Modéliser avec le langage UML
Diagramme de Classes

Laure Gérard

Laure.gerard@grenoble-epsi.fr

De l’approche fonctionnelle à l’approche objet

Approche fonctionnelle – Approche intuitive

- Raisonnement en terme de fonctions
- Séparation des données et du code de traitement

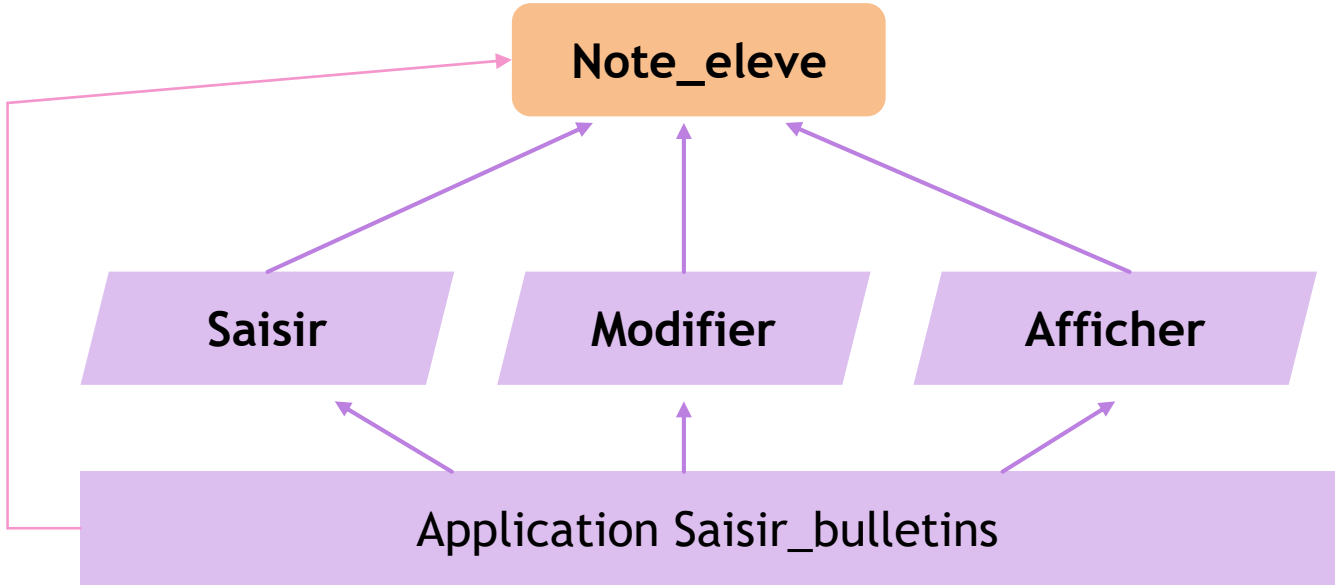
- Logiciel :
  - Hiérarchie de fonctions qui fournissent les services attendus
  - Ensemble de données qui représentent les concepts manipulés

- Avantage : factorisation des comportements communs du logiciel c'est-à-dire la réutilisation de fonctions existantes (génériques) pour réaliser une nouvelle fonction.

Approche fonctionnelle – Exemple

- **Ecrire** le programme permettant
  - Saisir des notes d'une classe (25 élèves).
  - Afficher des notes (Nom de l'élève, note et appréciation).
  - Rechercher et modifier la note d'un élève.
- Pour la saisie des notes, le professeur devra entrer le nom de l'élève suivi de sa note. Pour cela, **définir** une structure Note_eleve composée du nom de l'élève, de sa note et de son appréciation.

Approche fonctionnelle - Exemple




```
14  // Définition de la structure Note_eleve
15  struct Note_eleve{
16      char nom[100];
17      int note;
18      Appreciation appreciation;
19  };
```

```
void saisieNotes(Note_eleve *note);
void modifier_note(Note_eleve *note);
void afficherBulletin(Note_eleve *n);
```

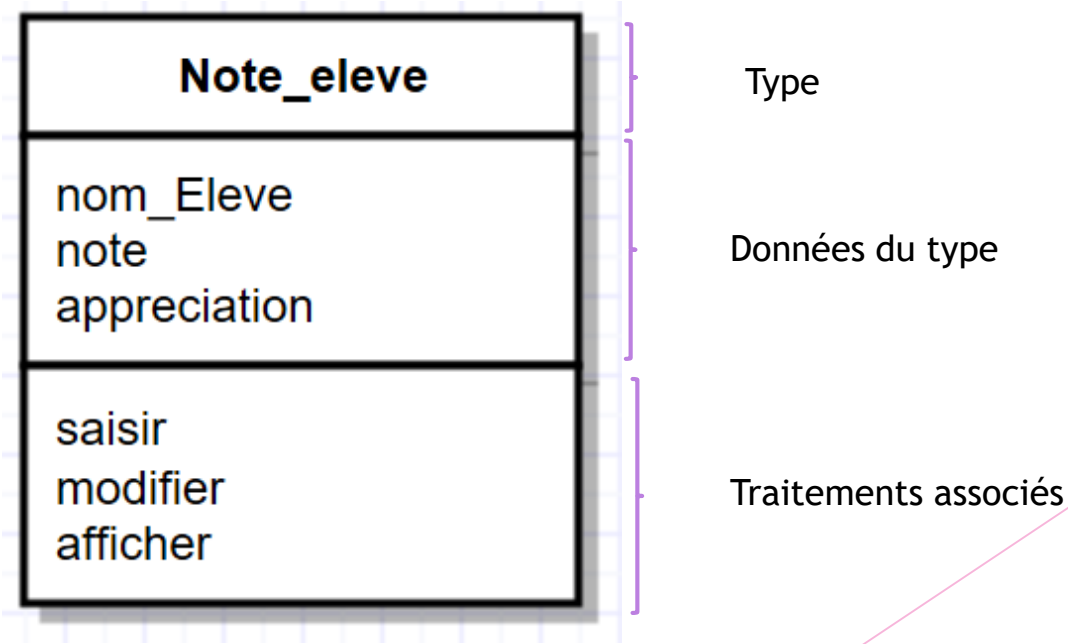
Approche fonctionnelle – les limites

- Ensemble de modules fonctionnels qui manipulent des données
  - Communication entre les fonctions par passage de paramètres ou variables globales
  - => accès libre aux données par n'importe quelle fonction

- Evolution difficile
  - Fonctions interdépendantes : une mise à jour simple peut impacter un grand nombre de fonctions
  - Fonctions développées pour un type de données : une évolution du logiciel pour prendre en compte de nouveaux types de données nécessite
    - L'évolution des structure de données manipulées par les fonctions
    - L'adaptation des traitements

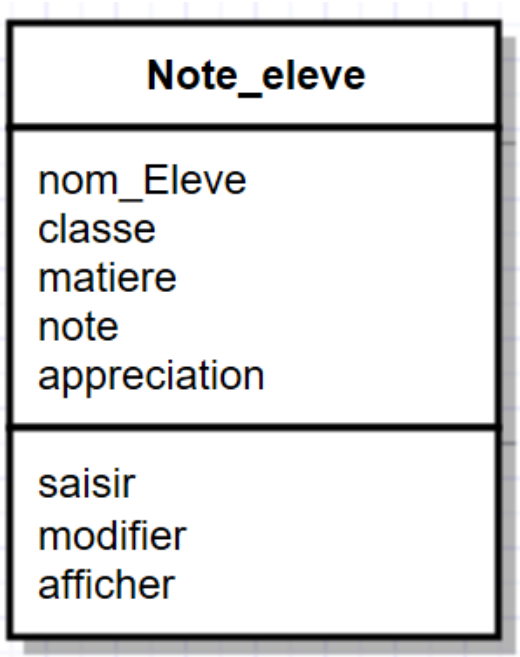
Comment améliorer cette approche ?

- Centraliser les données d'un type et les traitements associés au sein d'une même unité physique



Comment améliorer cette approche ?

- Centraliser les données d'un type et les traitements associés au sein d'une même unité physique



Objet

Transformation d'une **structure** de données **manipulée** par des **fonctions** en une **entité autonome** (objet) regroupant un ensemble de propriétés (attributs) et les traitements associés.

Approche Objet – Définitions des concepts

- Une approche orientée objet
  - Organisation d'un logiciel sous la forme d'une collection d'entités indépendantes (objets) regroupant une structure de données et les différents comportements/traitements.

