

# FORMATION

## Introduction POO & UML

---

23/10/2023

Glodie TSHIMINI  
contact@tshimini.fr



2itechacademy.com

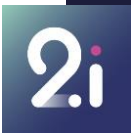
# PLAN

- I.** Les concepts de la programmation orientée objet
- II.** UML et généralités
- III.** Diagramme d'activité
- IV.** Diagramme de cas d'utilisation
- V.** Diagramme de classes
- VI.** Diagramme d'objets
- VII.** Diagrammes de composant et de déploiement

# PRÉAMBULE ([source image unsplash Jessica Mangano](#))

Le support sert principalement à **illustrer** les notions abordées avec beaucoup d'images et des diagrammes. Il est fortement recommandé de **prendre des notes** du cours effectué à l'oral.





# I. LES CONCEPTS DE LA PROGRAMMATION ORIENTÉE OBJET

---



[2itechacademy.com](https://2itechacademy.com)

# Abstraction

- L'abstraction est un principe qui permet de **ne retenir que les informations pertinentes pour modéliser un concept**. Autrement dit, on s'abstrait de tous les détails inutiles pour se focaliser uniquement sur l'essentiel.
- Par exemple, dans une application informatique, un utilisateur aura les caractéristiques suivantes e-mail, nom, prénom, date de naissance. On s'abstrait de représenter toutes les autres caractéristiques de sa personne, si n'est pas pertinent pour l'application.
- Autres exemples :
  - Numéro de sécurité sociale pour les systèmes de santé
  - Numéro de compte pour les systèmes bancaires
  - Numéro fiscal de reference pour les impôts
- Le principe d'abstraction s'applique également sur la modélisation avec des diagrammes UML en gardant le niveau de details adequate en fonction de la phase du projet.

# 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** qui vont collaborer 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.

## Source image Jordan Opel



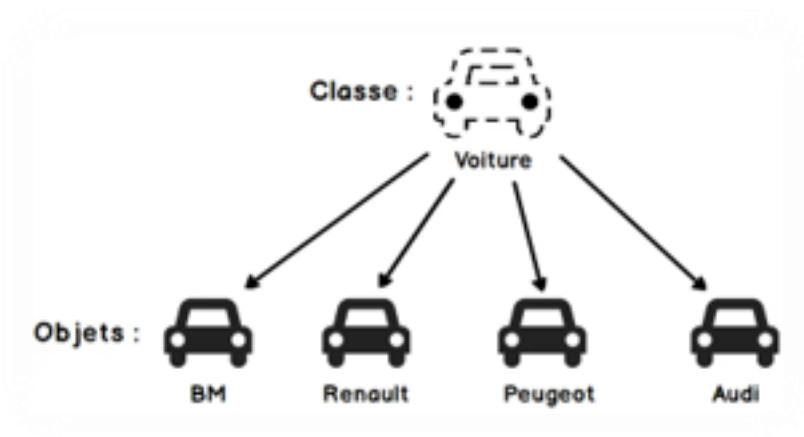
- Une personne peut être représentée comme un objet en informatique.
  - Caractérisée par un ensemble d'attributs :
    - Couleur des yeux
    - Taille
    - Poids
  - Peut réaliser un certain nombre d'actions :
    - Marcher
    - Courir
    - Parler
    - Etc.

# 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 futures objets.
- Prenons des exemples de la vie courante :
  - Moule à gâteau
  - Plan ayant servi à construire des maisons
  - Template d'un CV
  - Template du dossier professionnel



[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 :
  1. Un nom
  2. Des **attributs**
  3. Des **comportements**

# EXERCICE

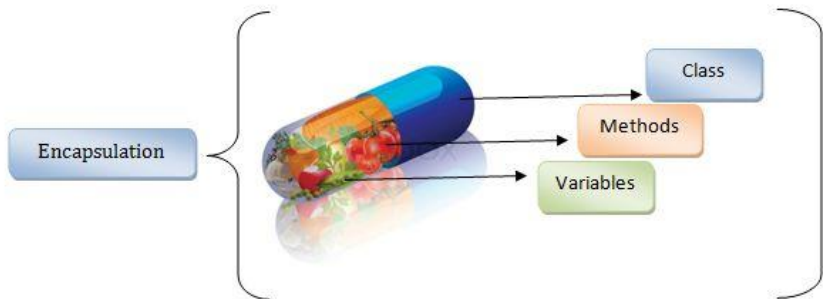




# EXERCICE 1

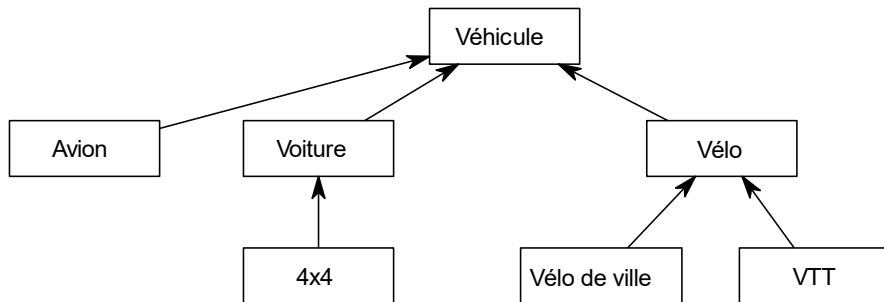
1-exercices/exercice1.md

[Source image logicmojo](#)



- On parle d'encapsulation, lorsqu'un objet est lui-même en capacité de connaître ses propres attributs et comportements.
- Parfois, on aura besoin de **cacher** une partie des attributs et comportements d'un objet.
- Plusieurs niveaux d'encapsulation :
  - **Privé** : attributs et/ou comportements accessibles uniquement par l'objet lui-même
  - **Protégé** : accessibles par l'objet lui-même et ses descendants (classes filles)
  - **Public** : accessibles par tout le monde
- Des exemples de la vie courante :
  - ADN
  - Numéro de série
  - Le solde de son compte

## Source de l'image perso-esiee



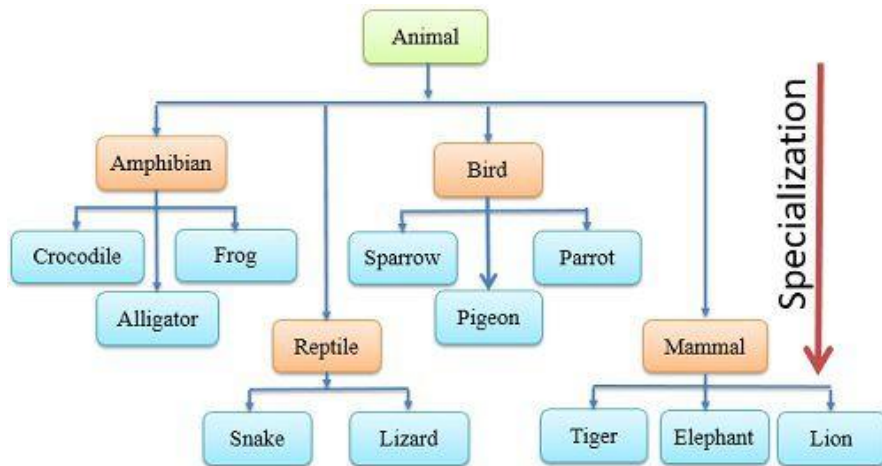
Une voiture est *classe* donc un *modèle*, lui-même créé à partir d'un autre modèle le véhicule.

Donc on peut dire qu'une voiture est un véhicule.

On peut également dire qu'un avion est un véhicule.

# Généralisation et spécialisation

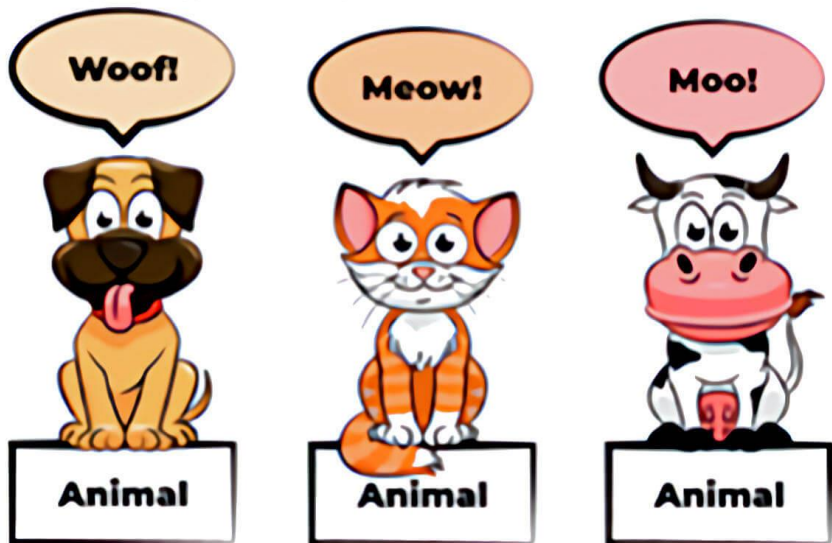
[Source image letsstudytogether](https://www.letsstudytogether.com)



- 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 un *Serpent*, *Lézard*, etc.
- *Reptile*, *Mammifères*, *Oiseaux*, *Amphibien* sont des spécialisations de la classe *Animal*. Elles sont appelées **sous-classe** ou **classes filles**.

