

ib  
cegos

# Initiation à la programmation





# Présentation

- Consultant Formateur et Développeur depuis 2017
  - ▶ Certifié Codebase IT Expert Trainer
  - ▶ Certifié Codebase IT Expert Trainer At POE
  - ▶ Certifié SCRUM Professional Developer I

- J'interviens dans la formation et développement sur les thèmes suivants
  - ▶ Gestion de projet
  - ▶ HTML & CSS & SASS & UX
  - ▶ JavaScript et son écosystème
  - ▶ PHP et son écosystème
  - ▶ CMS (WordPress et PrestaShop)
  - ▶ SEO
  - ▶ Versioning (Git, GitHub, GitLab)
  - ▶ Modélisation UML et Merise
  - ▶ Bases de données



## PRÉAMBULE (source image unsplash Jessica Mangano)

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



# Plan de la formation

J 1 - Matin

DAILY

ALGORITHMIE

J 2 - Matin

DAILY

PROGRAMMATION ORIENTEE OBJET

J 3 - Matin

DAILY

UML

J 1 – Après midi

ALGORITHMIE

J 2 – Après midi

TRAVAUX PRATIQUES

J 3 – Après midi

UML

# PLAN du cours sur l'algorithmie et la programmation

- I. Introduction
- II. Variables, opérateurs et tableaux
- III. Structures de contrôle
- IV. Fonction
- V. Tableaux multidimensionnels
- VI. Programmation Orientée Objet (POO)

# I. INTRODUCTION



# QU'EST-CE QU'UN ALGORITHME ?

- ❑ **Suite d'opérations élémentaires permettant d'obtenir le résultat final déterminé à un problème.**

*(source : Apprendre à programmer Christophe Dabancourt).*

- ❑ Décrit un traitement sans l'exécuter sur une machine.
- ❑ Fournit un résultat identique dans des conditions similaires.
- ❑ Exemples du quotidien
  - ❑ Recette de cuisine ;
  - ❑ Aller d'un point A à un point B .

# QU'EST-CE QU'UN PROGRAMME ?

- ❑ **Exécution de l'algorithme sur une machine.**
- ❑ Pour que le programme puisse être exécuté, il faut utiliser un langage que la machine peut comprendre : un **langage de programmation**.
- ❑ Suite d'instructions qui sont évaluées par le processeur sur lequel tourne le programme.
- ❑ Les **instructions** utilisées dans le programme représente le **code source**.





# DIFFÉRENCES ENTRE LANGAGES DE BAS ET HAUT NIVEAU

## BAS NIVEAU

- Un langage de bas niveau est un langage qui est considéré comme **plus proche du langage machine** (binaire) plutôt que du langage humain.

Il est en général plus difficile à apprendre et à utiliser mais **offre plus de possibilité d'interactions** avec le hardware de la machine.

## HAUT NIVEAU

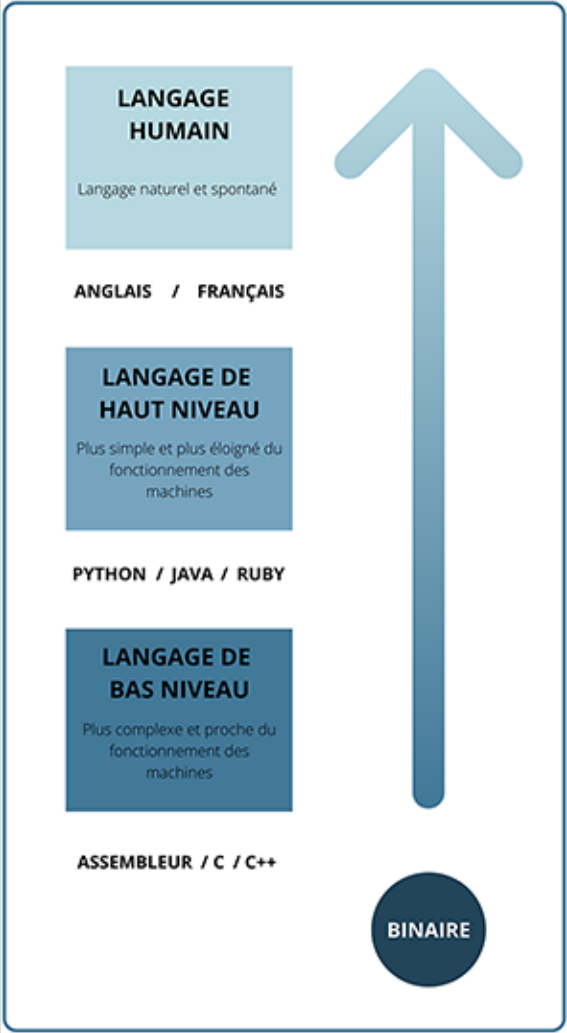
- Un langage de haut niveau est le contraire, il **se rapproche plus du langage humain** et est par conséquent **plus facile à appréhender**.

Cependant, les **interactions sont limitées** aux fonctionnalités que le langage met à disposition.



# ILLUSTRATION

[Source image Machine Learning et Deep Learning - ENI](#)





# DIFFÉRENCES COMPILATION ET INTERPRÉTATION

## COMPILATION

- ❑ La compilation d'un programme consiste à **transformer toutes les instructions en langage machine** avant que le programme puisse être exécuté.
- ❑ Par conséquent il sera nécessaire de refaire la compilation après chaque modification du code source.

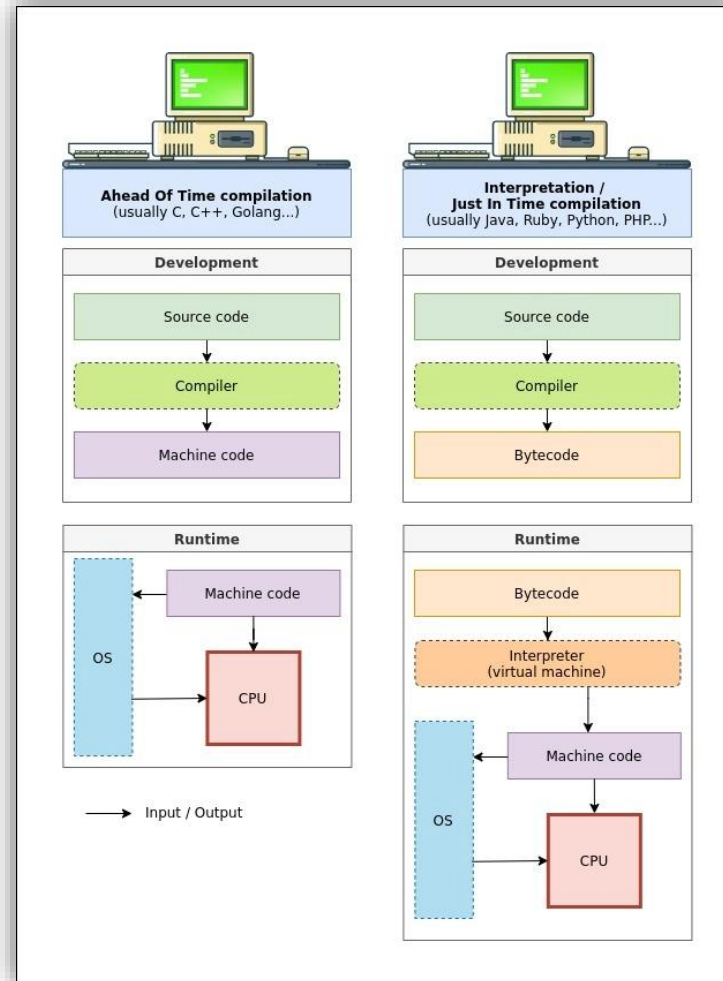
## INTERPRÉTATION

L'interprétation d'un programme consiste à **traduire les instructions en temps réel** (on run time).

Dans ce cas le code source est lu à chaque exécution et par conséquent les changements apportés au code seront pris en compte immédiatement.

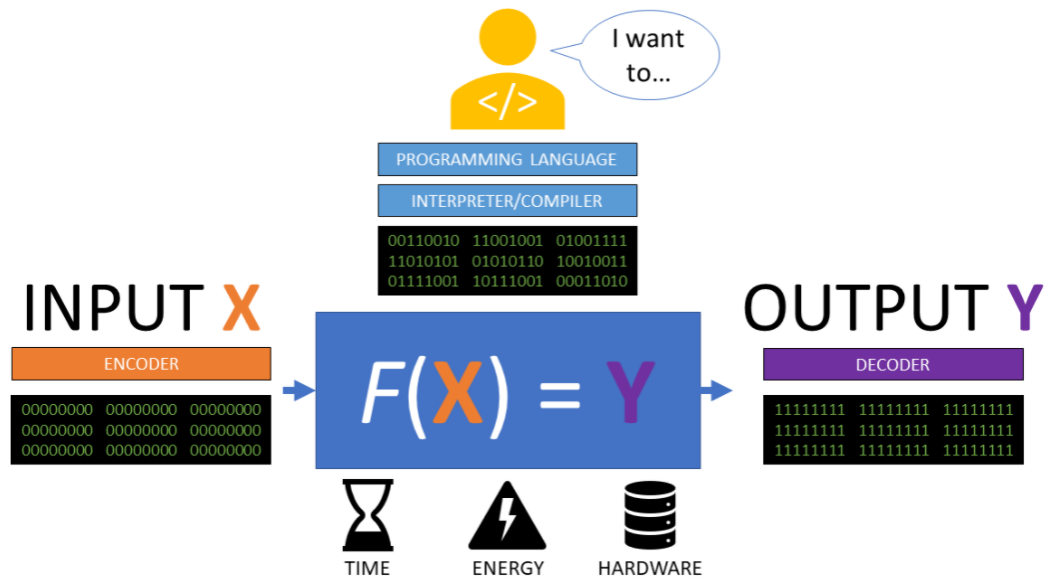
# ILLUSTRATION

[Source image thevaluable.dev](http://thevaluable.dev)



# RESUMÉ

[Source image edutechwiki](#)



- ❑  $F(X) = Y$
- ❑  $X$  représente l'*input* (**entrée**)
- ❑  $Y$  représente l'*ouput* (**sortie**)
- ❑  $F()$  représente la **fonction** qui permet de transformer  $X$  en  $Y$



# LA PROGRAMMATION

- ❑ Les éléments fondamentaux :
  - ❑ Les variables
  - ❑ Les opérateurs
  - ❑ Les structures de contrôle
  - ❑ Les fonctions
  - ❑ Les tableaux (ou array)
  - ❑ Les objets

## II. Variables, opérateurs et tableaux





# Variables et constantes

## Variable

- Une information (donnée) qui est stockée dans la RAM de la machine qui va exécuter le programme.
- Exemple manipulation des variables en JavaScript
  - Mot-clé ***let*** suivi du nom de la variable
  - Initialiser ou déclarer une variable : *let firstName;*
  - Initialiser et affecter ou assigner une valeur : *let lastName = 'Tshimini'*
  - Modifier la valeur : *lastName = 'Doe'*

## Constante

- Variable non-modifiable
- Exemple de manipulation d'une constante avec JavaScript
  - Mot-clé ***const***
  - Déclarer une constante: *const PI = 3.14;*

Les déclarations de variables ou des constantes doivent être précises - Attention à la casse !

- *lastname* et *lastName* sont 2 variables différentes.



# Convention de nommage

Les conventions ci-dessous varient en fonction des langages de programmation et des entreprises

camelCase

Génér. pour  
les variables  
et fonctions

PascalCase

Génér. pour les  
noms des  
classes

UPPERCASE

Génér. pour les  
variables  
globales et  
constantes

kebab-case

Génér. pour les  
noms des  
ressources  
Web, les  
classes CSS,  
etc.

snake\_case

Génér. pour  
les variables,  
fonctions et  
les noms des  
attributs des  
tables en  
base de  
données



# TYPE OU DOMAINE DE DÉFINITION

- ❑ Ensemble des valeurs que peut prendre une variable
- ❑ Primitifs :
  1. Un caractère alphanumérique (**char**)
    - Par extension une suite de caractère alphanumérique (**string**)
    - Utilisées pour représenter du texte
    - *Exemples : "a", "hello world", "5"*
  2. Chiffres(**number**), nombre entier, à virgule flottante, etc
    - Utilisés surtout avec des opérateurs mathématiques
    - Exemples : 10 , 12.5
  3. Valeurs booléennes (**booleans**)
    - valeurs dichotomiques (soit vrai, soit faux)  
*Exemples : **true**, **false***

# OPÉRATEURS

- ❑ Opérateur d'**affectation** ou d'**assignation** : = ou <=
- ❑ Les opérateurs **mathématiques** : +, -, /, DIV et MOD (%)
- ❑ Les opérateurs de **comparaison** : égalité (==), différence (!=), majeur (>) ou mineur (<)
- ❑ Les opérateurs **logiques** : AND (&) , OR (||) , NOT (!) .
- ❑ Opérateur de **concaténation** : +
- ❑ Opérateurs d'**incrément** : ++
- ❑ Opérateurs de **décrément** : --

# Concaténation, Transtypage

## concaténation

- Mettre bout à bout des chaînes de caractère
- Exemple en JavaScript

```
const firstName = 'Glodie'
const lastName = 'Tshimini'
// concaténation avec l'opérateur de concaténation +
console.info(firstName + ' ' + lastName)
// Concaténation avec ${} et ``
console.info(`${firstName} ${lastName}`)
// Glodie Tshimini
```

## Transtypage

- Changer le type d'une information par un autre type
- Exemple en JavaScript

```
let num = '3' // string
num = parseInt(num) // number
num = parseFloat(num) // number
num = num.toString() // string

// mot clé typeof sur une variable ou constante retourne son type
console.log(typeof num)
```



# Exercices 1 et 2

1-exercices/0-algos-poo/ex1.md

1-exercices/0-algos-poo/ex2.md

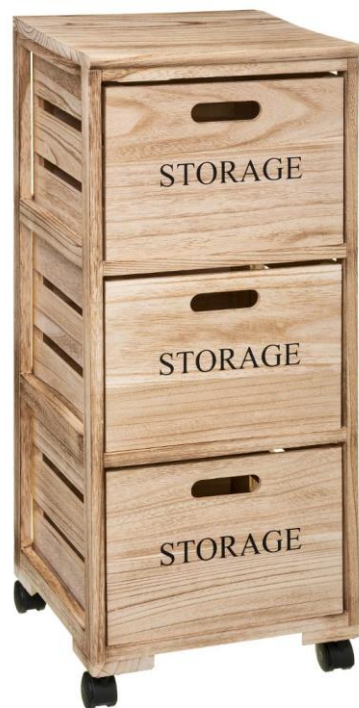


**30 min**



# Tableaux

- Liste indexée (ordonnée) d'éléments normalement appartenant au même domaine de définition (type)



## Déclaration d'un tableau :

- `Variable monTableau = [ ];`

## Remplissage d'un tableau :

- `monTableau[0] = 4;` : Mettre 4 à la première position 0
- `Lire monTableau[1])` : Accès à l'élément situé à l'indice 2 du tableau (aucune valeur a été affectée à cet indice dans cet exemple)

