Cursus Front

Introduction à la POO et modélisation UML

M2I FORMATION 2022

Glodie Tshimini

PLAN

Glodie Tshimini : contact@tshimini.fr

PLAN

I.

Les concepts de la pogrammation orientée objet

Approche orientée objet

- L'approche procédurale qui consiste à résoudre un problème informatique de manière séquentielle avec une suite d'instructions à exécuter et l'utilisation des fonctions.

- L'approche objet demande une réflexion plus poussée pour concevoir et développer une solution **réutilisable** et **évolutif** (maintenable). De plus, elle garantit une **protection** des données que l'on verra un peu plus tard lorsqu'on abordera la notion d'**encapsulation**.
- Elle utilise des **objets** pour résoudre le même problème.
- En informatique, un objet est une **entité** qui possède un ensemble d'**attributs** qui détermine sa structure et un ensemble de **méthodes** qui déterminent son comportement.

Objet

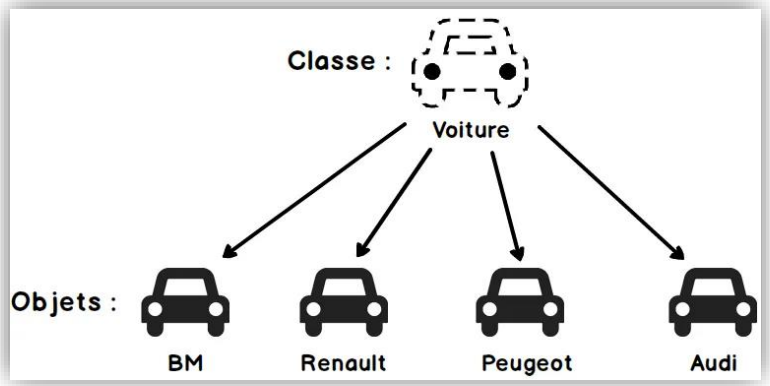
- Une personne peut être représentée comme un objet en informatique.
  - Une personne est caractérisée par un ensemble d'attributs :
    - Couleur des yeux
    - Taille
    - Poids
    - Etc.
  - Une personne peut réaliser différentes actions :
    - Marcher
    - Courir
    - Parler
    - Etc.

Classe

- Glodie, Christophe sont des personnes, ils possèdent les mêmes caractéristiques et comportements, cependant chacun est unique et indépendant de l'autre.
- On parle de **classe** pour désigner le **modèle** qui a servi à créer des objets de même type.
- Autrement dit, il désigne la **structure** et le **comportement communs** des objets.
- Prenons des exemples de la vie courante :
  - Moule à gâteau
  - Plan ayant servi à construire des maisons
  - Template d'un CV

Classe et objet

- La **classe** est le **modèle** permettant de créer un ou plusieurs objets.
- On dit alors qu'un **objet** est une **instance** d'une classe.
- Une **classe** possède :
  - Un **nom**
  - Des **attributs**
  - Et des **comportements**



Source image waytolearnx.com

Exercice

Glodie Tshimini : contact@tshimini.fr

Exercice 1

1-exercices/exercice1.md

Abstraction

- L’abstraction est un principe qui permet de **ne retenir que les informations pertinentes pour notre modèle**. Autrement dit, on s’abstrait de tous les détails utiles pour se focaliser uniquement sur l’essentiel.

- Par exemple dans une application informatique, l’objet utilisateur A ou la classe Users aura les caractéristiques suivantes e-mail, nom, prénom, date de naissance, etc. On s’abstrait de représenter toutes les caractéristiques qui le désignent.
- Autres exemples :
  - Numéro de sécurité sociale pour les systèmes de santé
  - Numéro de compte pour les systèmes bancaires

Encapsulation

- Parfois, on aura besoin de **cacher** certains attributs et comportements propres (privés) d'un objet.
- On parle alors d'encapsulation, c'est-à-dire seul l'objet lui-même est en capacité de connaître ses attributs et comportements cachés de l'extérieur.

- Des exemples de la vie courante :
  - ADN
  - Numéro de série
  - Le solde de son compte
- Plusieurs niveaux d'encapsulation :
  - **Privé**
  - **Protégé**
  - **Public**