# Polymorphisme

## polymorphisme animal



[www.aquaportail.com](http://www.aquaportail.com)

- Lorsque les sous-classes peuvent implémenter (réaliser) les **comportements** à leur façon selon leurs spécificités, on parle alors de **polymorphisme**.
- Autrement dit le comportement « *crier* » pour un *animal* peut prendre plusieurs formes.

## Source image ouest-france



- 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



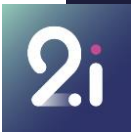
# EXERCICE





# EXERCICE 2

1-exercices/exercice2.md



## II. UML ET GÉNÉRALITÉS

---



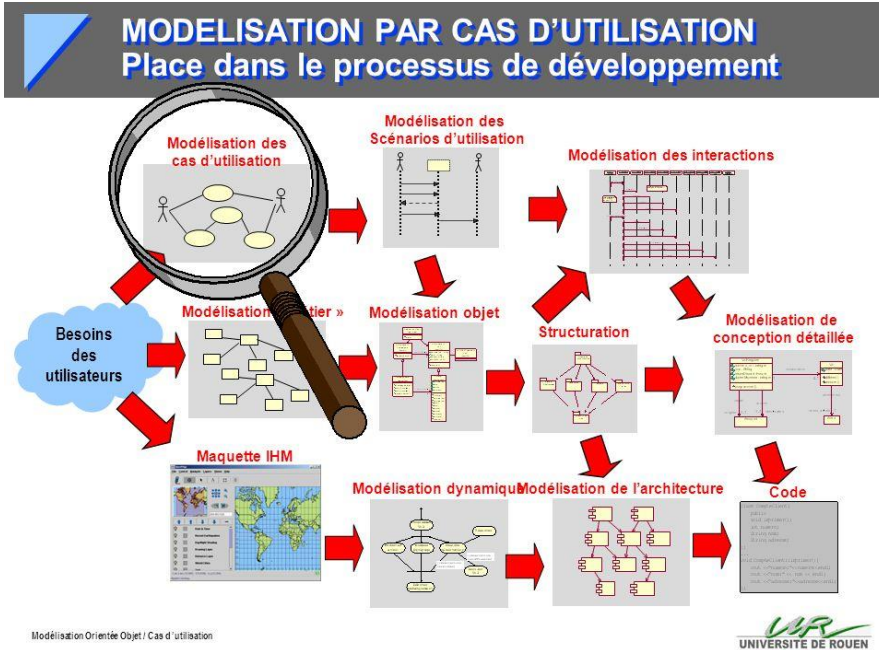
[2itechacademy.com](https://2itechacademy.com)

[Source image wikipédia](#)



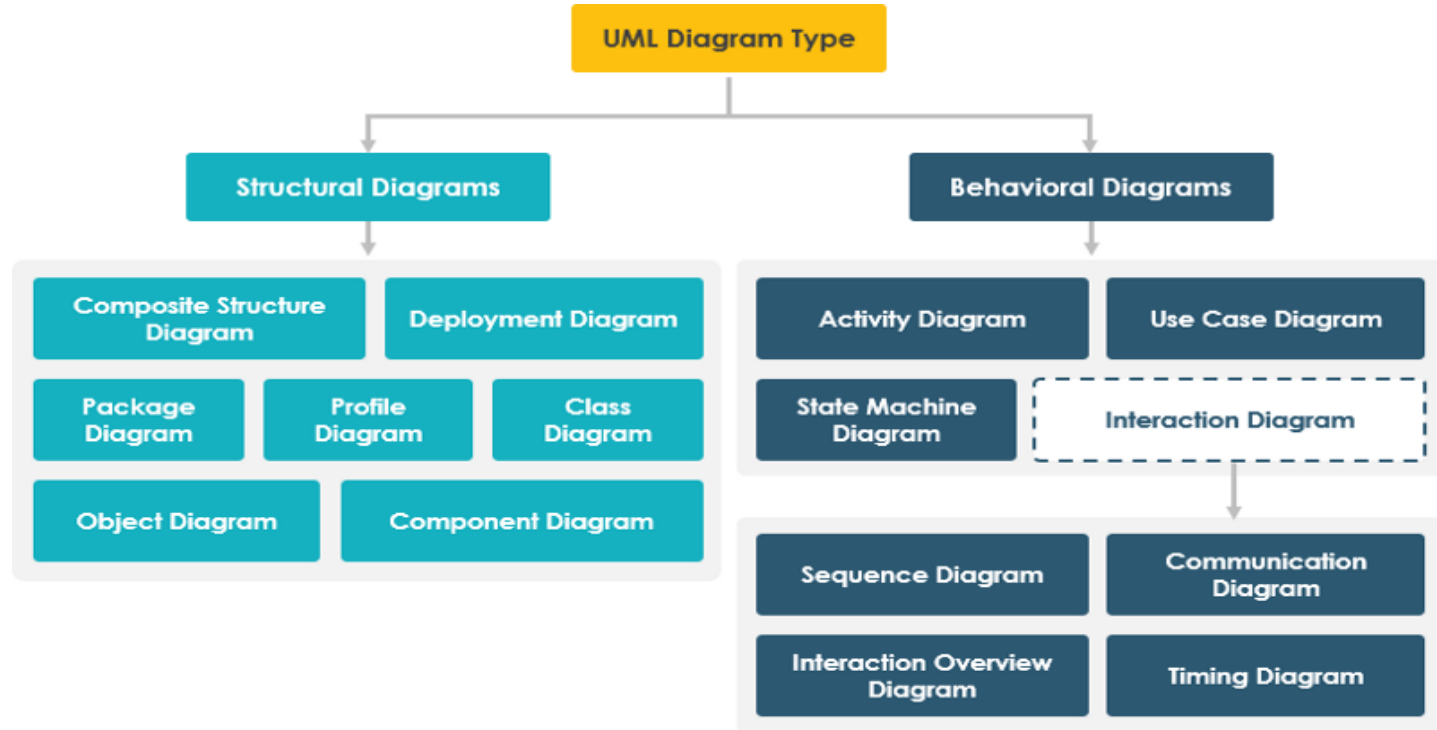
- Unified Model Language (*Langage de modélisation objet unifié*)
  - 1997 : UML 1
  - 2006 : UML 2
- Fusion de 3 méthodes
  - BOOCH
  - OMT
  - OOSE
- Langage graphique qui permet de modéliser une application informatique avec des diagrammes
- Plusieurs méthodes de gestion de projet dont les plus connues, le RUP et 2TUP intègre UML dans le processus

