

Ice break

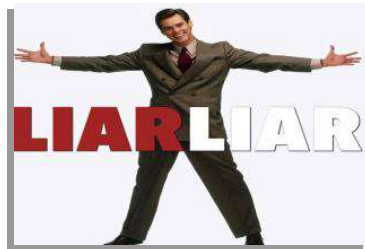
Jeu : un mensonge et deux vérités

Dites deux vérités et une mensonge sur vous et l'audience se chargera de trouver le mensonge ainsi que les deux vérités.

Exemples :

- J'ai quatre enfants,
- J'ai gagné un concours de beauté une fois,
- Je suis allé en Chine,

Pour plus d'infos > <https://fr.wikihow.com/jouer-au-jeu-1-mensonge-et-2-vérité>



Objectifs

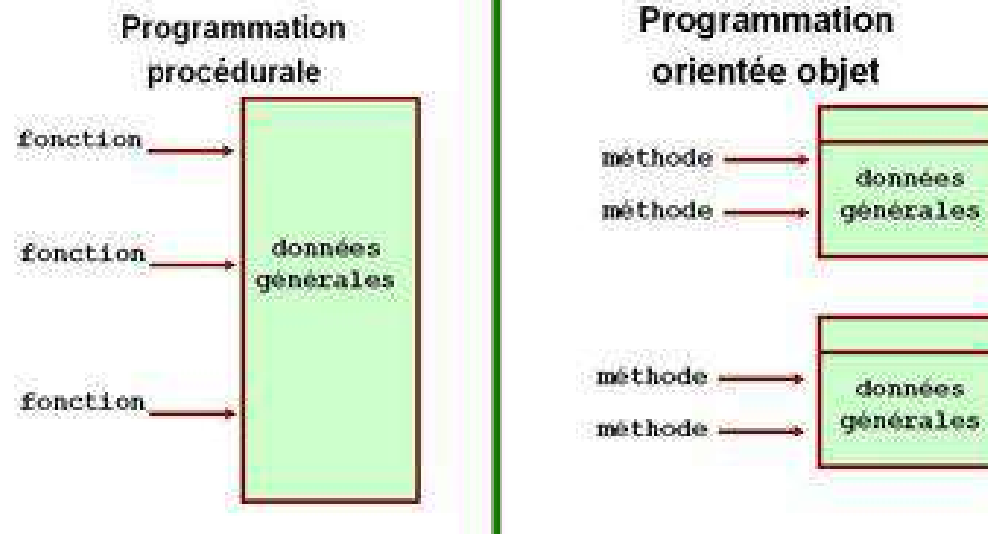
- Apprendre les concepts de base théoriques d'UML et leur représentation graphique.
- Illustrer la modélisation logiciel à l'aide de certains diagrammes UML en s'appuyant sur des exemples et des exercices pratiques.

Plan

1. **Concepts de l'approche objet**
2. Modélisation UML
3. Modélisation objet élémentaire avec UML
 - Diagrammes de cas d'utilisation
 - Diagrammes de classes
 - Diagrammes d'objets
 - Diagrammes de séquences