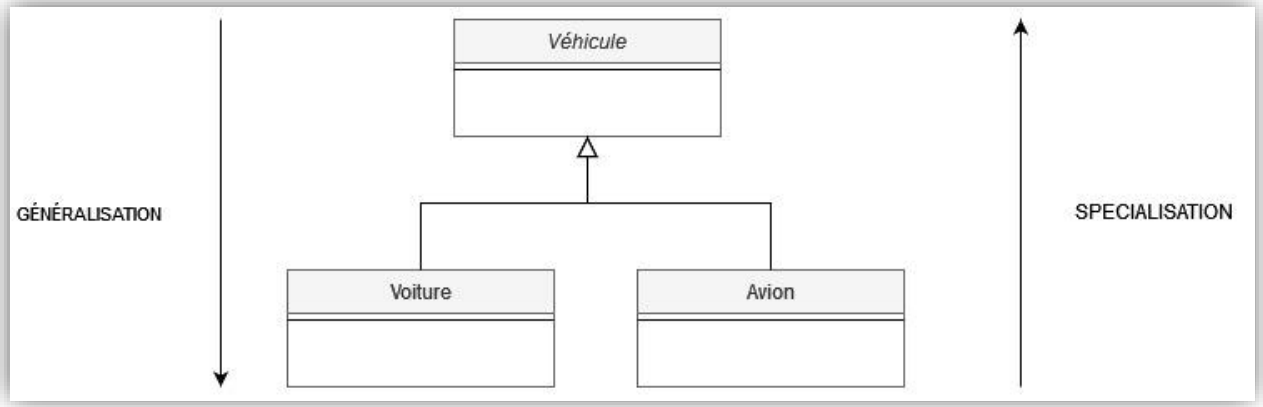
Héritage

- Une voiture est classe donc un modèle, lui-même crée à partir d'un autre modèle un véhicule. Donc on peut dire qu'une voiture est un véhicule. On peut également dire qu'un avion est un véhicule.
- Un véhicule est ?

Source image freepik

Généralisation et spécialisation

- D'une part, une **voiture** est une **spécialisation** d'un véhicule.
- D'autre part, l'avion est lui aussi une spécialisation d'un véhicule.
- La **classe véhicule** est appelée **classe mère ou superclasse**, car elles possèdent des **caractéristiques** et **comportements communs** à une voiture et un avion au niveau de la classe.
- Voiture et avion sont appelées **sous-classe** ou **classes filles**.



Polymorphisme

- Prenons l'exemple d'un poisson et d'un chat :
  - Tous les deux sont des **spécialisations** de la classe Animal
  - Tous les deux peuvent se déplacer
  - Cependant, un chat pour se déplacer marche sur ses 4 pattes
  - Le poisson utilise ses nageoires (nage) pour se déplacer.
- Lorsque les sous-classes peuvent implémenter (réaliser) les **comportements** à leurs façons selon leurs spécificités, on parle alors de **polymorphisme**.
- Autrement dit le comportement « se déplacer » peut prendre plusieurs formes.

Composition

- Un objet A peut-être **composé** de plusieurs objets B, on parle de composition.
- L'objet A est un **composé**.
- Les objets B sont des **composants**.

- Il existe 2 types de composition
  - **Composition faible ou agrégation**
    - Les objets B existent indépendamment de l'objet A
  - **Composition forte** : A l'inverse
    - Les objets B n'existent pas indépendamment de l'objet A. La suppression de l'objet A entraîne la suppression des objets B

Exercice

Glodie Tshimini : contact@tshimini.fr

Exercice 2

1-exercices/exercice2.md

II.

UML et généralités

Langage UML

- Langage représentant graphiquement les objets
- **Unified Model Language** (Langage de modélisation objet unifié)
  - 1997 : UML 1
  - 2006 : UML 2

