FORMATION

IT - Digital - Management

m2iformation.fr

UML & Les fondamentaux de la POO

Nehemie “Alfred” BALUKIDI
CTO/Software Engineering & Data|ML/AI

Pourquoi étudier l'UML?

- Tout ne s'arrête pas avec la maîtrise d'un langage de programmation ou d'un framework.
- Être en mesure de concevoir des logiciels.
- Connaître et appliquer les principes de la POO.
- Décrivez votre conception dans un langage normalisé en utilisant un langage graphique qui facilite la communication et la collaboration.

1.Rappels sur les méthodologies de développement

Pourquoi choisir une méthodologie?

A partir du moment où le logiciel à développer est plus qu'un simple "Hello world", et qu'en plus cela implique une ou des équipes, l'organisation devient un facteur incontournable.

Le manque d'un **processus de développement** défini finira pas mener au chaos.

Différentes approches ont été pensées pour résoudre le problème lié à l'organisation et processus à adopter dans l'industrie du développement logiciel.

Les 2 méthodologies qui vont nous intéresser sont : Le **modèle Waterfall** et **l'approche agile**.

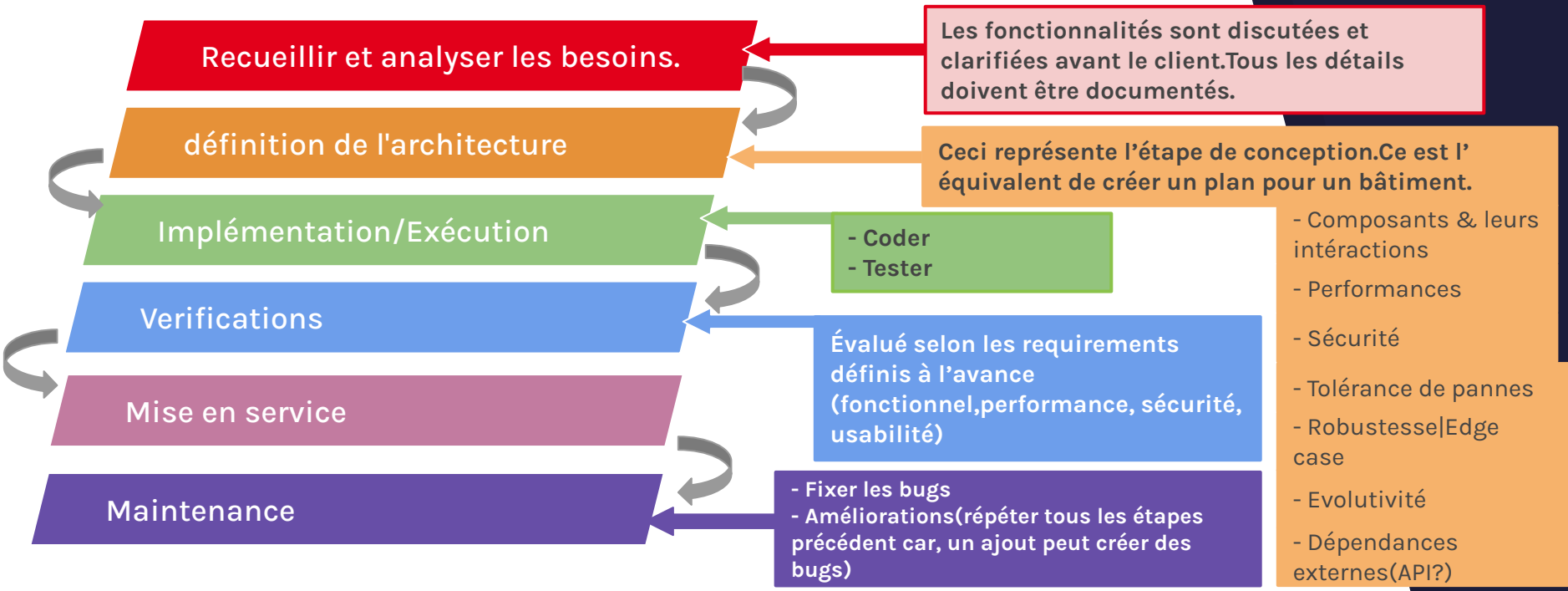
La méthodologie Waterfall(Cascade)

Cette approche veut qu'un plan détaillé puisse exister avant de commencer à écrire du code.

Les requirements sont fixés à l’avance, et aucun changement n’est possible durant la phase de développement.

La méthodologie Waterfall(Cascade)

Le Waterfall est un modèle linéaire qui définit les phases.Avant d'entamer une phase suivante, la phase précédente doit d'abord être terminée.



La méthodologie Waterfall(Cascade)

Cette approche est adaptée aux situations suivantes:

- Systèmes médicaux
- systèmes militaires : Par exemple, le développement d'un système de contrôle d'arme, les requirement doivent être définis à l'avance et être stable.Et changement les requirement durant le développement peut être très coûteux.

