

FORMATION

Introduction POO & UML

12/06/2023

Glodie TSHIMINI
contact@tshimini.fr

PLAN

- I. Les concepts de la programmation orientée objet
- II. UML et généralités
- III. Diagramme de cas d'utilisation
- IV. Diagramme de classes
- V. Diagramme d'objets

I. LES CONCEPTS DE LA PROGRAMMATION 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.

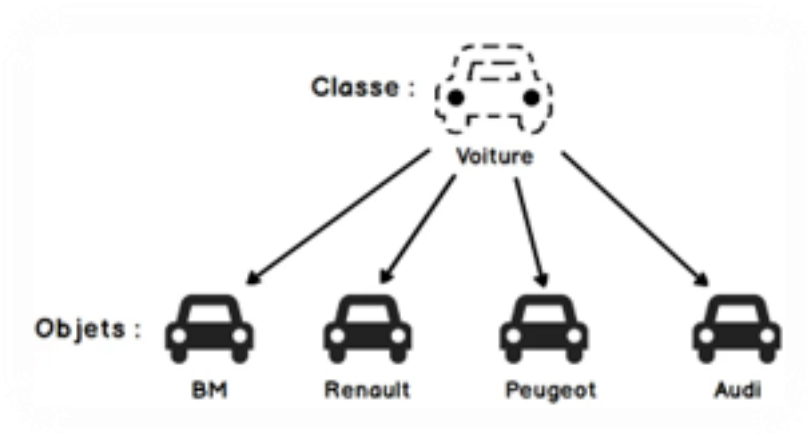
- Une personne peut être représentée comme un objet en informatique.
 - Une personne est caractérisée par un ensemble d'attributs :
 - Couleur des yeux
 - Taille
 - Poids
 - Etc.
 - Une personne peut réaliser différentes actions :
 - Marcher
 - Courir
 - Parler
 - Etc.

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

[Source image waytolearnx.com](http://waytolearnx.com)



- 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**

EXERCICE





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

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.

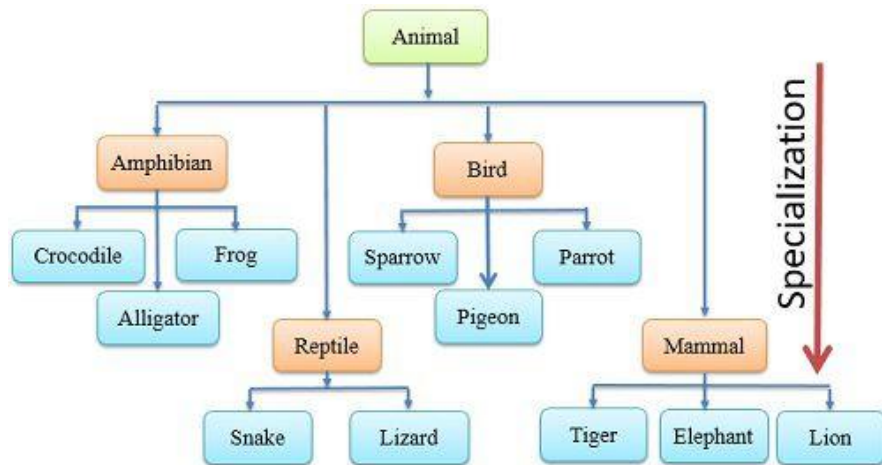
Source image freepik

- Un véhicule est ?



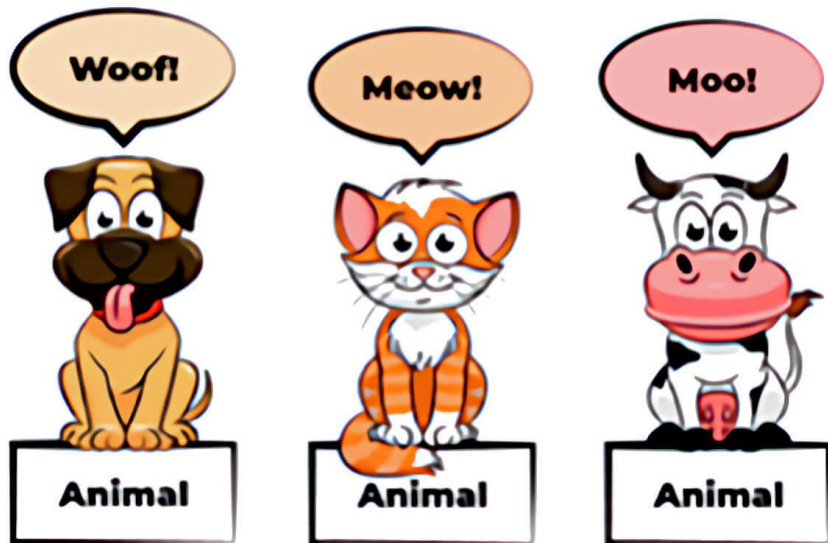
Généralisation et spécialisation

[Source image letsstudytogether](https://www.letsstudytogether.com)



- D'une part, un serpent est une spécialisation d'un reptile.
- D'autre part, un reptile est une généralisation d'un serpent, lézard, etc.
- La classe **Reptile** est appelée **classe mère** ou **superclasse**, car elles possèdent des **caractéristiques et comportements communes** à une serpent, lézard, etc.
- Reptile, mammifères, oiseaux, amphibien sont des spécialisations de la classe Animal. Elles sont appelées **sous-classe** ou **classes filles**.

polymorphisme animal



www.aquaportail.com

- Prenons un autre exemple, le poisson et le pêcheur
 - Tous les deux sont des **spécialisations** de la classe *Vivant*
 - Tous les deux peuvent se déplacer
 - Cependant, le pêcheur utilise ses pieds
 - Le poisson utilise ses nageoires (nage)
- 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.