Définition du paradigme orienté objet

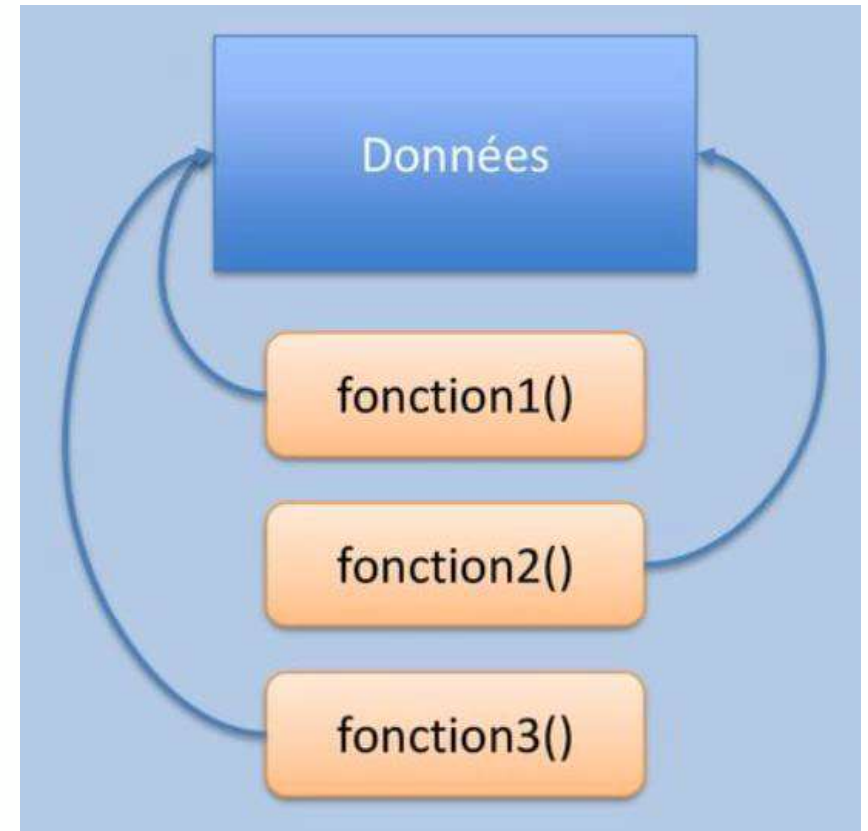
- La **programmation orientée objet** (POO) est un modèle de programmation informatique qui met en œuvre une conception basée sur les objets.
- La POO se différencie de la **programmation procédurale**, qui est basée sur l'utilisation des **procédures** et de la programmation fonctionnelle, qui elle, se repose entièrement sur le concept de **fonction**.



POO vs Programmation procédurale (1/2)

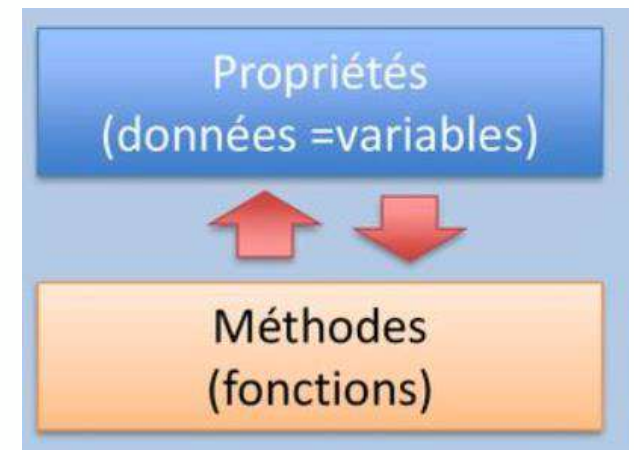
La programmation revient :

- Définir des variables
- Ecrire des procédures (fonctions) pour les manipuler sans associer explicitement les unes aux autres.
- Exécuter un programme se réduit à appeler ces procédures dans un ordre décrit par la séquence des instructions et en leur fournissant les données nécessaires à l'accomplissement de leurs tâches.



POO vs Programmation procédurale (2/2)

- Au fur et à mesure que les programmes écrits avec une approche procédurale grossissent, ils deviennent **fragiles** et mettent en danger les **données**.
- La **POO** c'est un peu comme le GOLF
 - Les concepts de base sont simples à apprendre mais :
 - Les utiliser tous et de manière élégante est beaucoup plus difficile.
 - 4 concepts de bases :
 - **Propriétés** (les variables) et **méthodes** (les fonctions)
 - Classe et objet
 - Encapsulation
 - Héritage