Utiliser le water fall si les requirements sont clairs et ne changent pas souvent, car le périmètre du projet est fixe.

Le seul inconvénient avec ce modèle vient de sa nature linéaire, qui ne permet pas d'introduire les changements entre différentes phases.

L’approche Agile

L'approche est surtout une manière de penser basée sur le manifeste agile qui prône la flexibilité pour s'adapter aux changements; une approche qui marche très bien pour les projets où les attentes peuvent changer rapidement et fréquemment et où tous les requirements ne peuvent être définis à l’avance.

Lien du manifeste Agile : https://agilemanifesto.org/iso/fr/manifesto.html

Par exemple, un projet de création d’un réseau social peut changer très rapidement selon la concurrence, etc..Il est impossible pour le client de décrire exactement tout ce qu’il faut.

L’approche Agile

La grande différence entre l’approche Agile et l’approche en Waterfall est qu’avec cette approche, les logiciels sont livrés par incrément de manière itérative au lieu de livrer un seul produit final à la fin.

Le travail est découpé en plusieurs itérations(sprint dans le cas de SCRUM).

Ici, la phase de test(recette) n’est pas séparée de la phase de développement.

Les méthodologies les plus connues de l’approche agile sont :

- SCRUM
- KANBAN

Conclusion

Aucune de ces 2 approches ne peut réellement décrire exactement toutes les étapes d'un processus de développement logiciel.

Cependant on en a besoin pour synchroniser et organiser les activités relatives au développement logiciel; activités qui ne se limitent pas qu'à l' écriture du code, mais incluent aussi la conception, gestion de projet, budgétisation, test, documentation, déploiement et maintenance.

Le waterfall est perçu comme rigide et bureaucratique.

Il y a des situations où le waterfall n'est pas adapté, particulièrement quand le degré d'incertitude est élevé et quand toutes les questions ne peuvent pas être répondues immédiatement.

2. Concepts de base de la programmation orienté objet.

Paradigmes de programmation

1950

Programmation non structurelle

L'exécution du code est séquentielle,le code est écrit comme un seul bloc entier. L'ensemble du programme est considéré comme une seule unité.
Le désavantage ici est que le code devient rapidement illisible et difficile à maintenir.Ceci conduit à un **code spaghetti**.

1960

Programmation structurelle

Le code est découpé en plusieurs sous-routines appelées aussi fonctions**(doivent êtres bien nommées)** ,modules ou encore sous-programmes. Cette approche nous évite la duplication du code, et le code est plus lisible.Ce qui facilite la maintenance et le débogage.

1980

Programmation orienté objet

L'idée est ici de **découper une partie du programme dans un objet autonome**.Système objet est associé à une partie du système.
Un o**bjet est comme un programme autonome en soi**, car un objet opère sur ses propres données a un rôle spécifique.

Objet

L'approche objet est une approche qui s'inspire des objets de la vie réelle, comme par exemple, une voiture.

Attributs(identité)
- couleur : "rouge"
- type : "Audi",
- modèle : 2019
- prix : 99580

Méthodes(comportement)
- accélerer()
- ralentir()
- tourner()
- freiner()

Attributs(identité)
- couleur : "jaune"
- type : "Mercedes",
- modèle : 300
- prix : 598989

Méthodes(comportement)
- accélerer()
- ralentir()
- tourner()
- freiner()

Dans la phrase : La voiture rouge accélère.

- **voiture :** C'est l'objet
- **Accélère :** le comportement
- **Rouge :** une propriété

Classes

La première étape dans la construction d'un système avec la POO est d'identifier les potentiels objets, leurs attributs et leurs responsabilités.

Ceci nous amène donc à la définition d'un **plan** pour la création d'objets. Dans la terminologie de la POO, ce plan est ce qu'on appelle une **classe**.

Si par exemple, dans notre programme, on veut avoir un objet voiture, on doit pouvoir décider de ce que sera une voiture dans le contexte de notre programme et ce qu'elle peut faire. La classe dira par exemple qu'une voiture est défini par le nom du fabricant et le modèle, sans savoir ce que sera le nom ou le modèle.

Les classes peuvent être utilisées dans d'autres projets par d'autres programmeurs. Mais selon les requirements de votre projet, vous pouvez être appelé à créer de nouvelles classes.

4 principes fondamentaux de la POO

La programmation orienté objet se repose essentiellement sur 4 principes :

- L’abstraction
- L’encapsulation
- L’héritage
- Le polymorphisme

Abstraction

L’abstraction consiste à décrire des problèmes complexes en de termes simples tout en ignorant les détails; en d’autres termes, mettre l'accent sur l'information essentielle, en mettant de côté l'information sans importance.

