fait ce graphe, on représente les flèches vers le bas autant que possible.

  ⇒ Si on ne met pas la flèche, cela veut dire que c'est fléché vers le bas.

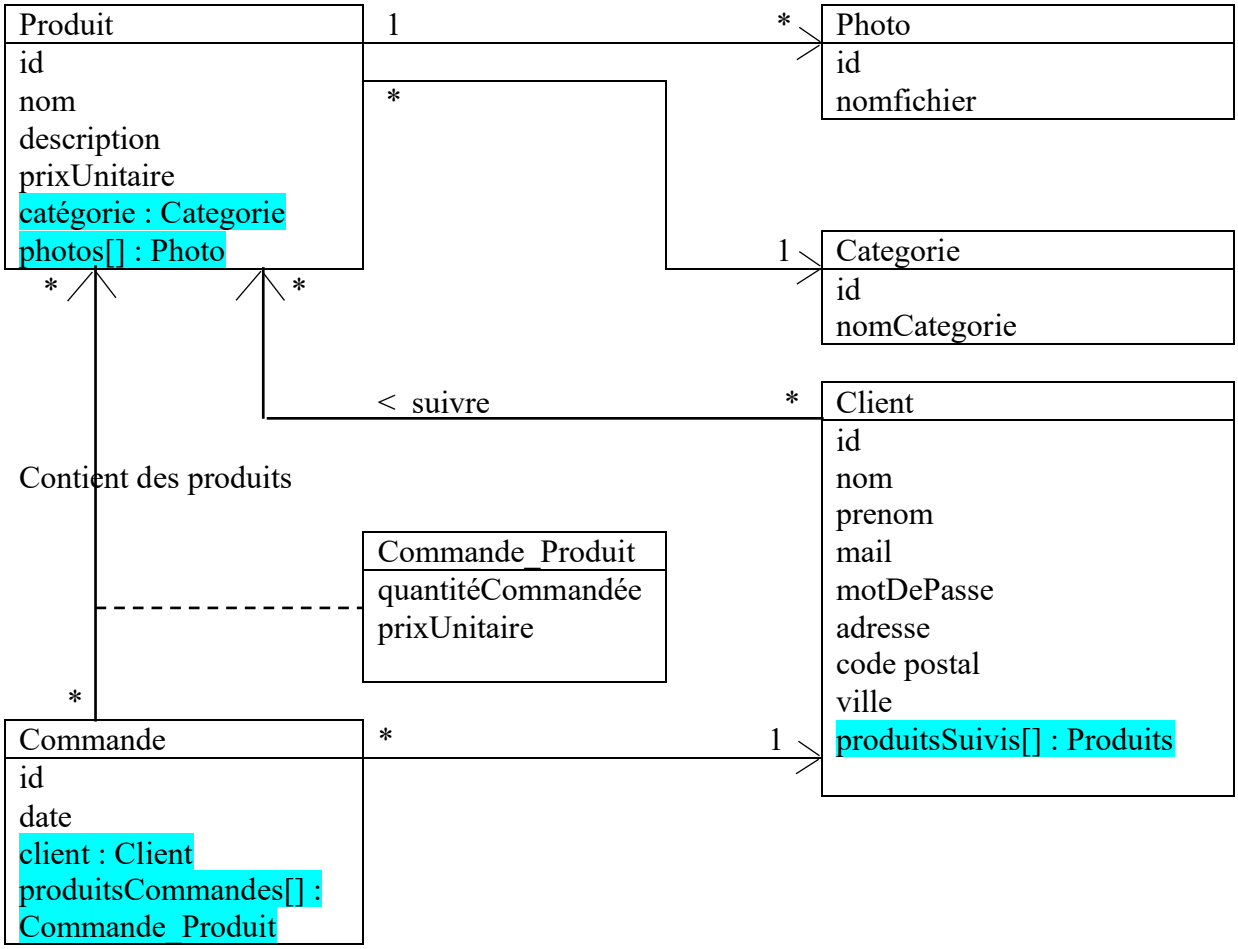


Génération de code SQL : chatGPT (slide 52)

```
DROP DATABASE if exists Boutique;
CREATE DATABASE Boutique;
USE Boutique;

-- Table Categories
CREATE TABLE Categories (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nomCategorie VARCHAR(255) NOT NULL
);

-- Table Produits
CREATE TABLE Produits (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nom VARCHAR(255) NOT NULL,
    description TEXT,
    prix DECIMAL(10,2),
    id_categorie INT,
    FOREIGN KEY (id_categorie) REFERENCES Categories(id)
);

-- Table Photos
CREATE TABLE Photos (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nomFichier VARCHAR(255) NOT NULL,
    id_produit INT,
    FOREIGN KEY (id_produit) REFERENCES Produits(id)
);

-- Table Clients
CREATE TABLE Clients (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nom VARCHAR(255) NOT NULL,
    prenom VARCHAR(255) NOT NULL,
    mail VARCHAR(255) UNIQUE NOT NULL,
    motDePasse VARCHAR(255) NOT NULL,
    adresse VARCHAR(255) NOT NULL,
    code_postal VARCHAR(20) NOT NULL,
    ville VARCHAR(100) NOT NULL
);

-- Table Clients_Produits
CREATE TABLE Clients_Produits (
    id_client INT,
    id_produit INT,
    PRIMArY KEY(id_client, id_produit),
    FOREIGN KEY (id_client) REFERENCES Clients(id),
    FOREIGN KEY (id_produit) REFERENCES Produits(id)
);
```

```
-- Table Commandes
CREATE TABLE Commandes (
    id INT PRIMARY KEY AUTO_INCREMENT,
    dateCommande DATE NOT NULL,
    id_client INT,
    FOREIGN KEY (id_client) REFERENCES Clients(id)
);

-- Table Commandes_Produits
CREATE TABLE Commandes_Produits (
    id_commande INT,
    id_produit INT,
    quantite INT,
    prixUnitaire DECIMAL(10,2),
    PRIMArY KEY(id_commande, id_produit),
    FOREIGN KEY (id_commande) REFERENCES Commandes(id),
    FOREIGN KEY (id_produit) REFERENCES Produits(id)
);
```

Diagramme de classes avec choix des circulations pour les classes (slide 53)



Explications des liens :

- Un produit a 1 catégorie et des photos.
- Un client a des produitsSuivis
- Une commande est faite par 1 client et a des produitsCommandés

Remarques :

- Dans le code de l'application, on aura des objets « Commande », « Produit » et « Client ».
- Les objets « Photo », « Categorie » et « Commande_Produit » n'existeront que dans les objets « Commande », « Produit » et « Client »

Génération de classes Java (slide 54)

Si on demande à chatGPT de générer les classes pour ce diagramme de classe métier, on obtient pour Commande :

```
import java.util.Date;
import java.util.List;

public class Commande {
    private int id;
    private Date date;
    private Client client;
    private List<CommandeProduit> produitsCommandes;

    // Constructeurs
    public Commande() {}

    public Commande(int id, Date date, Client client,
List<CommandeProduit> produitsCommandes) {
        this.id = id;
        this.date = date;