## Source image slideplayer



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

# Les diagrammes UML ([source image cybermedian](#))

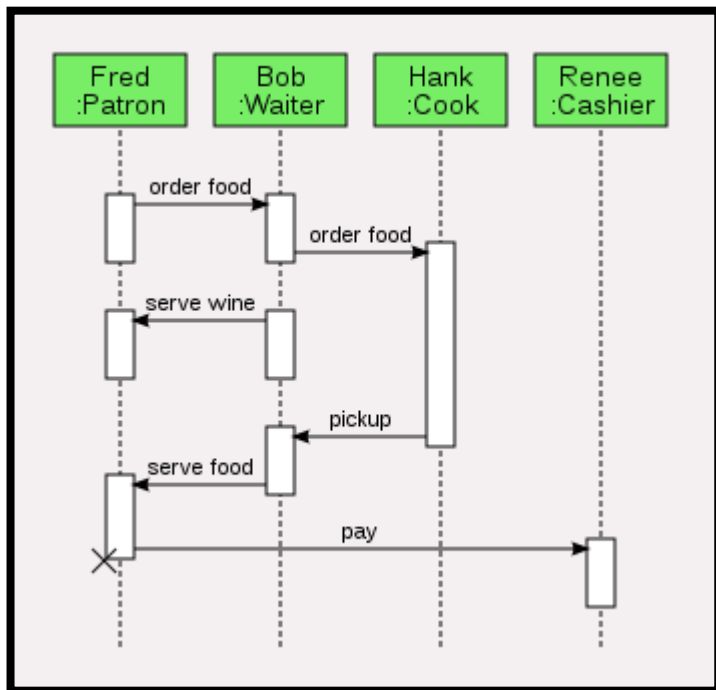


# Les diagrammes UML

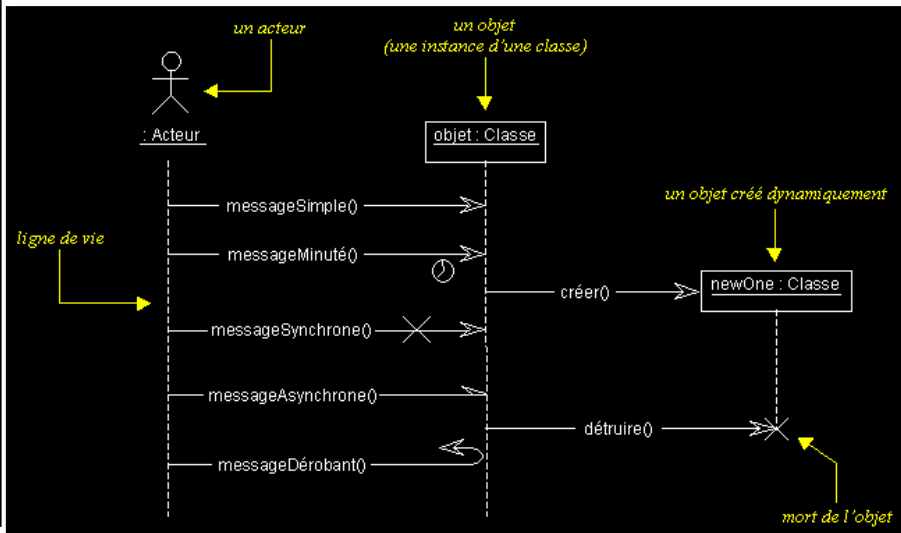
- **Diagramme de cas d'utilisation**
    - Représente le système d'un point de vue de l'utilisateur
  - **Diagramme de séquence**
    - Représente d'un cas d'utilisation en intégrant la notion du temps
  - **Diagramme de communication**
    - Autre représentation du diagramme de séquence
  - **Diagramme d'état-transition**
    - Représente les différents états d'un objet durant son cycle de vie
  - **Diagramme d'activité**
    - Représente séquentiellement et conditionnellement les états de plusieurs objets
- **Diagramme de classe**
    - Représente de la structure interne du système
  - **Diagramme objet**
    - Permet de vérifier le diagramme de classes
  - **Diagramme de package**
    - Regroupe et sépare les classes dans des sous-ensembles qui communiquent entre elle.
  - **Diagramme des composants**
    - Représente les composants du système
  - **Diagramme de déploiement**
    - Représente les composants matérielles du système

# Diagramme de séquence

[Source image wikipédia](#)



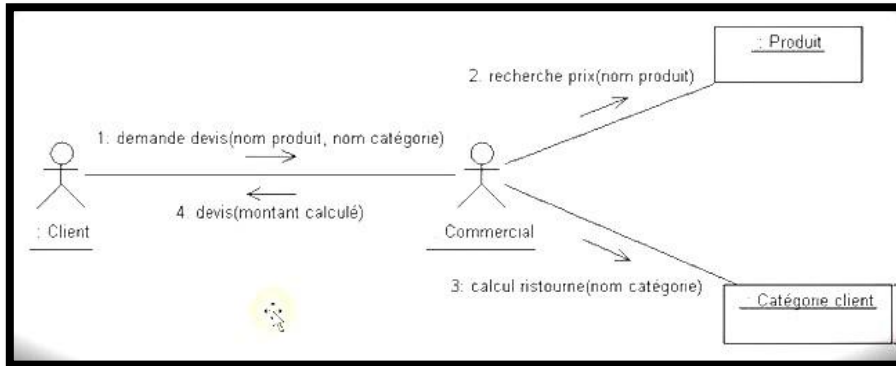
[Source image UML.free.fr](#)



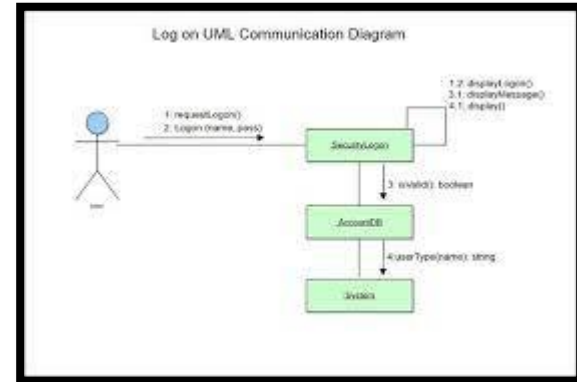


# Diagramme de communication

[Source image celamrani](#)

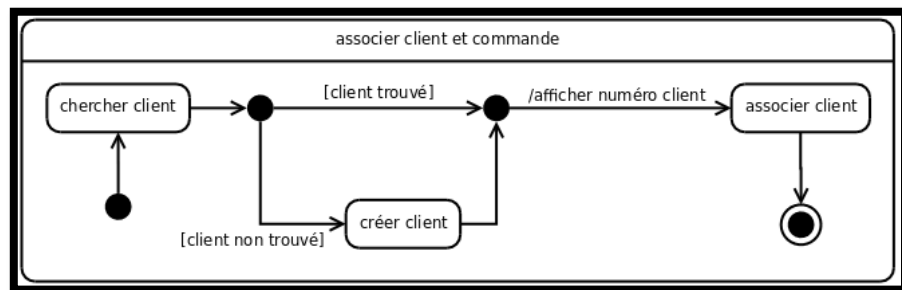


[Source image GitMind](#)