## Exemple d'une implémentation avec le langage JavaScript

```
170  const numbers = [1, 2, 3, 4, 5]
171  const names = ['Fatou', 'Maria', 'Solange', 'Sarah']
172  // Tableau à 2 dimensions
173  const persons = [
174    ['Fatou', 'Paris', 30, true],
175    ['Eric', 'Nancy', 56]
176  ]
177  console.log(numbers[0], numbers[5]) // 1 et undefined
178  console.log(names[1]) // Maria
179  console.log(persons[1]) // ['Eric', 'Nancy', 56]
```

### **III. Structure de contrôle**



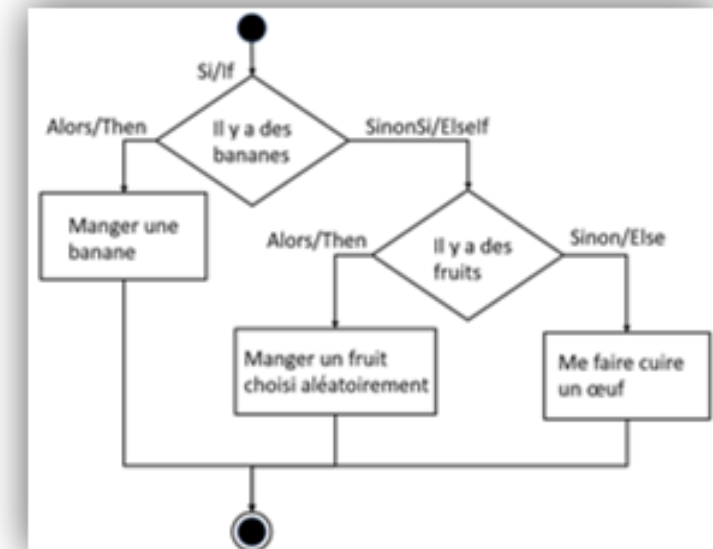
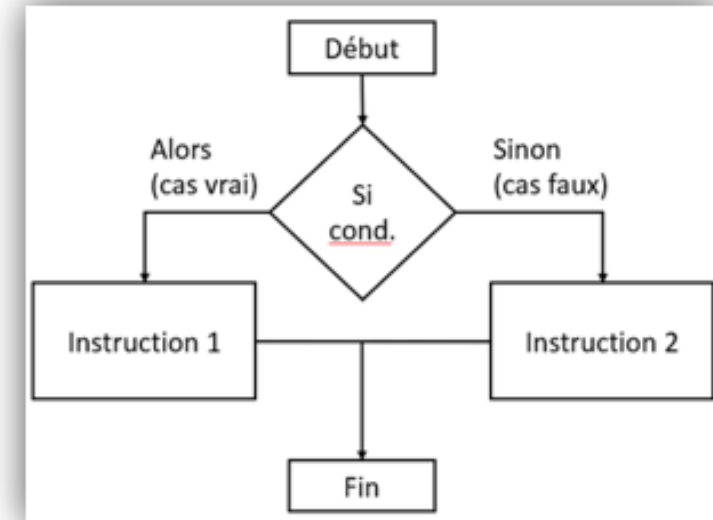




## Structure conditionnelle : *IF-ELSEIF-ELSE*

### Source image pro.du.code

- Exécution d'une **seule instruction** de la structure parmi les différentes « branches » possibles
- L'ordre des instructions à une importance
- La première instruction qui vérifie la condition sera exécutée (pas les autres)



## Exemple avec le langage JavaScript

```
69  const age = 100
70  if(age < 18) {
71      console.log('Mineur')
72  } else if(age >= 18 && age < 55) {
73      console.log('Majeur')
74  } else if(age >= 55 && age < 100) {
75      console.log('Senior')
76  } else {
77      console.log('Centenaire')
78  }
```

# Structure conditionnelle : *SWITCH*

- ❑ Identique à *if*, plus lisible dans les cas où il y a une vérification sur le contenu d'une chaîne de caractère
- ❑ Plus performant au niveau de l'exécution du programme

## Exemple avec le langage JavaScript

```
80  const season = 'winter'
81  switch(season) {
82      case 'winter':
83          console.log('Manteau')
84          break
85      case 'summer':
86          console.log('Tee-shirt')
87          break
88      case 'autumn':
89          console.log('Parapluie')
90          break
91      case 'spring':
92          console.log('Trench')
93          break
94      default:
95          console.log('Autre')
96          break
97  }
```



# Exercices 3 et 4

1-exercices/0-algos-poo/ex3.md

1-exercices/0-algos-poo/ex4.md



30 min



## Structures itératives (boucles)

Parcourir un tableau d'éléments en exécutant un bloc de code tant qu'une certaine condition est vérifiée

1. **FOR** : **compteur; condition; incrementation du compteur**) { *les instructions à effectuer* };
  - ❑ Boucle adaptée lorsqu'on **connaît le nombre d'itération à effectuer**
2. **WHILE** : **while (condition)** { *les instructions à effectuer et l'incrémentation selon les cas d'usage* };
  - ❑ Boucle adaptée pour parcourir des éléments lorsqu'on **ignore le nombre d'itération à effectuer**
3. **DO WHILE** : **do** { *les instructions à effectuer et l'incrémentation selon les cas d'usage* } **while (condition)** ;
  - ❑ Similaire à while sauf qu'il s'exécute au moins une fois

Le mot-clé **break** permet de sortir d'une boucle prématurément.



# BOUCLE POUR

1. Un compteur initialise le début de la boucle.
2. Une condition de sortie (évaluation de la condition).
3. Incrémentation du compteur (changement de la valeur du compteur).

❑ Exemple :

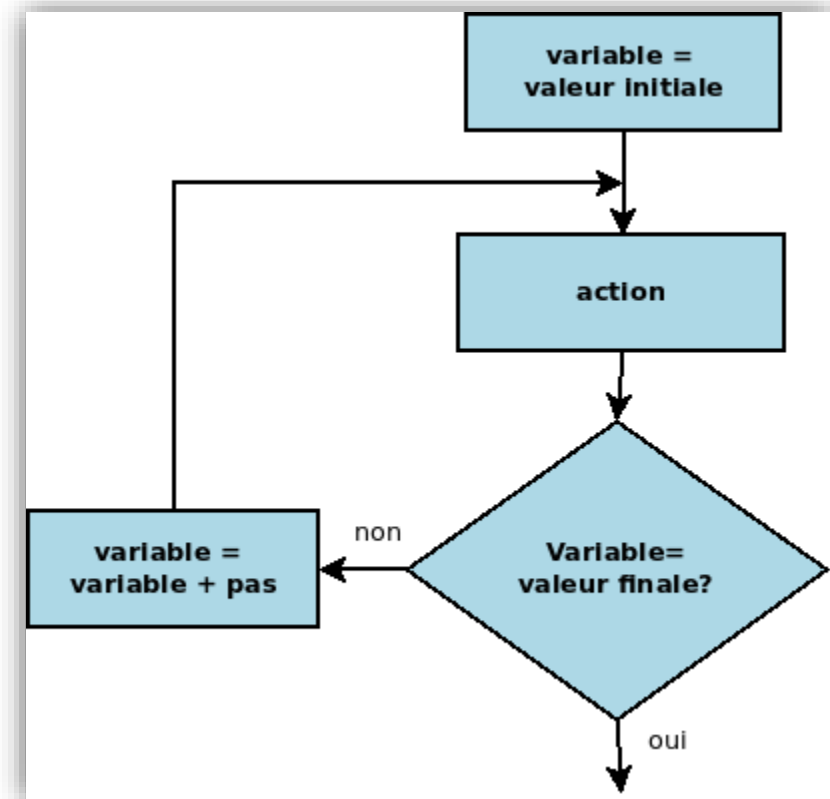
*DebutPour*

*Pour compteur de 1 à 10, pas de 1 :*

*Affiche 5 \* compteur*

*FinPour*

Source image Zeste STI 2D



# BOUCLE WHILE (TANT QUE)

- ❑ S'exécute tant qu'une condition est respectée.
- ❑ Utilise un **opérateur d'incrément** pour éviter une **boucle infinie**.
- ❑ Exemples d'utilisation :

*Tant que la liste n'est pas totalement parcouru :*

*Affiche l'élément de la liste*

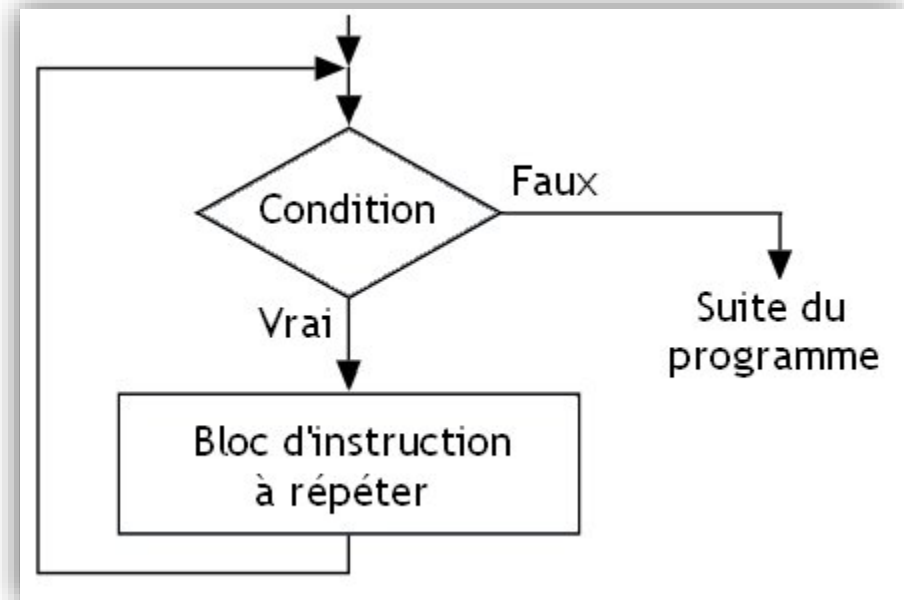
*Fin de Tant que*

*Tant que ce chiffre ne dépasse pas 16 :*

*Réalise un calcul*

*Fin de Tant que*

Source image Zeste de savoir



# BOUCLE DO WHILE (FAIRE TANT QUE)

- ❑ La boucle **Faire...tant que** aussi appelée

## Répéter...tant que

- ❑ Similaire à la boucle Tant Que.
- ❑ **La condition est évalué à la fin.**
- ❑ **S'exécute au moins une fois.**

- ❑ Exemple :

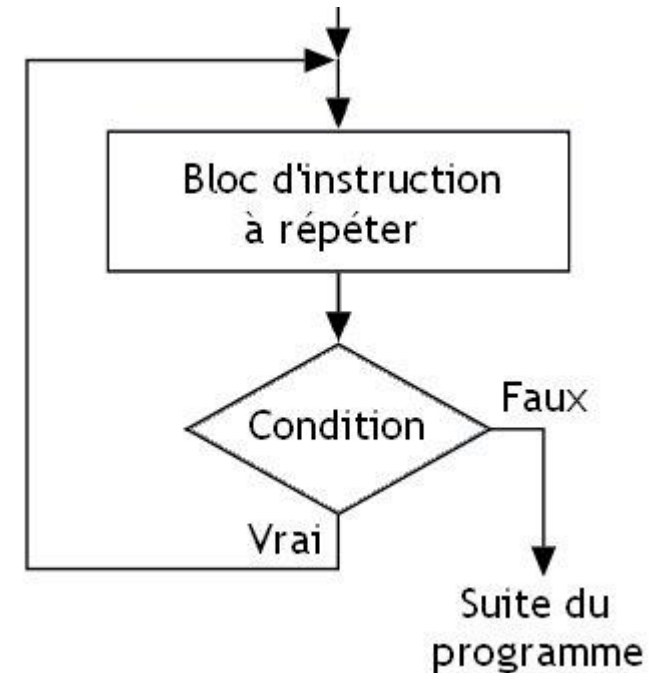
*Faire :*

*Ecrire 'Entrez un nombre  $\geq$  à 10'*

*Lire nombre*

*Tant que (nombre  $<$  10)*

## Source image Zeste de savoir





# Exemple avec le langage JavaScript

```
100  const cities = ['Paris', 'Marseille', 'Lille', 'Lyon', 'Nantes']
101
102  for(let i = 0; i < cities.length; i++) {
103      console.log('Ville avec la boucle for : ', cities[i])
104  }
105
106  let i = 0
107  while(cities.length > i) {
108      console.log('Ville avec la boucle while : ', cities[i])
109      i++// attention en cas d'oubli de l'incrémentatation => boucle infini
110  }
111
112  do {
113      console
114      .log('Je m\'exécute au moins une fois même si la condit. est fausse')
115  } while(false)
```



# Exercices 5 et 6

1-exercices/0-algos-poo/ex5.md

1-exercices/0-algos-poo/ex6.md



30 min

## IV. Fonction





# Fonction

- ❑ *programme dans le programme*
- ❑ On utilise des fonctions pour **regrouper des instructions et les appeler sur demande** :  
chaque fois qu'on a besoin de ces instructions, il suffira d'appeler la fonction au lieu de répéter toutes les instructions.
- ❑ Pour accomplir ce rôle, le **cycle de vie d'une fonction comprend 2 phases** :
  1. Une phase unique dans laquelle la fonction est **déclarée**  
On définit à ce stade toutes les instructions qui doivent être groupées pour obtenir le résultat souhaité.
  2. Une phase qui peut être répétée une ou plusieurs fois dans laquelle la fonction est appelée puis **exécutée**.



## EXEMPLES FONCTIONS

---

Fonction somme (nb1: entier, nb2: entier) : entier

Variable resultat : entier

Début

    resultat <- nb1 + nb2

    retourne resultat

Fin

***somme(10, 20) // retourne le résultat 30 que l'on peut stocker dans une variable du programme principal***  
***somme(152,265)***

---

Fonction bonjour () : vide

Début

    écrire("Bonjour ")

Fin

***bonjour() // affiche Bonjour sur le périphérique de sortie***  
***bonjour()***

---

Fonction presentation(nom: chaine, prenom: chaine, age: entier) : vide

Début

    écrire("Je m'appelle ", prenom, " ", nom, ", j'ai ", age, " ans.")