        this.client = client;
        this.produitsCommandes = produitsCommandes;
    }

    // Getters
    public int getId() { return id; }
    public Date getDate() { return date; }
    public Client getClient() { return client; }
    public List<CommandeProduit> getProduitsCommandes() { return
produitsCommandes; }

}
```

Exercices * -- * avec attributs (slide 55)

Le but de ces exercices est de comprendre les principes de la modélisation d'un DCM.
Ce sont des exemples à vocation pédagogiques.

- Pour la situation décrite : faites un DCM et justifiez les cardinalités pour chaque association.
  - ⇒ Côté SQL : faites un MLD et faites le graphe des tables.
  - ⇒ Côté POO : écrivez tous les attributs de chaque classe (ceux issus des associations)
    - ⇒ Réfléchissez à la présence d'agrégation ou de composition.
    - ⇒ Réfléchissez à orienter vos associations => modifiez la classe côté POO en conséquence.

Classes – v2

On gère des classes de collège.
Une classe à un niveau (6è, 5è, etc.), un nom (A, B, etc.) et une année (23-24, 24-25, etc.)
Dans l'école, il y a des élèves. Les élèves sont inscrits dans une classe.

Cinémas

On gère plusieurs cinémas.
Un cinéma est constitué de salles. Ils ont un nom et une adresse.
Les salles ont un numéro unique dans chaque cinéma (salle 1, 2, 3). Elles ont un nombre de places.
Dans les salles le cinéma projette des films qui ont un titre, une durée, une langue, une version (VO ou VF).
Un film est projeté à une certaine date et heure.

4 – Héritage (xx slides)

A faire : pour le moment : voir ici dans ce document.

5 - UML – Exercices Diagramme de classes techniques

0 : Qu'est-ce qu'un diagramme de classes techniques

- C'est un diagramme de classes avec des méthodes et potentiellement des orientations.
- On l'utilise pour des petites analyses de quelques classes pour organiser son code technique.
- On peut le faire sur papier ou avec draw.io (ou avec un modeler UML).
- Les modeler UML permettent de faire du « reverse ingenering » : à partir du code Objet, par exemple du Java, ils peuvent générer un diagramme de classe.

  ⇒ A partir d'un code SQL de création de BD, ils peuvent générer un MCD ou un DCM.

1 : Jeu vidéo avec joueur et arme

- Un personnage de jeu vidéo a un nom, une force, une localisation (des coordonnées X et Y), une expérience, des dégâts et une bourse (un nombre de pièces).
- Le joueur peut se déplacer et peut tourner (changer de direction).
- Le joueur peut combattre un autre personnage. Le combat modifie l'expérience et les dégâts.
- Le joueur va utiliser des armes. Les armes ont un nom, un type, une puissance et une valeur. Le nom et le type sont des données fixes. La puissance et la valeur varient au cours du jeu.
- Le joueur peut porter une arme. Les armes peuvent être « dans la nature ». Un personnage peut aussi stocker des armes.
- Le joueur peut changer d'arme portée en en prenant une autre dans son stock et stockant celle qu'il portait avant.
- Le joueur peut vendre une arme de son stock.