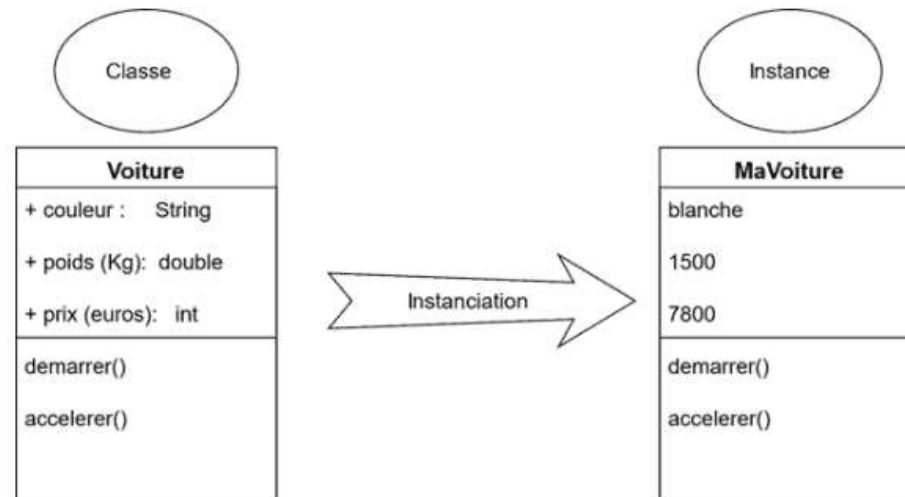


Avantage du paradigme POO

- Avantages du paradigme orienté objet :
 - **La modularité** : les objets forment des modules compacts regroupant des données et un ensemble d'opérations.
 - **L'abstraction** : les objets de la POO sont proches de celles du monde réel. Les concepts utilisés sont donc proches des abstractions familières que nous exploitons.
 - **Productivité et ré-utilisabilité** : plus l'application est complexe et plus l'approche POO est intéressante. Le niveau de réutilisabilité est supérieur à la programmation procédurale.
 - **Sûreté** : l'encapsulation et le typage des classes offrent une certaine robustesse aux applications.

Le concept de classe

- Le premier concept de la POO est la classe qui est une structure abstraite décrivant les objets du monde réel sous deux angles.
 - ses propriétés (ses caractéristiques) et ses méthodes (les actions qu'elle peut effectuer ou son comportement).
- La classe est une sorte de moule ou de modèle.
- Les objets sont construits à partir de ce moule (classe) par un processus appelé **instanciation**.
- Toutes les instances de classe s'appellent des objets.

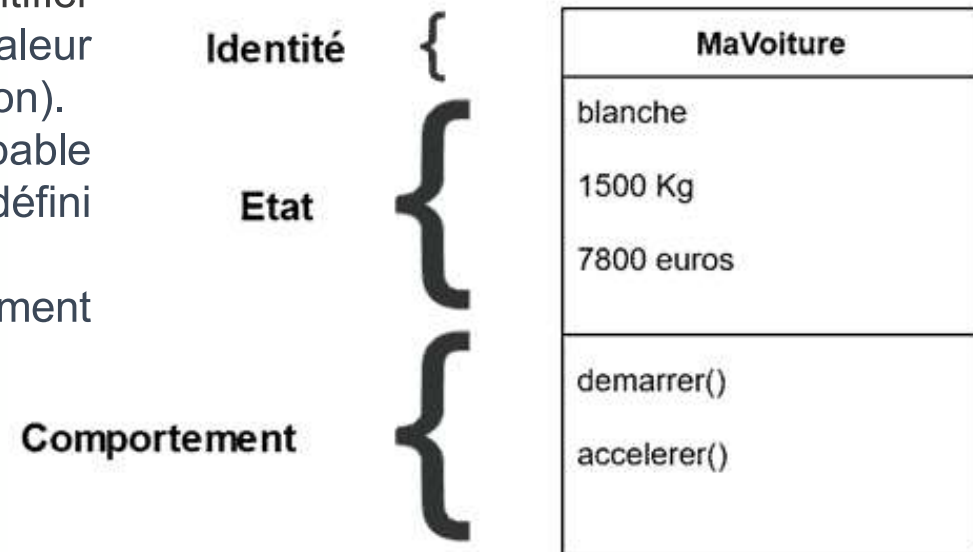


Le concept d'objet (1/2)