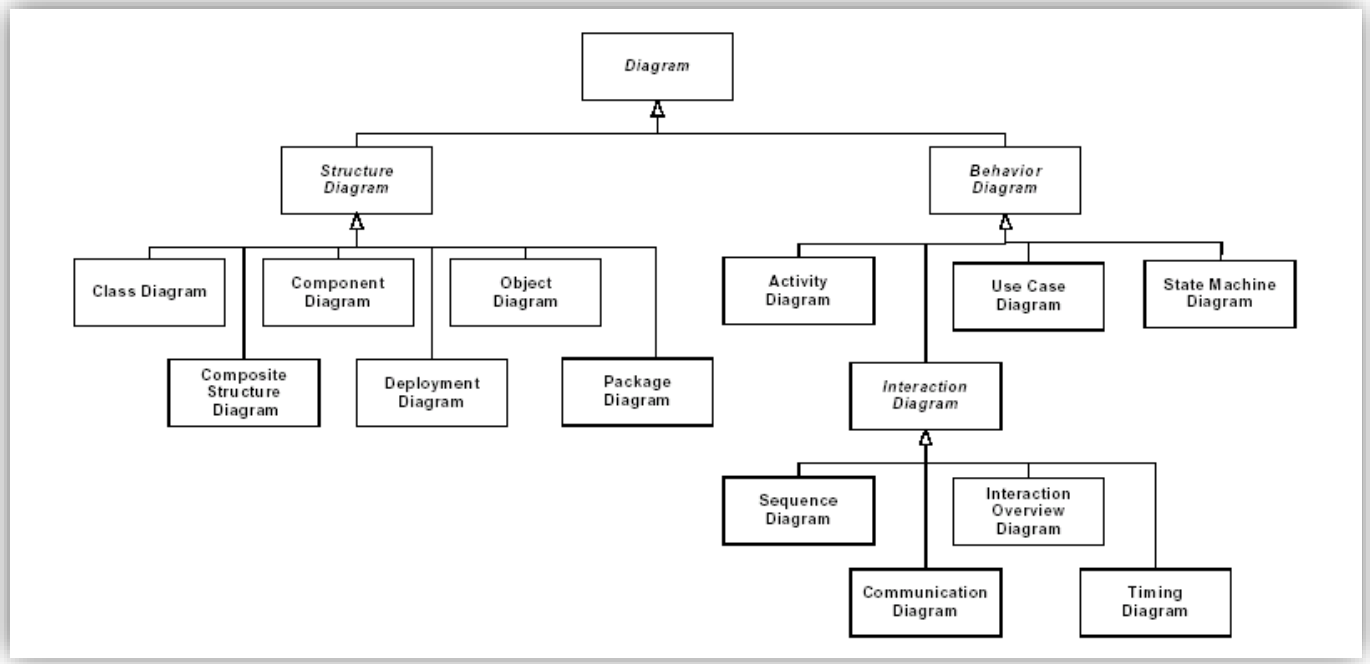
- Fusion de 3 méthodes
  - BOOCH
  - OMT
  - OOSE

Langage UML

- Pour **modéliser**
  - **Applications** utilisant un langage orienté objet
  - **Bases de données**

- Pour **communiquer**
  - Humains (échanger, spécifier, **documenter**)
  - Machines (représenter partiellement ou intégralement un système)

Les diagrammes du langage UML



Source https://fr.m.wikipedia.org/wiki/Fichier:Uml_hierarchie_des_diagrammes.png

III.

Diagramme de cas d’utilisation

Principes

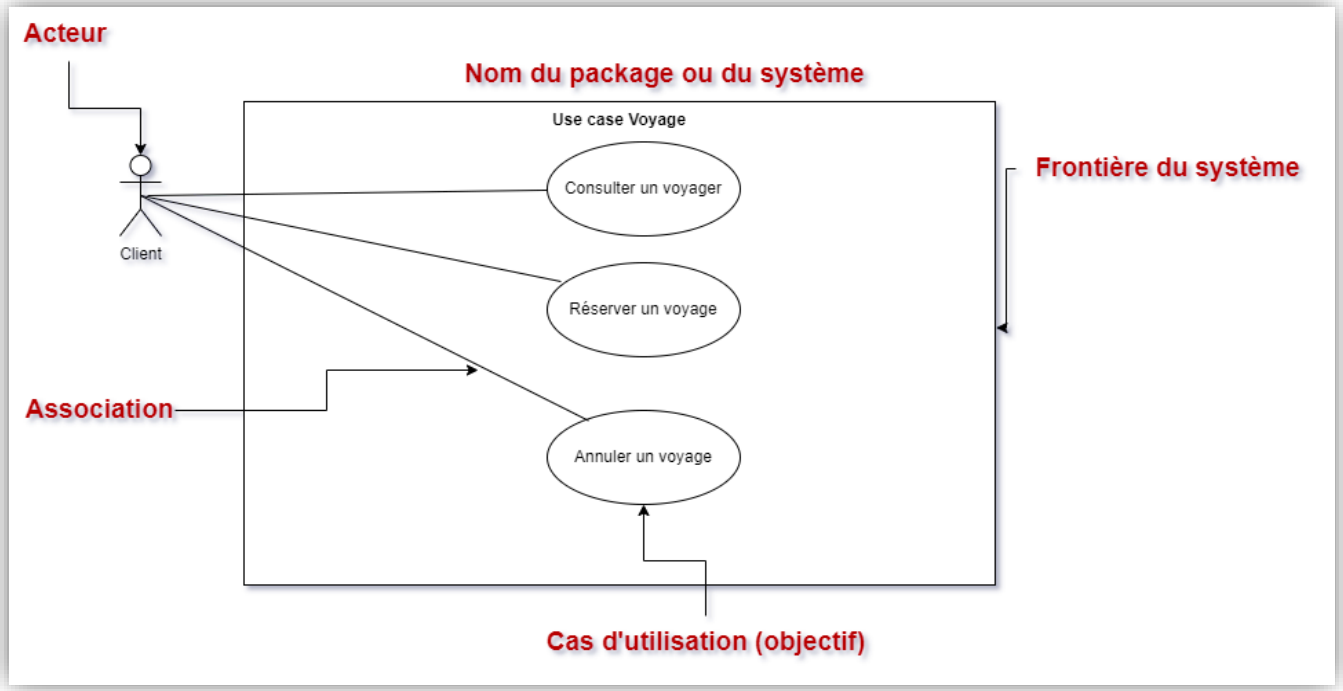
- Schématiser l'expression des besoins.