⇒ Faire le diagramme des classes techniques avec les méthodes et les paramètre des méthodes. Mettez la navigabilité des associations.

⇒ Écrire le code SQL de la BD correspondante.

6 - UML – Exercices Diagramme de classes métier

Qu'est-ce qu'un diagramme de classes métier

- C'est un diagramme des classes qui correspondant à la BD.
  - ⇒ Dans un premier temps, on ne met pas de méthodes.
  - ⇒ Dans un premier temps on ne met pas avec d'orientations.
- On le fait avec un modeler UML.
- Les modeler UML permettent de faire du « reverse ingenering » : à partir du code Objet, par exemple du Java, ils peuvent générer un diagramme de classe.
  - ⇒ A partir d'un code SQL de création de BD, ils peuvent générer un MCD ou un DCM.

Principes

- Pour la situation décrite : faites un DCM et justifiez les cardinalités pour chaque association.
  - ⇒ Côté SQL : faites un MLD et faites le graphe des tables.
  - ⇒ Côté POO :
    - ⇒ Faites le DCM en ajoutant tous les attributs de chaque classe (ceux issus des associations)
    - ⇒ Réfléchissez à la présence d'agrégation ou de composition.
    - ⇒ Réfléchissez à orienter vos associations => modifiez la classe côté POO en conséquence.

- Les exercices peuvent être faits sur papier, sur draw-io ou sur votre modeler UML préféré !

0 : Exercices de base

- On fait un DCM.

  ⇒ Ensuite, on réfléchit à un diagramme de classes techniques : ajout de méthodes et orientation.

- Exercices **Employés** et **Livres** : §1.1 : **asso 1 -- ***

  -> slide 29 : Employés, Livres

- Exercice **Projets** : §1.2 : **asso * -- * sans attribut**

  -> slide 41 : Projets, Classes

- Exercices **Classes**, **Cinémas** : §1.3 : **asso * -- * avec attributs**

  -> slide 55 : Cinémas

1 : Application de gestion des cours d'un institut de formation.

- Les cours sont organisés en modules, chaque module est caractérisé par un numéro de module, un intitulé, une durée en heures et un type.
- Les étudiants suivent des enseignements portant sur plusieurs modules. Chaque étudiant est caractérisé par un numéro d'inscription unique, un nom, un prénom et une adresse (avec code postal et ville) et une date de naissance. Un étudiant est évalué trois fois pour chaque module et possède une note de fin de module.
- Chaque étudiant appartient à un groupe caractérisé par un code, une spécialité et le nombre d'étudiants qu'il comporte.
- Un enseignant intervient dans un module pour un groupe donné à une date donnée, chaque enseignant est caractérisé par un code, un nom, un prénom et une adresse.
- Un enseignant intervient habituellement dans plusieurs modules.
- On désire aussi mémoriser le nombre d'heures effectué par chaque enseignant dans un module donné pour un groupe donné. Chaque séance est prévue pour une certaine durée (1h30 ou 2 h). Et on enregistre la présence de chaque enseignant aux séances (l'enseignant le fait lui-même et un administrateur valide).

2 : Gestion d'une chaîne hôtelière

- Vous devez réaliser le système de réservation d'un groupe hôtelier disposant d'une centrale de réservation nationale.

- Après entretien avec le directeur des relations clients voici les notes dont vous disposez :

- Un hôtel est caractérisé par un nom, une adresse, une ville et une catégorie d'hôtel (de une à cinq à étoiles).

- Un hôtel appartient à une seule société qui est identifiée par son numéro de SIRET, son nom et son adresse. Cette société peut également être la propriété d'une autre qui elle-même peut appartenir à une société mère.
- Une société peut être propriétaire de plusieurs hôtels

.

- Une chambre est caractérisée par son numéro, un nombre de lits simples, et un nombre de lits doubles.

- Un client réserve une chambre d'hôtel avec une date de début et une date de fin d'occupation.
- Un client peut réserver plusieurs chambres d'hôtel à la même date.
- Enfin un client est caractérisé par un numéro de client, un nom, une adresse, et une ville de résidence.