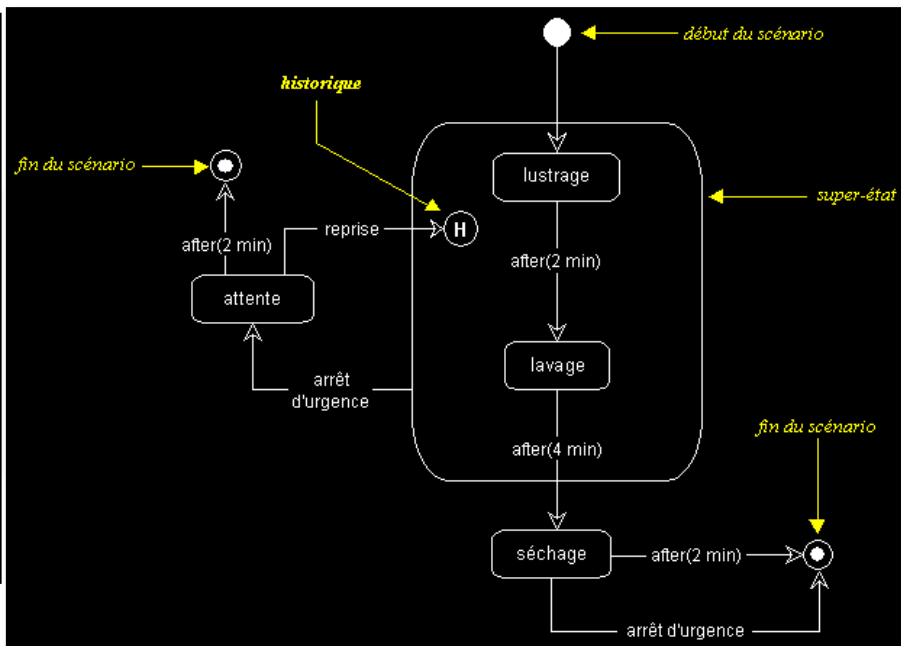


# Diagramme d'état-transition

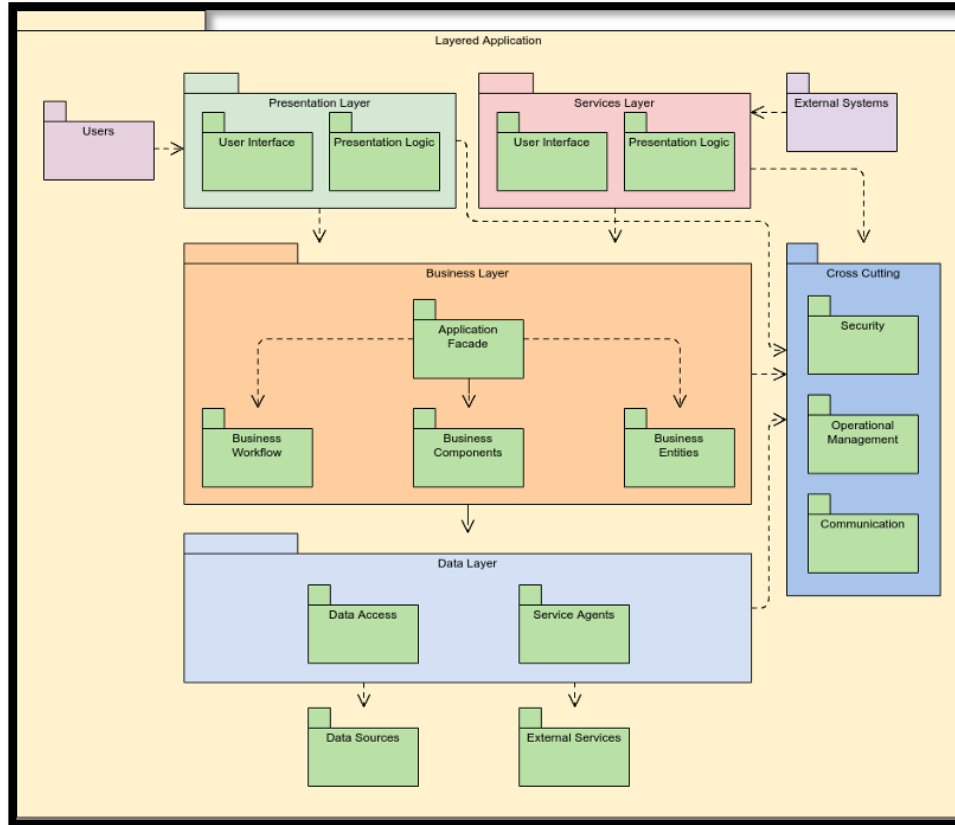
[Source image laurent audibert](#)

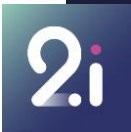


[Source image UML.free.fr](#)



# Diagramme de package ( [source image cybermedian](#) )





## III. DIAGRAMME D'ACTIVITÉ

---

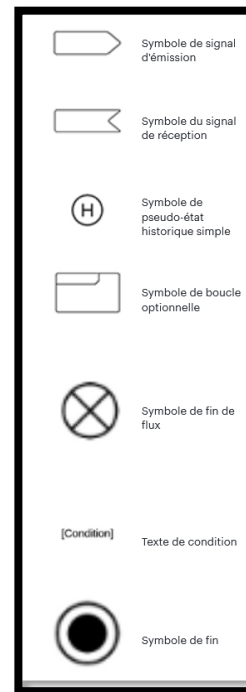
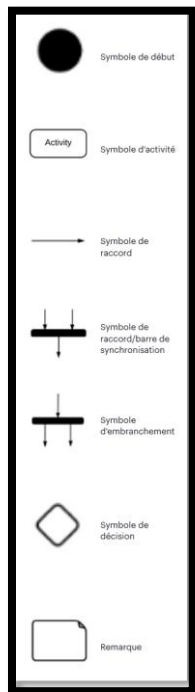


[2itechacademy.com](https://2itechacademy.com)

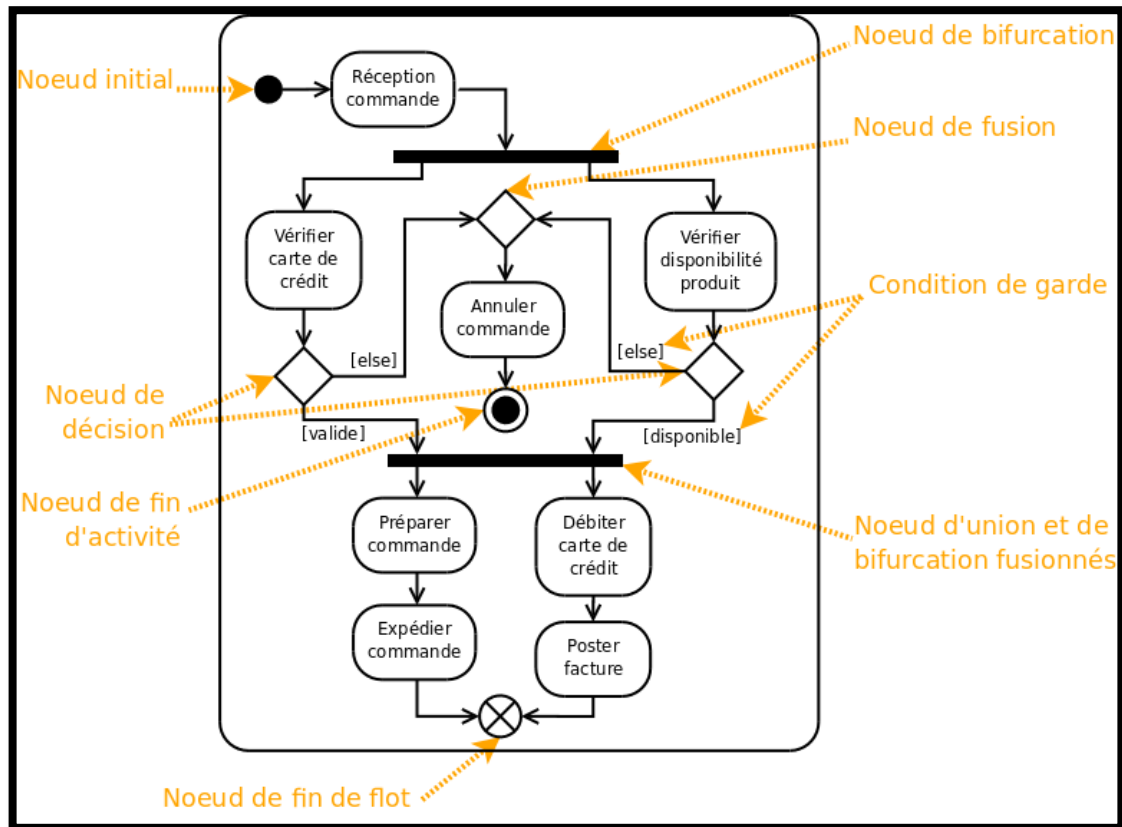
# Diagramme d'activité

- Représentation séquentielle et éventuellement conditionnelle des états de plusieurs objets associé à une activité spécifique.
- Le diagramme d'activité
  - Représente également les différents transitions entre les activités.
  - Représente l'exécution de plusieurs activités en parallèle
- Séquentielle
  - Attendre la fin d'une activité avant de commencer une nouvelle
- Conditionnelle
  - Certains activités sont possibles uniquement lorsqu'une ou plusieurs conditions sont satisfaites.

# Formalisme (source image lucid-chart)



# Exemple ([source Laurent-Audibert](#))



# EXERCICE







# EXERCICE 3

1-exercices/exercice3.md



## IV. DIAGRAMME DE CAS D'UTILISATION

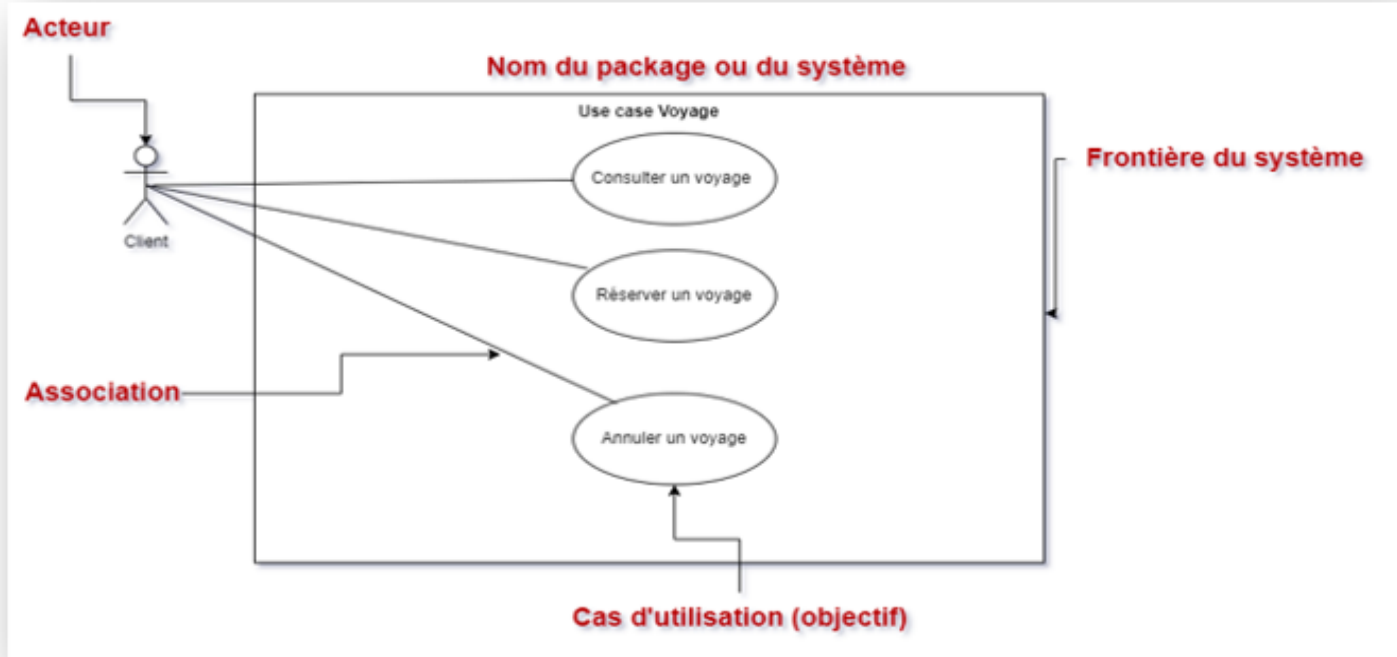
---



[2itechacademy.com](https://2itechacademy.com)