Fin

***presentation("Tshimini", "Glodie", 32) // affiche Je m'appelle Glodie Tshimini, j'ai 32 ans.***

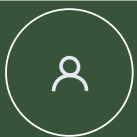
# Exemple avec le langage JavaScript

```
136 // Déclaration sans paramètre et retourne une valeur
137 function helloWorld() {
138     return 'Hello World'
139 }
140 // Déclaration avec un paramètre et ne retourne pas une valeur
141 function getStatus(age) {
142     if(age >= 18) console.log('Majeur')
143     else console.log('Mineur')
144 }
145
146 const hello = helloWorld()
147 console.log(hello) // Hello World
148 getStatus(20) // Majeur
```



# Exercice 7

1-exercices/0-algos-poo/ex7.md



30 min

## **V. Tableaux multidimensionnels**





# TABLEAUX MULTIDIMENSIONNELS

## Source image eprojet

### Tableau Multidimensionnel

```
Array
(
    [0] => Array
        (
            [prenom] => Julien
            [nom] => Cottet
        )
    [1] => Array
        (
            [prenom] => Nicolas
            [nom] => Lafaye
        )
)
```

<u>indice</u>	<u>valeur</u>
0	Array
1	Array

<u>indice</u>	<u>valeur</u>
prenom	Julien
nom	Cottet

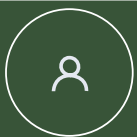
<u>indice</u>	<u>valeur</u>
prenom	Nicolas
nom	Lafaye

- ❑ Un tableau dont les éléments sont d'autres tableaux.
- ❑ Il n'y a **pas de limite au niveau du nombre des dimensions**, cependant **au-delà de 2 dimensions**, c'est **plus en plus difficile** pour les humains de **visualiser les informations**.
- ❑ On parle généralement de tableau à N dimensions, avec N le nombre des dimensions.
- ❑ Il faut des **boucles imbriquées** pour parcourir un tableau multidimensionnel



# Exercise 8

1-exercices/0-algos-poo/ex8.md



45 min



# Jeux blocky

1-exercices/0-algos/jeux/README.md



15 min

# **VI. PROGRAMMATION ORIENTÉE OBJET (POO)**





# 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.

# Objet

## 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
    - Etc.
  - Peut réaliser un certain nombre **d'actions** :
    - Marcher
    - Courir
    - Parler
    - Etc.



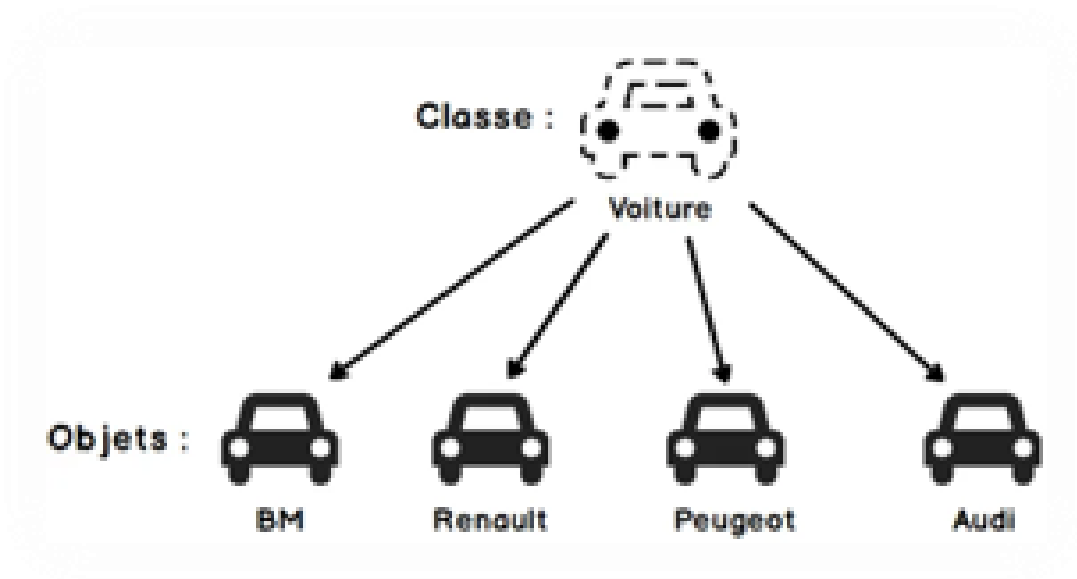
## Classe

- 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



# Classe et objet

Source image 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



# Exercise 9

1-exercices/0-algos-poo/ex9.md

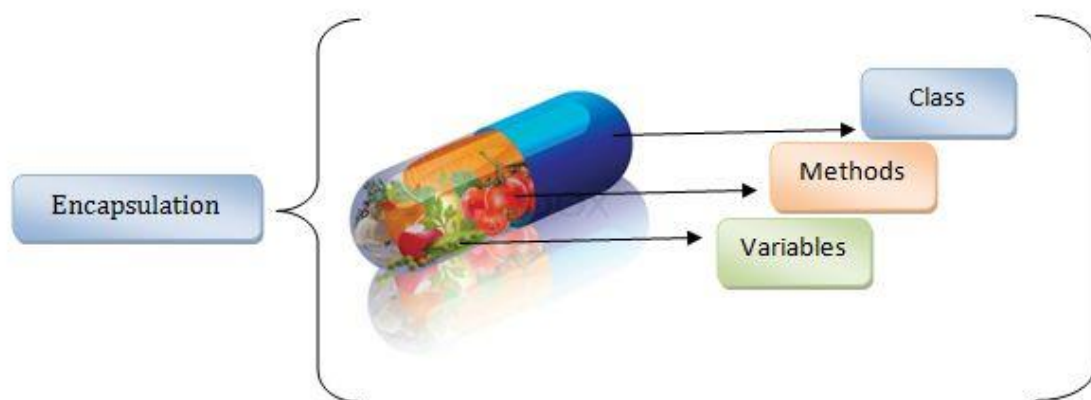


10 min



# Encapsulation

Source image logicmojo

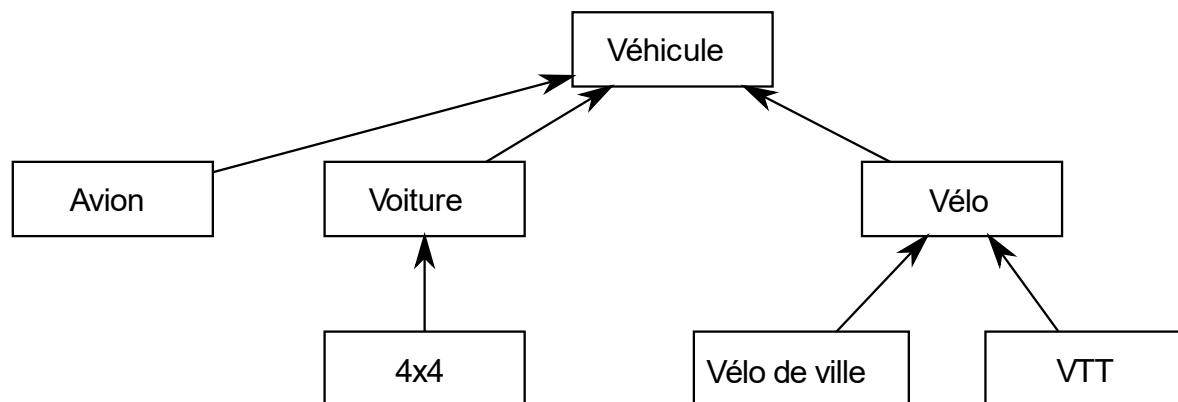


- On parle d'encapsulation, lorsqu'un objet est lui-même en capacité de connaître ses propres attributs et comportements.
- Parfois, on aura besoin de **cacher** une partie des attributs et comportements d'un objet.
- Plusieurs niveaux d'encapsulation :
  - **Privé** : attributs et/ou comportements accessibles 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



# Héritage

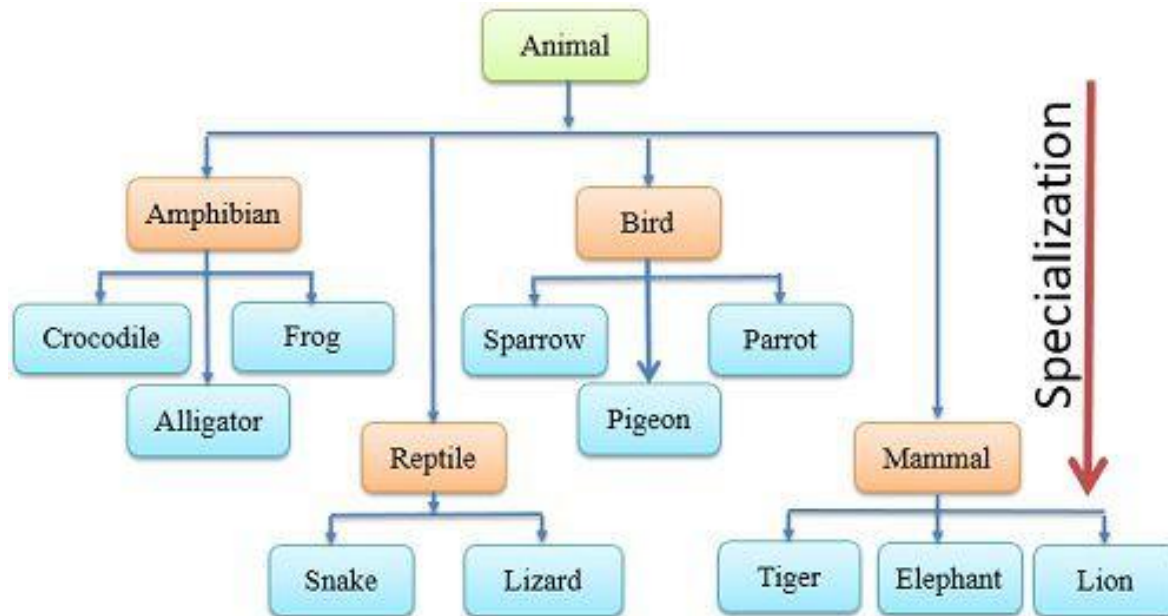
## Source de l'image perso-esiee



- Une voiture est *classe* donc un *modèle*, lui-même crée 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

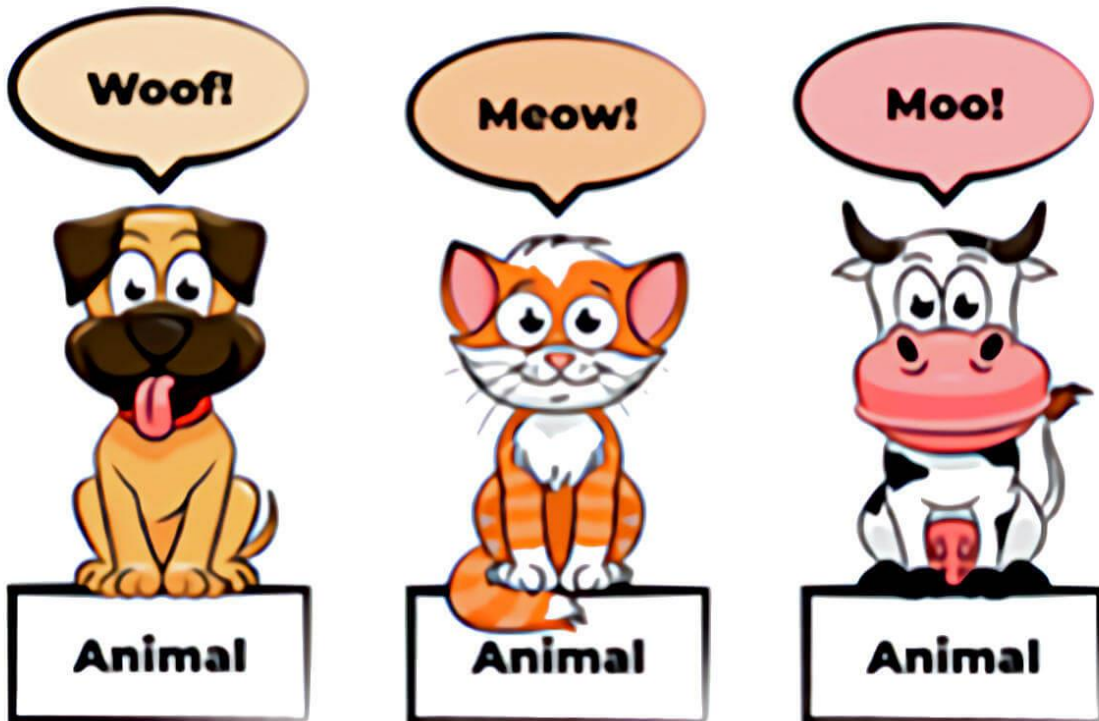
Source image letsstudytogether



- 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.

# Composition

Source image pari-et-gagne



- ❑ 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



# Exercise 10

1-exercices/0-algos/ex10.md



10 min





ib  
cegos

UML



# PLAN du cours sur UML

- I. UML et généralités
- II. Diagramme d'activité
- III. Diagramme de cas d'utilisation
- IV. Diagramme de classes
- V. Diagramme d'objets
- VI. Diagrammes de composant et de déploiement