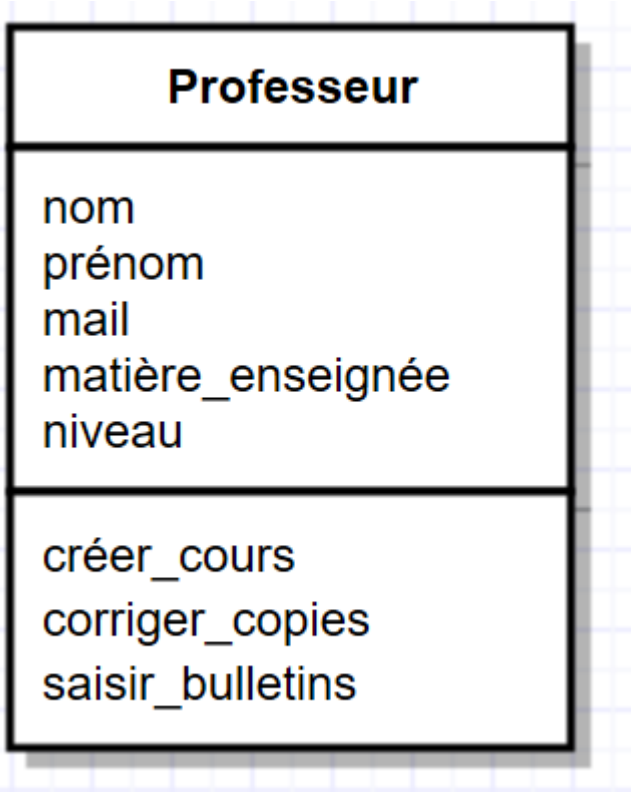
- Un objet
  - Un entité définie par un nom
  - Un ensemble d'attributs caractérisant son état
  - Un ensemble d'opérations (méthodes) définissant son comportement

Approche Objet – Définitions des concepts

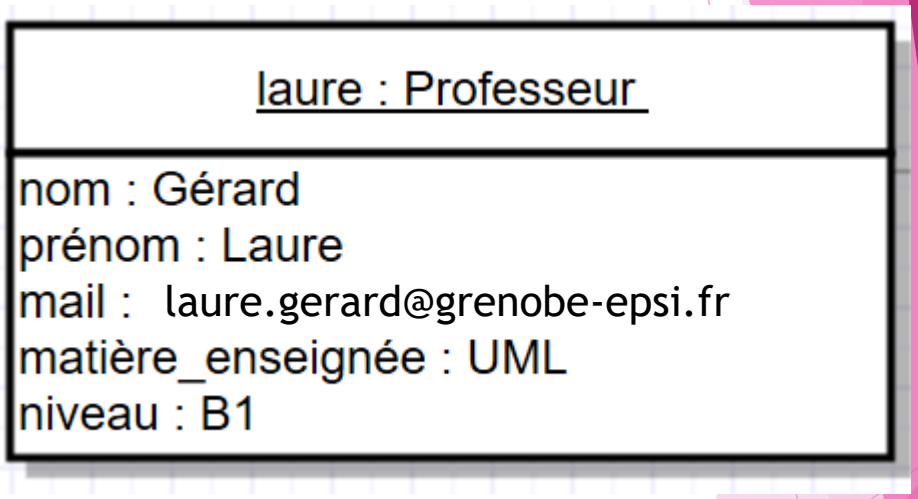
- Classe
  - Description abstraite d'un ensemble d'objets possédant les mêmes propriétés (attributs et méthodes).
  - Exemples : classe Etudiant, classe Professeur

- Un objet est une instance de classe, c'est-à-dire une occurrence d'un type abstrait.
  - Exemples : Laure Gérard est un objet instance de la classe Professeur. Vous, en tant qu'étudiant EPSI, vous êtes des objets instances de la classe Etudiant.

Approche Objet – Définitions des concepts



Classe Professeur qui regroupe les données et les traitements



objet laure qui est une instance de la classe Professeur

Approche Objet – Définitions des concepts

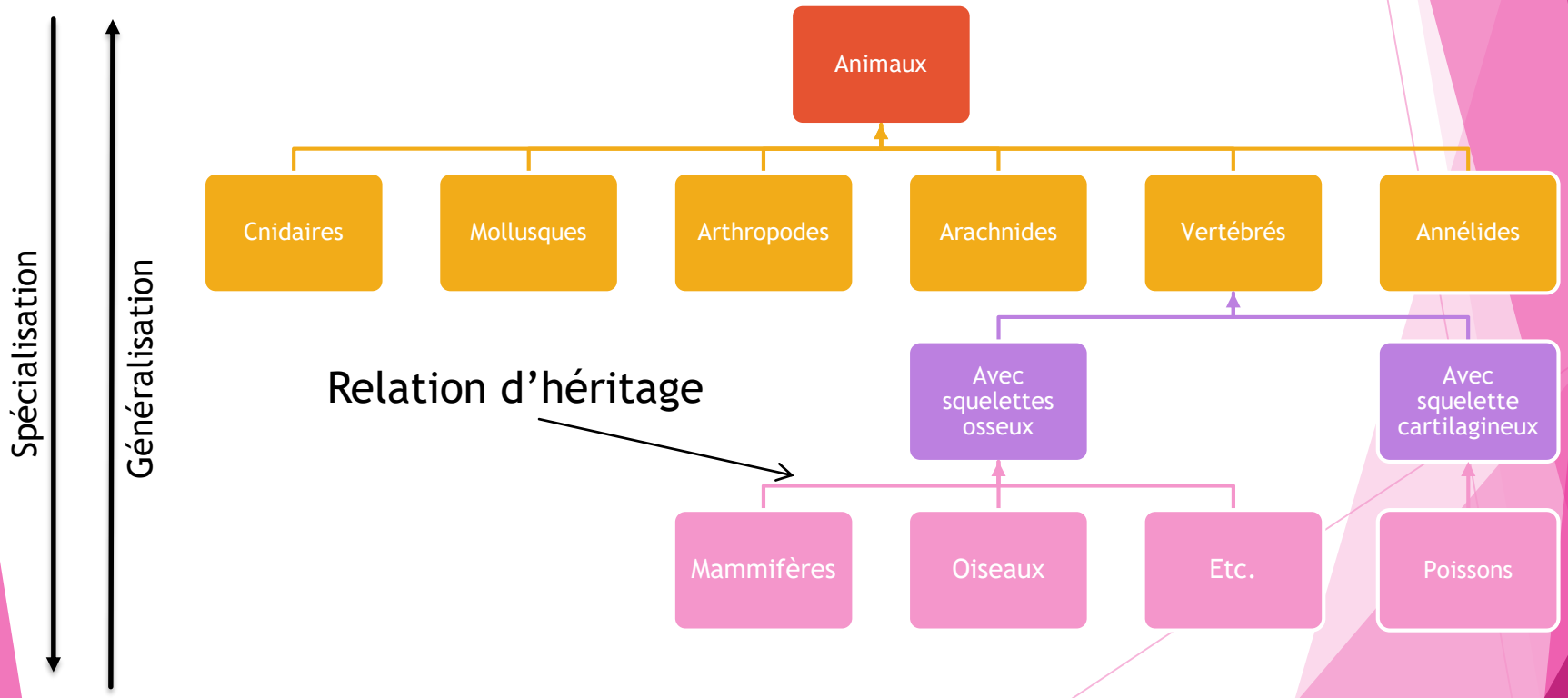
- L'**Encapsulation** consiste à masquer les détails d'implémentation d'un objet, en définissant une interface (vue externe d'un objet qui définit les services accessibles aux utilisateurs de l'objet)
  - Facilité l'évolution d'une application, car stabilise l'utilisation des objets.
  - Garantit l'intégrité des données, car interdit l'accès direct aux attributs des objets par exemple.

Approche Objet – Définitions des concepts

- **Héritage**
  - Mécanisme de transmission des propriétés d'une classe, c'est-à-dire attributs et méthodes, vers une sous-classe.
  - Evite la duplication et encourage la réutilisation.
  - Conséquences :
    - Une classe peut être spécialisée en d'autres classes pour y ajouter des caractéristiques spécifiques.
    - Plusieurs classes peuvent être généralisées en une classe qui les factorise, pour regrouper les caractéristiques communes d'un ensemble de classes.

Approche Objet – Définitions des concepts

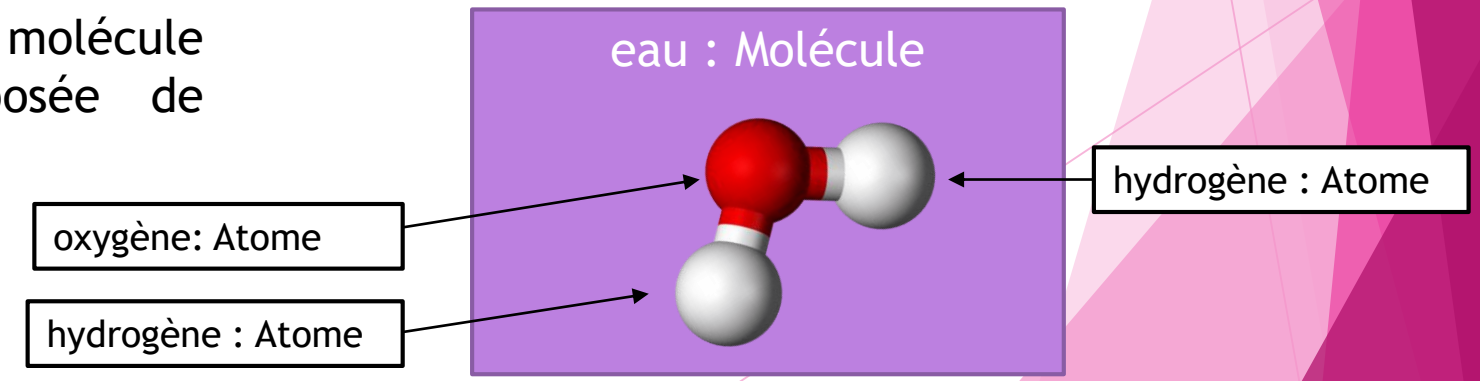
- Exemple d'héritage : classification des animaux



Approche Objet – Définitions des concepts

- **Agrégation**
  - Relation entre deux classes spécifiant que les objets d'une classe sont des composants de l'autre classe.
  - Permet
    - de définir des objets composés d'autres objets.
    - d'assembler des objets de base pour construire des objets plus complexes.

Exemple : molécule d'eau composée de trois atomes



Approche objet - Histoire

- Concepts stables issus du terrain
  - 1967 : 1e langage de programmation (Simula) à implémenter le concept de type abstrait, à l'aide de classes.
  - 1976 : 1e implémentation des concepts fondateurs de l'approche objet par Smalltalk.
  - 1980 : 1e compilateur C++
  - Depuis de nombreux langages orientés objets académiques ont validés les concepts objets tels que Objectif C, Eiffel, etc.