Dans le contexte de l’application Uber, le concept voiture fait penser à la marque, au modèle et au numéro de plaque.

Le fonctionnement interne du mécanisme de démarrage n’est pas un détail important.

Encapsulation & dissimulation de données

Encapsulation & dissimulation de données

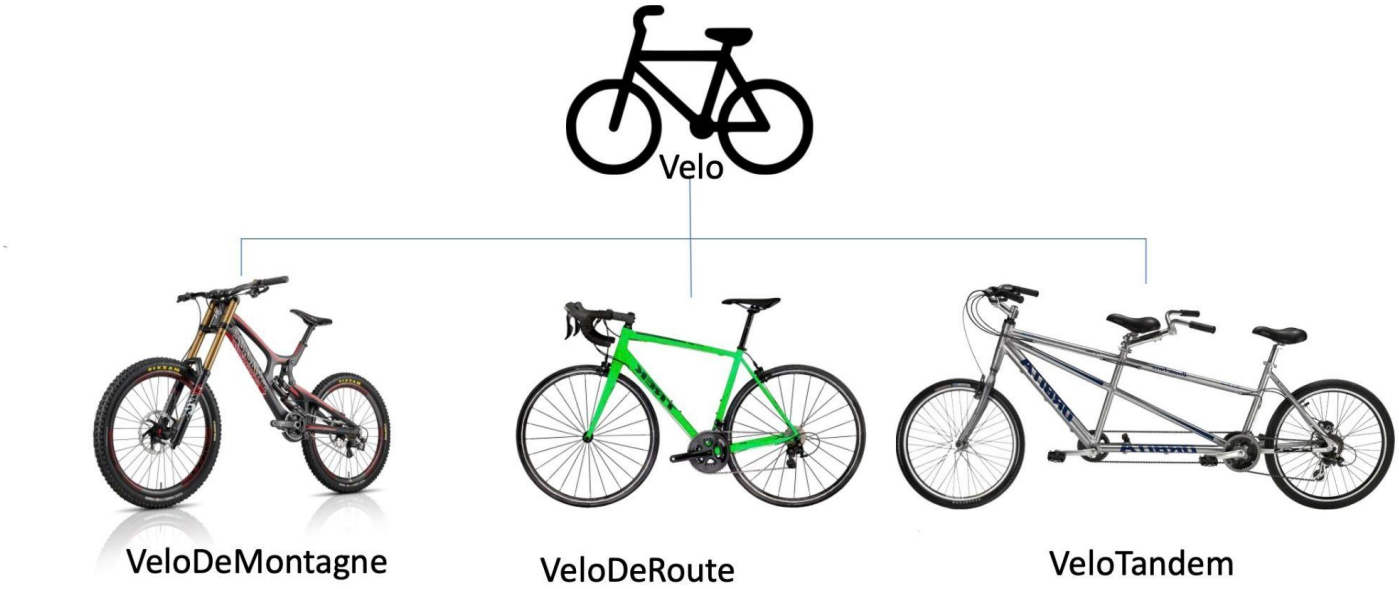
En POO l’encapsulation est le fait de grouper dans une classe les propriétés et les méthodes.

Pour qu’un utilisateur manipule un iphone, il n’a pas besoin de connaître l'électronique, ou le fonctionnement interne de l’appareil.

Si les composants électronique sont cachés, cela empêche l'utilisateur d’effectuer de mauvaises manipulations volontairement ou non.Cela réduit les dépendances(couplage) entre les classes.

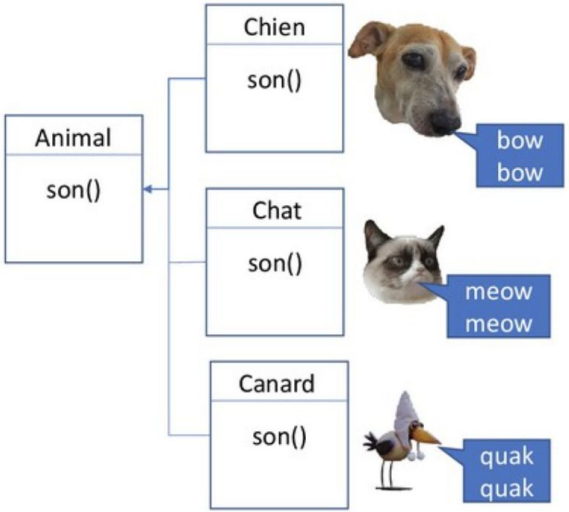
Héritage

L’héritage nous permet de réutiliser du code en minimisant la duplication.



Polymorphisme

Le polymorphisme fait référence à la possibilité d'avoir pour méthode, différents comportements .



Polymorphisme

Il en existe 2 formes:

- **La redéfinition :** Il s'agit d'avoir des implémentations spécialisées dans les classes enfants, des méthode définies dans les classes parents.