# I. UML ET GÉNÉRALITÉS





# Mind Mapping UML

# Langage UML

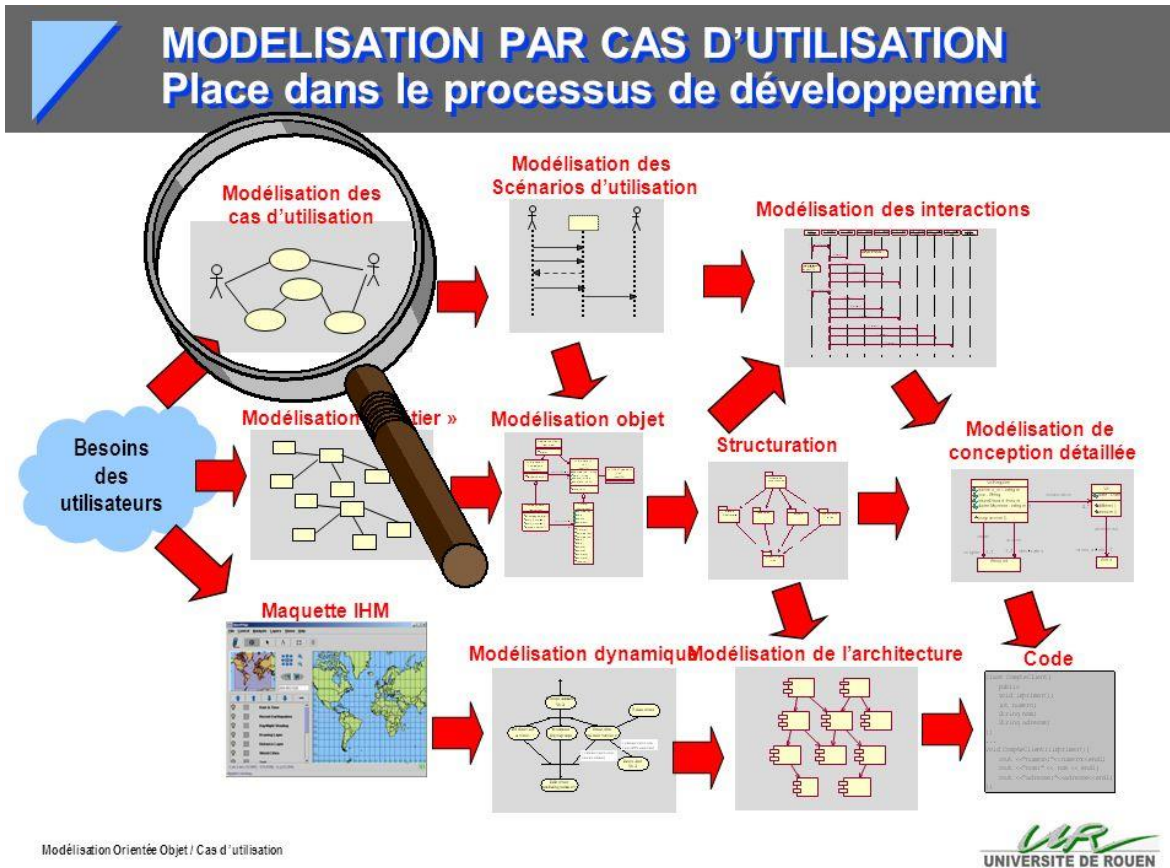
Source image wikipédia



- ***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

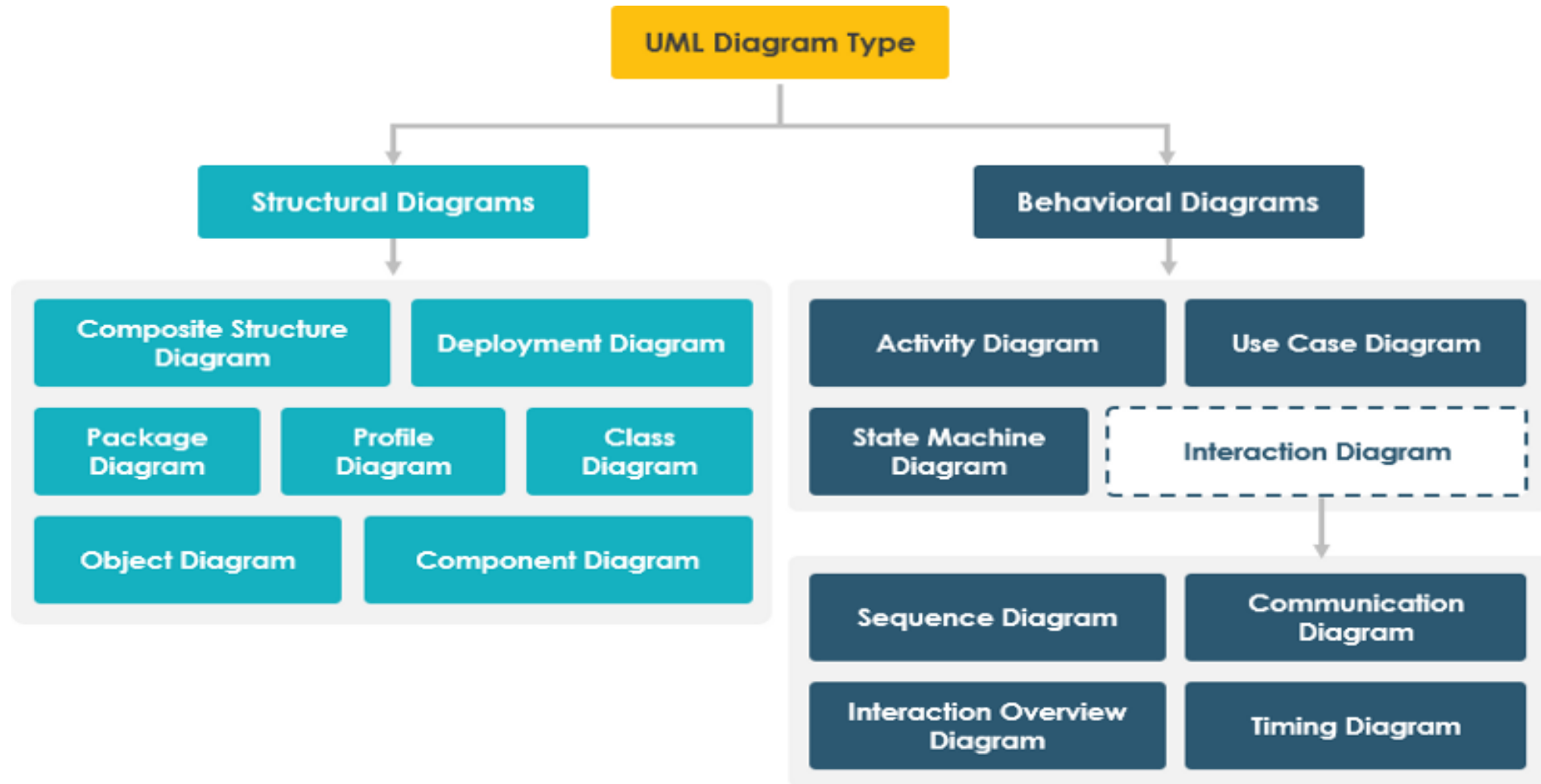
# Langage UML

## 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)





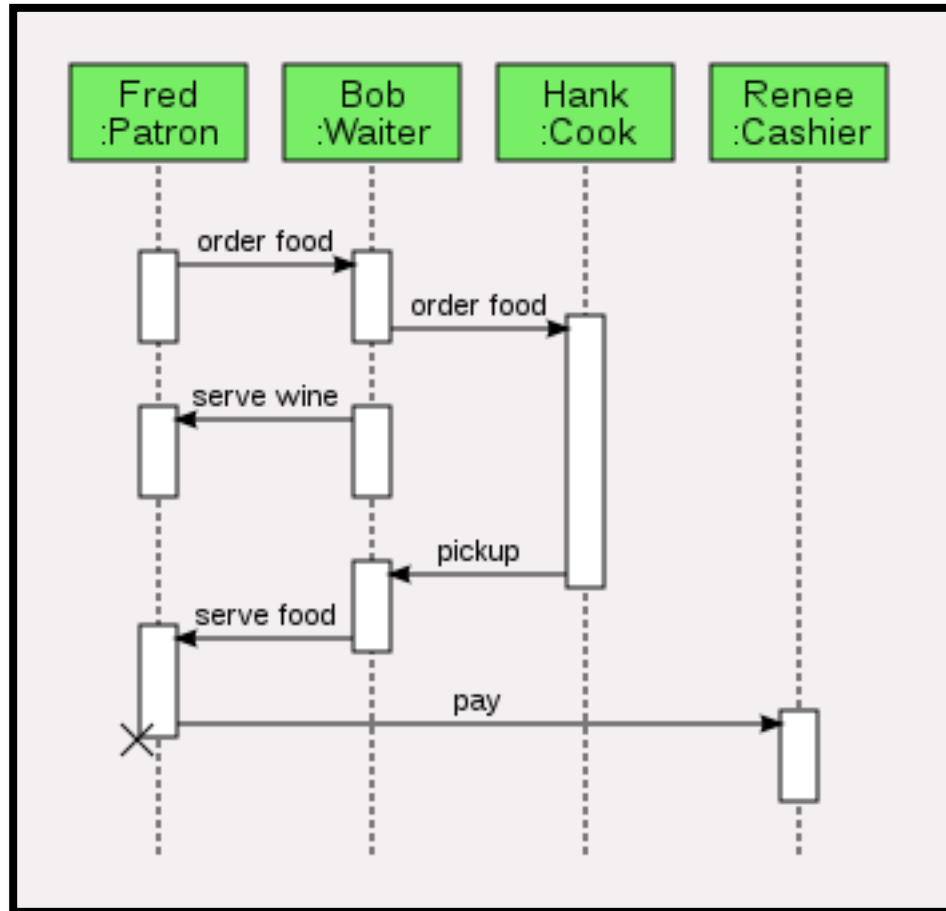
# Les diagrammes UML

- ❑ **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

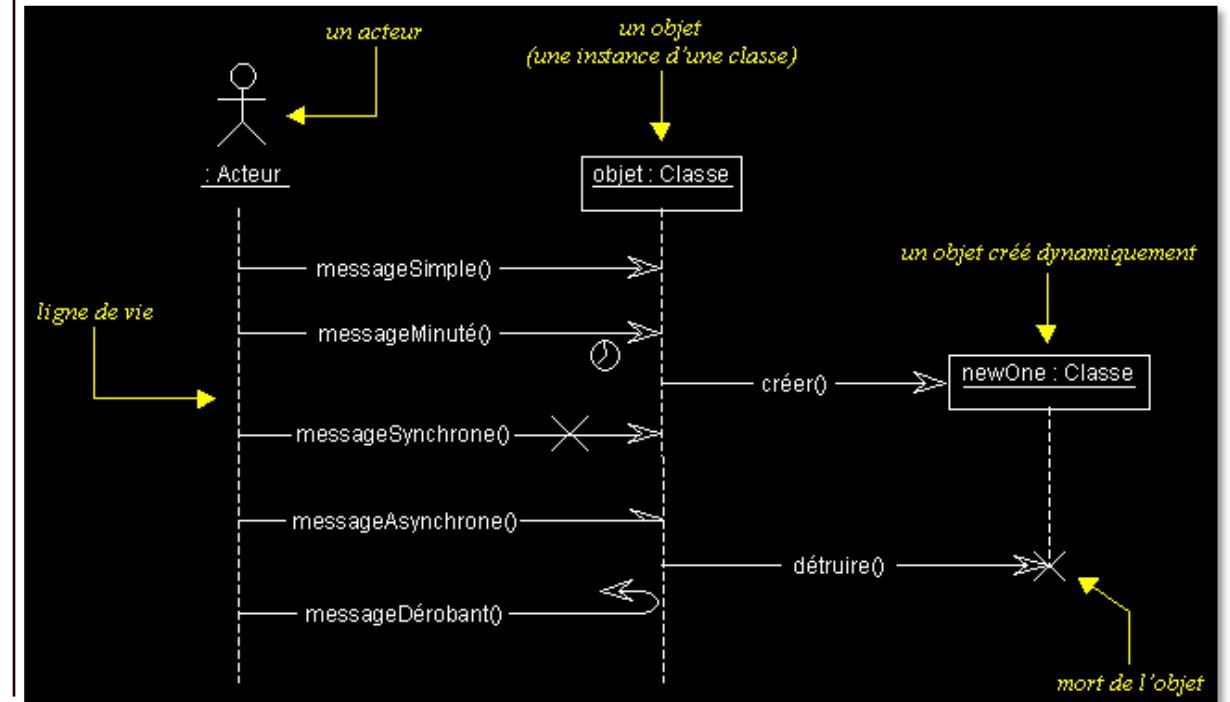


# Diagramme de séquence

Source image wikipédia

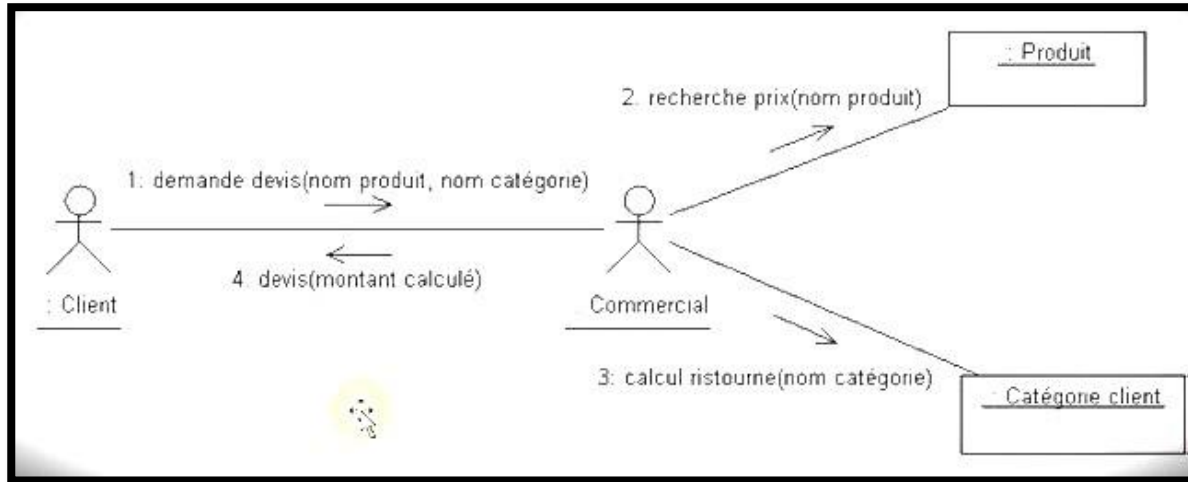