[Source image doretdeplatineshop](#)



- 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**
 - Les objets B n'existent pas indépendamment de l'objet A. La suppression de l'objet A entraîne la suppression des objets B

EXERCICE





EXERCICE 2

1-exercices/exercice2.md

II. UML ET GÉNÉRALITÉS

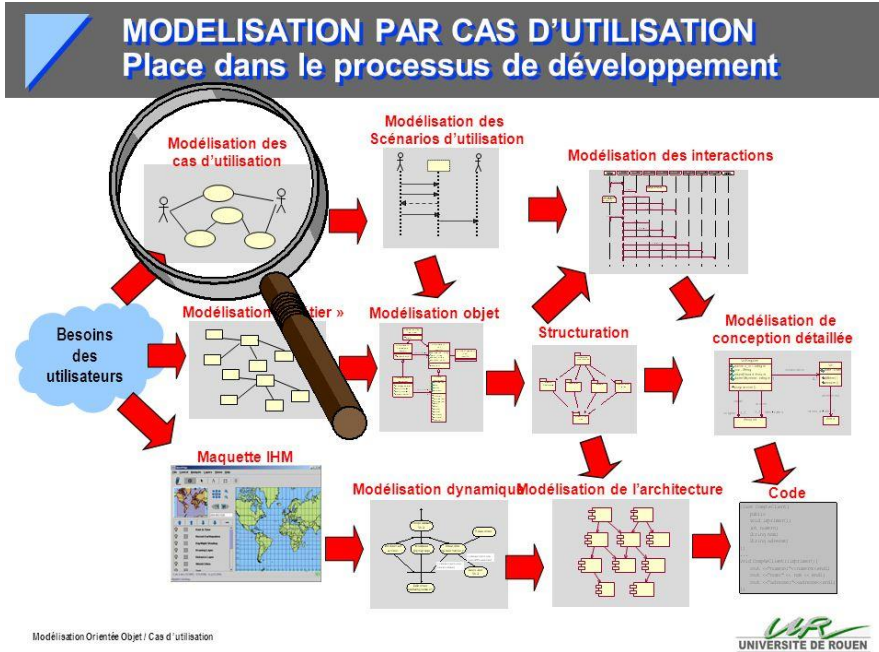


[Source image wikipédia](#)



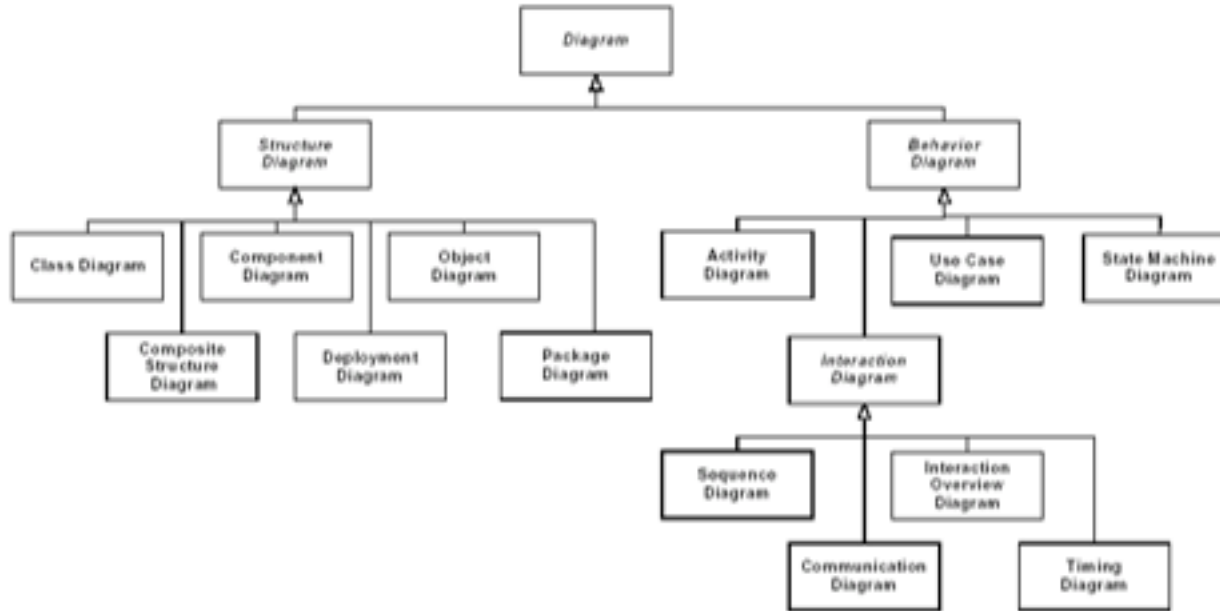
- Un objet A peut-être composé de plusieurs objets B, on parle de composition.
- Langage représentant graphiquement les objets
- Unified Model Language (Langage de modélisation objet unifié)
 - 1997 : UML 1
 - 2006 : UML 2
- Fusion de 3 méthodes
 - BOOCH
 - OMT
 - OOSE