Approche objet – Besoins

- Un langage pour exprimer les concepts utilisés pour
  - Représenter des concepts abstraits
  - Limiter les ambiguïtés et les incompréhensions
  - Faciliter l'analyse
- Une démarche d'analyse et de conception pour
  - Réaliser une conception et une implémentation objet sans effectuer une analyse fonctionnelle
  - Définir les vues qui permettent de couvrir tous les aspects d'un système.

Méthodes objet et UML

Méthodes et consensus

- Années 70 : premières méthodes d'analyse proposant une découpe fonctionnelle et hiérarchique d'un système.
- Années 80 : approche systémique offrant une modélisation des données et des traitements.
- 1990-1995 : premières méthodes objet
- 1995 : premiers consensus
  - OMT de Rumbaugh : vues statiques, dynamiques et fonctionnelles d'un système.
  - OOD de Booch : vues logiques et physiques du système
  - OOSE de Jacobson : méthodologie reposant sur l'analyse des besoins des utilisateurs, couvrant tout le cycle de développement.
- 1995-1997 : Unification et normalisation des méthodes

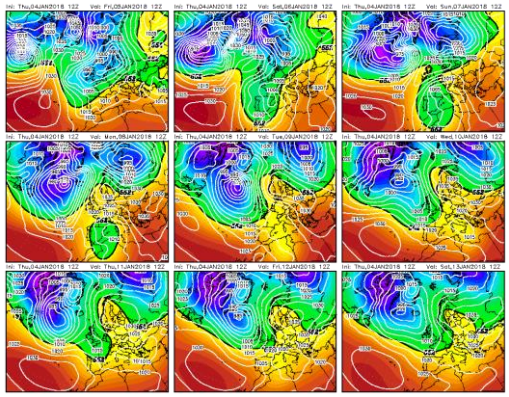
UML (Unified Modeling Langage)

- Langage visuel formel et normalisé
- Cadre de l'analyse objet à travers
  - Différentes vues complémentaires d'un système
  - Plusieurs niveaux d'abstraction permettant de mieux appréhender la complexité dans l'expression des solutions objets.
- Support de communication performant
  - Notation graphique permettant d'exprimer visuellement une solution objet.
  - Aspect formel de la notation limite les ambiguïtés et incompréhensions
  - Aspect visuel facilite la comparaison et l'évaluation de solutions.
  - Langage universel

UML – Démarche de modélisation

Qu’est-ce qu’un modèle ?

- Abstraction de la réalité
- Vue subjective mais pertinente de la réalité
- Exemple : modèle météorologique



Observation de données
Identification des caractéristiques importantes

Prédiction des conditions climatiques pour les jours à venir

Démarche – Préconisations des auteurs de l'UML

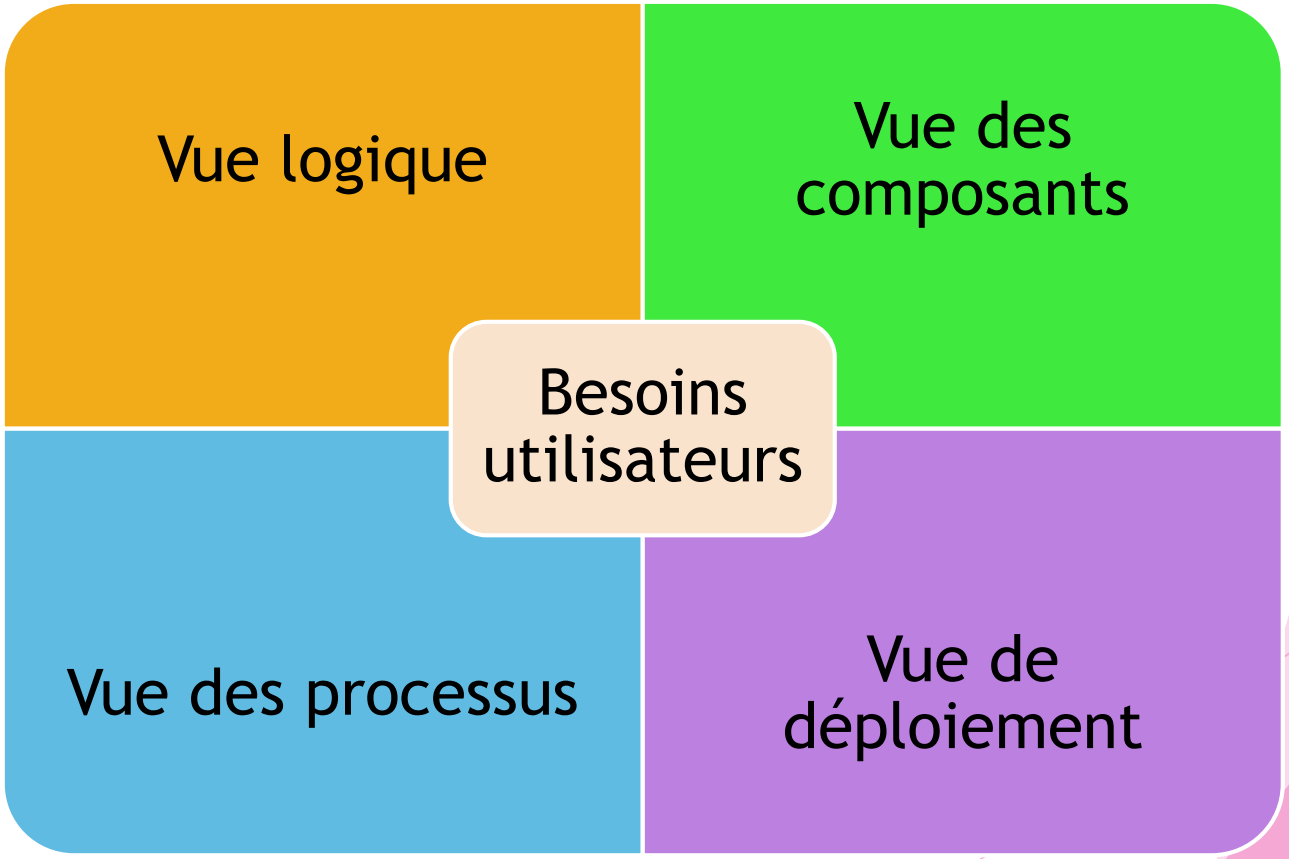
- La démarche doit être
  - **Itérative** et **incrémentale**
  - Guidée par les **besoins des utilisateurs** du système
  - Centrée sur l'**architecture logicielle.**

Démarche – Besoins des utilisateurs

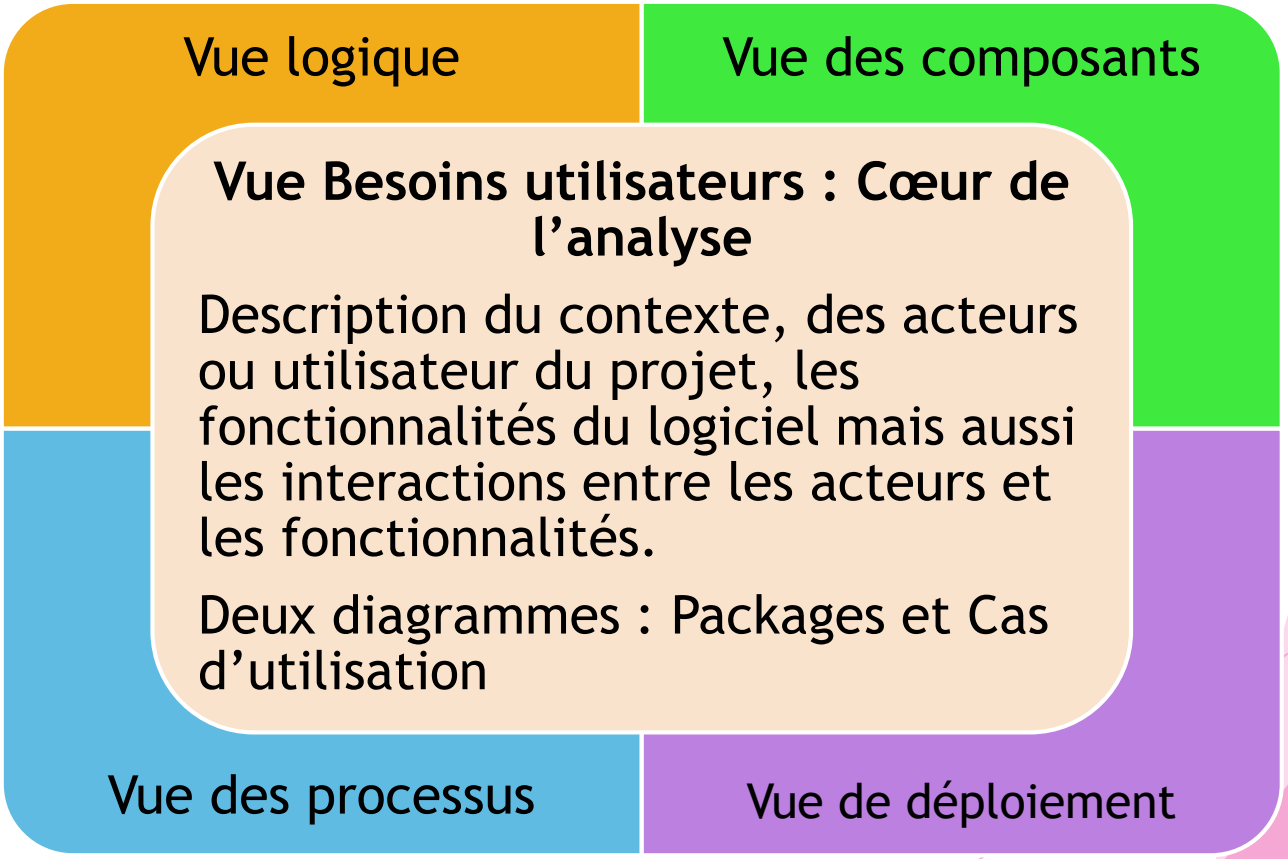
- La définition des modèles guidée par les utilisateurs
  - Objectif du système est de répondre aux besoins des utilisateurs du système

- Besoins utilisateurs = fil conducteur tout au long du cycle de développement.
  - Phases d'analyse : besoins utilisateurs sont clarifiés, affinés et validés.
  - Phases de conception : on s'assure que les besoins utilisateurs ont bien été compris et pris en compte.
  - Phases de test : on vérifie que les besoins des utilisateurs sont satisfaits.