- Application vue du point de vue des acteurs.
- Répondre aux questions **Qui** et **Quoi** ?

- Délimiter le **périmètre fonctionnel**.
- Servir pour réaliser des **tests fonctionnels**.

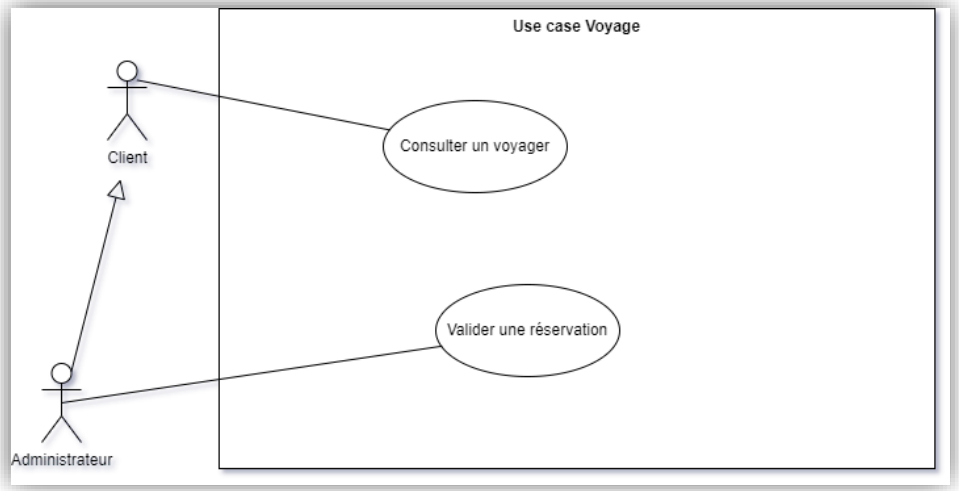
- On peut s'en servir pour impliquer le client dans la conception.
- Construire des interfaces **IHM** (d'autres diagrammes UML sont plus adaptés).

Syntaxe


Acteur
Nom du package ou du système
Use case Voyage
Frontière du système
Consulter un voyager
Client
Réserver un voyage
Association
Annuler un voyage
Cas d'utilisation (objectif)

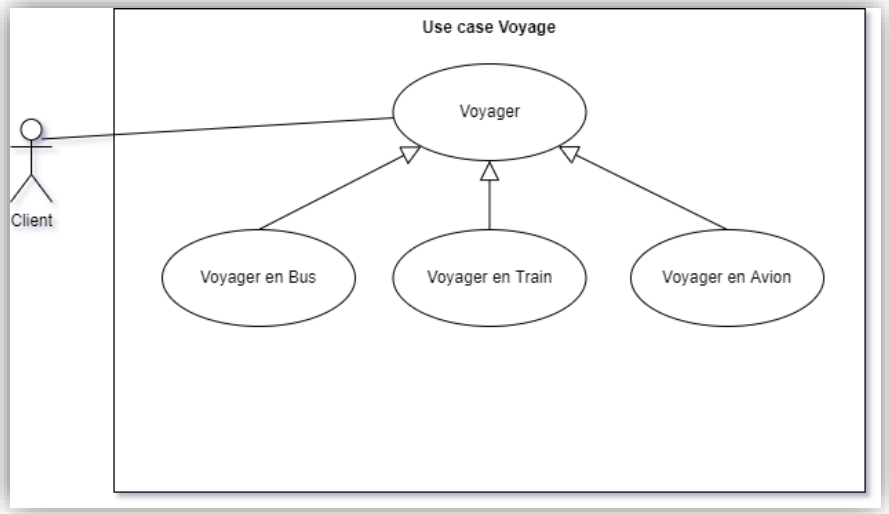
Héritage entre acteur

- Un Administrateur est un client, il hérite de tous les cas d'utilisation qu'un client peut réaliser.
- L'inverse est faux, c'est-à-dire qu'un client n'est pas un administrateur, dans notre exemple, il ne peut pas valider une réservation.