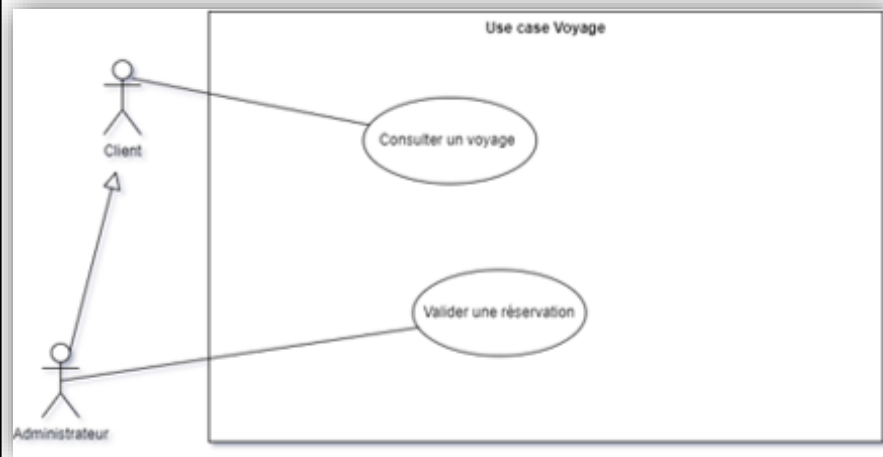
# Principes

- Schématiser l'expression des besoins d'un point utilisateur. Autrement dit c'est la représentation du système vu par l'utilisateur.
- Répondre aux questions **Qui** et **Quoi** ?
- Objectifs
  - Délimiter le périmètre fonctionnel.
  - Servir pour réaliser des tests fonctionnels.
  - Impliquer et communiquer avec le client.
  - Construire des interfaces IHM (d'autres diagrammes UML sont plus adaptés).
  - Communiquer entre membres de l'équipe



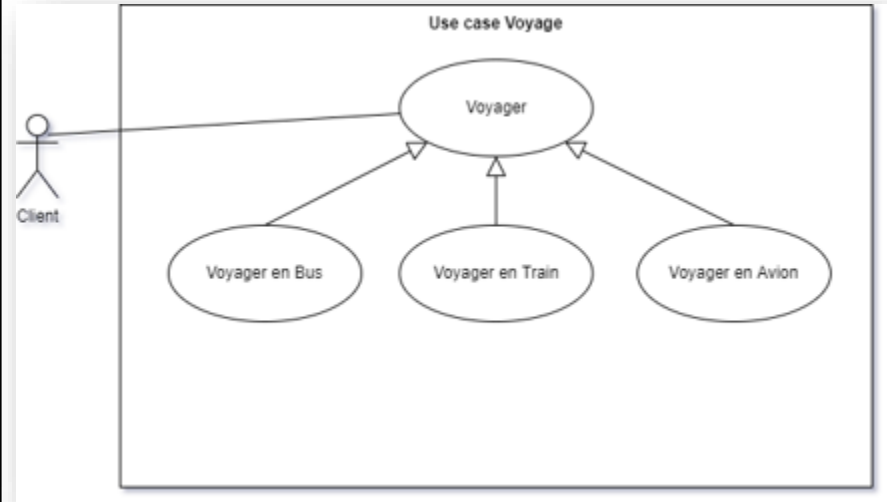
# Héritage entre acteurs

- 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.



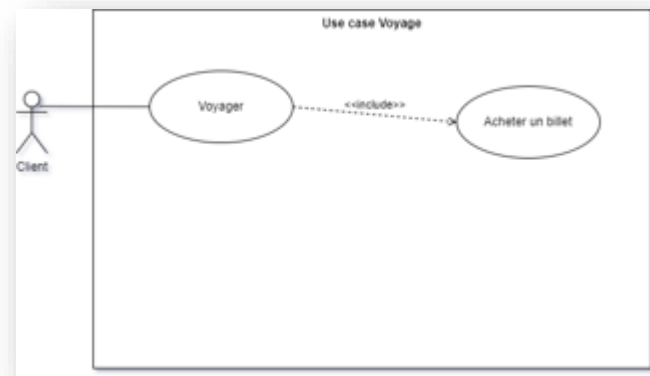
# Héritage entre cas d'utilisation

- 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.



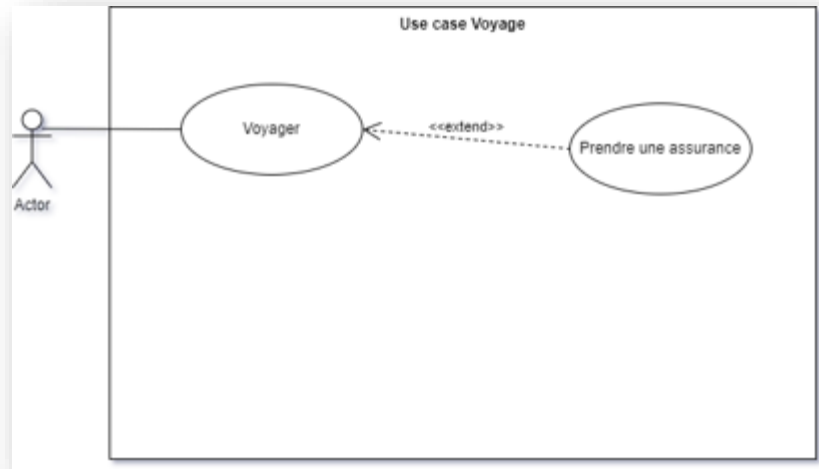
# Include : cas additionnel obligatoire

- La relation d'inclusion entre deux cas d'utilisation signifie que la réalisation d'un cas d'utilisation implique obligatoirement la réalisation d'un autre cas d'utilisation
- Pour « voyager », il faut obligatoirement « acheter un billet » .
- Généralement, les cas d'inclusion ne répondent pas directement à un besoin primaire de l'acteur.



# 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, ils apportent un peu plus de clarté au diagramme mais ils peuvent également surcharger le diagramme.
- On peut s'en passer pour gagner en lisibilité.





# EXERCICE





# EXERCICE 4

1-exercices/exercice4.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 ».

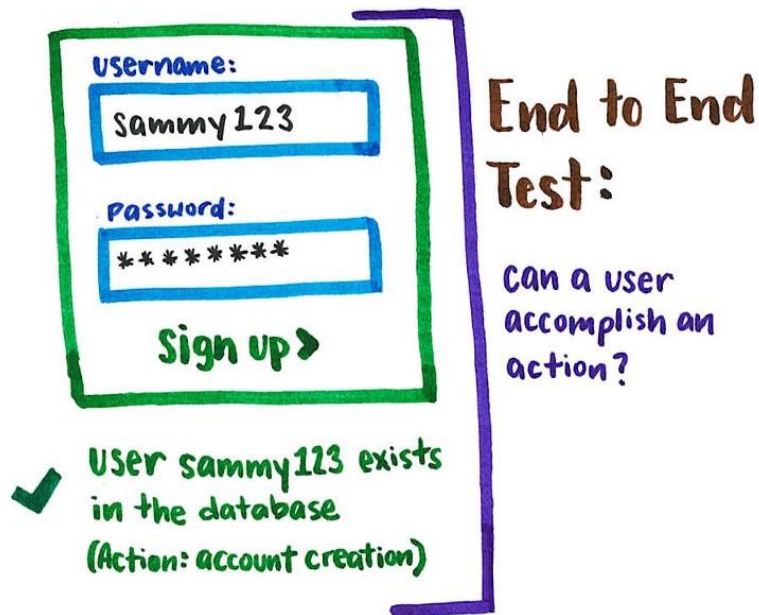
Titre	Ajouter un domaine
Acteurs	Élève
Description	Lorsqu'un utilisateur du système veut accéder à l'application, il doit saisir son login et son mot de passe : ensuite le système vérifie s'ils sont corrects ou pas afin d'autoriser ou bien refuser l'accès.
Description des scénarios	<p><b>Scénario nominal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