Démarche – un modèle d'architecture



Démarche – un modèle d'architecture



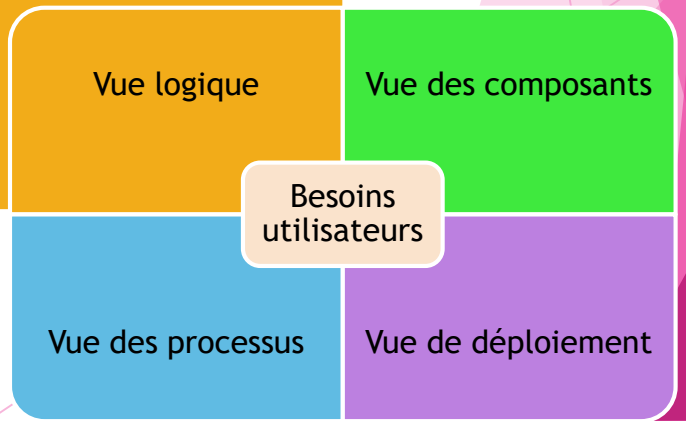
Démarche – un modèle d'architecture

Vue logique : vue de haut niveau

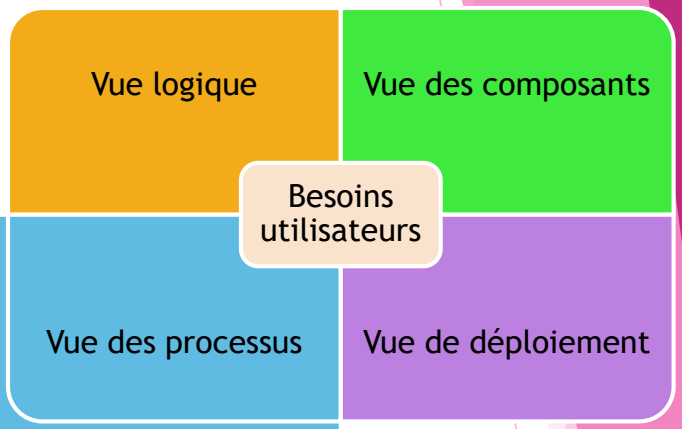
- Identification des éléments du domaine liés au(x) métier(s), leurs relations et interactions.
- Organisation des éléments du domaine en « catégorie » pour une meilleure organisation du projet.

Deux diagrammes :

- Diagramme de classes
- Diagramme d'objets



Démarche – un modèle d'architecture



Vue des processus
Mise en évidence de

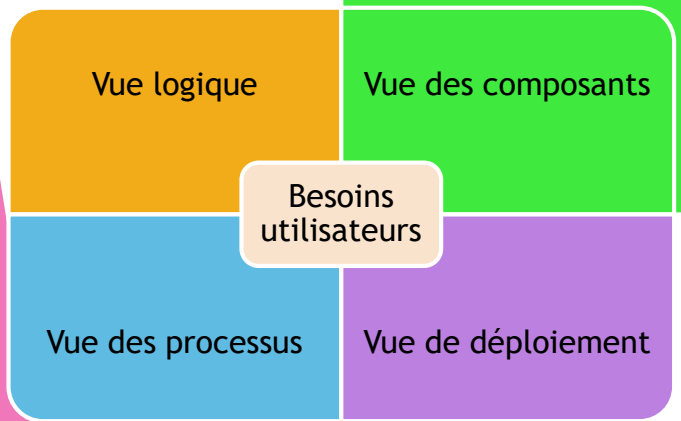
- la décomposition du système en processus et actions ;
- les interactions entre les processus ;
- la synchronisation et la communication des activités parallèles.

Cinq diagrammes : diagramme de séquences, d'activité, de collaboration, d'interaction et de temps

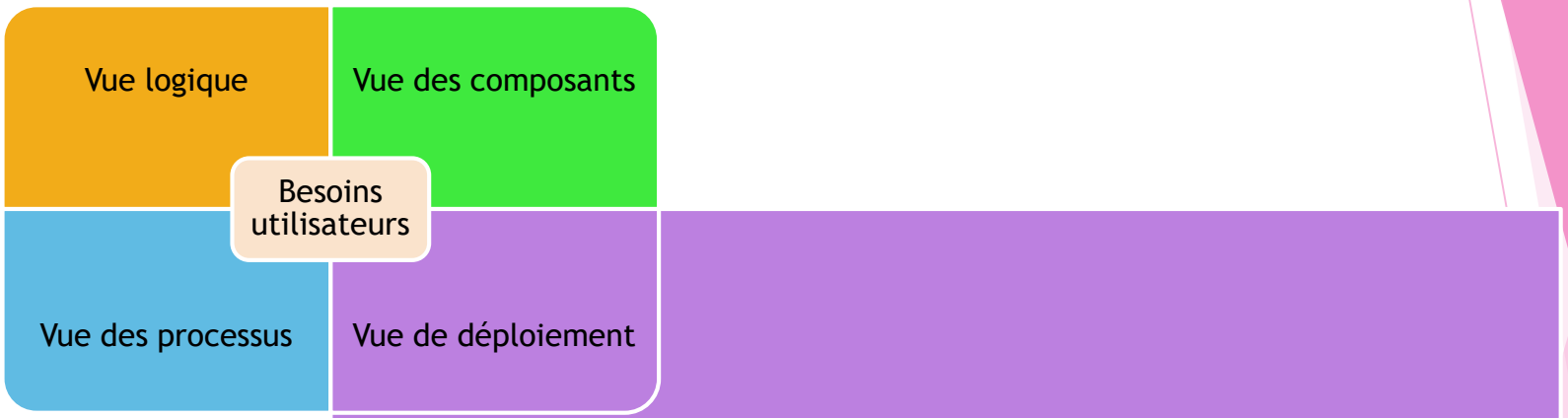
Démarche – un modèle d'architecture

Vue des composants : vue de bas niveau
Mise en évidence des différentes parties qui composeront le futur système (fichiers sources, bibliothèques, bases de données, exécutables, etc.).

Deux diagrammes : diagramme de structure composite et diagramme de composants.



Démarche – un modèle d'architecture

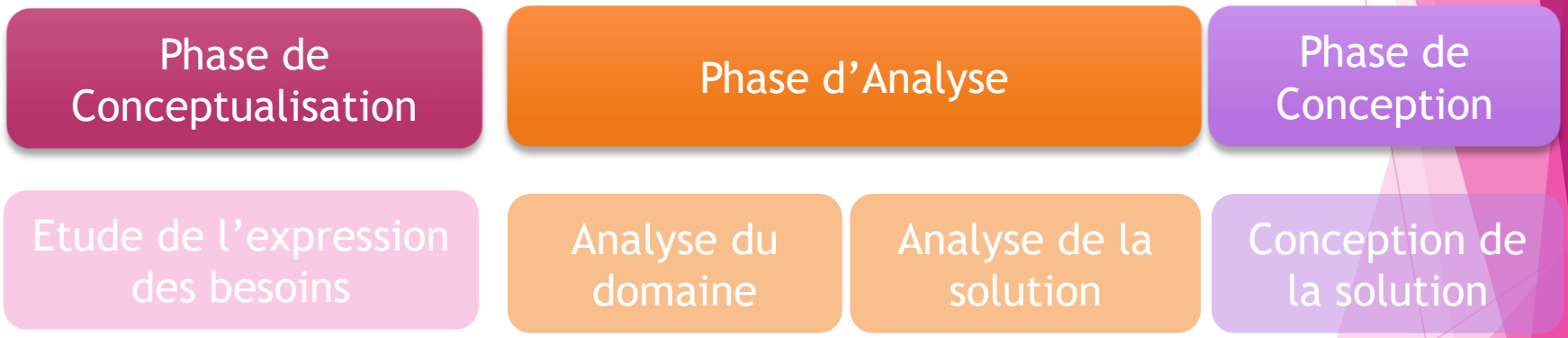


Vue de déploiement
Description des ressources matérielles et de la répartition des parties du logiciel sur ces éléments.

Un diagramme : diagramme de déploiement

Concrètement, comment faire ?

- Dans un processus de développement logiciel



Concrètement, comment faire ?

Phase de Conceptualisation | Phase d'Analyse | Phase de Conception

- Point d'entrée : dossier d'expression des besoins utilisateur / client.
- Objectifs :
  - Définir le contour du système à modéliser.
  - Comprendre et définir les fonctionnalités principales du système afin d'en fournir une meilleure compréhension.
  - Fournir une base à la planification du projet.
- Comment ?
  - On doit capturer les besoins principaux des utilisateurs.
  - Il ne faut pas chercher l'exhaustivité, mais clarifier, filtrer et organiser les besoins.

Concrètement, comment faire ?