3 : Gestion d'excursions et de randonnées

- Une association est spécialisée dans l'organisation des excursions et randonnées dans la nature (montagnes, forêts, déserts...) dans différentes régions.
- Cette association compte un certain nombre d'abonnés, appelés aussi des randonneurs qui sont les seuls à pouvoir sortir en excursion.
- Les excursions sont organisées selon un planning très précis et prévu à l'avance, le principe d'organisation est le suivant :
- Une excursion, fixée par une date de départ et une date de retour, porte un nom (Circuit du Ceci, de Cela, etc…). Chaque excursion possède un point de départ situé dans une région donnée et un point d'arrivée situé soit dans la même ou dans une autre région.
- Chaque excursion possède un tarif déterminé et un nombre maximum de randonneurs à ne pas dépasser.
- Un randonneur peut s'inscrire à plusieurs excursions s'il le souhaite.
- Enfin, une excursion est menée par un ou plusieurs guides, et chacun d'entre eux possède un numéro, un nom, et un téléphone portable pour être joignable à tout moment par les randonneurs.

4 : Gestion des emprunts dans une bibliothèque

- Une bibliothèque gère les emprunts des livres de ses adhérents.

- Les livres ont un titre, un auteur, un éditeur, une année d'édition, un ISBN (c'est l'identifiant du livre édité, quel que soit l'exemplaire).
- On peut avoir plusieurs exemplaires du même livre (plusieurs livres identiques) mais aussi plusieurs éditions différentes de la même œuvre.
- Une œuvre est caractérisée par son titre, son auteur (on considère qu'il n'y en a qu'un ou que c'est une œuvre anonyme), une langue originale (qui n'est pas forcément connue), une date de première édition (qui n'est pas forcément connue).
- Une édition est caractérisée par l'année d'édition, un ISBN (c'est l'identification de l'édition du livre), une langue.

- Les adhérents ont un nom, une adresse, une adresse mail et éventuellement un téléphone.

- On souhaite archiver tous les emprunts. Un livre ne peut pas être rendu le jour même de son emprunt. La durée maximum d'emprunt est de 21 jours. C'est une règle qui peut changer. Un adhérent ne peut pas avoir plus de 15 emprunts en cours. Tout retard dans les rendus bloque la possibilité de nouveaux emprunts.

- Les livres sont rattachés à des genres (jeunesse, SF, fantastique, roman, bande dessinée, philosophie, politique, histoire, etc.). Une œuvre peut être rattachée à plusieurs genres.

- À tout moment, on connaît le nombre de livres actuellement empruntés par chaque adhérent ainsi que le nombre de livres actuellement en retard de rendu.

- La bibliothèque souhaite pouvoir faire des statistiques sur la pratique des abonnés (nombre de livres empruntés par an, répartition des emprunts par genre, nombre d'emprunts par livre, etc.)

- De plus, les abonnés peuvent commander des livres. Ils peuvent en commander 5 au maximum. Une commande peut être annulée ou honorée si le livre commandé a finalement été emprunté. La bibliothèque souhaite garder la trace de toutes les commandes effectuées, qu'elles aient été annulées ou honorée.

5 : Application de ventes aux enchères

- On souhaite développer un système d'enchères électroniques. Les premières fonctionnalités décrites sont les suivantes :

- Tout utilisateur du système (vendeur ou acheteur) doit préalablement s'inscrire en fournissant une adresse email qui lui servira d'identificateur unique pour ses futures interventions, un mot de passe choisi par lui qui lui servira d'authentification pour ses futures interventions et des renseignements d'état-civil : nom, prénom, adresse. Une fois inscrit, l'utilisateur peut mettre à jour son profil.

- Les utilisateurs peuvent mettre en vente des objets. Pour cela, ils doivent fournir leur identificateur, leur mot de passe, la catégorie de l'objet pour une recherche par thème, un intitulé court de l'objet, une description détaillée de l'objet avec ses défauts et ses qualités, une mise à prix en euros, la date et heure de début de l'enchère, la date et heure de fin, une ou plusieurs photos s'il le souhaite. Le système fournit en retour un numéro de lot unique pour l'objet à vendre.

- Un utilisateur peut consulter les objets qu'il a proposé à la vente, l'historique et le résultat des enchères sur cet objet. Il peut aussi consulter toutes les ventes auxquelles il a participé.

- Un utilisateur ayant proposé un objet à la vente peut réviser la mise à prix à la baisse tant qu'il n'y a aucune enchère sur l'objet et que l'enchère est encore en cours.

- Les utilisateurs peuvent effectuer des recherches parmi les objets mis en vente selon les critères suivants : catégorie de l'objet, mots de la description de l'objet, fourchette de prix.

- Un utilisateur peut enchérir sur un objet. Pour être valable, l'enchère doit être supérieure de 0,5 euro à la plus haute enchère déjà effectuée ou à la mise à prix si c'est la première enchère ; l'annonce ne doit pas être expirée ; l'utilisateur ne doit pas être le vendeur de l'objet. Pour suivre l'évolution de l'enchère, le système garde aussi la date de chaque enchère.

- Quand l'échéance de fin d'enchère tombe, le système informe le vendeur dans tous les cas, et l'acheteur s'il y a lieu, du résultat de l'enchère.

- Si un vendeur veut remettre en vente un objet non vendu, il doit créer un nouvel objet dans le système avec ses caractéristiques propres.