### Le format est libre

- 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



## 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 5

1-exercices/exercice5.md



# V. DIAGRAMME DE CLASSES

---



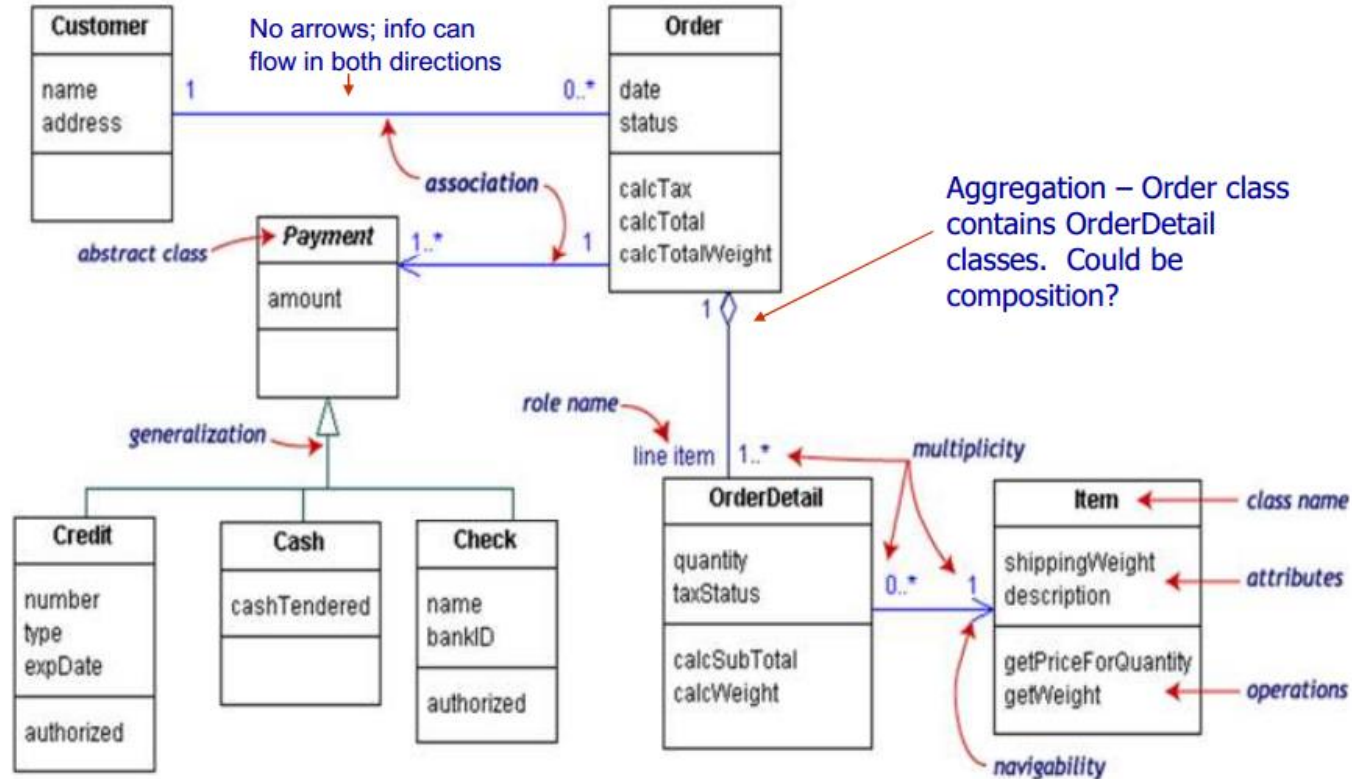
[2itechacademy.com](https://2itechacademy.com)

# Principes

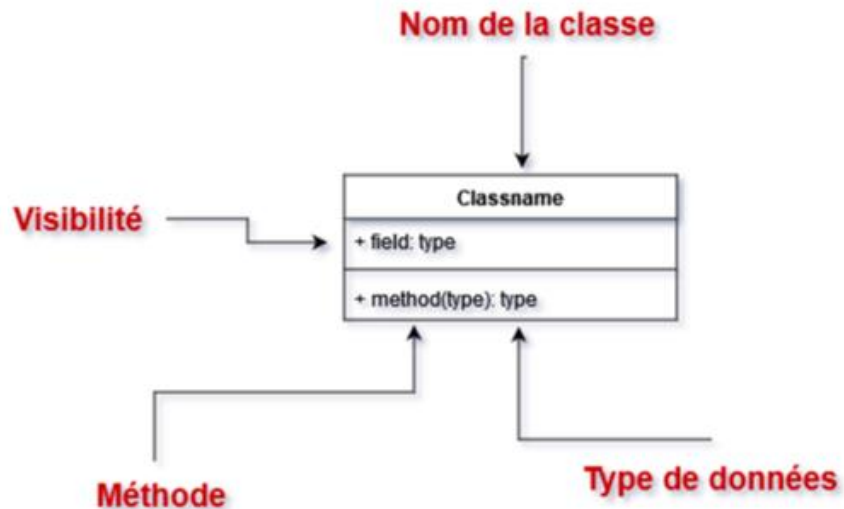
- Schématiser la structure interne d'un système qui sera implémenté plus tard à l'aide d'un langage de programmation orienté objet (POO)
  - *Classes*
  - *Attributs*
  - *Opérations*
  - *Relations*
- Autrement dit, représente les *données* et les *traitements* du système.
- Modéliser des bases de données relationnelles ou objet.
- Le niveau d'abstraction ou du détail dépend de vos objectifs et de la phase à laquelle le projet se trouve.



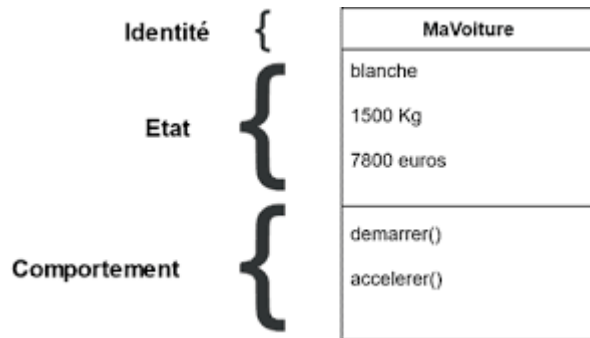
# Mapping diagramme de classe : [source image stackexchange](#)



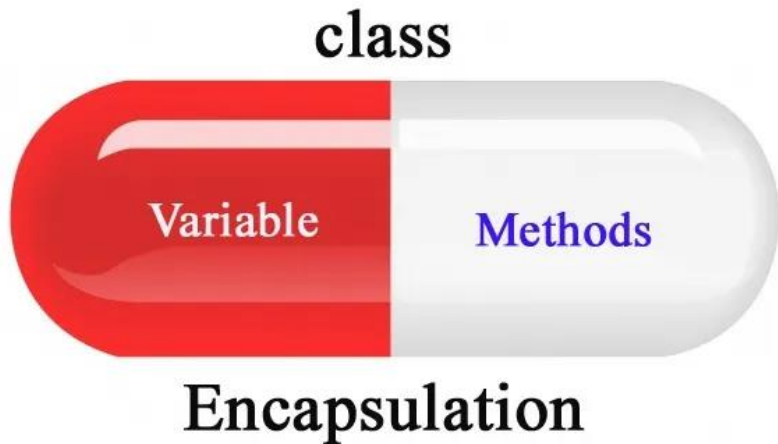
# Zoom sur une classe



[Source image data-transitionnumerique](#)

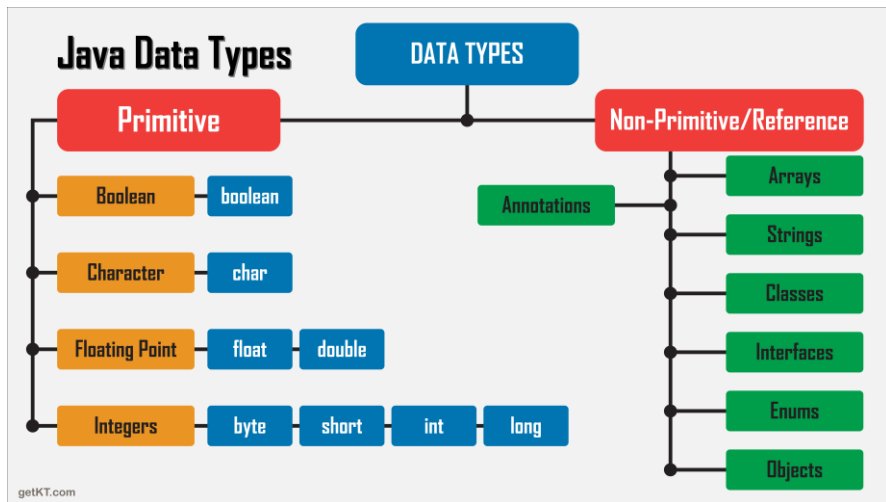


[Source image code4coding](#)



- 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.