- **La surcharge :** Fournir plusieurs implémentations différentes d'une méthode(additionner 2 nombres, additionner plusieurs nombres).

3. Analyse et conception orienté objet.

Fondamentaux

Quelque soit l'approche ou la méthodologie utilisé dans la gestion d'un projet, voici les différentes qui doivent nécessairement être suivis:

1. Recueillir les besoins et identifier le problème à résoudre
2. Décrire le système
3. Identifier les entités
4. Modéliser le système

En ce qui concerne les 2 premières étapes, il n'y a pas forcément besoin d'un outil spécifique, car des notes écrites peuvent être suffisants.

3.1. Analyser les besoins

Il s’agit de l'étape de :

- Identification du problème à résoudre
- Clarification des fonctionnalités demandées
- Définir le périmètre du projet(en agile ceci peut changer ou être affiné)

Cette étape implique beaucoup d’échanges avec le client, et parfois plusieurs allers et retours pour afin de s’assurer d’avoir bien capturé les besoins.

Quand on arrive à une entente sur la compréhension du problème, cela doit être clairement documenté.

3.1. Analyser les besoins

En ce qui concerne les besoins ou requirements, il y en a 2 types :

- **Fonctionnelles :**
  Il s'agit de décrire ce que fait l'application.Par exemple c'est une application de vente en ligne, l'application doit afficher le prix selon la devise du pays de l'utilisateur en se basant sur les paramètres du système, ou par exemple, afficher selon la langue de l'appareil si celle-ci est supportée.

- **Non fonctionnelles :**
  Il ne s'agit pas de ce que peut faire l'application.C'est par exemple la performance, tout ce qui est requirements légaux(car on va peut être stocker des données personnelles), les version de IOS supportés(IOS 9 et les suivants), limiter des requêtes inutiles pour limiter des frais sur les données utilisé et préserver la batterie.

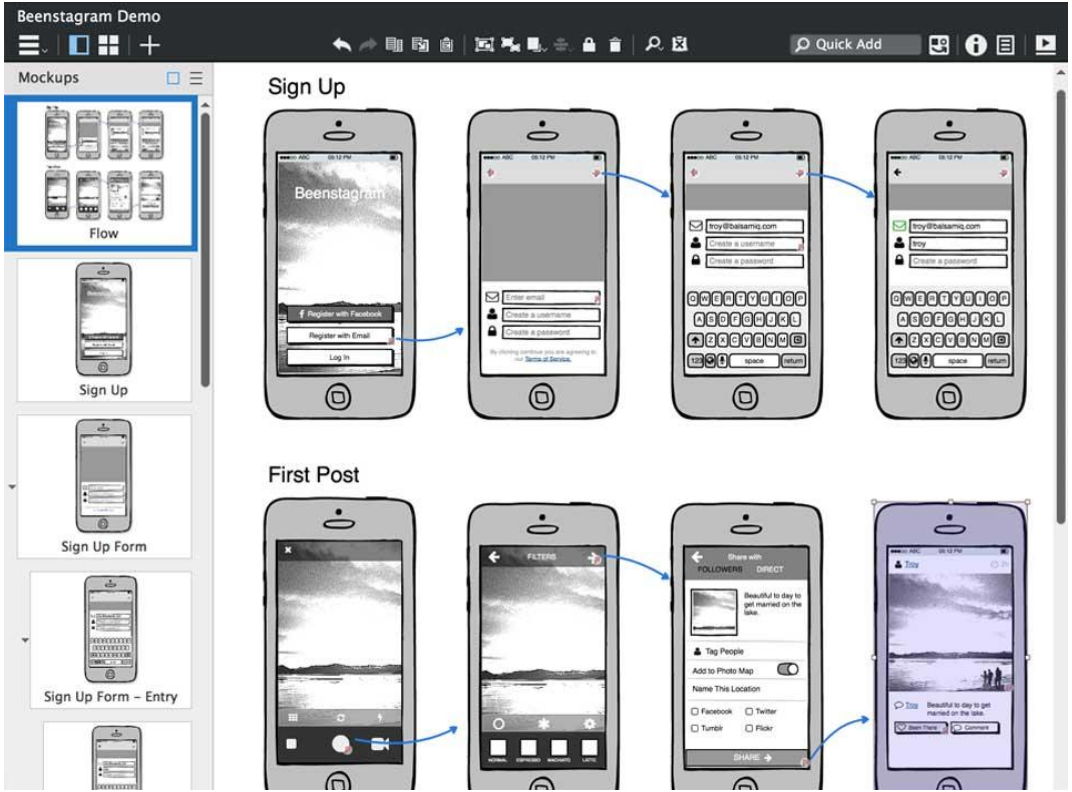
3.2. Décrire le système

Après la phase précédente, on peut alors choisir une approche Agile ou en waterfall selon la nature du projet.Ce qu'il faut juste noter est qu'en de choix des méthodologies agiles, on peut ne pas décrire de manière très détaillé les fonctionnalités du système.