- Un **objet** représente une entité du monde réel qui se caractérise par un ensemble de **propriétés (attributs)**, des **états significatifs** et un **comportement**.
- Le **comportement** d'un objet est caractérisé par l'ensemble des opérations qu'il peut exécuter en réaction aux messages provenant des autres objets.
- **Exemple** : Considérons l'employé Durand, n°1245, embauché en tant que ingénieur dans le site N.
- Cet objet est caractérisé par la liste de ses attributs et son état est représenté par les valeurs de ses attributs :
 - n° employé : 1245
 - nom : Durand
 - qualification : ingénieur
 - Lieux de travail : site N
- Son comportement est caractérisé par les opérations qu'il peut exécuter :
 - entrer dans l'organisme
 - changer de qualification
 - changer de lieu de travail
 - sortir de l'organisme

Le concept d'objet (2/2)

- Un **objet** est une instance de classe.
- Pour faire le parallèle avec le monde réel, l'objet c'est un peu comme une maison bâtit sur la base d'un plan particulier. Tant que les architectes se réfèrent à ce plan, ils produiront toujours les mêmes maisons.
- Un objet est caractérisé par :
 - **une identité** : l'identité doit permettre d'identifier sans ambiguïté l'objet. Chaque objet a une valeur par défaut (lorsqu'elle est indiquée à l'instanciation).
 - **des méthodes** : chaque objet est capable d'exécuter les actions ou le comportement défini dans la classe.
 - Ces actions sont traduites en POO concrètement sous forme de **méthodes**.



Encapsulation

- Les attributs et les méthodes d'un objet qui constituent sa structure interne ne sont pas en général pas accessibles aux autres objets : c'est le principe de l'encapsulation.
- Par exemple, pour pouvoir modifier la couleur d'une voiture, il faudra lui ajouter une méthode publique, **changerCouleur**, qui s'occupera de changer la valeur de son attribut **couleur**.
- Les autres objets, n'ont ainsi plus besoin de savoir comment changer la couleur de la voiture, ils se contentent d'appeler la méthode **changerCouleur**.

Plan

1. Concepts de l'approche objet
- 2. Modélisation UML**
3. Modélisation objet élémentaire avec UML
 - Diagrammes de cas d'utilisation
 - Diagrammes de classes
 - Diagrammes d'objets
 - Diagrammes de séquences

Bibliographie — UML

- UML 2.0, guide de référence
 - James Rumbaugh, Ivar Jacobson, Grady Booch
 - Edition Campus Press (2005)
- UML 2.0
 - Benoit Charoux, Aomar Osmani, Yann Thierry-Mieg
 - Edition Pearson, Education France (2008)
- UML 2.0 Superstructure et UML 2.0 Infrastructure
 - OMG (Object Management Group)
 - www.uml.org (2004)

Matériel et logiciel

- Systèmes informatiques :
 - **80 % de logiciels**
 - 20 % de matériel
- Depuis quelques années, la fabrication du matériel est assurée par quelques fabricants seulement.
 - le matériel est relativement fiable
 - le marché est standardisé

Les problèmes liés à l'informatique sont essentiellement des problèmes logiciel.

La *"crise du logiciel"*

- Étude sur 8 380 projets (Standish Group, 1995) :
 - succès : 16 %;
 - problématique : 53 % (budget ou délais non respectés, défaut de fonctionnalités) ;
 - échec : 31 % (abandonné).

Le taux de succès décroît avec la taille des projets et la taille des entreprises.

- Génie Logiciel (Software Engineering) :
 - comment faire des logiciels de qualité ?
 - qu'attend-on d'un logiciel ?
 - quels sont les critères de qualité ?