Source image UML.free.fr

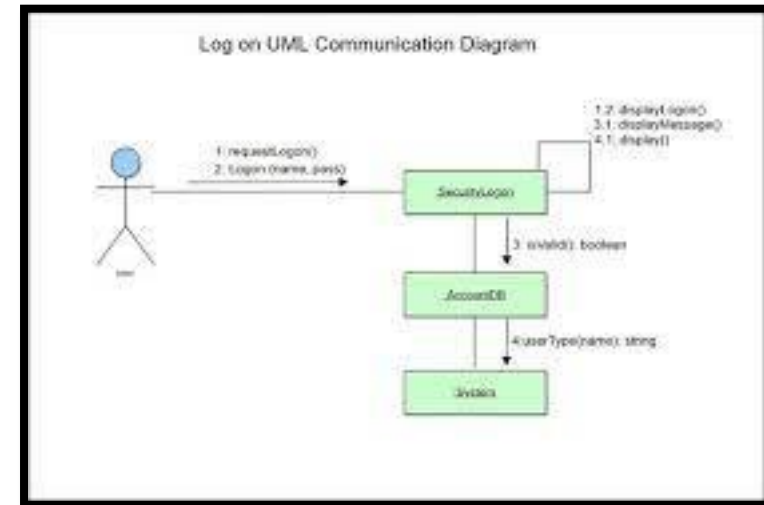


# Diagramme de communication

Source image celamrani

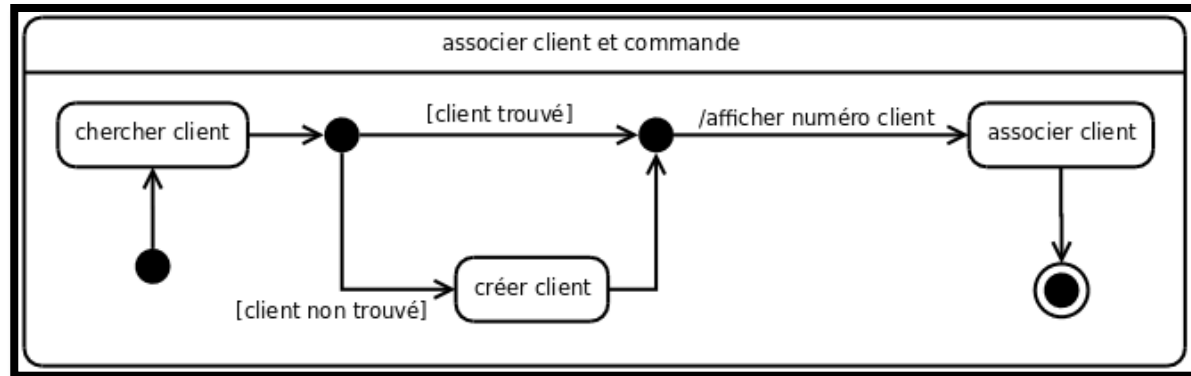


Source image GitMind

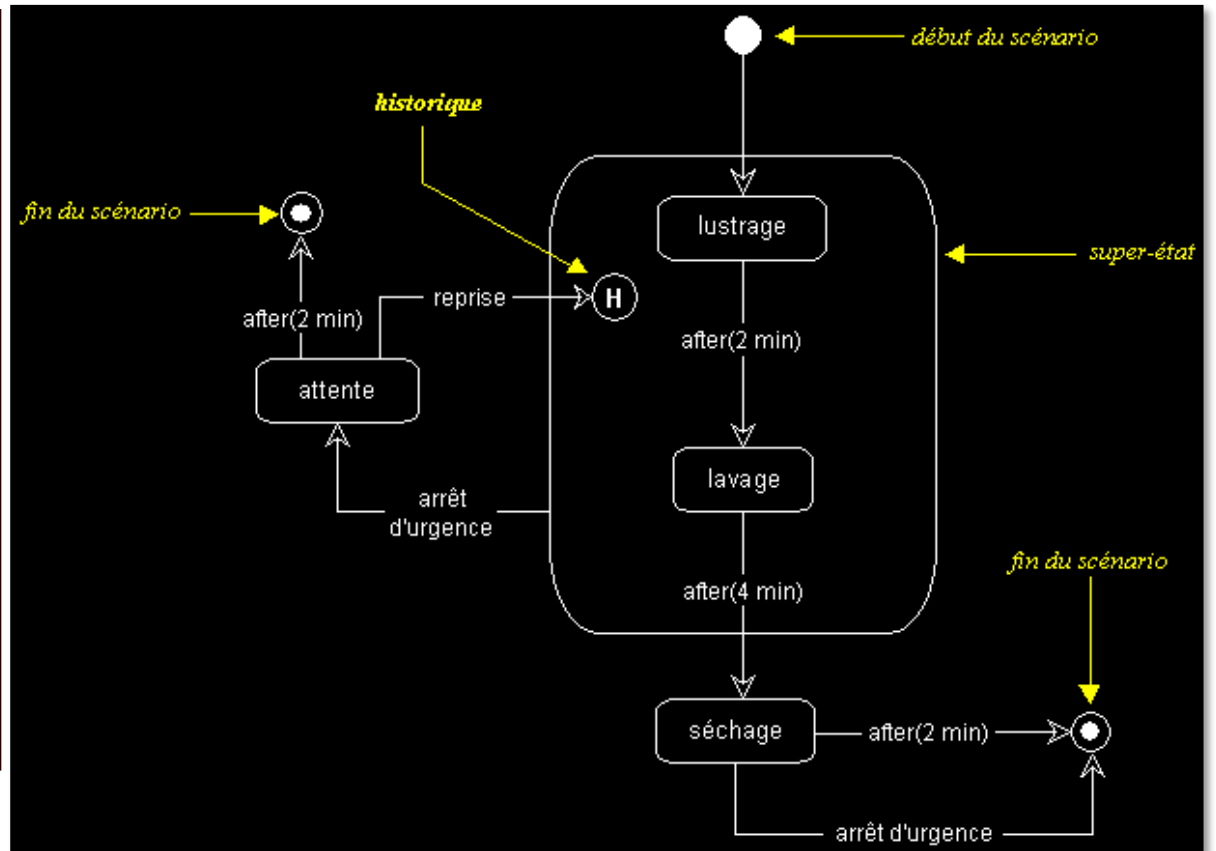


# Diagramme d'état-transition

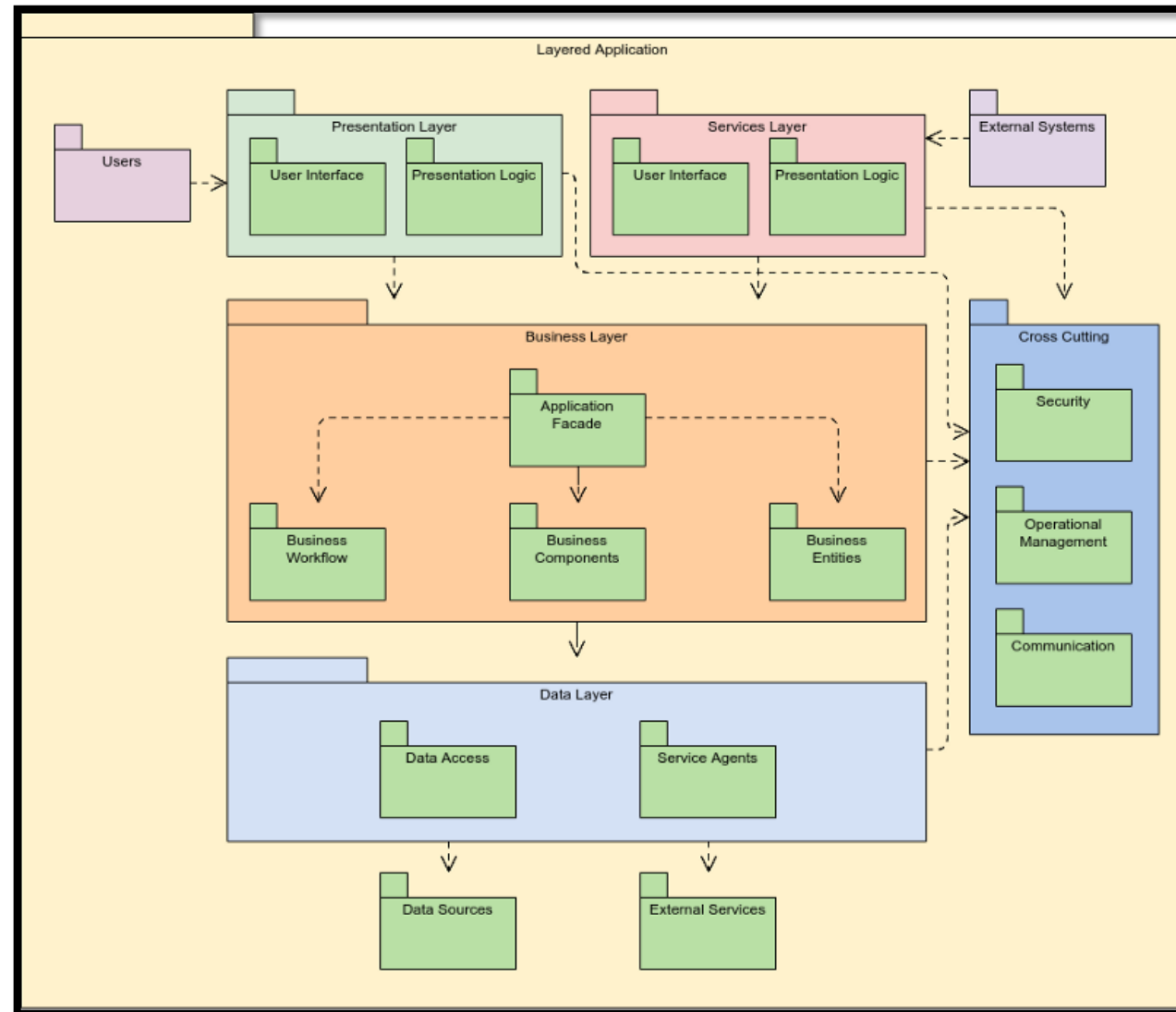
Source image laurent audibert



Source image UML.free.fr



# Diagramme de package (source image cybermedian)





# Exercise 1

1-exercices/1-uml/ex1.md



20 min

## II. DIAGRAMME D'ACTIVITÉ



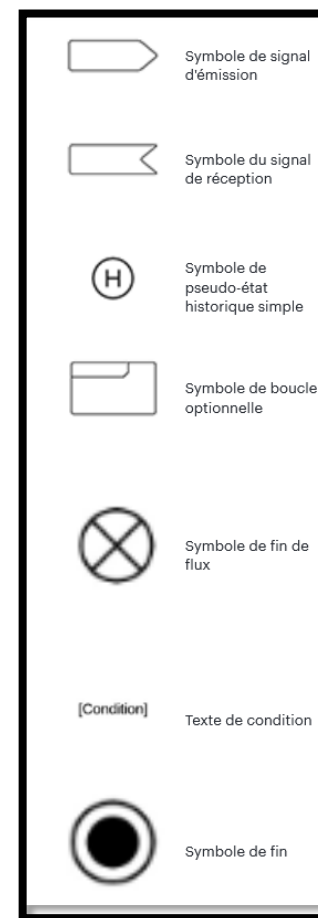
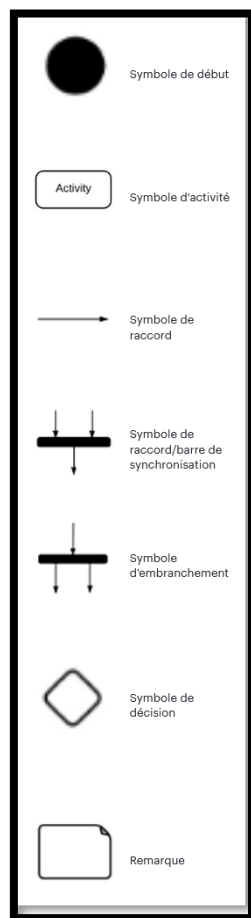