Source image slideplayer



- Pour modéliser
 - Applications utilisant un langage de programmation orienté objet
 - Bases de données
- Pour communiquer
 - Humains (échanger, spécifier, documenter)
 - Machines (représenter partiellement ou intégralement un système)

Les diagrammes UML



[Source image wikipedia](https://en.wikipedia.org/wiki/UML_Diagram_Hierarchy)

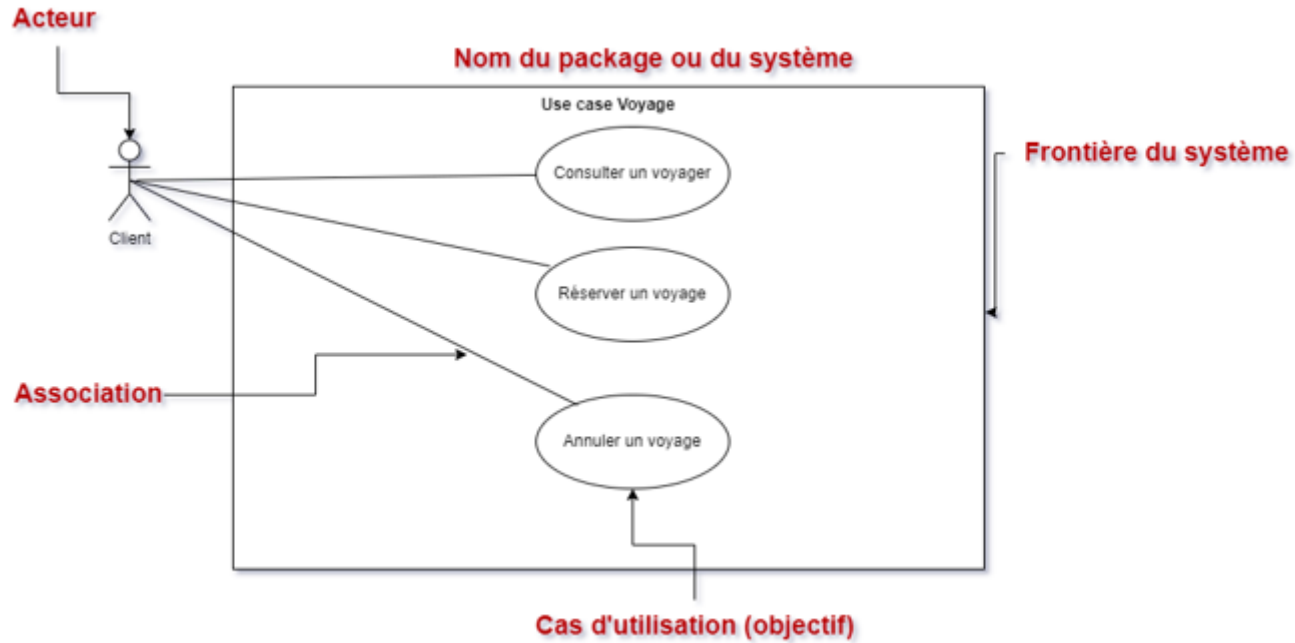
III. DIAGRAMME DE CAS D'UTILISATION



Principes

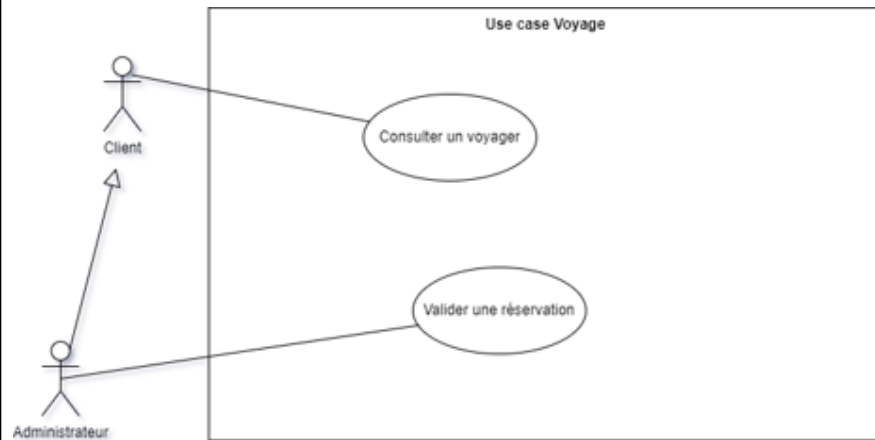
- Schématiser l'expression des besoins.
- Application vue du point de vue des acteurs.
- Répondre aux questions Qui et Quoi ?
- Délimiter le périmètre fonctionnel.
- Servir pour réaliser des tests fonctionnels.
- On peut s'en servir pour impliquer le client dans la conception.
- Construire des interfaces IHM (d'autres diagrammes UML sont plus adaptés).