Phase de Conceptualisation | **Phase d'Analyse** | Phase de Conception

- Point d'entrée : modèle des besoins clients, c'est-à-dire les cas d'utilisation en UML.
- Objectif : construction de modèles pour bien comprendre les exigences en terme de fonctionnalités.
- Deux phases :
  - Analyse du domaine pour capturer les connaissances générales de l'application
  - Analyse de l'application pour capturer les aspects informatiques de l'application visibles pour les utilisateurs.
- Résultats : modèles de classes (aspects orientés données du système), d'états (aspects temporels, comportementaux et de « contrôle » du système) et d'interactions (collaboration entre les objets).

Concrètement, comment faire ?

Phase de Conceptualisation | Phase d'Analyse | Phase de Conception

- Point d'entrée : modèles d'analyse.
- Objectif : déterminer comment l'application doit être réalisée.
- Deux phases :
  - Conception du système : réaliser l'architecture du système.
  - Conception des classes : affiner les modèles d'analyse.
- Résultats : modèles de classes, d'états et d'interactions détaillés.

Etude de cas

Réalisation du modèle de domaine ou modèle statique d'analyse

Phase de Conceptualisation	Phase d'Analyse	Phase de Conception

Etude de l'expression des besoins	Analyse du domaine	Analyse de la solution	Conception de la solution

Editeurs UML

- StarUML : http://staruml.io/
- ArgoUML : https://argouml.fr.softonic.com/
- BoUML : http://www.bouml.fr/
- Modelio : https://www.modelio.org/
- Gliffy : https://www.gliffy.com/go/html5/launch

Etude de cas : Système de réservation de vol

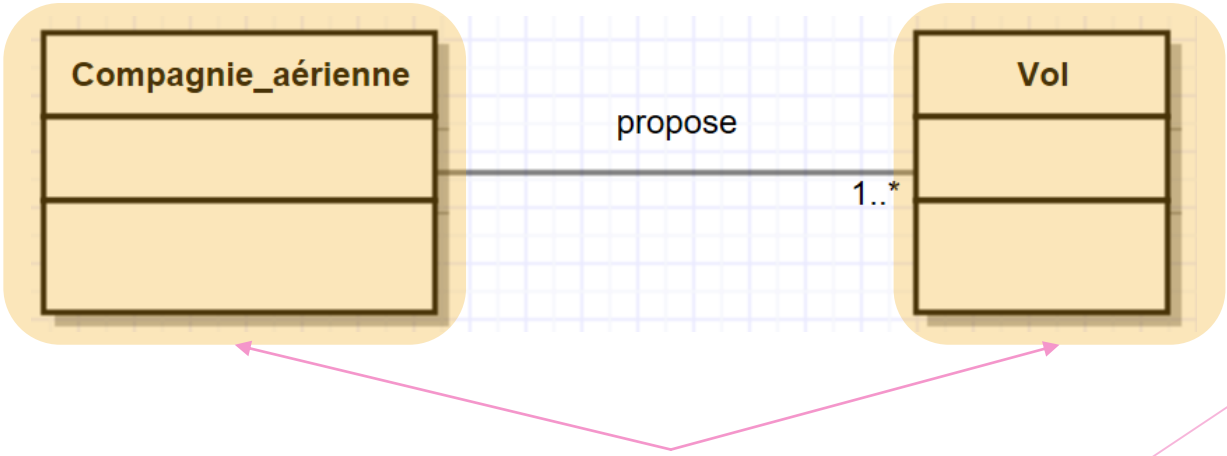
- Connaissances du domaine issues de la phase de conceptualisation :
  - Des compagnies aériennes proposent différents vols.
  - Un vol est ouvert à la réservation et refermé sur ordre de la compagnie.
  - Un vol a un jour et une heure de départ, et un jour et une heure d'arrivée.
  - Un vol a un aéroport de départ et un aéroport d'arrivée.
  - Chaque aéroport dessert une ou plusieurs villes.
  - Un vol peut comporter des escales dans des aéroports.
  - Une escale a une heure d'arrivée et une heure de départ.
  - Une réservation concerne un seul vol et un seul passager.
  - Une réservation peut être annulée ou confirmée.
  - Un client peut réserver un ou plusieurs vols, pour des passagers différents.

Analyse du domaine : proposition de méthode

- Etape 1 : Etude des phrases une à une pour
  - Définir les classes
  - Définir les associations
  - Définir les états des classes
  - Définir les opérations
  - Définir les attributs
- Comment les représenter ?
  - Un diagramme de classes est le point central dans le développement orienté objet. Son objectif est de décrire la structure des entités manipulées par les utilisateurs.
  - Il met en œuvre les classes contenant des attributs et des opérations, reliées par des associations ou des généralisations.

« *Des compagnies aériennes proposent différents vols.* »

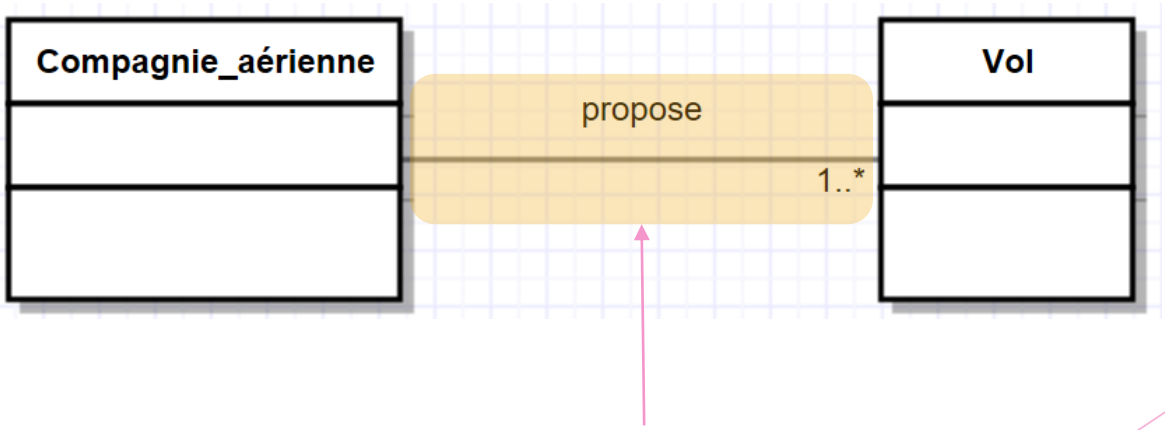
- Deux concepts importants dans le monde réel
  - Compagnie_aérienne
  - Vol
- Comment modéliser cette phrase?



Classe : description abstraite d'un ensemble d'objets possédant les mêmes caractéristiques.

« *Des compagnies aériennes proposent différents vols.* »

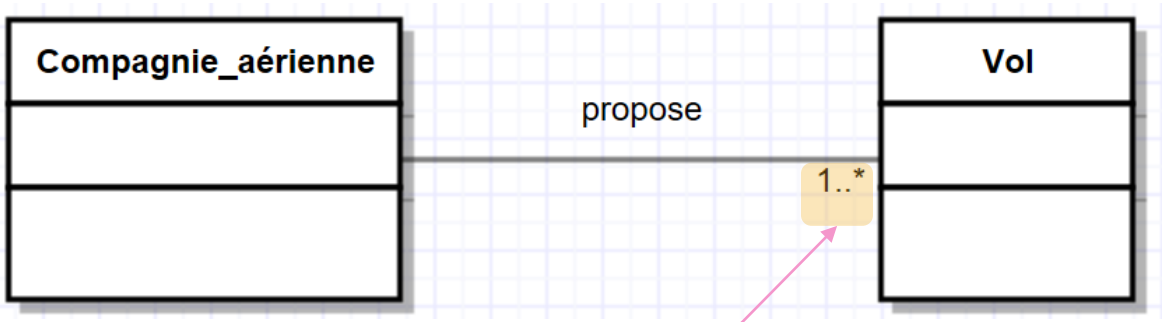
- Deux concepts importants dans le monde réel
  - Compagnie_aérienne
  - Vol
- Comment modéliser cette phrase?



Association : relation sémantique durable entre deux classes, nommée par un verbe.

« *Des compagnies aériennes proposent différents vols.* »

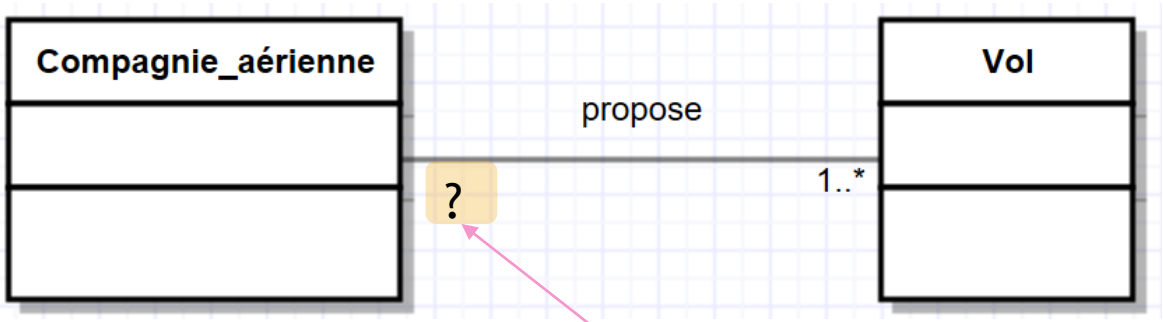
- Deux concepts importants dans le monde réel
  - Compagnie_aérienne
  - Vol
- Comment modéliser cette phrase?



Multiplicité : nombre d'objets qui peuvent participer à une relation avec un objet de l'autre classe.