# Diagramme d'activité

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

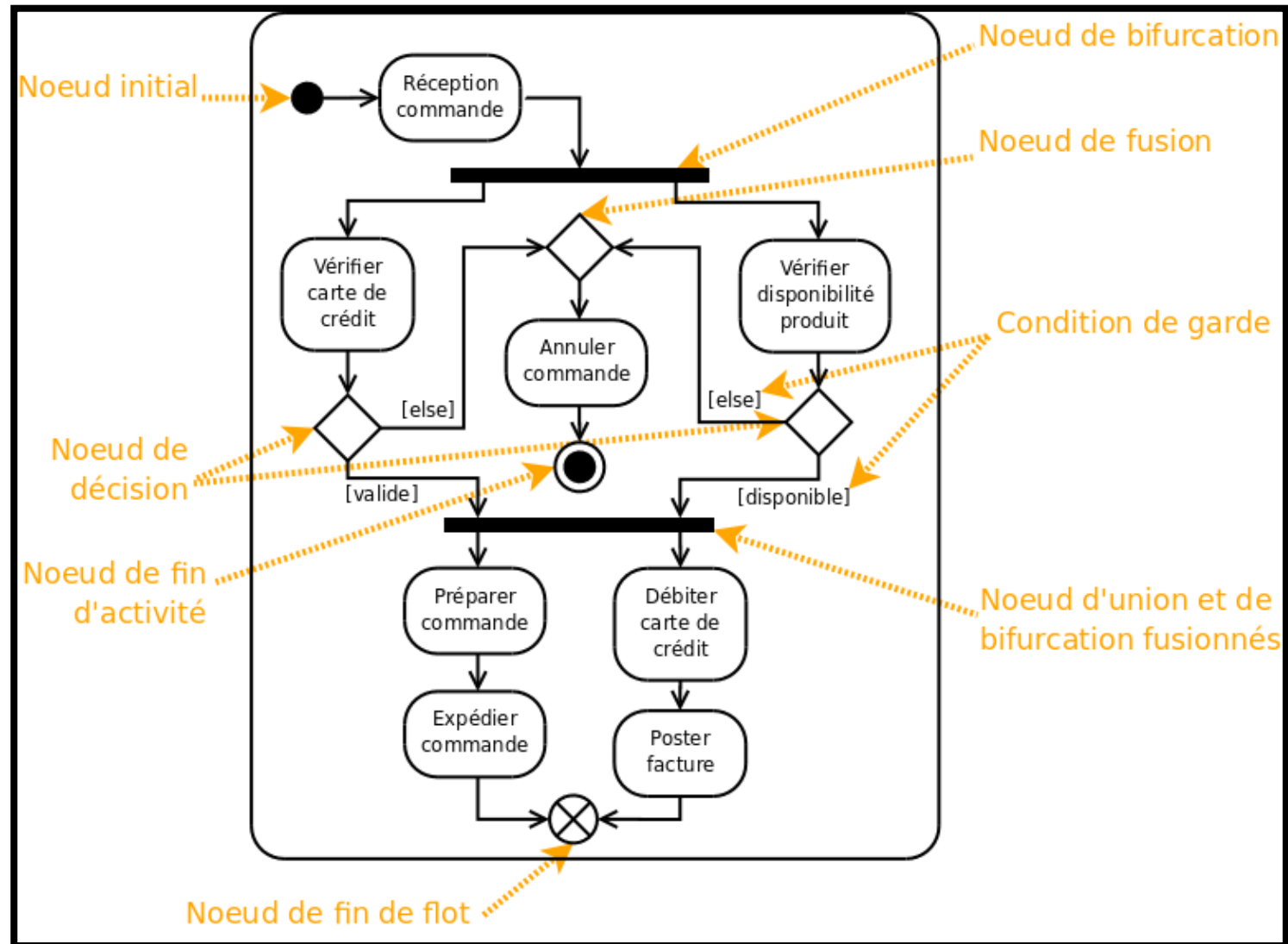


# Formalisme (source image lucid-chart)





## Exemple (source Laurent-Audibert)





# Exercise 2

1-exercices/1-uml/ex2.md



30 min

### **III. DIAGRAMME DE CAS D'UTILISATION**

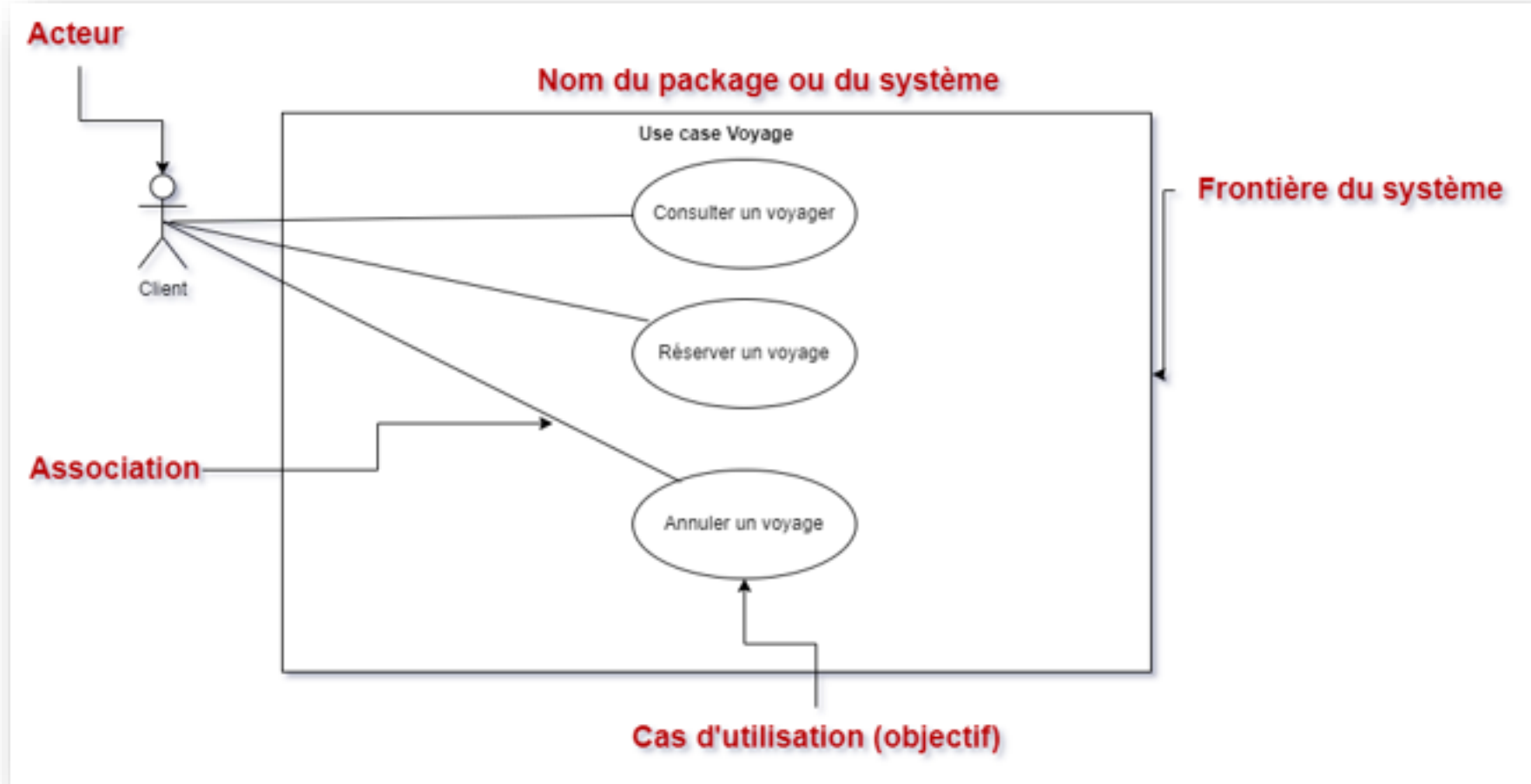




# Principes

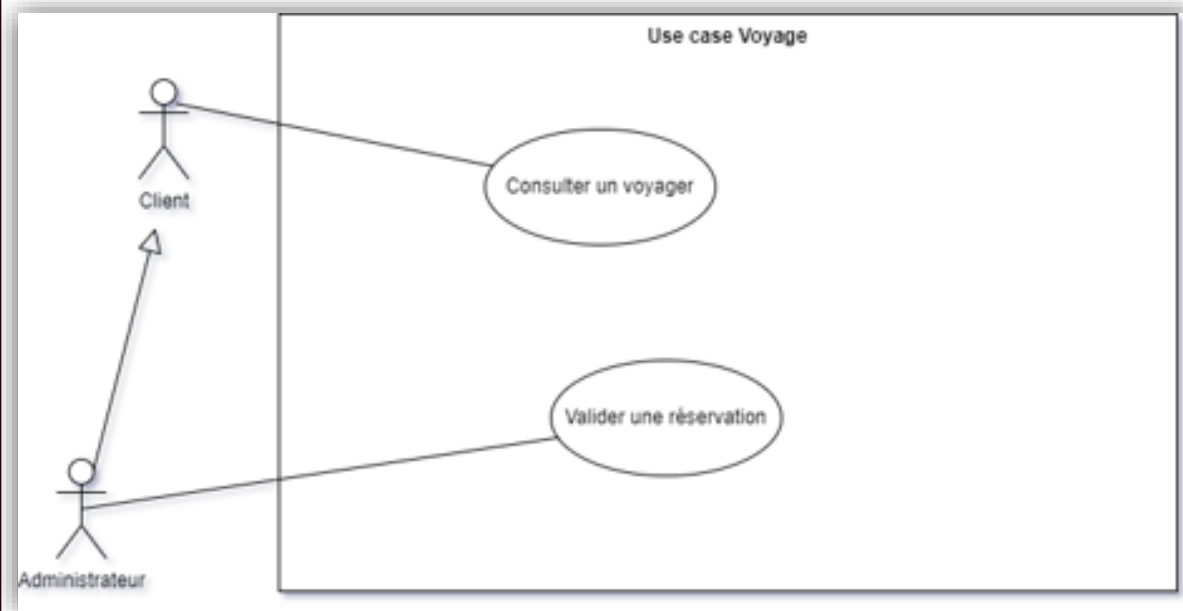
- ❑ Schématiser l'expression des besoins d'un point utilisateur. Autrement dit c'est la représentation du système vu par l'utilisateur.
- ❑ Répondre aux questions **Qui** et **Quoi** ?
- ❑ Objectifs
  - ❑ Délimiter le périmètre fonctionnel.
  - ❑ Servir pour réaliser des tests fonctionnels.
  - ❑ Impliquer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

# Syntaxe



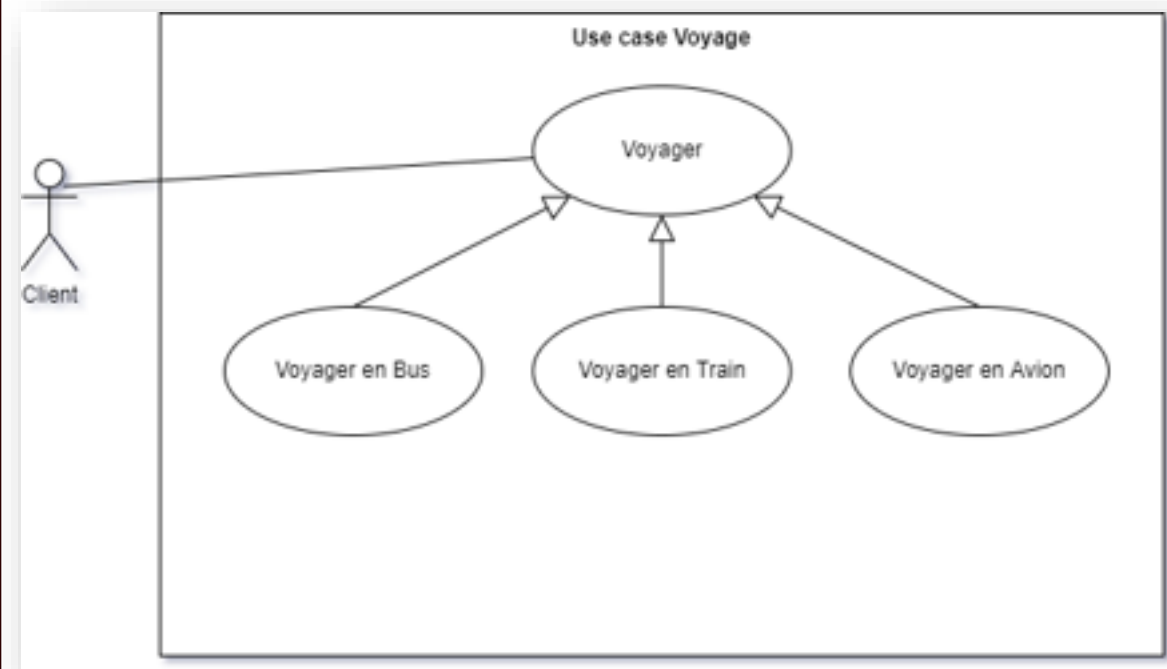
## Héritage entre acteurs

- 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.



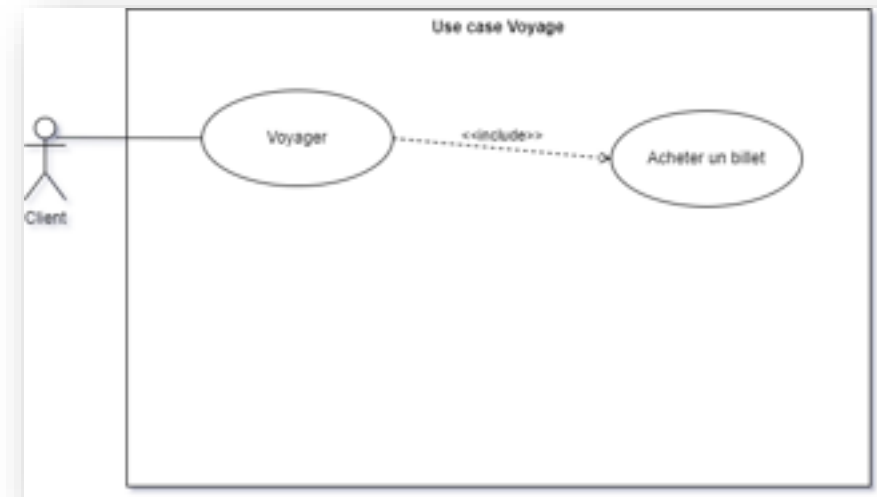
## Héritage entre cas d'utilisation

- 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

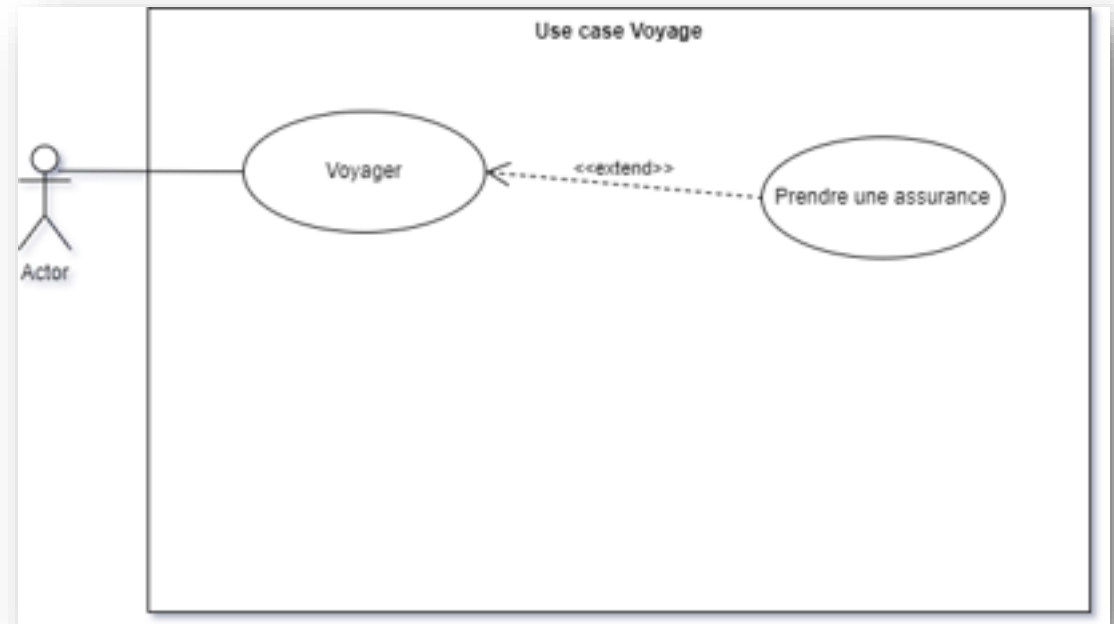
- La relation d'inclusion entre deux cas d'utilisation signifie que la réalisation d'un cas d'utilisation implique obligatoirement la réalisation d'un autre cas d'utilisation
- Pour « voyager », il faut obligatoirement « acheter un billet » .
- Généralement, les cas d'inclusion ne répondent pas directement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





# Exercise 3

1-exercices/1-uml/ex3.md



30 min

# Cas d'utilisation détaillé

## Source image scribd

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

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

<b>Titre</b>	<b>Ajouter un domaine</b>
<b>Acteurs</b>	Élève
<b>Description</b>	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.
<b>Description des scénarios</b>	<b>Scénario nominal :</b> <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> <b>Scénario alternatif :</b> <b>A1 : Email ou mot de passe non valide :</b> <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>
<b>Pré condition(s)</b>	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 blue. A green checkmark is placed to the left of 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**