Syntaxe



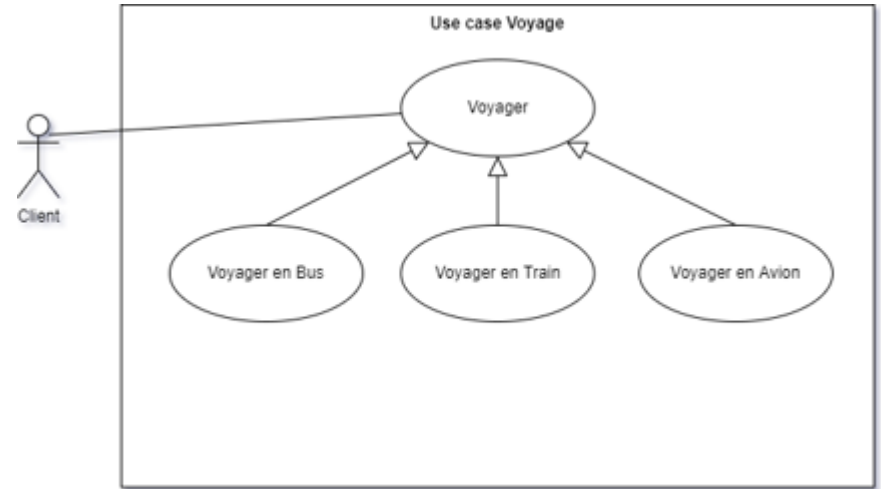
Héritage entre acteurs

- 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.



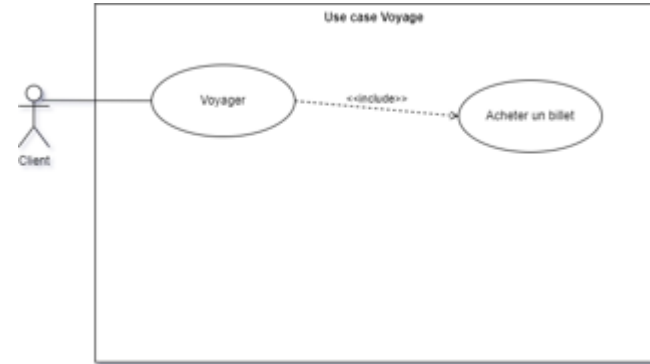
Héritage entre cas d'utilisation

- 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.



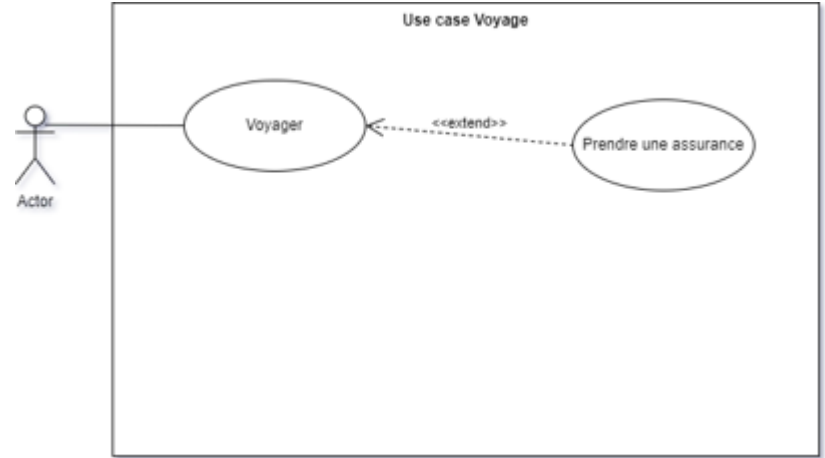
Include : cas additionnel obligatoire

- La relation d'inclusion 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.



Extend : 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 gagner en lisibilité.



EXERCICE





EXERCICE 3

1-exercices/exercice3.md

Source image scribd

Description textuelle des cas d'utilisation « S'authentifier »

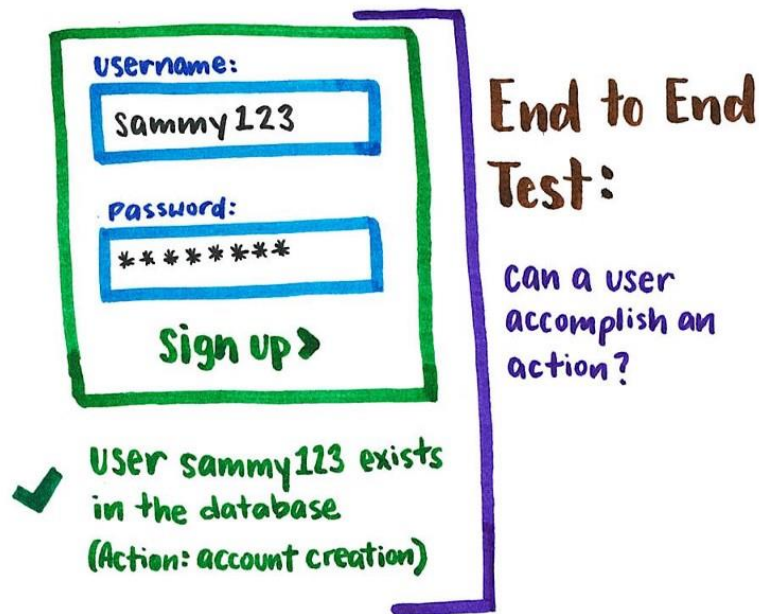
Le tableau suivant décrit la description textuelle du cas d'utilisation « S'authentifier ».