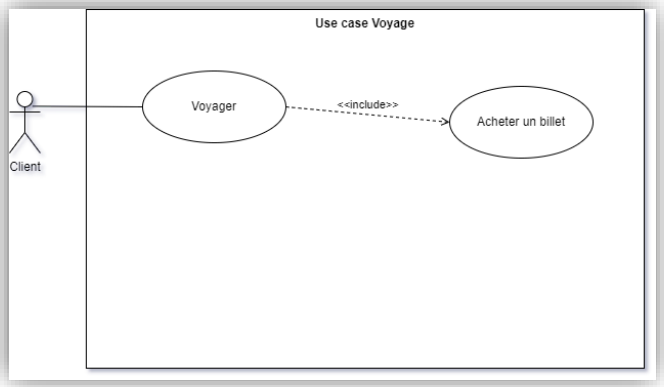
Héritage entre cas d'utilisation

- L'héritage entre les cas d'utilisation est possible. Dans notre cas, voyager en bus ou voyager en train ou voyager en avion sont des spécifications d'un voyage.



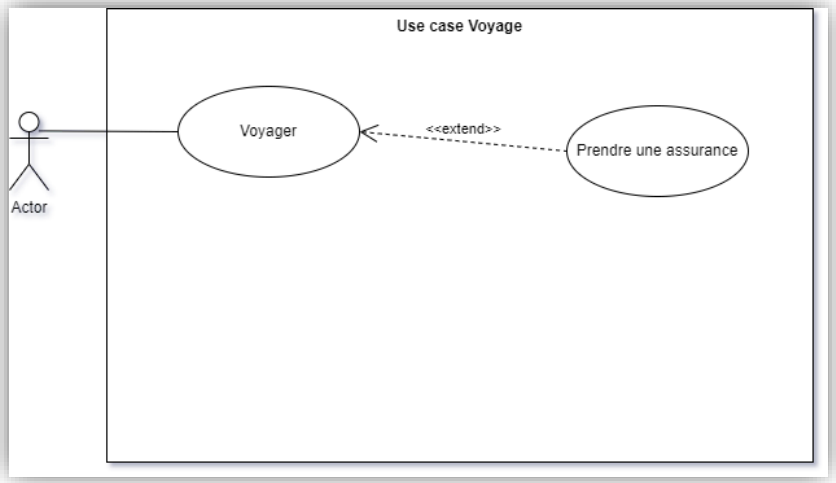
Cas additionnel obligatoire

- La relation d'exclusion entre deux cas d'utilisation signifie que pour la réalisation du cas d'utilisation « voyager » puisse se faire, il faut obligatoirement que le cas d'utilisation « acheter un billet » se réalise également.
- Généralement, les cas **inclus** ne répondent pas directement à un besoin primaire de l'utilisateur.



Cas additionnel optionnel

- La relation d'extension s'applique lorsqu'il y a un cas d'utilisation de base qui peut être étendue par un autre cas d'utilisation.
- Contrairement à l'inclusion, **l'extension n'est pas obligatoire**.
- L'inclusion et l'extension ne sont pas obligatoires dans le diagramme, on peut s'en passer pour garder en **lisibilité**.



Exercice

Glodie Tshimini : contact@tshimini.fr

Exercice 3

1-exercices/exercice3.md

Cas d'utilisation détaillé

- Nom du cas d'utilisation (UC)
- Description courte UC
- Acteur(s) impliqué(s)

- Pré-conditions
- Post-conditions

- Scénario nominal
- Scénarios alternatifs
- Scénarios d'erreurs

Intérêts

- Avoir des informations pour réaliser son diagramme de classe spécifique au cas d'utilisation
  - **Entités**
  - **Attributs**

- Scénarios pour les **tests fonctionnels**

Exercice

Glodie Tshimini : contact@tshimini.fr

Exercice4

1-exercices/exercice4.md

IV.