# Exercise 4

1-exercices/1-uml/ex4.md



15 min

## **IV. DIAGRAMME DE CLASSES**

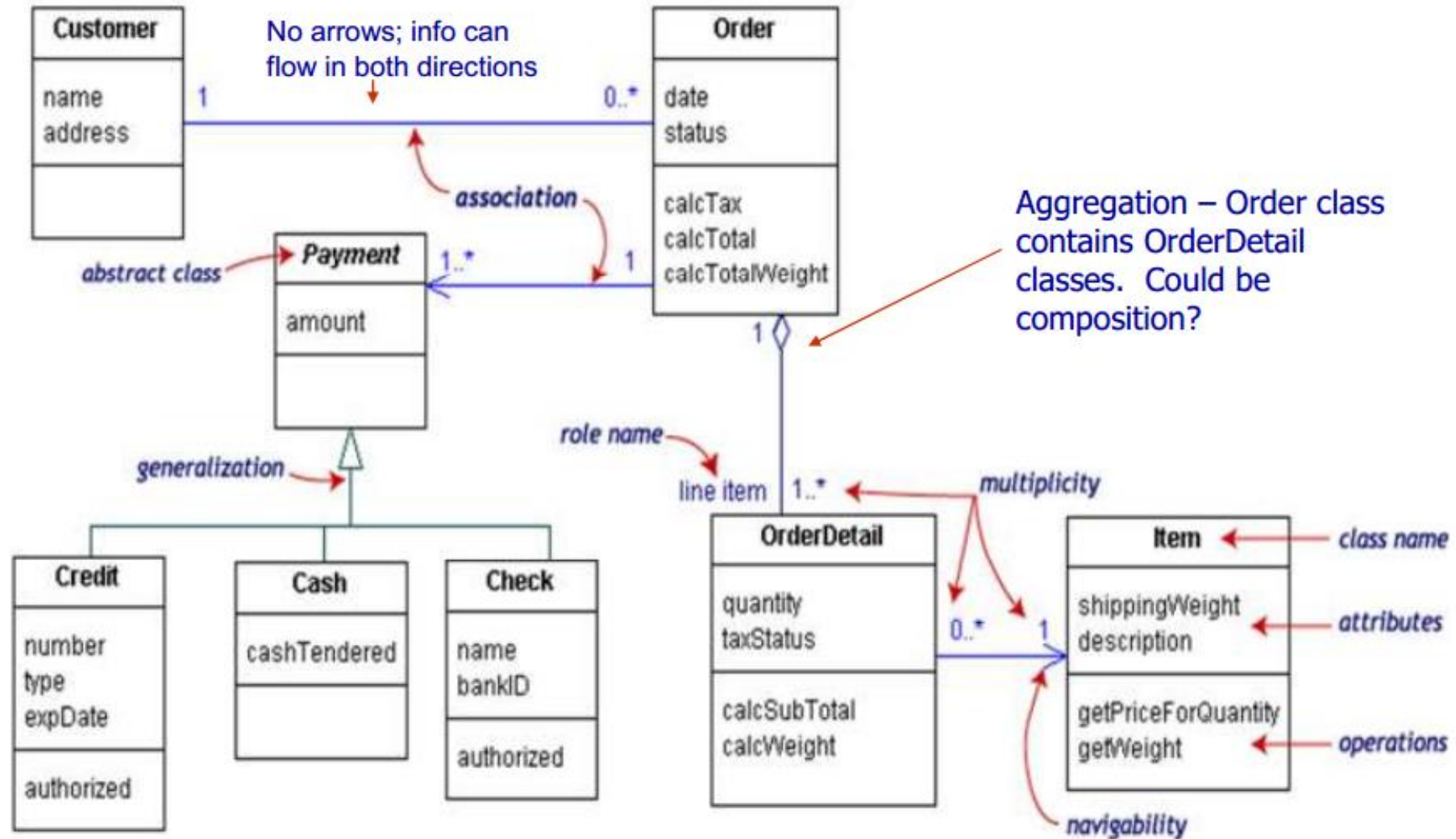




# Principes

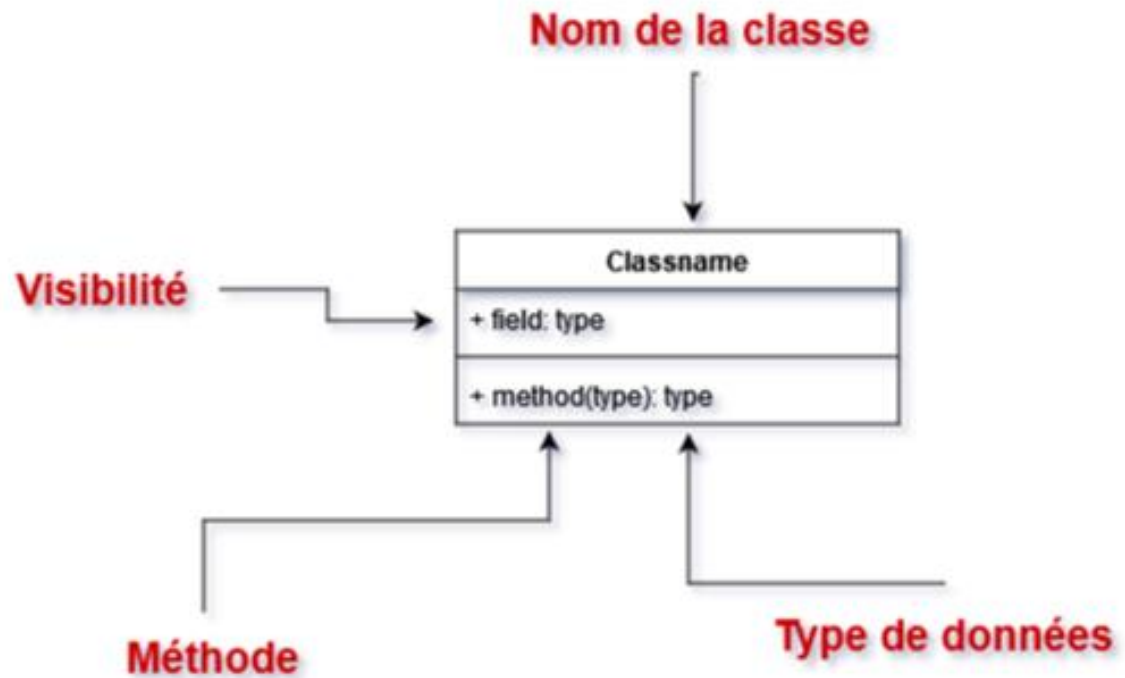
- ❑ Schématiser **la structure interne d'un système** qui sera implémenté dans un langage de programmation orienté objet
  - ❑ Classes
  - ❑ Attributs
  - ❑ Opérations
  - ❑ Relations
- ❑ Autrement dit, représente les **données** et les **traitements** du système.
- ❑ Modéliser des bases de données relationnelles 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

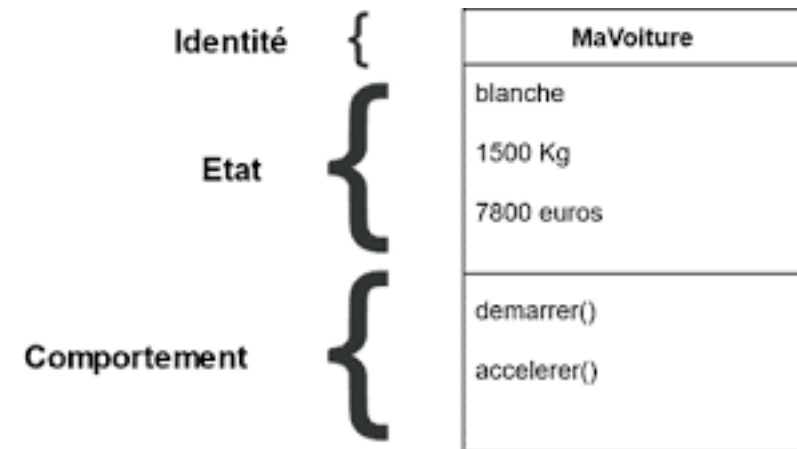




# Zoom sur une classe



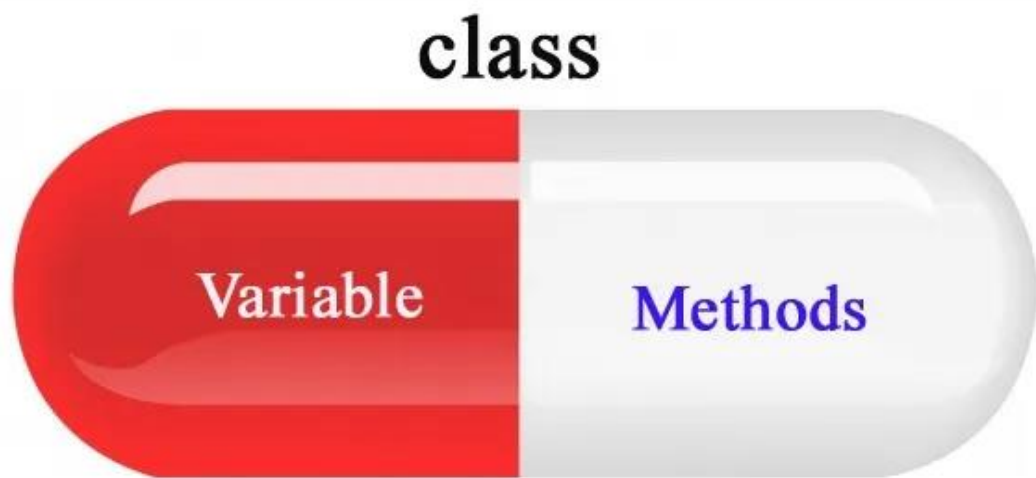
## Source image data-transitionnumerique





# Encapsulation

Source image code4coding

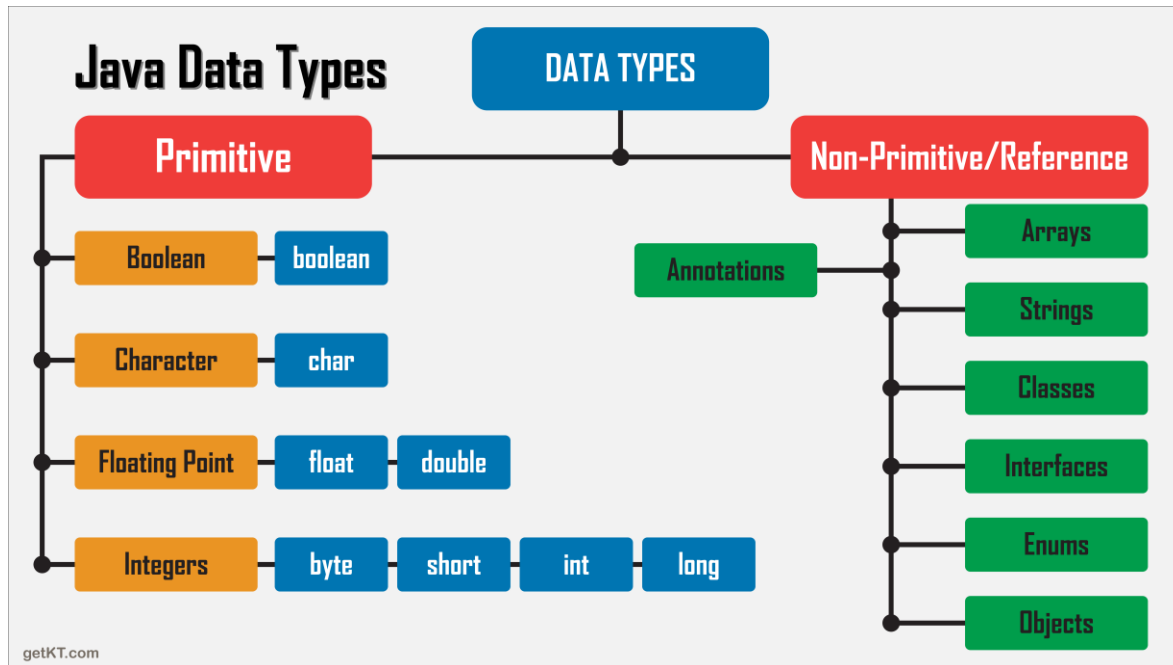


Encapsulation

- **Public (+)** : accessible par tous les autres objets.
- **Privé (-)** : accessible uniquement au sein de la classe.
- **Protégé (#)** : accessible uniquement au sein des classes filles.

# Types de données

Source image getkt



❑ On utilise les types **primitifs** de l'algorithmie et éventuellement les énumérations (liste fermée des données)

❑ Int

❑ 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.




# Associations

[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**



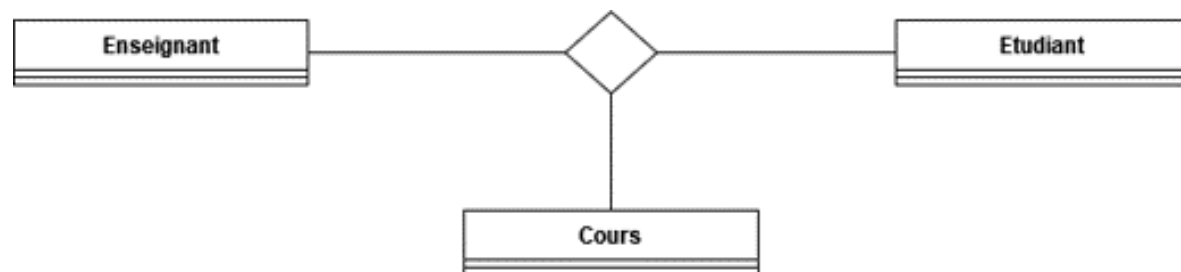
## Association binaire



- ❑ 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**.



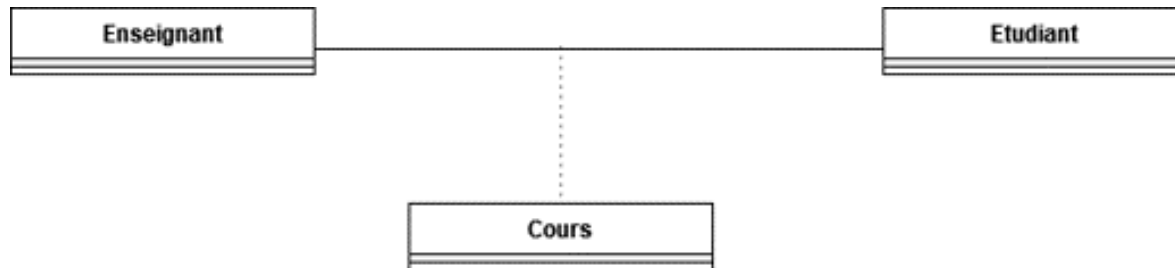
## Association N-aire



- ❑ 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.

```

classDiagram
    class Person {
        - name: string
    }
    class Enfant
    class Parent
    Person "*" -- "2" Enfant
    Person -- "2" Parent

```

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



# 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.

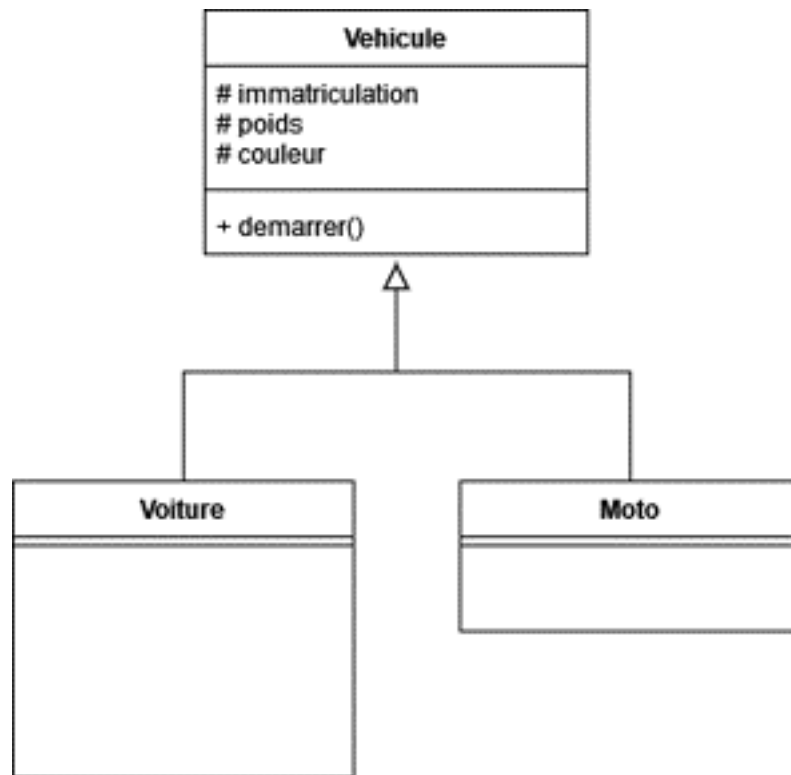


## Agrégation forte



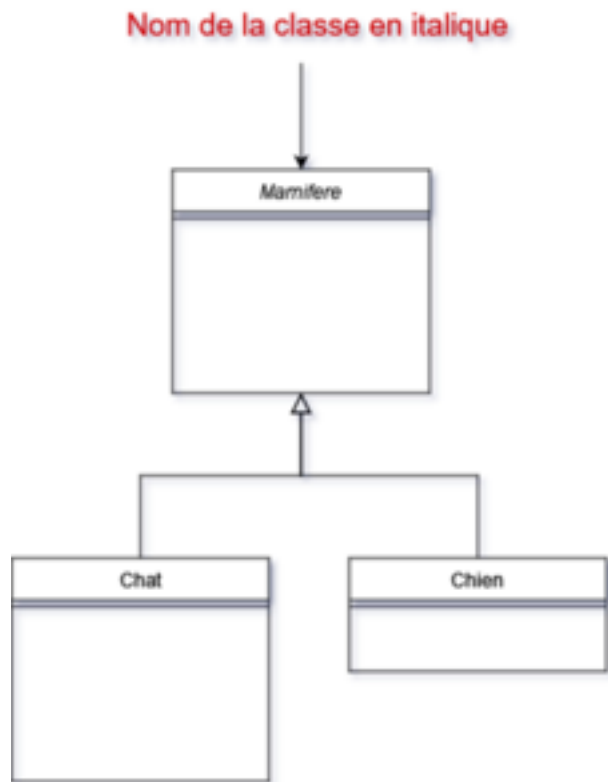
- ❑ **Composition ou agrégation forte.**
- ❑ Suppression d'une instance de la classe qui domine entraîne la suppression des instances liées par cette relation.