Titre	Ajouter un domaine
Acteurs	Élève
Description	Lorsqu'un utilisateur du système veut accéder à l'application, il doit saisir son login et son mot de passe : ensuite le système vérifie s'ils sont corrects ou pas afin d'autoriser ou bien refuser l'accès.
Description des scénarios	<p>Scénario nominal :</p> <ol style="list-style-type: none"> 1. L'utilisateur demande l'accès au système, en cliquant sur le bouton « Se connecter ». 2. Le système redirige l'utilisateur vers la page mmm.com. 3. L'utilisateur introduit son email et son mot de passe de son compte TA. 4. Si l'utilisateur est identifié, le système affiche l'interface de « Accueil ». <p>Scénario alternatif :</p> <p>A1 : Email ou mot de passe non valide :</p> <ol style="list-style-type: none"> 1. Le système affiche un message d'erreur « Votre identifiant ou votre mot de passe est incorrect ».
Pré condition(s)	L'utilisateur doit avoir un compte TA.

- Nom du cas d'utilisation (UC)
- Description courte UC
- Acteur(s) impliqué(s)
- Pré-conditions
- Post-conditions
- Scénario nominal
- Scénarios alternatifs
- Scénarios d'erreurs

Cas d'utilisation détaillé

Source image freecodecamp



The diagram shows a hand-drawn sign-up form with a green border. Inside, there are two input fields: 'Username:' with the value 'sammy123' and 'Password:' with masked characters '*****'. Below the fields is a 'Sign up >' button. To the right of the form, the text 'End to End Test:' is written in brown, followed by the question 'Can a user accomplish an action?' in purple. At the bottom left, a green checkmark is next to the text 'User sammy123 exists in the database (Action: account creation)' in green.

Username:
sammy123

Password:

Sign up >

End to End Test:

Can a user accomplish an action?

✓ User sammy123 exists in the database
(Action: account creation)

Avantages

- Avoir des informations pour réaliser son diagramme de classe spécifique au cas d'utilisation
 - Entités
 - Attributs
- Source d'information pour la réalisation des IHM (Interface home-machine, pour simplifier les écrans)
- Scénarios pour les tests fonctionnels