- On décrit les fonctionnalités du système du point de vue de l'utilisateur(acteur).Dans une approche Agile, on peut se limiter aux users stories.
- On va réaliser si besoin des wireframes et prototypes

L'idée c'est d'avoir une manière de bien communiquer sa compréhension du problème au client.

Décrire le système



3.3. Identifier les entités à manipuler

Il s’agit des acteurs qui vont avoir des rôles spécifiques dans l'application.

Par exemple, c’est une application de vente en ligne, on aura besoin d’une classe qui va représenter un produit et posséder les attributs suivants:

- nom
- prix
- Description

Ou une classe responsable de la communication avec le serveur avec la possibilité de récupérer les articles, les mettre à jour ou les supprimer.

Ceci est rendu facile par ce qui est fait dans les phases précédentes.

3.4. Modéliser le système

C’est ici qu’intervient la conception orientée objet.

Il s’agit de créer par exemple de:

- représentations graphique de nos classes, leurs attributs et comportements
- modéliser les interactions entre les objets

Le langage que nous allons donc choisir c’est l’UML car il offre une notation graphique et standard.

4.UML

UML

UML n’est pas un langage de programmation, mais un système de notation graphique standard qui facilite la collaboration entre développeurs, mais aussi avec le client car certains diagrammes permettent spécifiquement de communiquer avec le client.

Ceci permet donc d’avoir une vision claire du système à concevoir sans avoir à écrire du code.