- Par ailleurs, les utilisateurs peuvent aussi sélectionner certains objets dont ils veulent suivre l'enchère. Le système enverra un mail d'alerte à l'utilisateur 24 heures avant la fin de l'enchère. L'utilisateur pourra supprimer les éléments de sa sélection.

2 : COMPLEMENTS THEORIQUES – 1 -
LES 4 TYPES DE RELATIONS ENTRE LES CLASSES

- On retrouve cette présentation, plus détaillée, dans le cours de P.O.O.
- On y trouve des exercices orientés « codage ».

0 - Présentation des relations

- Les objets, et les classes qui leur correspondent, peuvent avoir des relations entre elles.
- Il y a principalement 4 types de relations entre les classes :

  - L'agrégation
  - La composition
  - L'énumération
  - L'héritage

- Agrégation et composition sont 2 relations assez semblables : c'est une relation « avoir ».
- L'énumération est une sorte de composition.
- L'héritage est une relation à part : c'est une relation « est un ».
- On va expliquer ce que sont ces 4 relations avec des exemples. Il faut déjà retenir le vocabulaire.

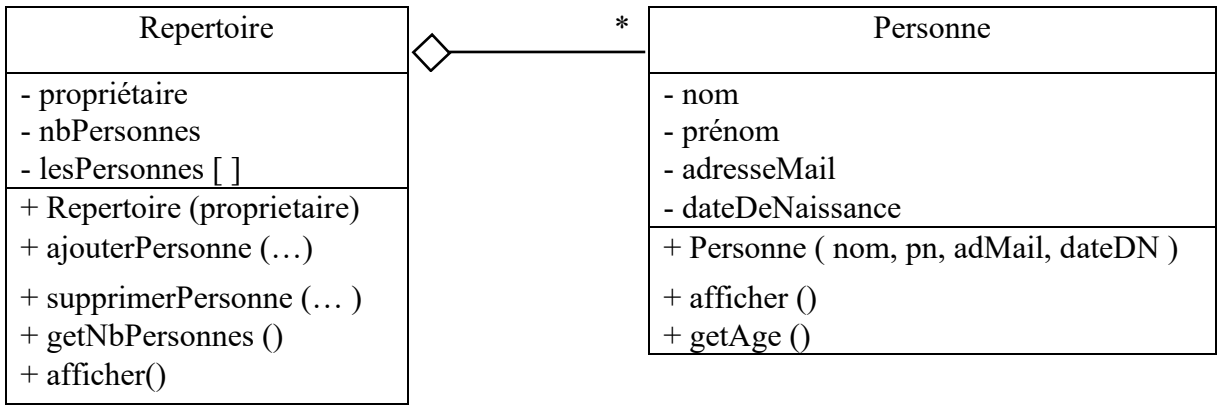
1 - Agrégation

Rappel : il y a principalement 4 types de relations entre les classes :

- L'agrégation
- La composition
- L'énumération
- L'héritage

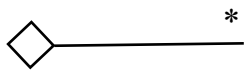
Exemple d'agrégation : un répertoire de personnes

Diagramme de classes UML



- Le répertoire **agrège** des personnes. Cela veut dire qu'il **contient une liste** (collection, tableau, etc.) de personnes.
- C'est une **agrégation** et pas une composition. Cela veut dire que **si on supprime le répertoire**, on ne doit **pas forcément supprimer les personnes** : elles sont peut-être utilisées dans un autre répertoire.
- Syntaxe UML de l'agrégation :

  ⇒ La relation entre Repertoire et Personne matérialisée graphiquement par le lien :

  ⇒ on met un losange du côté de **l'agrégat** (le ◇ **conteneur)** :

  ⇒ on met une étoile du côté des **éléments agrégé (**les **contenus)** : « * ». « * » veut dire plusieurs : le répertoire agrège (contient) plusieurs personnes.

- Subtilités de syntaxe :

  ⇒ La collection s'appelle: « lesPersonnes ». C'est une convention simple : « les » pour la collections, suivi du nom de la classe. Des crochets « [ ] » pour montrer que c'est une collection.

  ⇒ On n'est pas obligé de la mettre dans le diagramme de classe : la relation de composition dit qu'elle existe.

  ⇒ On met les paramètres des méthodes si c'est facile, rien s'il n'y en a pas.

  ⇒ On met « … » comme paramètres de méthodes si on ne sait pas encore ce qu'ils seront.

- Relation AVOIR

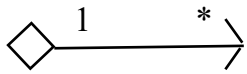
  ⇒ Une agrégation, comme une composition, ou les relations entre classes qui ne sont pas des héritages, est une relation AVOIR :

  ⇒ Un répertoire « a des » personnes.

  ⇒ Une personne a 1 répertoire (ou plusieurs, comme on veut).

- Subtilités UML

  ⇒ On ne s'intéresse pas aux répertoires des personnes, mais aux personnes dans le répertoire : la relation ne va que dans un sens.

  ⇒ On peut considérer que l'agrégation c'est le 1 et la flèche par défaut.