# Héritage entre les classes

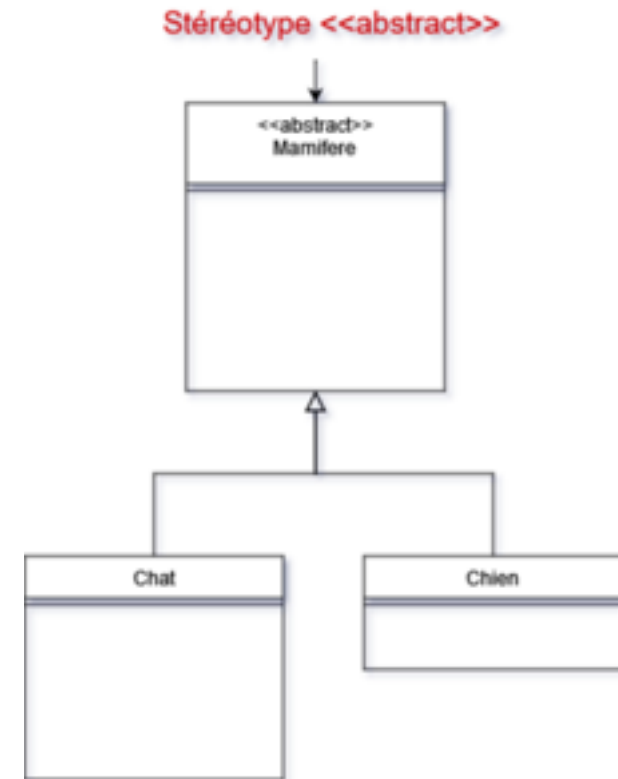


- ❑ Une **classe mère** contient des **caractéristiques communes** pour ses classes filles.
- ❑ **Généralisation** des attributs et des méthodes au sein d'une super classe.
- ❑ **Spécialisation** dans les sous-classes.

# Classe abstraite

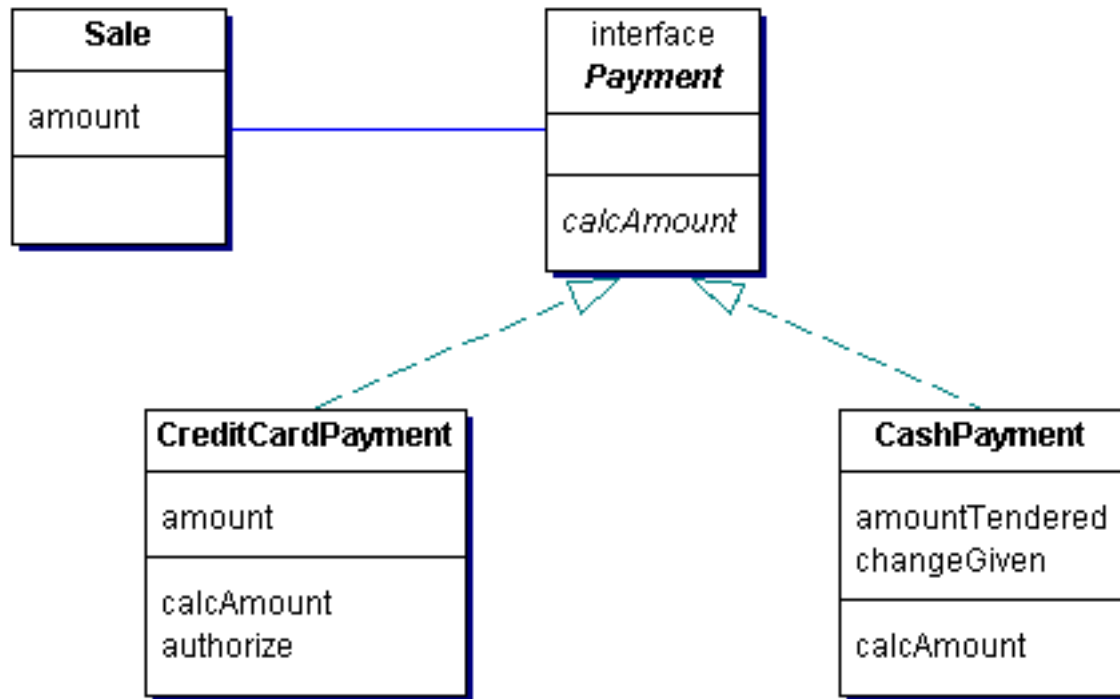


- ❑ Ne peut pas avoir d'instance
- ❑ Sert de base (mère) pour les classes dérivées (filles)



# Interface

## 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).



# Exercices 5 et 6

1-exercices/1-uml/ex5.md

1-exercices/1-uml/ex6.md



45 min

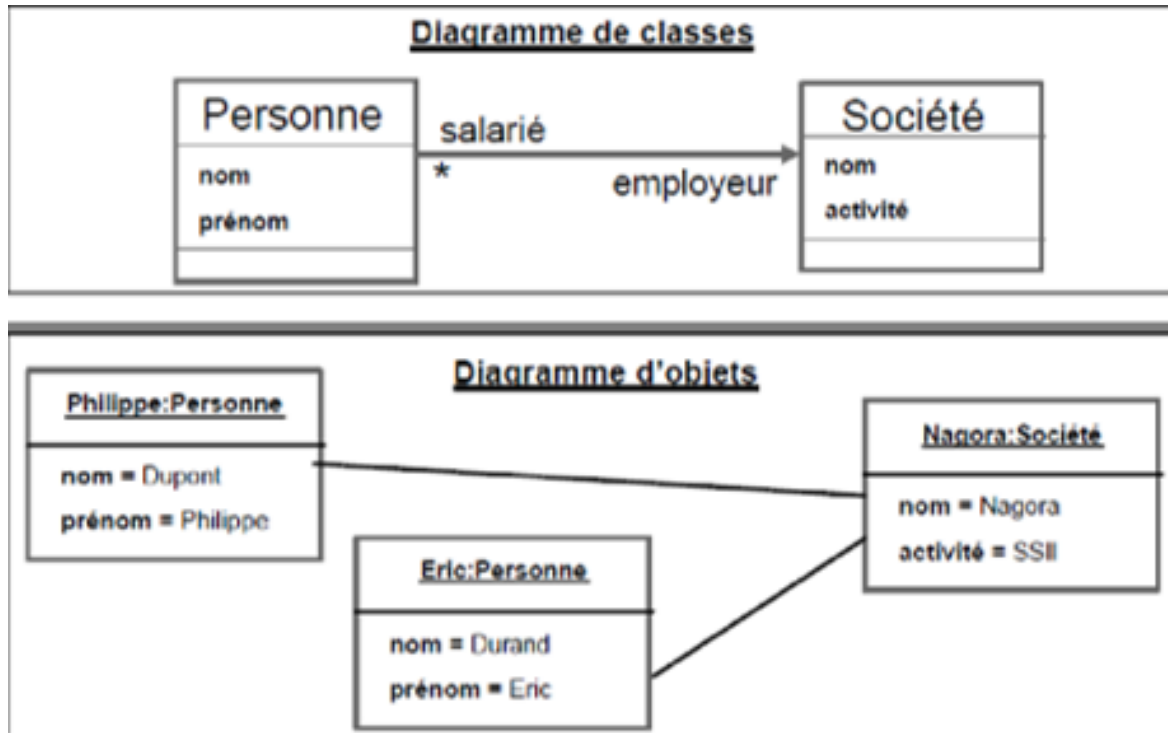
## **V. DIAGRAMME D'OBJETS**





# 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.



# Exercise 7

1-exercices/1-uml/ex7.md



30 min

## **VI. DIAGRAMMES DE COMPOSANT ET DE DEPLOIEMENT**





# Principes

## 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

# Exemple de diagramme de composant

(source image Bouchra Bouihi)

## Diagramme de Composant

### Exemple : Transfert des données par internet

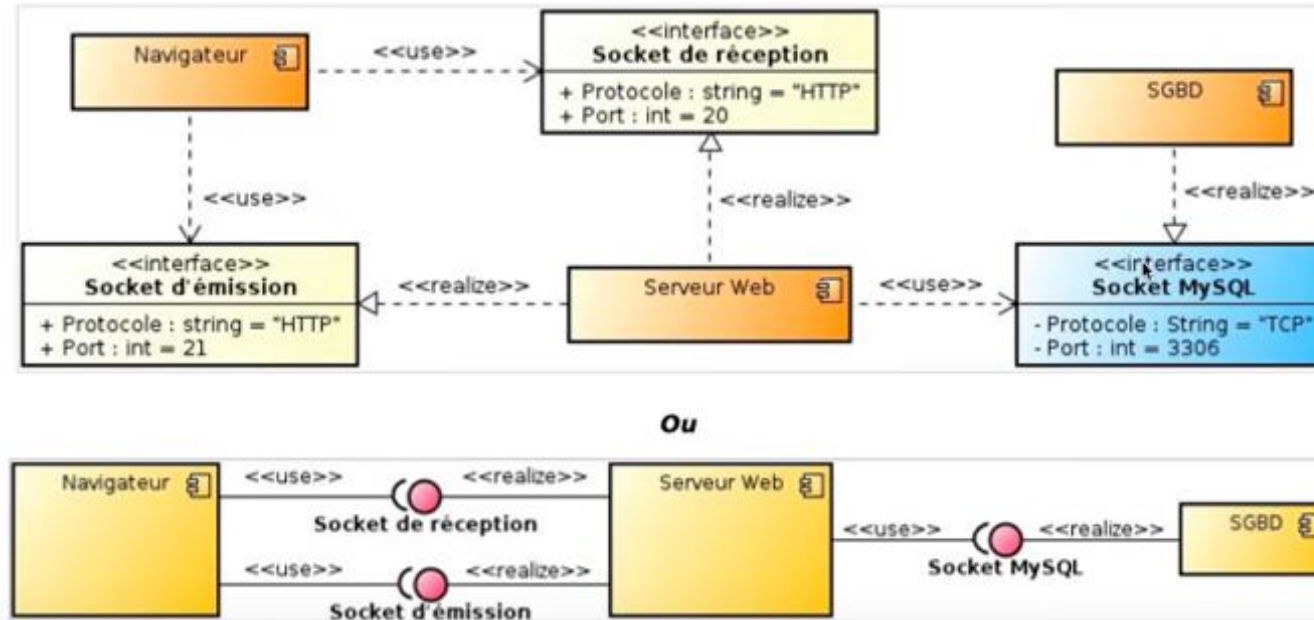
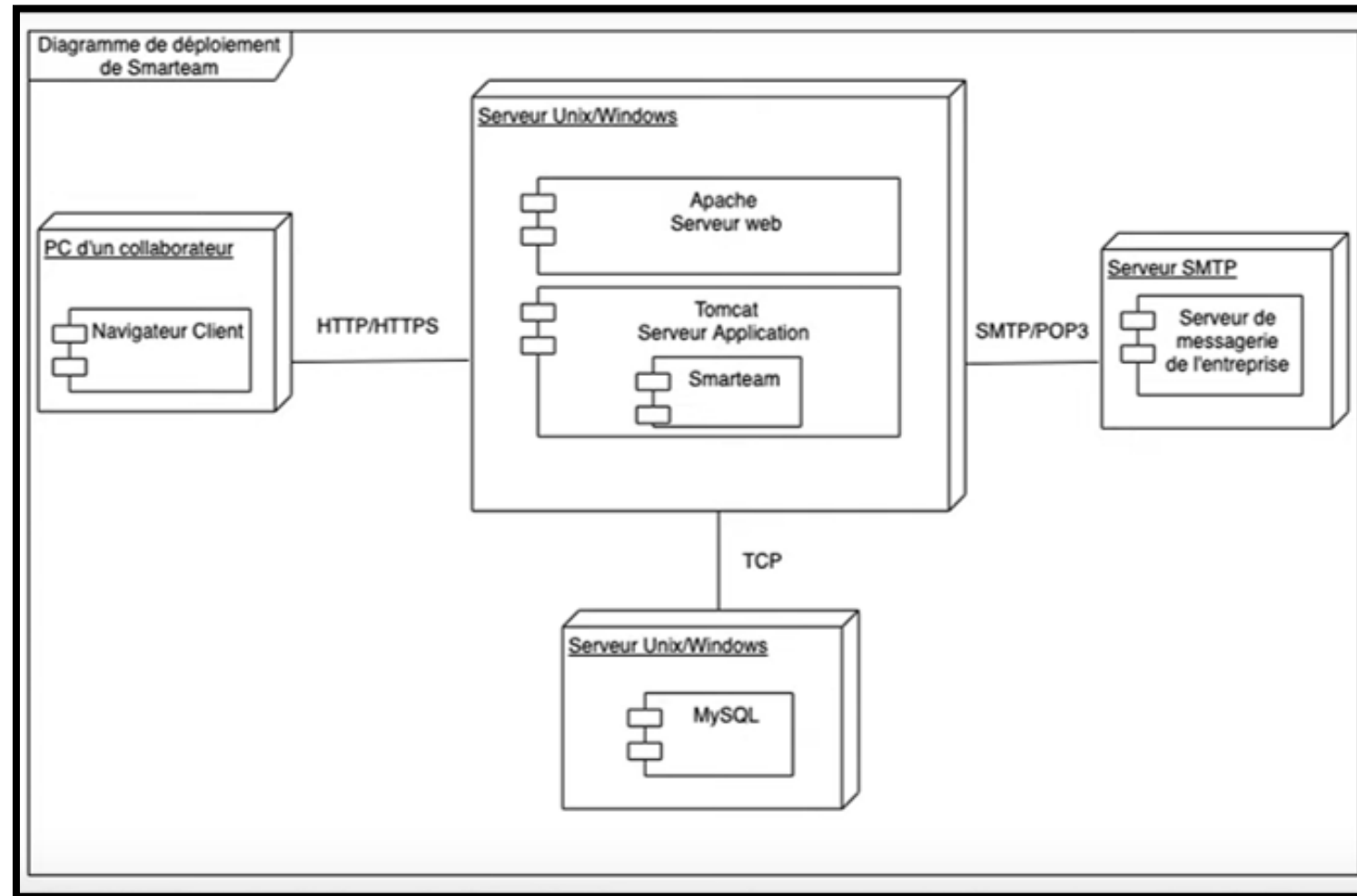


Image cours remy-manu

# Exemple diagramme de déploiement

(source image Bouchra Bouihi)





# Exercise 8

1-exercices/1-uml/ex8.md



45 min



**BONUS**





## MERISE VS UML : points communs

- Modélisation des SI
- Graphique
- Plusieurs niveaux d'étude (analyse, développement)
- Concepts similaires et formalisme (syntaxe) plus ou moins similaires

MERISE	UML
Entité	Classe
Propriété	Attribut
Cardinalités	Multiplicité
Relation	Association
Acteur	Acteur
Diagramme de flux	Diagramme d'activité



# MERISE VS UML : différences

## MERISE

- Une méthode qui sépare les flux, les données et les traitements
- Une méthode séquentielle

## UML

- Traitement conjoint des traitements et des données par le principe de l'encapsulation
- Un langage qui met à disposition plusieurs diagrammes à utiliser comme on l'entend
- Plus agile



## Utilisation UML VS MERISE

- Il n'y a pas des chiffres à l'appui, cependant UML par son envergure à l'international devrait logiquement être plus utilisé que la Merise made in méthode et donc principalement utilisé en France.
- De plus, UML offre plus de diagramme, le langage est plus flexible et adaptable (choix des diagrammes) en fonction des projets et niveau d'analyse.

FIN

Merci pour votre participation

Glodie Tshimini

[contact@tshimini.fr](mailto:contact@tshimini.fr)