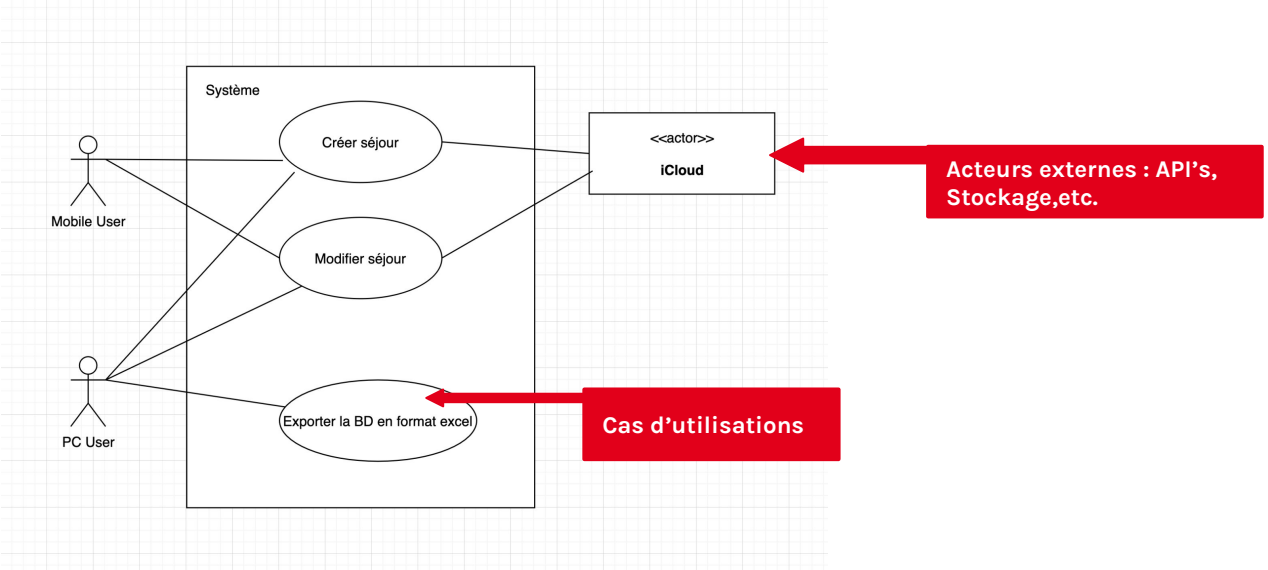
5. Préparer l’environnement

https://www.draw.io/

6. Diagramme de cas d'utilisation

Introduction au diagramme de cas d'utilisation

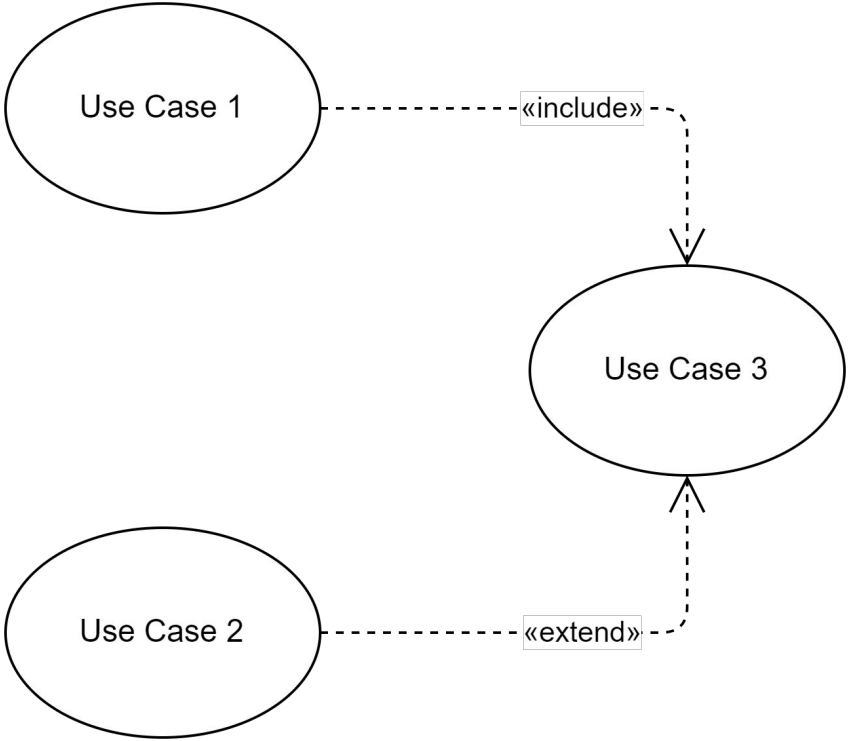
Permet de représenter les différentes fonctionnalités du système ainsi que les acteurs associés



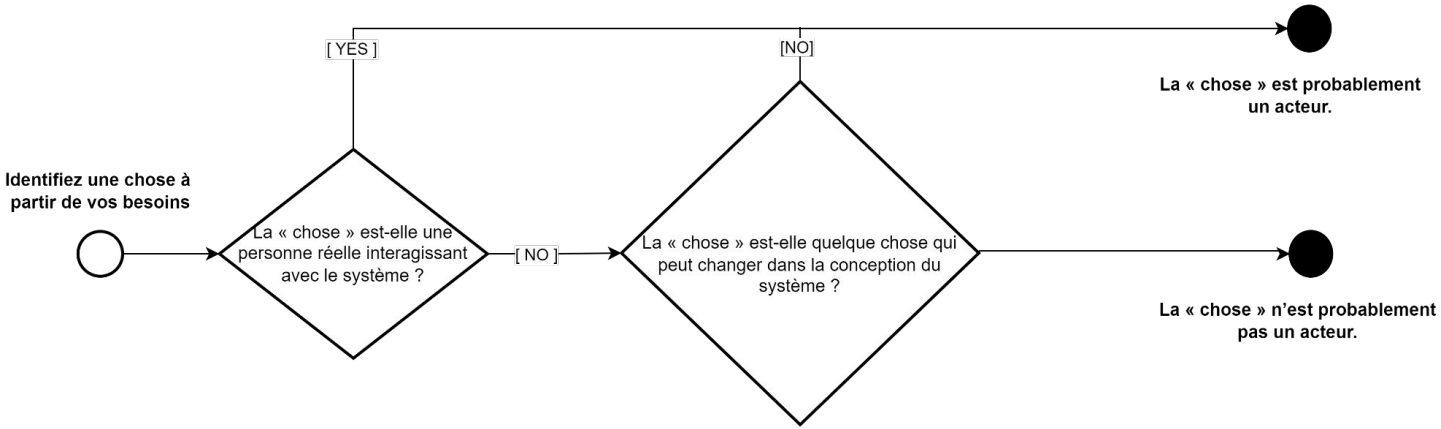
Use case(cas d'utilisation)

- Un cas d'utilisation représente une fonctionnalité ou un service fourni par le système à un acteur.
- Il décrit une interaction entre un acteur et le système, en vue d'atteindre un objectif précis.
- Les cas d'utilisation sont représentés graphiquement par des ellipses dans un diagramme de cas d'utilisation.