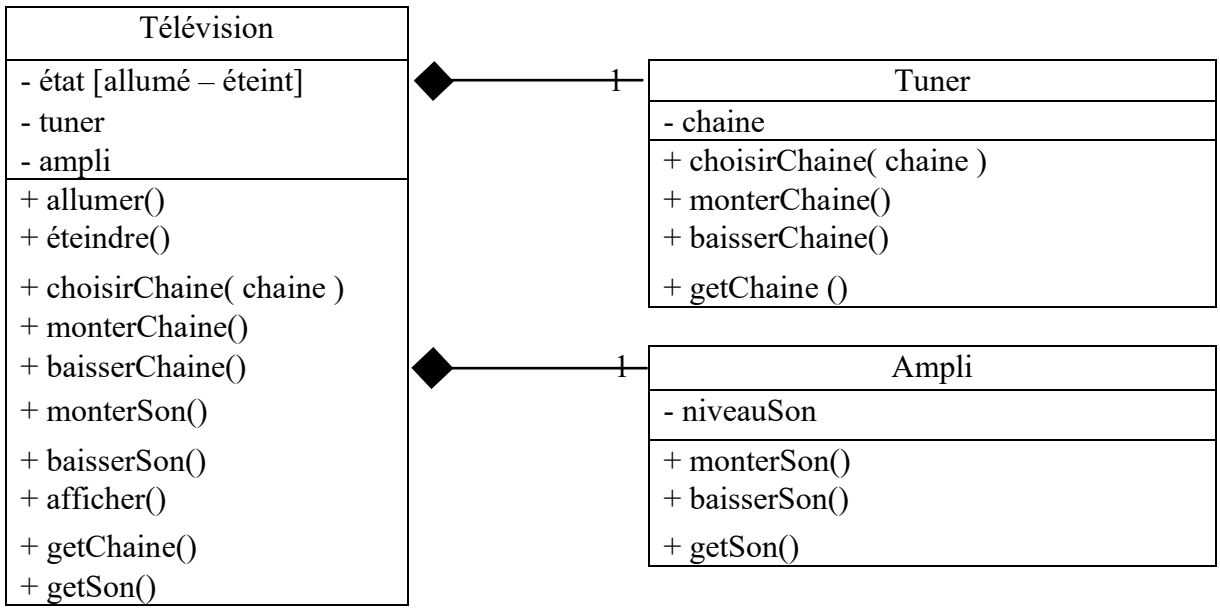
2 - Composition

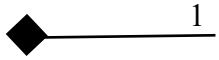
Rappel : il y a principalement 4 types de relations entre les classes :

- L'agrégation
- La composition
- L'énumération
- L'héritage

Exemple de composition : la télévision et ses composants (tuner et ampli)

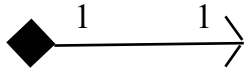
Diagramme de classes



- La classe Télévision est composée d'un objet « tuner » et d'un objet « ampli ».
- Cette relation est matérialisée dans le diagramme par le lien : 

Principes de la composition

- La télévision est composée d'un ampli et d'un tuner. Cela veut dire qu'elle contient un ampli et un tuner.
- C'est une **composition** et pas une agrégation. Cela veut dire que **si on supprime la télévision**, **on supprime aussi l'ampli et le tuner**. L'ampli et le tuner ne sont utilisés que par la télévision.
- Syntaxe UML :
  - ⇒ on met un losange du côté du **composé** (le **conteneur)** : ◆
  - ⇒ on met une 1 du côté des **composants**, les **contenus** : la télévision contient 1 tuner et 1 ampli. Ca pourrait être 2, 3, plusieurs.
- Subtilités
  - ⇒ On ne met **rien du côté du composé** : c'est toujours 1 par défaut.
  - ⇒ Si on ne met **rien du côté du composant**, c'est 1 par défaut.
- Relation AVOIR
  - ⇒ Une composition, comme une agrégation, ou les relations entre classes qui ne sont pas des héritages, est une relation AVOIR :
  - ⇒ Une télévision « a un » tuner.
  - ⇒ Un tuner « a 1 » télévision.
- Subtilités UML
  - ⇒ On ne s'intéresse pas à la télévision du tuner, mais au tuner de la télévision : la relation ne va que dans un sens.



  - ⇒ On peut considérer que la composition, c'est les 1 et la flèche par défaut.

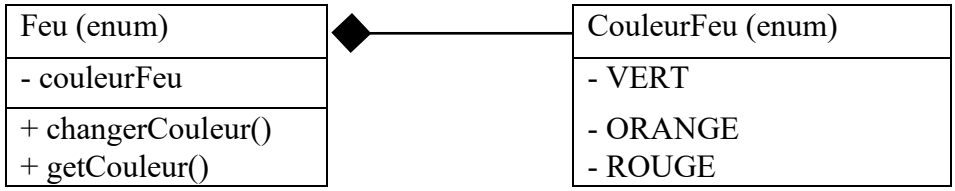
3 - Enumération

Rappel : il y a principalement 4 types de relations entre les classes :

- L'agrégation
- La composition
- L'énumération
- L'héritage

Classe énumération

Exemple



Technique

- On définit l'énumération comme une classe particulière : une classe d'énumération.
- Les attributs de l'énumération sont les valeurs possibles.
- Il n'y a pas de méthodes dans l'énumération.