Diagramme de classes

Principes

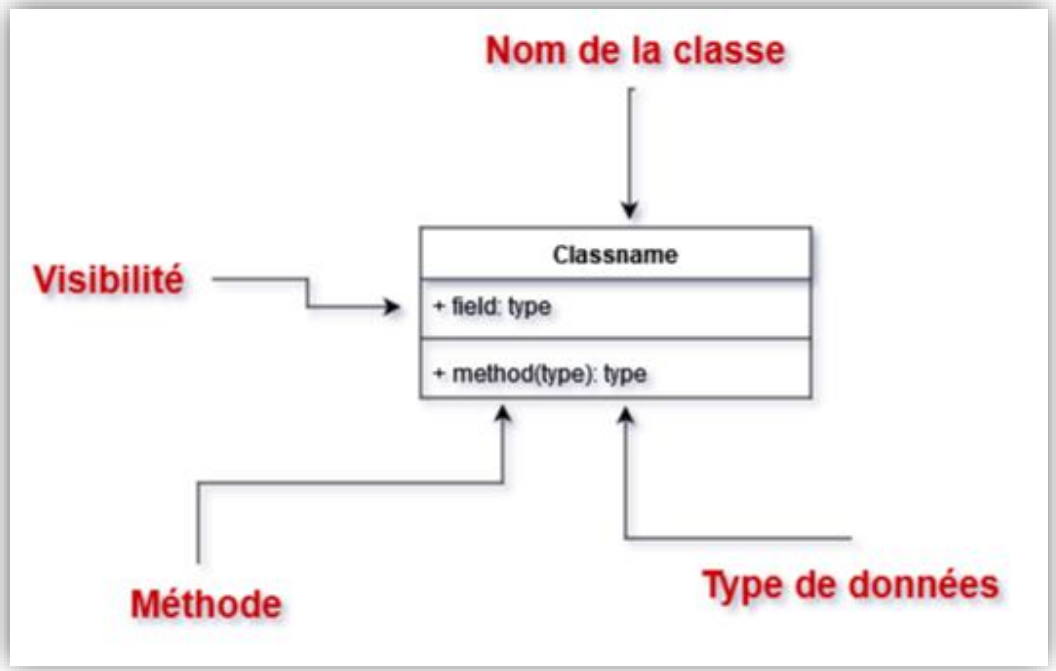
- Schématiser la structure interne d'un système qui sera implémenté dans un langage orienté objet
  - Classes
  - Attributs
  - Opérations
  - Relations

- Autrement dit, représente les données et les traitements du système.

- Modéliser des bases de données relationnelles ou objet.

- Le niveau d'abstraction ou du détail dépend de vos objectifs et de la phase à laquelle le projet se trouve.

Syntaxe


Nom de la classe
Classname
+ field: type
+ method(type): type
Visibilité
Méthode
Type de données

Encapsulation

- **Public** (+) : accessible par tous les autres objets.
- **Privé** (-) : accessible uniquement au sein de la classe.
- **Protégé** (#) : accessible uniquement au sein des classes filles ou paquetage.
- Paquetage (~) : accessible uniquement aux classes appartenant au package.

Types de données

- On utilise les types de base de l'algorithmie dite **primitifs** (on n'utilise pas les types spécifiques à un langage de programmation) et les énumérations (liste fermée des données)
  - Int
  - Float
  - Boolean
  - String

- On n'utilise pas non plus un type d'une de nos classes
  - C'est la relation entre les classes qui permet de dire que la classe A utilise la classe B.

Associations

- Détermine les liens entre les classes
  - **Association binaire** (entre 2 classes)
  - **Association n-aire** (entre n classes)
  - **Classe d'association**
  - **Association réflexive**

Association binaire

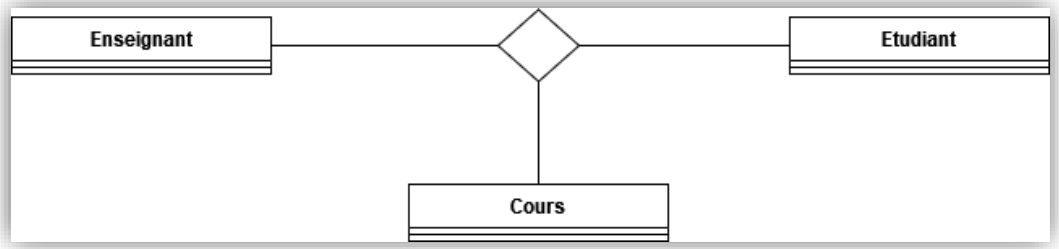
- La plus repandue et celle qu'il faut préférer par rapport aux autres (n-aire et classe d'association).

- La plus **lisible** et **compréhensible**.

Association N-aire

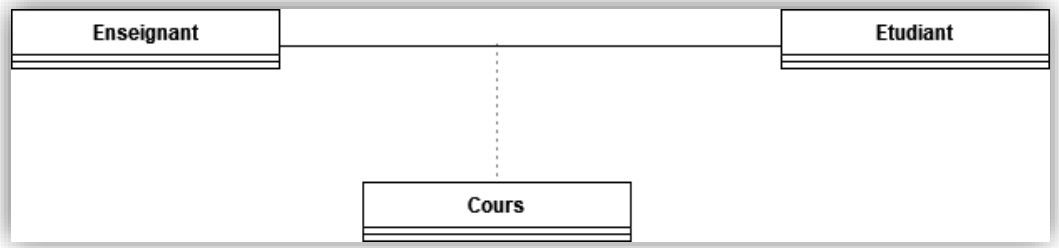
- Une association entre plusieurs classes (plus 2 classes)