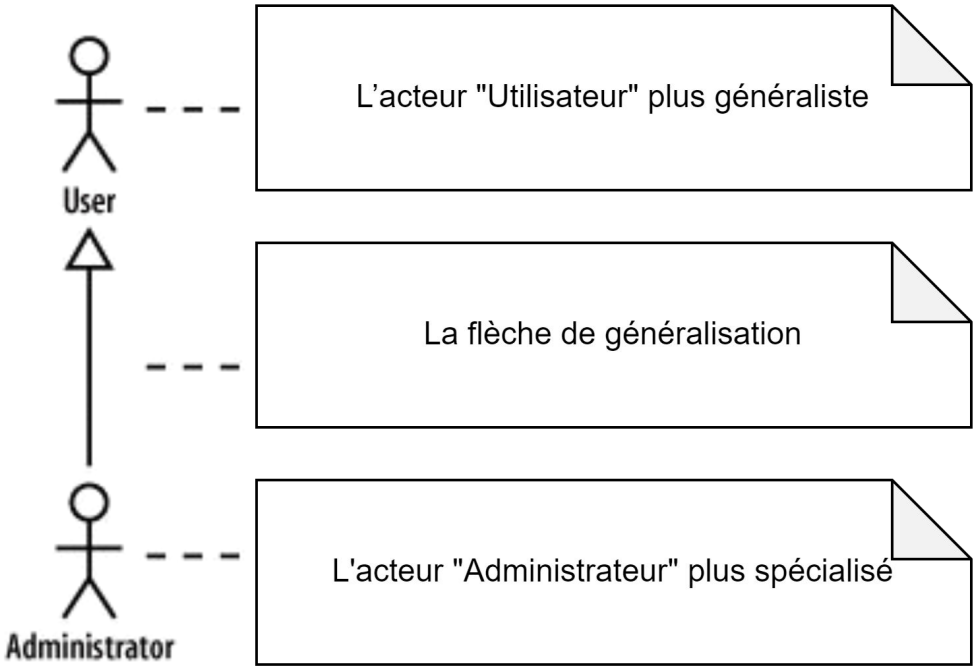
Relations de cas d'utilisation



Qui est un acteur?



Généralisation des acteurs



Atelier 1: DAB Use Case Diagram

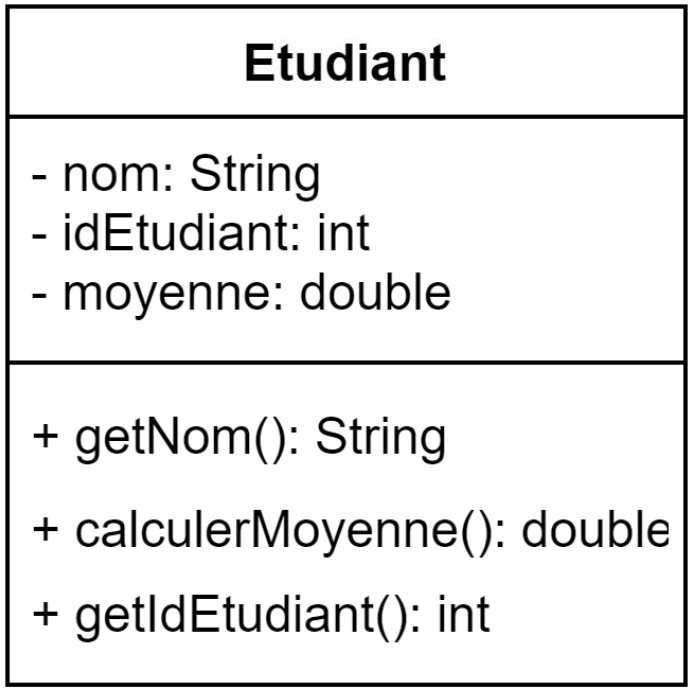
Utiliser les US à l'adresse https://gist.github.com/Olfredos6/e34c3eb2010a8cedf5a6dce86c3d1ee8 pour dessiner une Use Case Diagram