« *Des compagnies aériennes proposent différents vols.* »

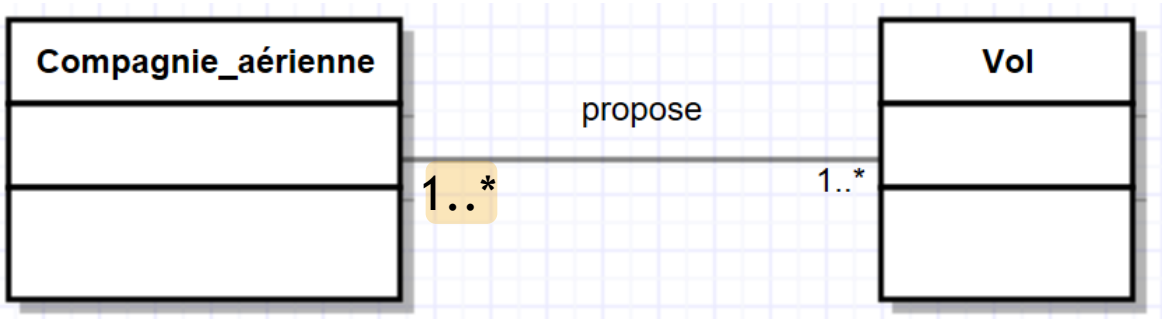
- Deux concepts importants dans le monde réel
  - Compagnie_aérienne
  - Vol
- Comment modéliser cette phrase?



Aucune indication sur la multiplicité du coté de la classe Compagnie_aérienne => retour vers les utilisateurs pour savoir.

« *Des compagnies aériennes proposent différents vols.* »

- Deux concepts importants dans le monde réel
  - Compagnie_aérienne
  - Vol
- Comment modéliser cette phrase?



=> Un vol est proposé le plus souvent par une seule compagnie aérienne, mais qu'il peut également être partagé entre plusieurs affréteurs.

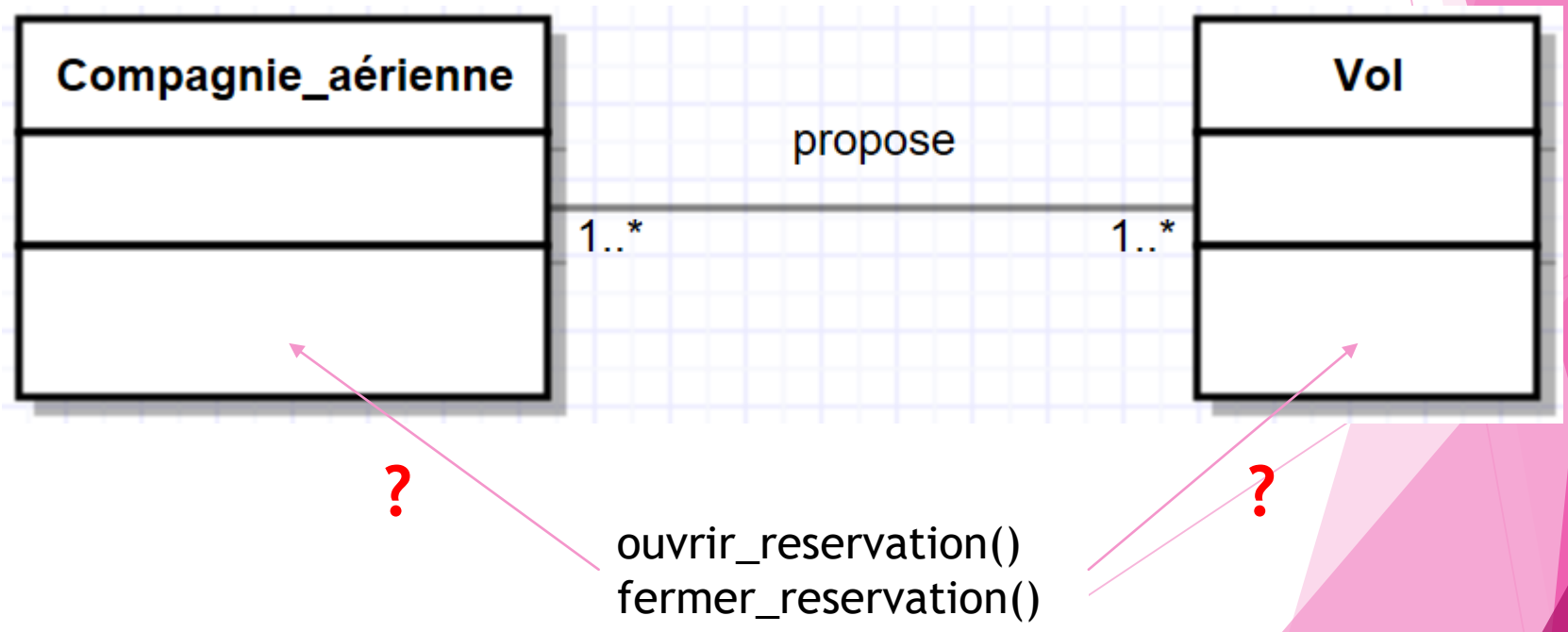
« *Un vol est **ouvert** à la réservation et **refermé** sur ordre de la compagnie* »

- Notions d'ouverture et de fermeture sont des concepts dynamique :
  - Changement d'état d'un objet Vol sur ordre d'un objet Compagnie_aérienne.

- Comment les modéliser?
  - Dans le diagramme de classes, les seuls concepts dynamiques disponibles sont les opérations.
  - Une opération représente un élément de comportement contenu dans une classe.

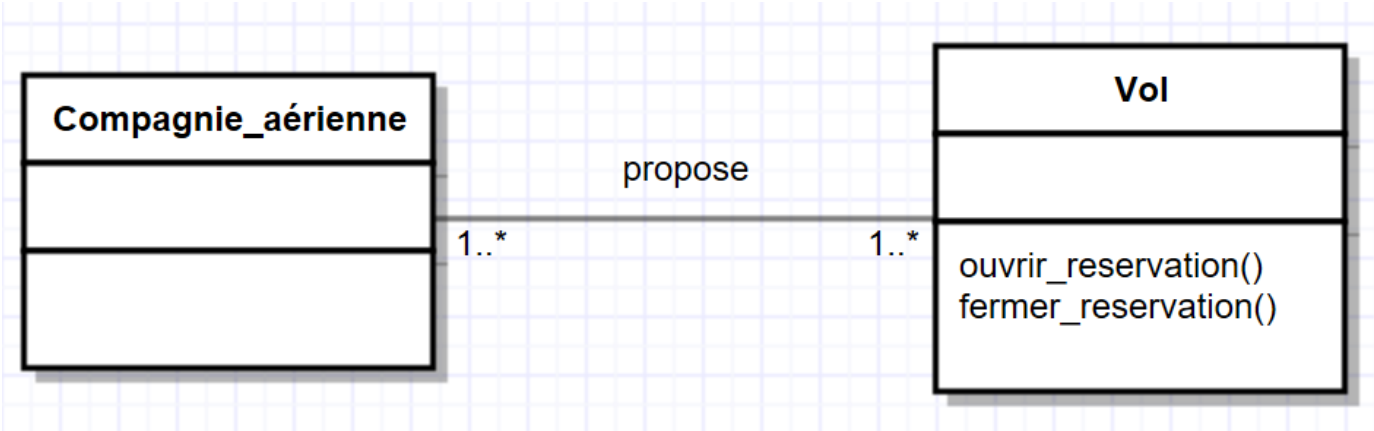
« *Un vol est **ouvert** à la réservation et **refermé** sur ordre de la compagnie* »

- Dans quelle classe ajouter les opérations ouvrir_reservation() et fermer_reservation() ?



« *Un vol est **ouvert** à la réservation et **refermé** sur ordre de la compagnie* »

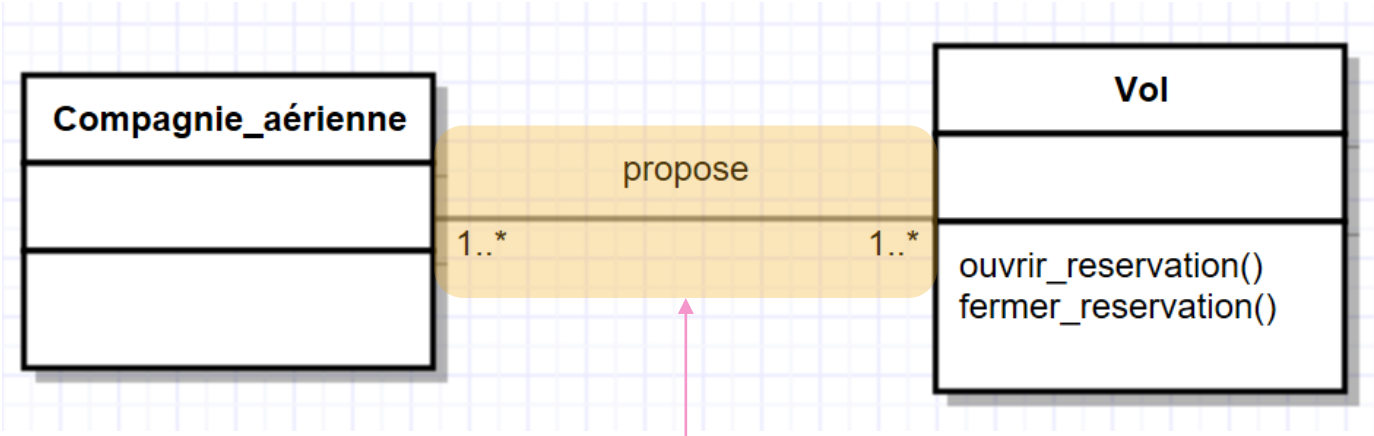
- Dans quelle classe ajouter les opérations ouvrir_reservation() et fermer_reservation() ?