- **Les classes existent indépendamment des uns des autres.**



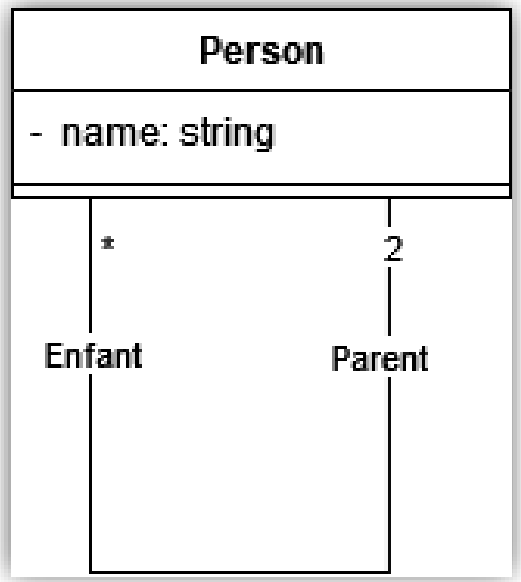
Classe d'association

- Une classe permet de faire l'association entre 2 autres classes.
- **La classe d'association existe uniquement via l'association entre les 2 classes.**



Association réflexive

- Une classe qui est associée à elle-même avec 2 **rôles** différents.
- La définition des **rôles est obligatoire** dans ce cas précis.



Exercice

Glodie Tshimini : contact@tshimini.fr

Exercice5

1-exercices/exercice5.md

Multiplicité

- Indique le **nombre d'objets liés par l'association :**
  - 0..1
  - 1

  - N..M : au minimum N et au maximum M
  - M : exactement M

  - 0..* ou *
  - 1..* : au moins une instance

Agrégation et composition

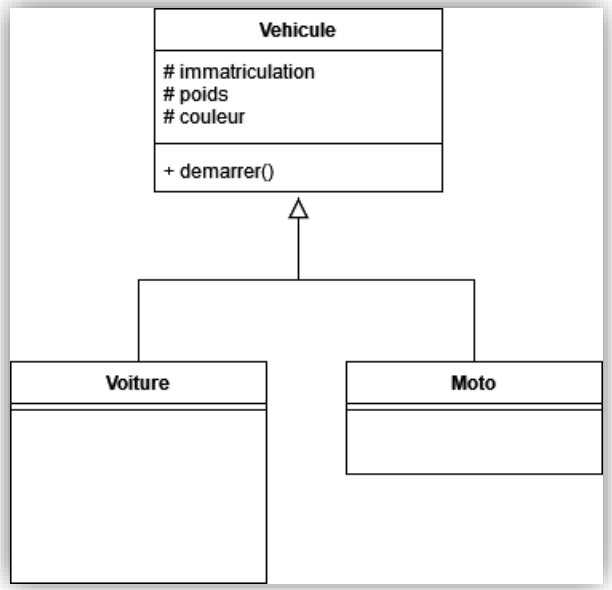
- Relation particulière entre une instance d'une classe A avec une ou plusieurs instances d'une autre classe B.
- La classe A "**domine**" la classe B.
- Ou la classe A "**contient**" la classe B.

Agrégation et composition

- Composition ou agrégation **forte.**
- Suppression d'une instance de la classe qui domine entraîne la suppression des instances liées par cette relation.

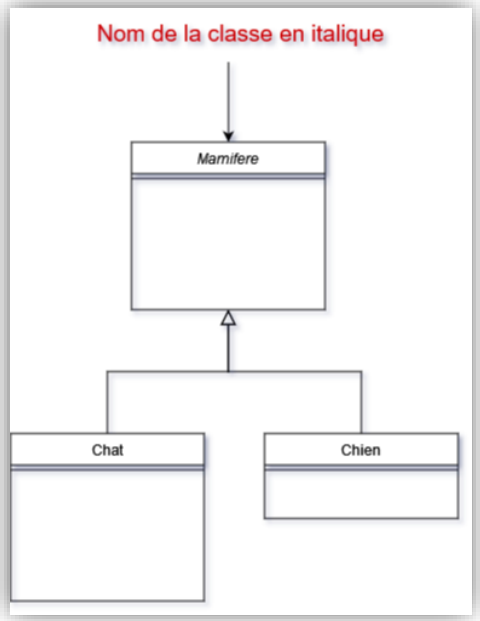
Héritage entre les classes

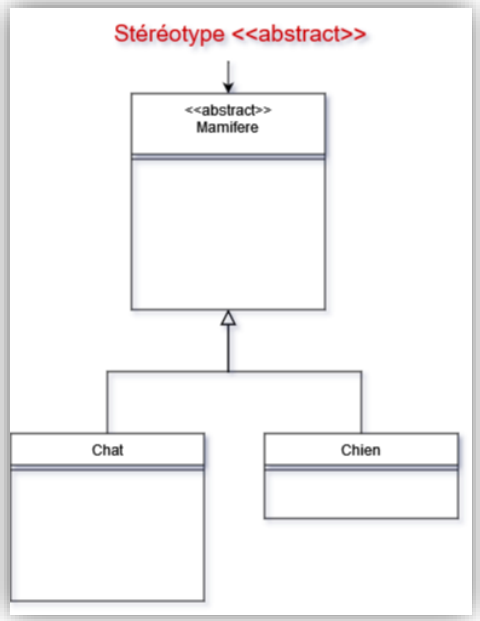
- Une classe mère contient des caractéristiques communes pour ses classes filles.
- **Généralisation** des attributs et des méthodes au sein d'une super classe.
- **Spécialisation** dans les sous-classes.