EXERCICE





EXERCICE 4

1-exercices/exercice4.md

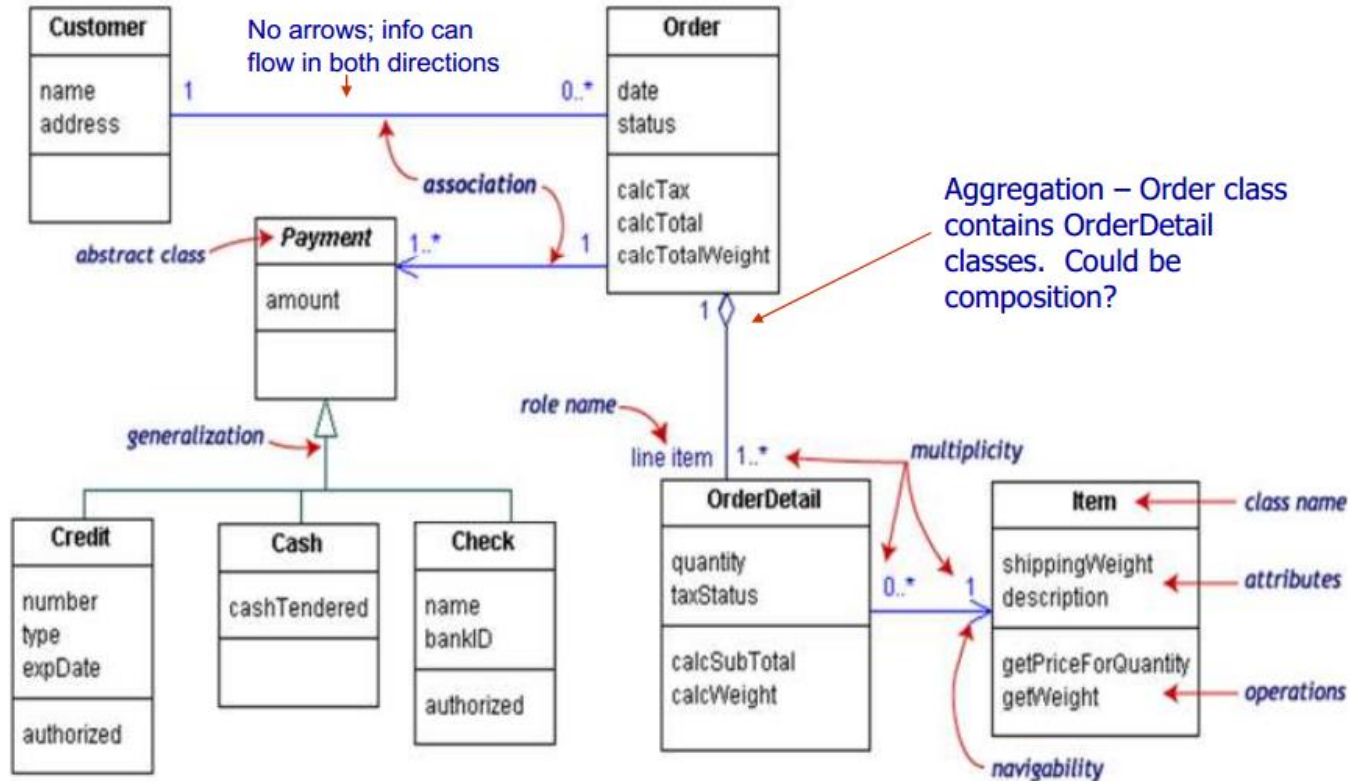
IV. DIAGRAMME DE CLASSES



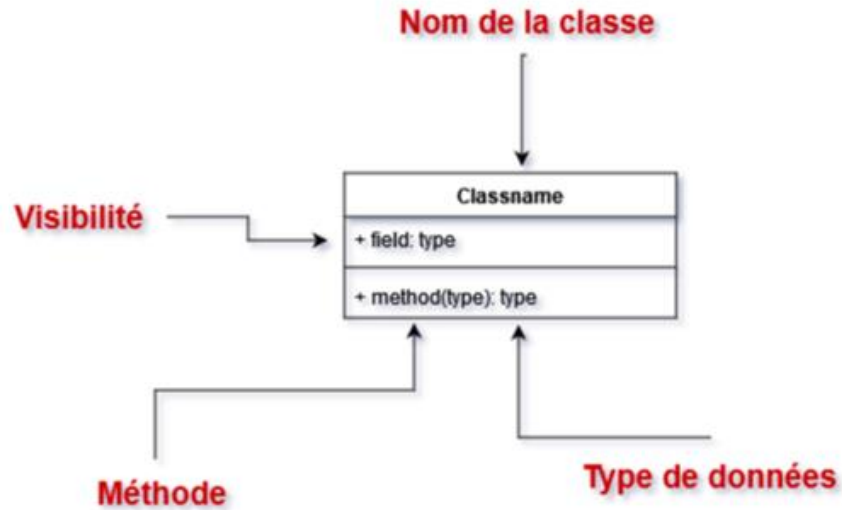
Principes

- Schématiser la structure interne d'un système qui sera implémenté dans un langage de programmation 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.

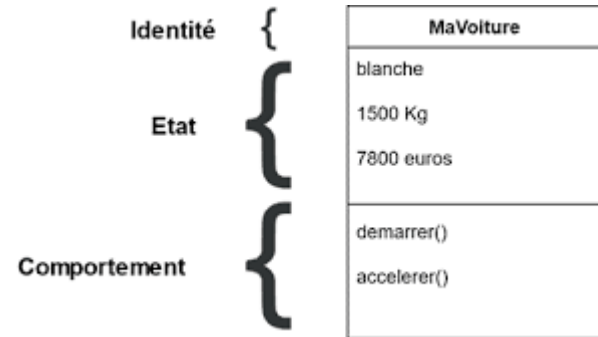
Mapping diagramme de classe : [source image stackexchange](#)



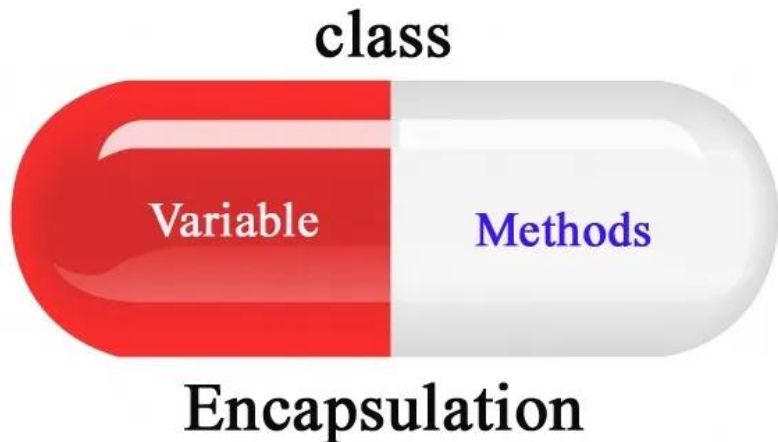
Zoom sur une classe



[Source image data-transitionnumerique](#)

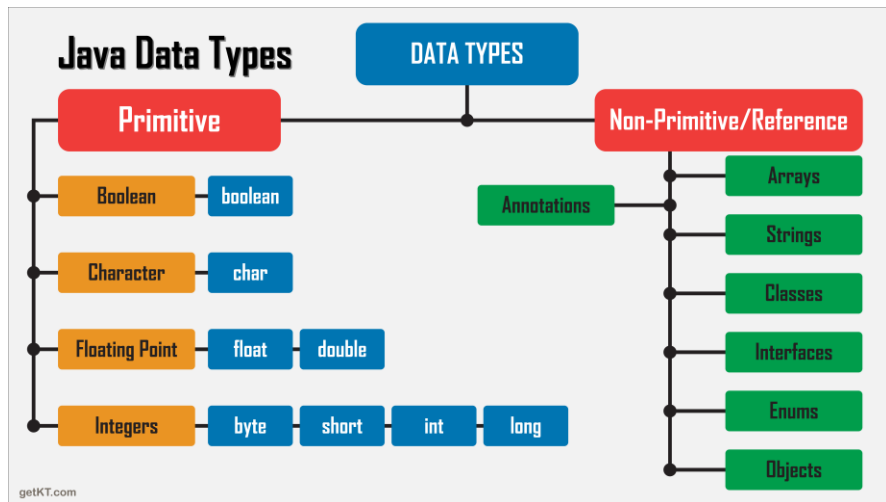


[Source image code4coding](https://sourceimagecode4coding.com)



- 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.