## Source image getkt



- On utilise les types primitifs de l'algorithmie et éventuellement les énumérations (liste fermée des données)
  - *Integer*
  - *Float*
  - *Boolean*
  - *String*
- On n'utilise pas les types spécifiques à un langage de programmation
- 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/les classe B.

[Source image stackoverflow](#)


Inheritance 

Dependency 

Aggregation 

Containment 

Association 

Directed Association 

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

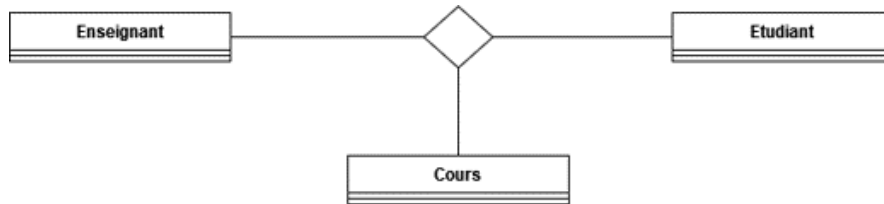
*PS : attention l'ordre des images à gauche ne correspond pas à la liste des associations listée à droite*

# Association binaire



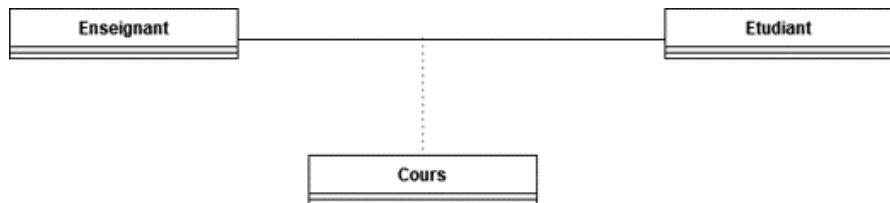
- La **plus rependue** et celle qu'il faut privilégier par rapport aux autres types d'association (n-aire et classe d'association).
- La **plus lisible et compréhensible**.

# Association N-aire



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

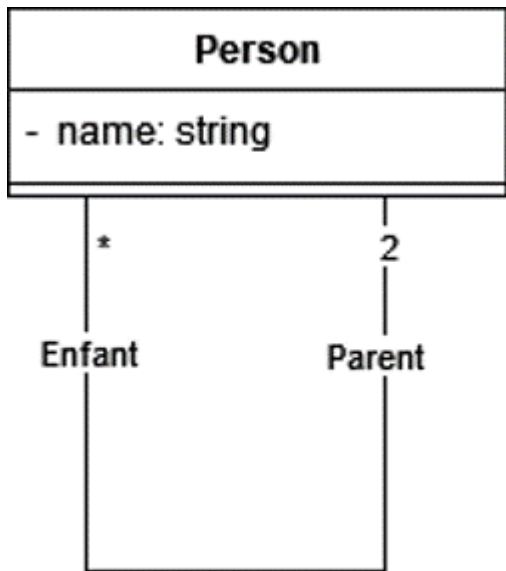
# Classe d'association



- 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 pour apporter plus de **clarté** à l'association.

# EXERCICE





# EXERCICE 6

1-exercices/exercice6.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

# EXERCICE



## EXERCICE 6

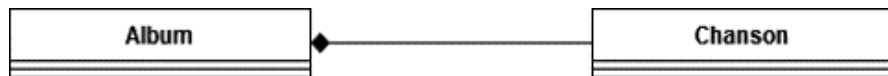
Reprendre l'exercice 6 pour y ajouter les multiplicités

# 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.

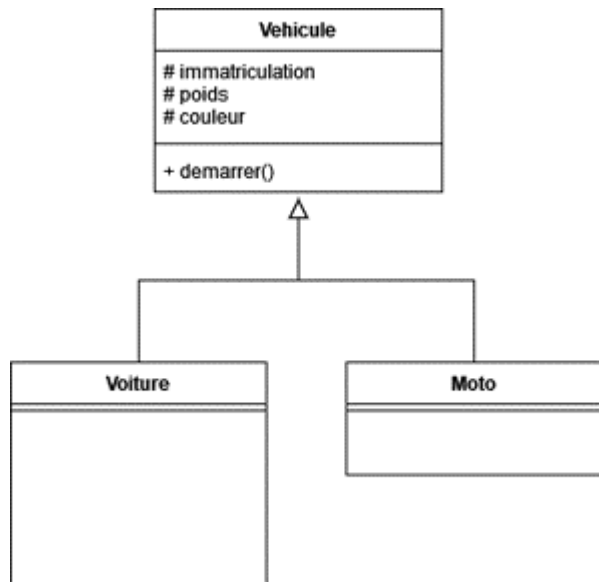
# Agrégation forte



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

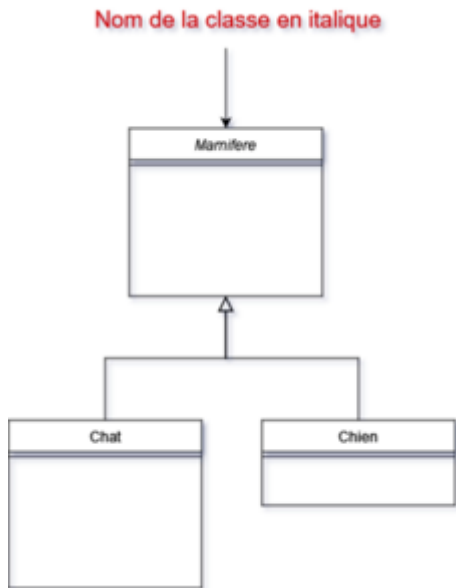


# Héritage entre les classes

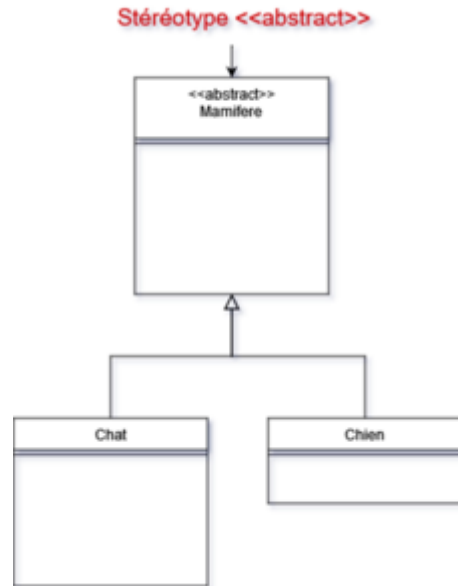


- Une classe mère contient des caractéristiques communes pour ses classes filles.
- Généralisation des attributs et des méthodes au sein d'une super classe.
- Spécialisation dans les sous-classes.

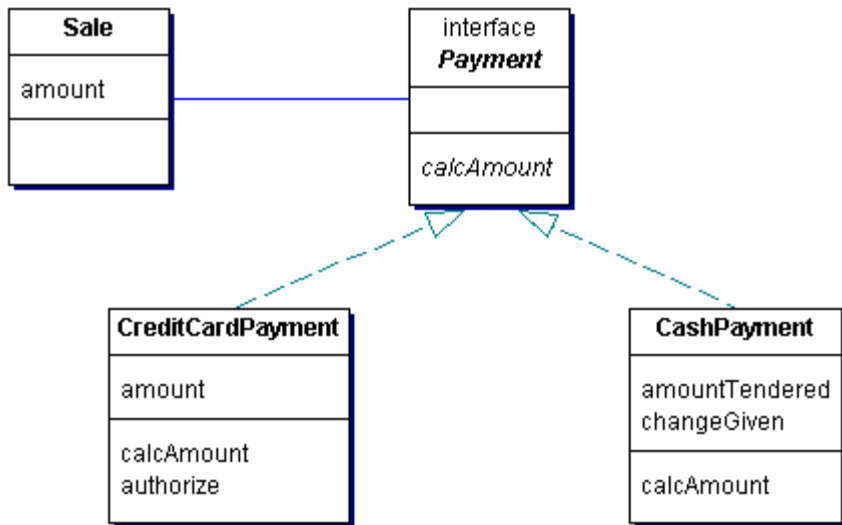
# Classe abstraite



- 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).
- Dans la programmation, les méthodes abstraites ont uniquement une **signature** (nom, paramètres et valeur de retour) et n'ont pas de corps (le contenu devra être écrit dans chaque classe qui va implémenté l'interface)