Classes abstraite

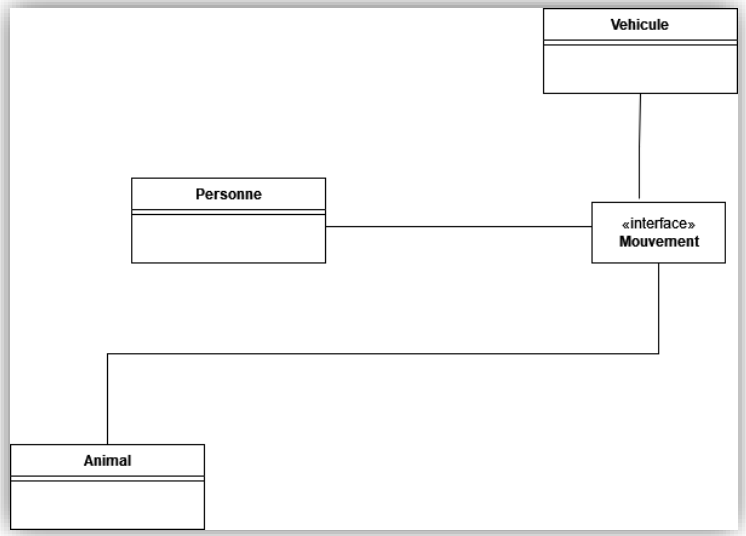
- Ne peut pas avoir d'instance
- Sert de base (mère) pour les classes dérivées (filles)





Interface

- Contrat que doit remplir une ou plusieurs classes.
- Toutes les méthodes au sein d'une interface sont abstraites (elles doivent être implémentées par la classe qui doit remplir le contrat).



Exercice

Glodie Tshimini : contact@tshimini.fr

Exercice6

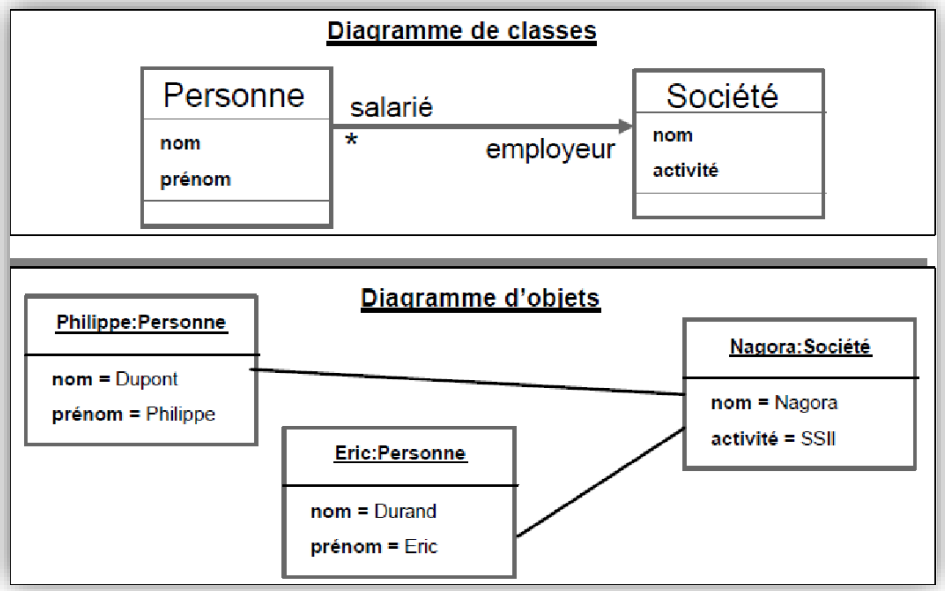
1-exercices/exercice6.md

V.

Diagramme d’objets

Diagramme d'objets

- Réaliser des **tests** de son diagramme de classes grâce à l'instanciation des objets.



Source image Mohamed Nemiche

Exercice

Glodie Tshimini : contact@tshimini.fr

Exercice7

1-exercices/exercice7.md

THE END.

Glodie Tshimini: contact@tshimini.fr