Source image getkt



- 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.

[Source image stackoverflow](#)


Inheritance 

Dependency 

Aggregation 

Containment 

Association 

Directed Association 

Realization 

- Détermine les liens entre les classes

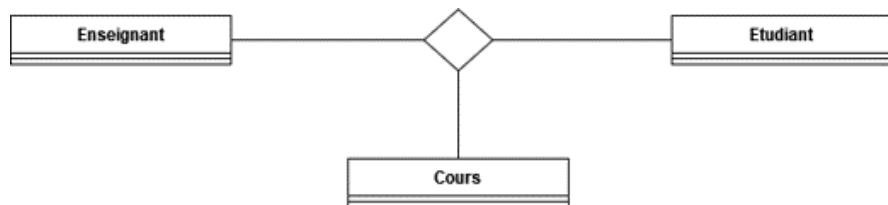
1. Association binaire (entre 2 classes)
2. Association n-aire (entre n classes)
3. Classe d'association
4. Association réflexive
5. Héritage
6. Agrégation

Association binaire



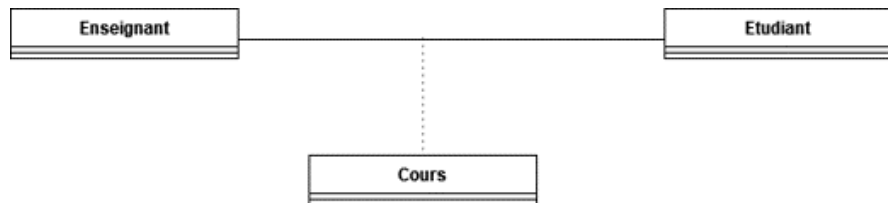
- La plus rependue et celle qu'il faut privilégier par rapport aux autres types d'association (n-aire et classe d'association).
- La plus lisible et compréhensible.

Association N-aire



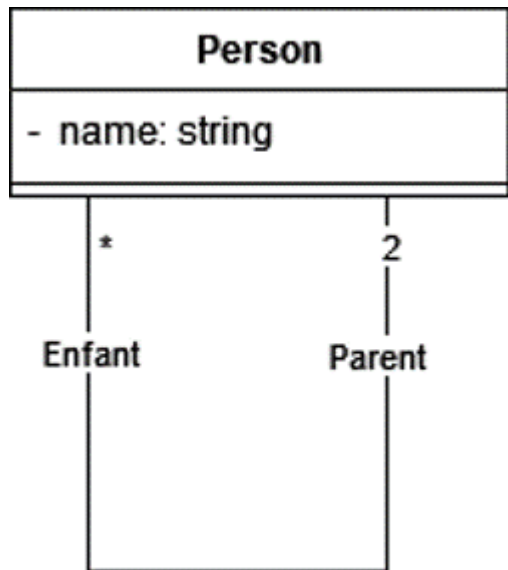
- Une association entre plusieurs classes (> 2 classes)
- Les classes existent indépendamment des uns et des autres.

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





EXERCICE 5

1-exercices/exercice5.md

Multiplicité

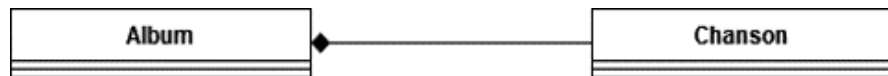
- Indique le nombre d'objets liés par l'association :
- Association un à un
 - 0..1
 - 1
- Association un à plusieurs
 - N..M : au minimum N et au maximum M
 - M : exactement M
- Association plusieurs à plusieurs
 - 0..* ou *
 - 1..* : au moins une instance

Agrégation et composition



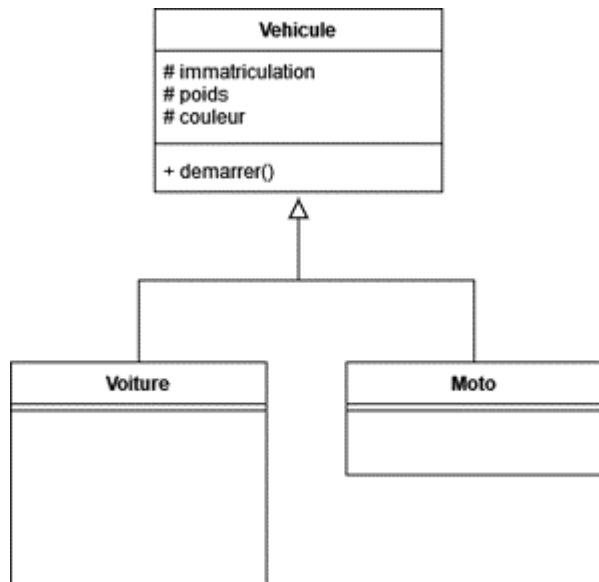
- Relation particulière entre une instance d'une classe A avec une ou plusieurs instances d'une autre classe B.
- La classe A "domine" la classe B.
- Ou la classe A "contient" la classe B.