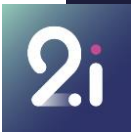
# EXERCICE





# EXERCICE 7

1-exercices/exercice7.md



## VI. DIAGRAMME D'OBJETS

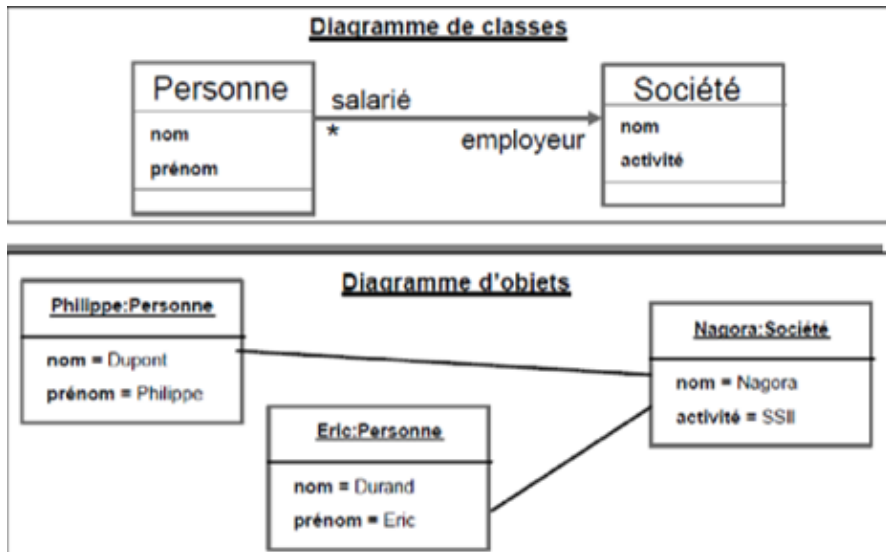
---



[2itechacademy.com](https://2itechacademy.com)

# Diagramme objet

**Source image Mohammed Nemiche**



- Réaliser des tests de son diagramme de classes grâce à l'instanciation des objets qui servent d'exemple pour vérifier par exemple les règles de gestion (RG).
- Les RG sont un ensemble d'exigences, comportements, restrictions etc. qui détermine le fonctionnement d'une entreprise ou d'une activité.

# EXERCICE

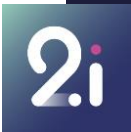






# EXERCICE 8

1-exercices/exercice8.md



## VII. DIAGRAMMES DE COMPOSANT ET DE DEPLOIEMENT

---



[2itechacademy.com](https://2itechacademy.com)

## Diagramme de composant

- Modélise la répartition des composants logiciels
- Les composants logiciels peuvent être
  - Modules
  - Fichiers
  - Exécutables (programme)
  - Librairies

## Diagramme de déploiement

- Modélise la répartition des composants logiciels dans les composants matérielles (physique)
- Un nœud est un composant matérielle (ressource physique)
- Un composant matériel est un élément qui "héberge" un composant logiciel
- Un composant logiciel est un élément qui fournit un service
- Les composants sont associées entre elles par une interface

# Interface

- Une interface est une association (communication) entre 2 composants
- Un composant est indépendant et remplaçable par un autre composant qui présente une interface similaire
- On distingue 2 types d'interface
  - Requis : obligatoire pour bénéficier d'un service, il est fourni par le composant lui-même.
  - Fournie : mise à disposition par le fournisseur (un autre composant) du service

*PS : à ne pas confondre avec l'interface (contrat) qu'on trouve dans un diagramme de classes.*

# Exemple de diagramme de composant

([source image Bouchra Bouihi](#))

## Diagramme de Composant

### Exemple : Transfert des données par internet

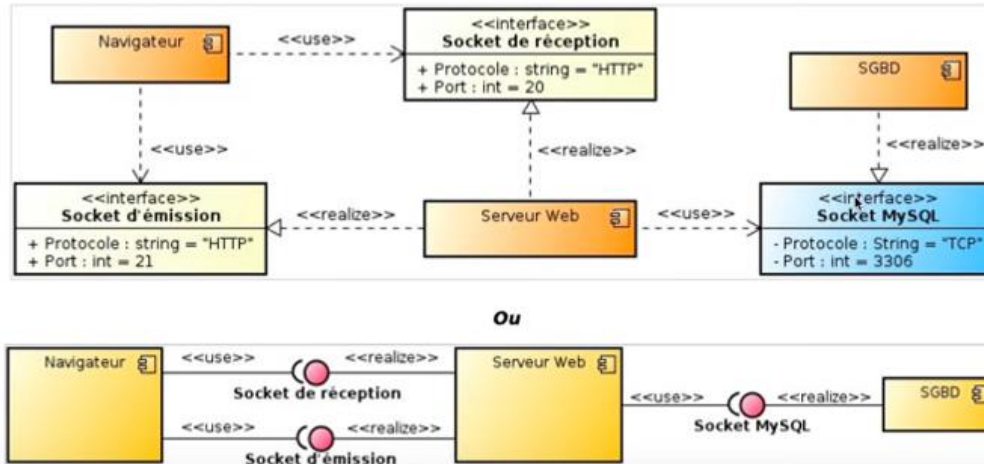
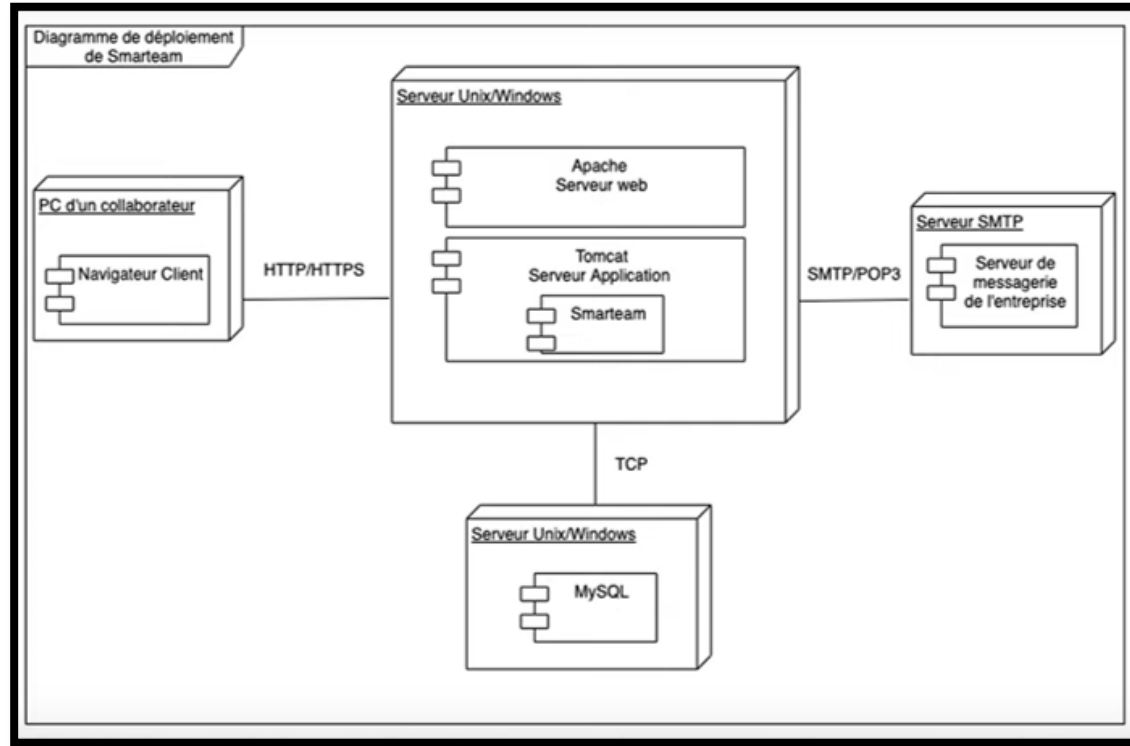


Image cours remy-manu

# Exemple diagramme de déploiement

([source image Bouchra Bouihi](#))



# EXERCICE

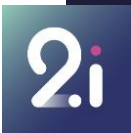




# EXERCICE 9

1-exercices/exercice9.md





**MERCI DE VOTRE ATTENTION ET PARTICIPATION**  
**Glodie Tshimini**  
**[contact@tshimini.fr](mailto:contact@tshimini.fr)**



[2itechacademy.com](https://2itechacademy.com)