7. Diagramme de classes

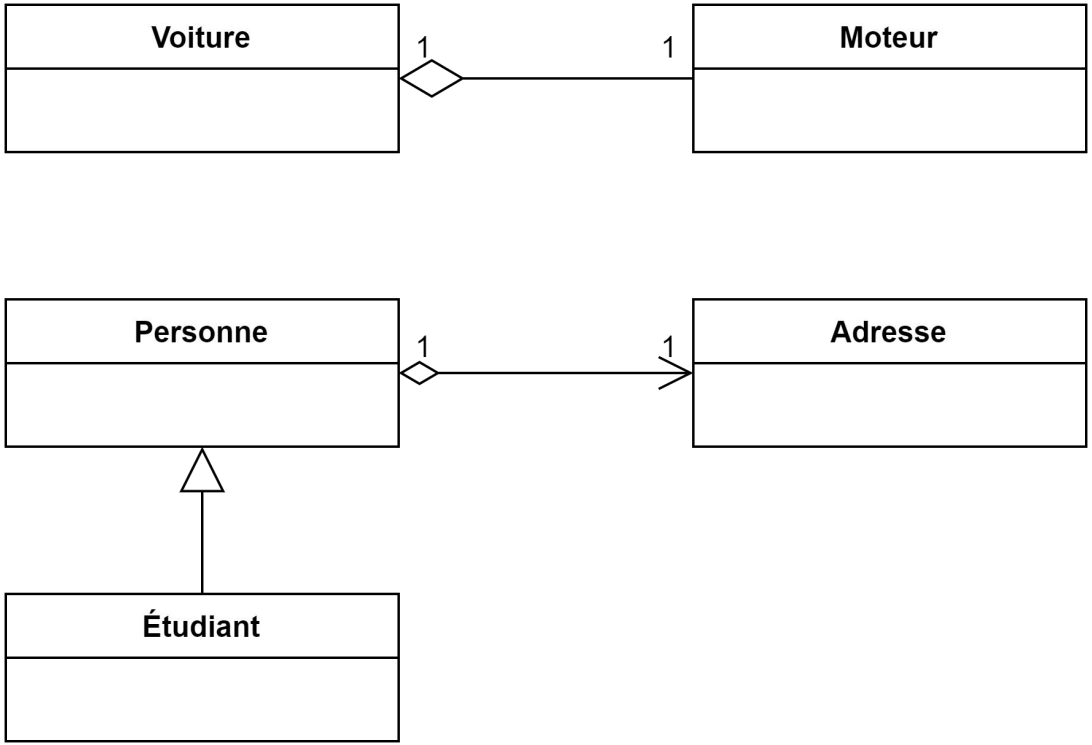
Introduction aux diagrammes de class

- Les diagrammes de classes sont un élément central d'UML.
- Ils représentent visuellement la structure d'un système en montrant les classes, leurs attributs, leurs méthodes et les relations entre elles.
- Les diagrammes de classes fournissent une vue statique du système et sont couramment utilisés dans la conception et l'analyse orientées objet.

Représentation d'une classe



Relations entre les classes



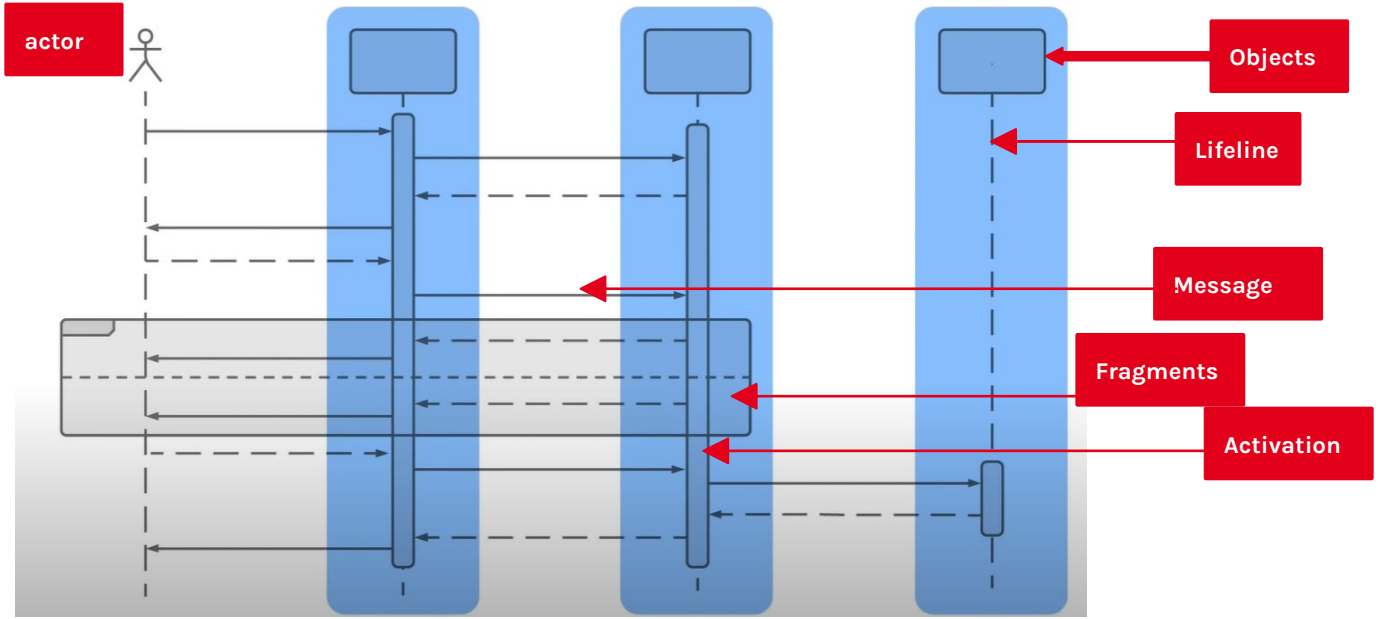
Atelier 2: DAB Class Diagram

Utilisez les US et le diagramme de cas d'utilisation du premier atelier pour modéliser un diagramme de classes."

8. Diagramme de séquençage

Sequence diagram

Permet de représenter visuellement la séquence des interactions entre différents objets ou acteurs dans un système logiciel ou un processus



Atelier 3: DAB Sequence Diagram

Représentez les interactions entre les différents composants du cas d'utilisation **Retirer de l’argent** à l'aide d'un diagramme de séquence.

FIN.