Agrégation forte



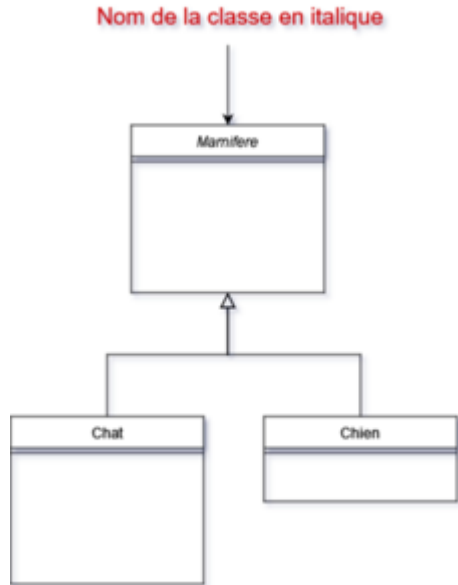
- Composition ou agrégation forte.
- Suppression d'une instance de la classe qui domine entraîne la suppression des instances liées par cette relation.

Héritage entre les classes

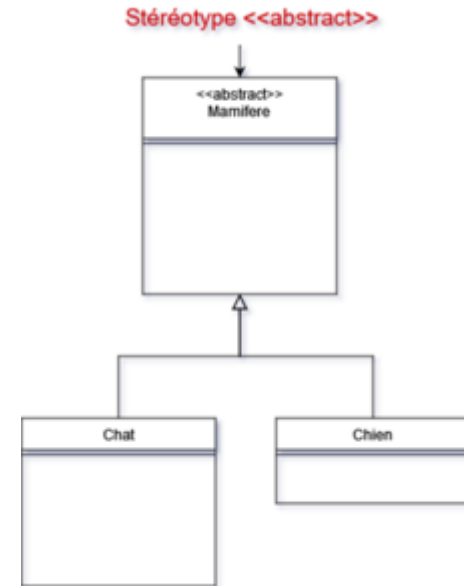


- Une classe mère contient des caractéristiques communes pour ses classes filles.
- Généralisation des attributs et des méthodes au sein d'une super classe.
- Spécialisation dans les sous-classes.

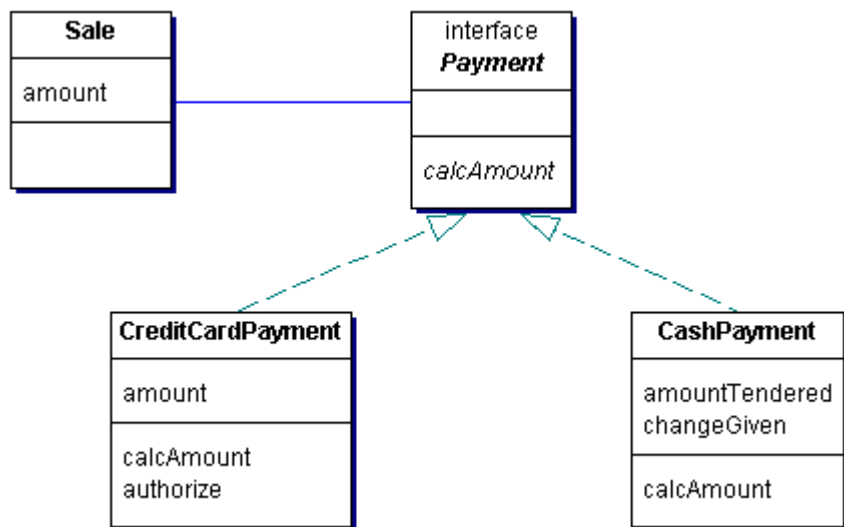
Classe abstraite



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



Source image informIT



- 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





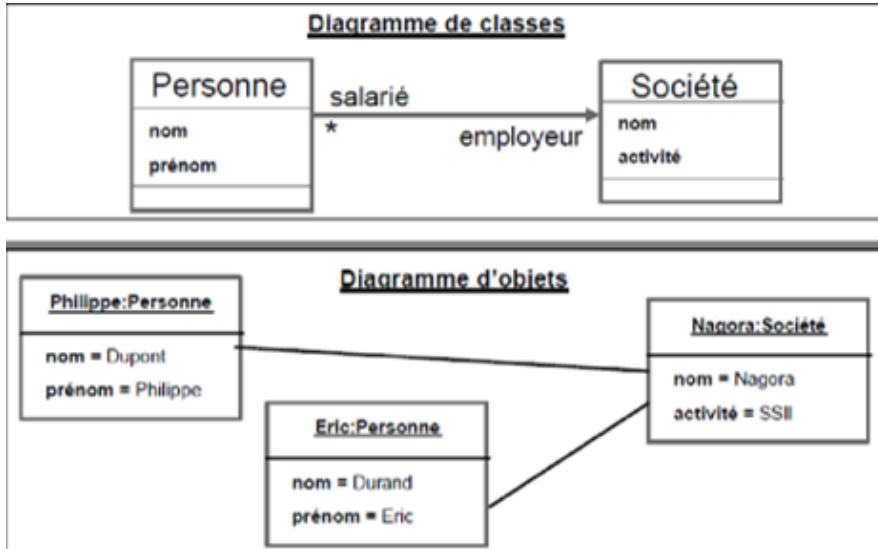
EXERCICE 6

1-exercices/exercice6.md

V. DIAGRAMME D'OBJETS



Source image Mohammed Nemiche



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

EXERCICE





EXERCICE 7

1-exercices/exercice7.md

MERCI DE VOTRE ATTENTION ET PARTICIPATION
Glodie Tshimini
contact@tshimini.fr