Principes

- En POO, on utilise souvent des classes d'énumération qui nous permettent de définir des petites listes de valeurs constantes.
- Ces valeurs sont utilisées pour fixer les valeurs possibles pour un attribut et ainsi éviter les erreurs.
- L'énumération devient alors un type pour un attribut.

4 – Héritage (retour au I-4)

Rappel : il y a principalement 4 types de relations entre les classes :

- L'agrégation
- La composition
- L'énumération
- L'héritage

Exemple conduisant à l'héritage

- On a une classe Chat et une classe Chien :

Chat
- nom - age - propriétaire - raceChat
+ crier() + manger() + ronronner()

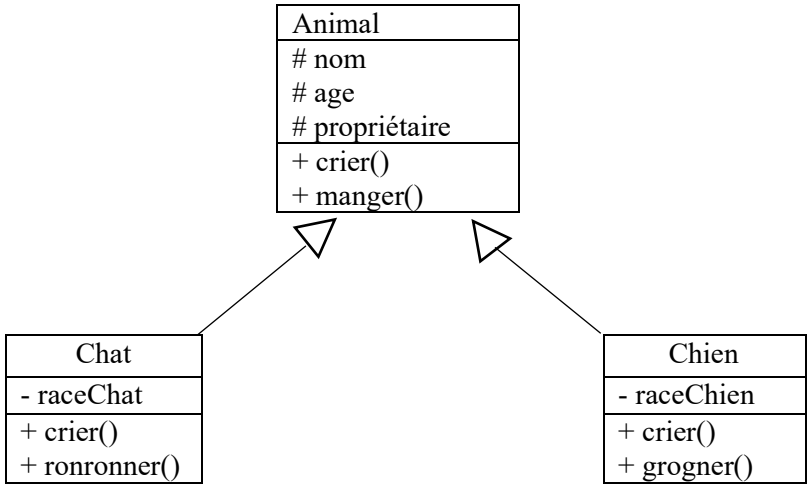
Chien
- nom - age - propriétaire - raceChien
+ crier() + manger() + grogner()

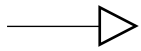
- On voit que les 2 classes ont des attributs et des méthodes en commun.
- Concernant les méthodes, on peut considérer que **manger()** est la même méthode pour le chat et pour le chien.
- Par contre, **crier()** est différente : le chat miaule et le chien aboie !

Principe de l'héritage : mettre le commun dans une classe de base

- Dès que 2 classes ont des attributs ou des méthodes en commun :
  - ⇒ on retire ces attributs et méthodes de leurs classes de départ pour créer une nouvelle classe avec les attributs et méthodes communes.
  - ⇒ On fait ensuite « hériter » les classes de départ de la nouvelle classe.

- Les classes Chat et Chien héritent de la classe Animal :

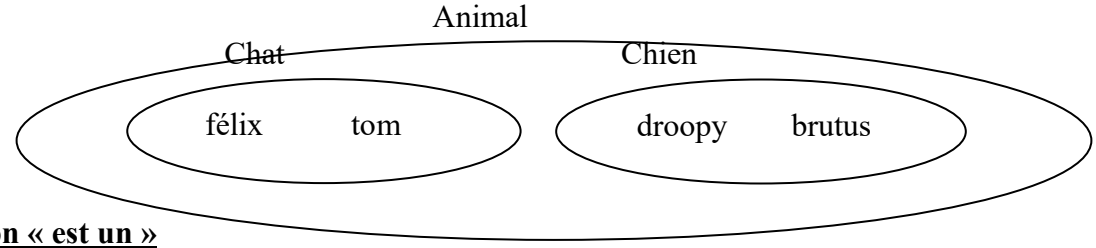


- La nouvelle classe est appelée : **classe de base** (ou **classe-mère** ou **classe-parent**).
- Les classes de départ sont appelées : **classe dérivée** (ou **classe-fille** ou **classe-enfant**).
- On dit que **les classes-enfant « héritent » de la classe parent.** En effet, tous les attributs et les méthodes de la classe-parent seront accessibles par la classe enfant.
- La flèche avec un triangle blanc au bout est une flèche d'héritage : 

Inclusion ensembliste, relation « est-un »

Inclusion ensembliste

L'inclusion ensembliste est ce qui traduit le mieux la notion d'héritage.
Un ensemble correspond à une classe. Un élément d'ensemble correspond à un objet. Un ensemble inclut dans un autre correspond à un héritage.



Relation « est un »

L'héritage, la flèche à triangle, l'inclusion sont des **relations « est un »**.
Un Chat « est un » animal. Félix « est un » Chat. félix est un Animal. Etc.