En orienté objet, l'objet sur lequel on peut réaliser un traitement doit le déclarer en tant qu'opération.
Les autres objets qui posséderont une référence dessus pourront alors lui envoyer un message qui invoque cette opération.

« *Un vol est **ouvert** à la réservation et **refermé** sur ordre de la compagnie* »

- Dans quelle classe ajouter les opérations ouvrir_reservation() et fermer_reservation() ?



L'association **propose** s'instanciera en un ensemble de liens entre des objets des classes **Compagnie_aérienne** et **Vol.**

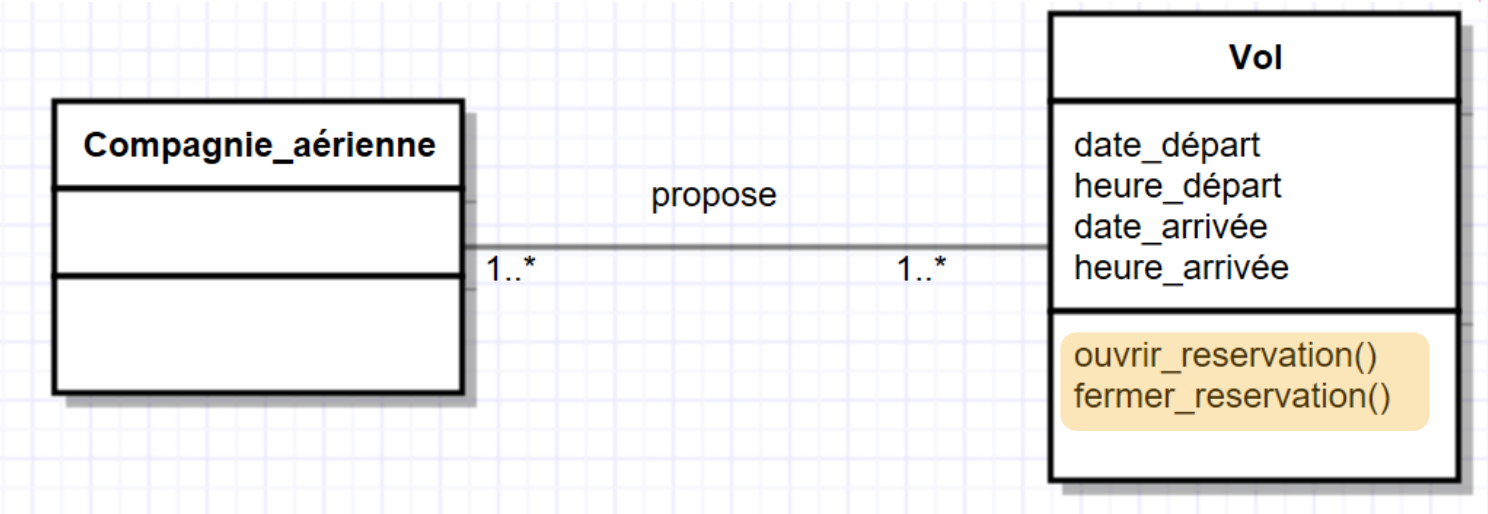
« *Un vol a un **jour** et une **heure** de départ et un **jour** et une **heure** d'arrivée.* »

- Les notions de date et heure sont de simples valeurs.

- Comment les modéliser?
  - Comme des attributs de la classe Vol.
- Pourquoi ?
  - Ce sont de simples types
  - Si l'on ne peut demander à un élément que sa valeur, alors il s'agit d'un simple attribut.
  - Si plusieurs questions s'y appliquent alors il s'agit d'un objet qui possède lui-même des attributs ainsi que des liens avec d'autres objets.

« Un vol a un jour et une heure de départ et un jour et une heure d'arrivée. »

- Les notions de date et heure sont de simples valeurs.



Exercice

- Comment modéliser la phrase suivante ?

« Un vol a un aéroport de départ et un aéroport d'arrivée. »

- Compléter votre diagramme de classe.

Exercice

- Comment modéliser cette phrase ?

« *Chaque **aéroport** dessert une ou plusieurs **villes**.* »

- Compléter votre diagramme de classe.

Exercice

- Comment modéliser les phrases suivantes ?

« Un vol peut comporter des escales dans des aéroports. »

« Une escale a une heure d'arrivée et une heure de départ. »

- Compléter votre diagramme de classe.

Exercice

- Comment modéliser les phrases suivantes ?

« *Une **réservation** concerne un seul **vol** et un seul **passager**.* »

« *Une **réservation** peut être **annulée** ou **confirmée**.* »

- Compléter votre diagramme de classe.

Exercice

- Comment modéliser la phrase suivante ?

« *Un **client** peut réserver un ou plusieurs **vols**, pour des **passagers** différents.* »

- Compléter votre diagramme de classe.

Identifier les attributs métier

- Comment faire ?
  - Lister les attributs indispensables pour chacune de classes.
  - Attention, ne pas faire références aux autres classes ! Il s'agit des associations et non des attributs.

- Exercice :
  - Quels sont les attributs métier des différentes classes ?
  - Compléter votre diagramme.

Identifier les attributs métier

- Attributs :
  - Compagnie_aérienne : nom
  - Aéroport : nom
  - Client : nom, prénom, adresse, mail, numero_tel
  - Infos_escale : heure_départ, heure_arrivée
  - Passager : nom, prénom, adresse, mail, numero_tel
  - Réservation : date, numéro
  - Ville : nom
  - Vol : numéro, date_départ, heure_départ, date_arrivée, heure_arrivée

Identifier les attributs dérivés

- Un attribut dérivé est une propriété dont la valeur peut être déduite à l'aide d'autres informations disponibles sur le modèle.

- Exemple : la notion de durée d'un vol
  - Importante pour le client et le passager
  - Redondante car déductible à partir de l'heure d'arrivée et de l'heure de départ.
  - => information dérivée donc il s'agit bien d'un attribut dérivé.

Et en conception ?

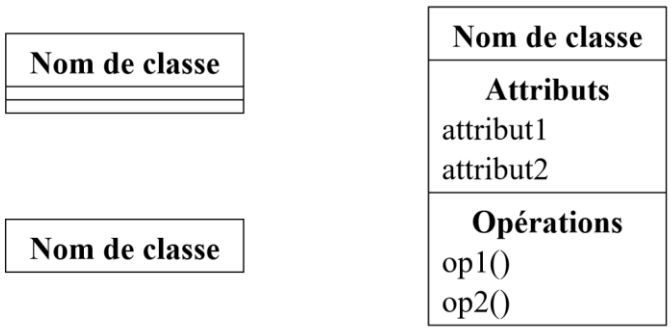
- Le concept d'attribut dérivé n'a pas d'existence en langage objet.
- Deux solutions laissées au concepteur :
  - Définir cet attribut comme un autre pour lequel les méthodes d'accès seront définis.
    - Ok si la valeur de l'attribut dérivé doit être disponible en permanence
  - Définir une méthode de calcul de cet attribut.
    - Ok si la fréquence de requête est faible et l'algorithme de calcul simple.

UML2 : notions de base

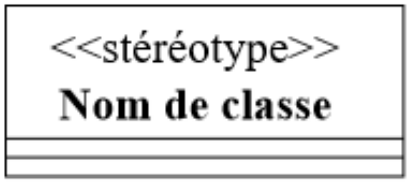
Résumé

La classe

- Elle est décrite par :
  - Un nom
  - Un stéréotype
  - Des attributs
  - Des opérations
  - Des exceptions
  - Etc.



La classe – Stéréotype



- Permet d'étendre les classes déjà existantes en leur donnant une signification sémantique différente, mécanisme proche de la généralisation/spécialisation sauf qu'il permet le changement de sémantique.
- Exemple : si la classe A est un stéréotype de la classe B, alors A se comporte comme B tout en ayant une signification sémantique différente.

La classe – Stéréotypes prédéfinis

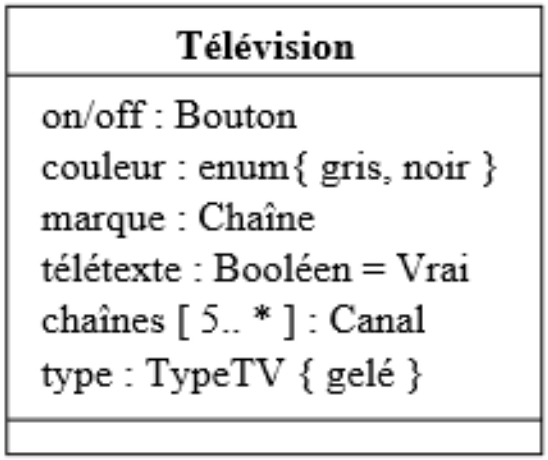
- « énumération » : classe définissant un ensemble d'identificateurs formant le domaine de valeur d'un type.
- « acteur » : classe modélisant un ensemble de rôles joués par un acteur.
- « interface » : classe contenant uniquement une description des opérations visibles.
- « exception » : classe modélisant un cas particulier de signal : les exceptions.

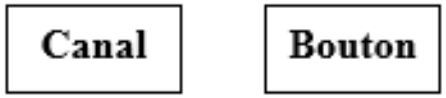
La classe - Attribut

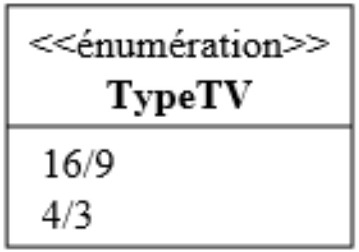
- Un attribut est une propriété de la classe définie par un nom, un type et éventuellement une valeur initiale.