Utilité

L'héritage est utilisé pour :

- **Factoriser le code** : ça évite d'avoir à écrire deux fois la même chose !
- **Utiliser un code existant déjà** : c'est une forme de factorisation.
- **Le polymorphisme** (possibilité qu'une même méthode se réalise différemment).

5 - UML – Exercices de conception de POO UML

Équation du second degré

Faites le diagramme de classes permettant de traiter l'étape 1 :

Étape 1 :

Créer une classe permettant de résoudre une équation du 2è degré.
Le but est d'arriver à cet usage :

```
Eq2 eq2 = new Eq2(1, -3, 2);
System.out.println(eq2); // Équation du second degré, 2 solutions : 1 et 2.
```

Ou :

```
Eq2 eq2 = new Eq2(1, 2, 5);
System.out.println(eq2); // Équation du second degré : 0 solution.
```

Ou :

```
Eq2 eq2 = new Eq2(0, 2, -6);
System.out.println(eq2); // Équation du premier degré : 1 solution : 3.
```

Complétez le diagramme de classes précédant pour traiter l'étape 2 :

Étape 2 :

On veut pouvoir faire :

```
Eq1 eq1 = new Eq1(2, -6);
System.out.println(eq1); // Équation du premier degré : 1 solution : 3.
```

3 : COMPLEMENTS THEORIQUES - 2 - CLASSE ABSTRAITES ET INTERFACES

- On retrouve cette présentation, plus détaillée, dans le cours de P.O.O.
- Les notions de classes abstraites et d'interface relève de la technique et pas du fonctionnel.

1 – Classe et méthode abstraite

Classe abstraite

Définition

- **Une classe abstraite** est une classe qui **ne peut pas être instanciée** : on ne peut pas créer d'objet correspondant à cette classe. Ce n'est donc pas un moule !

Pourquoi faire ?

- La classe Animal est utile pour **factoriser ce qui est commun**.
- La classe Animal pourrait être rendue abstraite pour éviter qu'on puisse créer des objets à partir de cette classe.
- La classe abstraite devient une partie du moule de la classe enfant.
- Ca permet de factoriser du code et d'être plus proche du réel (il y a des animaux, des chats et des chiens).

Dans le code

- En Java, on ajoute le mot clé « abstract » devant le mot clé « class ».
- En Python… le Python ne gère pas les classes abstraites. Il faudra faire « comme si ».

Méthode abstraite

Définition

- Une méthode abstraite est une méthode qui n'a qu'une en-tête mais pas de corps.
- Quand une classe contient une méthode abstraite, elle est forcément abstraite.

A quoi sert-elle ?

- Elle sert à obliger les classes enfants qui hériteront de la classe mère abstraite à définir la méthode abstraite. C'est donc une obligation qui est faite au programmeur qui utilise la classe abstraite.

Pourquoi faire ?

- Si on reprend le **modèle** Animal – Chat – Chien, on a la méthode crier () qui se trouve dans la classe animal et qui devrait être abstraite.
- Il faut avoir cette méthode dans la classe crier () **pour pouvoir utiliser le polymorphisme.**

2 - Interface

- Les interfaces sont des techniques utiles pour avoir un code le plus générique possible.

- Elles ne contiennent **que des méthodes abstraites** : aucun attribut d'instance.
- C'est donc un cas particulier de classe abstraite.
- La différence est qu' **une classe peut « implémenter » une ou plusieurs interfaces** alors que l'héritage à tout intérêt à être unique pour éviter les incohérences (et est unique en Java).
- Ca l'oblige donc à écrire le code des méthodes abstraites qui se trouvent dans l'interface.

- C'est une **notion complexe** qui permet de mettre en œuvre au mieux le **polymorphisme** et d'avoir un **code facile à faire évoluer**.

4 : COMPLEMENTS THEORIQUES - 3 -
NOTION DE DEPENDANCE

- On retrouve cette présentation, plus détaillée, dans le cours de P.O.O.
- La notion de dépendances relève de la technique et pas du fonctionnel.

1 – Dépendance

Définition

- **Une classe A dépend** d'une autre classe B quand elle l'utilise dans son code.

3 dépendances marquées par des associations déjà vues :

- Avoir un attribut de la classe B, donc qu'il y ait une association entre A et B.
  A ——1——> B : A contient un attribut b B, donc dépend de B
- Hériter de la classe B : A sera en partie constituer par B, donc en dépend.
  A ——▷ B : A hérite de B, donc dépend de B
- Implémenter une interface : si A implémente B, alors A dépend de B (la dépendance est limitée à l'en-tête des méthodes définies dans B).
  A - - - - - - - -▷ B : A implémente B, donc dépend de B

2 nouvelle dépendances qui se traduiront par une flèche orientée en pointillés

A - - - - - - - - -> B : A dépend de B

- Il y a dépendance si une méthode de A à un B en paramètre
- Il y a dépendance si une méthode de A instancie un B dans son code