- Syntaxe :

[visibilité] nom_attribut [multiplicité] : type [= valeur_initiale]







La classe - Opération

- Une opération est une spécification du comportement des instances de la classe.
- Cinq types d'opérations :
  - Constructeurs qui créent les objets
  - Destructeurs qui détruisent les objets
  - Sélecteurs (opérations de consultation) qui renvoient tout ou un partie de l'état d'un objet
  - Modificateurs qui changent tout ou une partie de l'état d'un objet
  - Itérateurs qui visitent l'état d'un objet ou le contenu d'une structure de données contenant des objets.

La classe – Opération (2)

- Syntaxe : [visibilité] nom_op ([arguments]) : type_retourné
  - Un argument est une description d'une valeur nécessaire à l'opération.
  - Le type retourné est le type de la valeur retournée par l'opération. Attention, si il n'est pas présent, l'opération ne retourne aucune valeur.

- Il est également possible de spécifier une **précondition** (condition qui doit toujours être vraie avant l'exécution de l'opération) et une **postcondition** (condition qui est toujours vraie après l'exécution de l'opération).

La classe - Visibilité

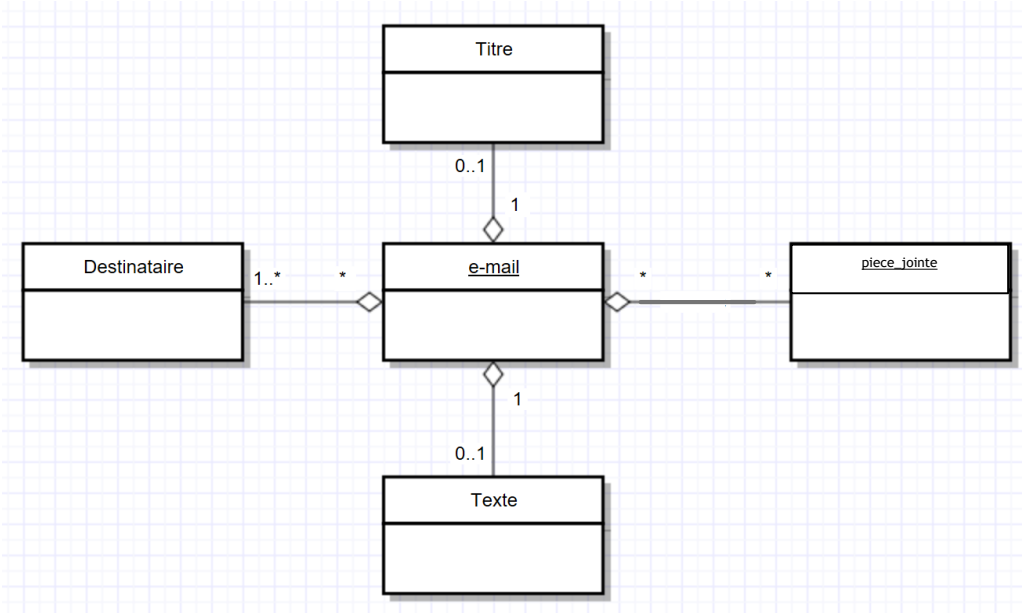
- 3 niveaux :
  - Public : l'élément est visible pour tous les clients/utilisateurs de la classe (noté par un +)
  - Protégé : l'élément est visible pour les sous-classes de la classe (noté par un #)
  - Privé : l'élément est visible pour la classe seule (noté par un -)

L’association

- Une association exprime une connexion sémantique bidirectionnelle entre deux classes.
- Décoration
  - Nommage : un texte unique décrivant la sémantique de l’association.
  - Rôles : texte spécifiant la fonction d’une classe pour une association donnée.
  - Cardinalités : précise le nombre d’instances qui participent à une relation.
  - Classe d’association : il s’agit d’une classe qui réalise la navigation entre les instances d’autres classes.

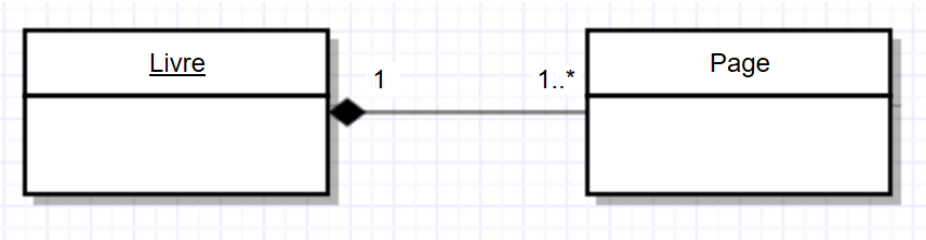
L’association – Type agrégation

- L'agrégation est une association non symétrique, qui exprime un couplage fort et une relation de subordination.
- Elle représente une relation de type "ensemble / élément".
- Une instance d'élément agrégé peut exister sans agrégat (et inversement) : les cycles de vies de l'agrégat et de ses éléments agrégés peuvent être indépendants.



L’association – Type Composition

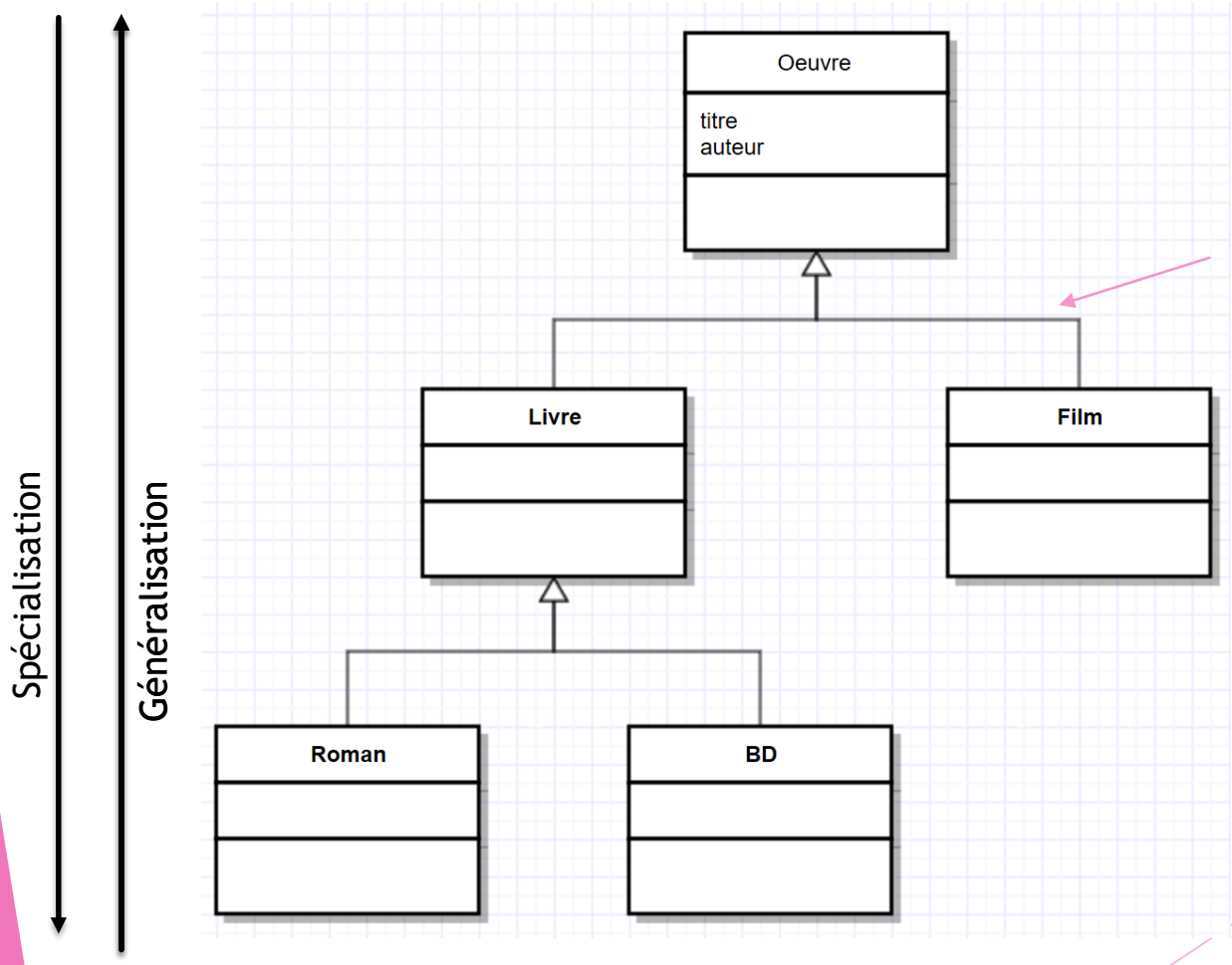
- La composition est une agrégation forte (agrégation par valeur).
- Les cycles de vies des éléments (les "composants") et de l'agrégat sont liés : si l'agrégat est détruit (ou copié), ses composants le sont aussi.
- A un même moment, une instance de composant ne peut être liée qu'à un seul agrégat.



L'Héritage

- Spécification
  - Démarche descendante
  - Consiste à étendre les propriétés d'une classe sous forme de sous-classes, plus spécifiques.
- Généralisation
  - Démarche ascendante
  - Consiste à factoriser les propriétés d'un ensemble de classes sous forme d'une super-classe plus abstraite.
- Classification
  - L'héritage, spécification et généralisation, permet la classification des objets.
  - Principe de substitution (Liksow, 1987) : « *Il doit être possible de substituer n'importe quel instance d'une super-classe, par n'importe quel instance d'une de ses sous-classes, sans que la sémantique d'un programme écrit dans les termes de la super-classe n'en soit affectée* »

L'Héritage - exemple



Fiche TD à réaliser