Critère de qualité d'un logiciel

- **Utilité**
 - Adéquation entre le logiciel et les besoins des utilisateurs.
- **Utilisabilité**
- **Fiabilité**
- **Interopérabilité**
 - Interactions avec d'autres logiciels.
- **Performance**
- **Portabilité**
- **Réutilisabilité**
- **Facilité de maintenance**
 - Un logiciel ne s'use pourtant, la maintenance absorbe une très grosse partie des efforts de développement.

Cycle de vie d'un logiciel

La qualité du processus de fabrication garantie de la qualité du produit.

- Pour obtenir un logiciel de qualité, il faut en maîtriser le processus d'élaboration.
 - La vie d'un logiciel est composée de différentes étapes.
 - La succession de ces étapes forme le cycle de vie du logiciel.
 - Il faut contrôler la succession de ces différentes étapes.

Étapes du développement d'un logiciel

1. Étude de faisabilité
2. Spécification
 - Déterminer les fonctionnalités du logiciel.
3. Conception
 - Déterminer la façon dont le logiciel fournit les différentes fonctionnalités recherchées.
4. Codage
5. Tests
 - Essayer le logiciel sur des données d'exemple pour s'assurer qu'il fonctionne correctement.
6. Maintenance

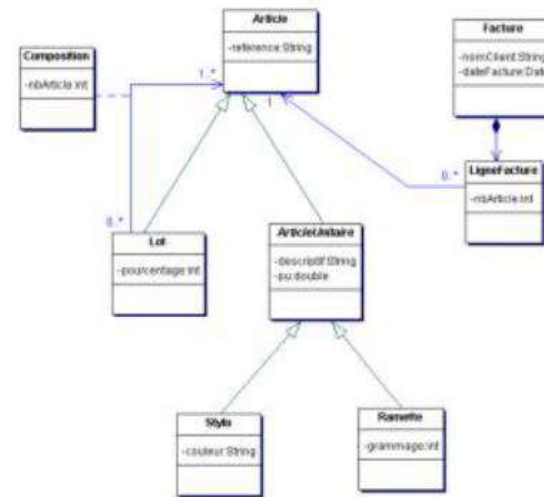
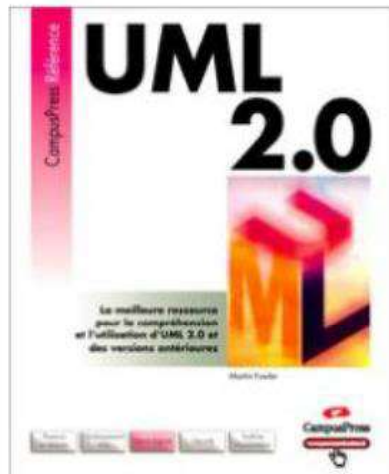
Modèle

Modèle = objet conçu et construit (artefact)

- Un modèle est une représentation abstraite de la réalité qui exclut certains détails du monde réel.
 - il permet de réduire la complexité d'un phénomène en éliminant les détails qui n'influencent pas son comportement de manière significative.
 - il reflète ce que le concepteur croit important pour la compréhension et la prédiction du phénomène modélisé, les limites du phénomène modélisé dépendent des objectifs du modèle.

Concevoir un modèle UML

- Avant de construire un bâtiment, il nous faut un plan.
- En programmation objet, il faut élaborer le schéma UML avant de commander le codage.



UML — Unified Modeling Language

- Au milieu des années 90, les auteurs de Booch, OOSE et OMT ont décidé de créer un langage de modélisation unifié avec pour objectifs :
 - modéliser un système des concepts à l'exécutable, en utilisant les techniques orientée objet ;
 - réduire la complexité de la modélisation ;
 - utilisable par l'homme comme par la machine :
 - représentations graphiques mais disposant de qualités formelles suffisantes pour être traduites automatiquement en code source ;
 - ces représentations ne disposent cependant pas de qualités formelles suffisantes pour justifier d'aussi bonnes propriétés mathématiques que des langages de spécification formelle (Z, VDM...).
- Officiellement UML est né en 1994.

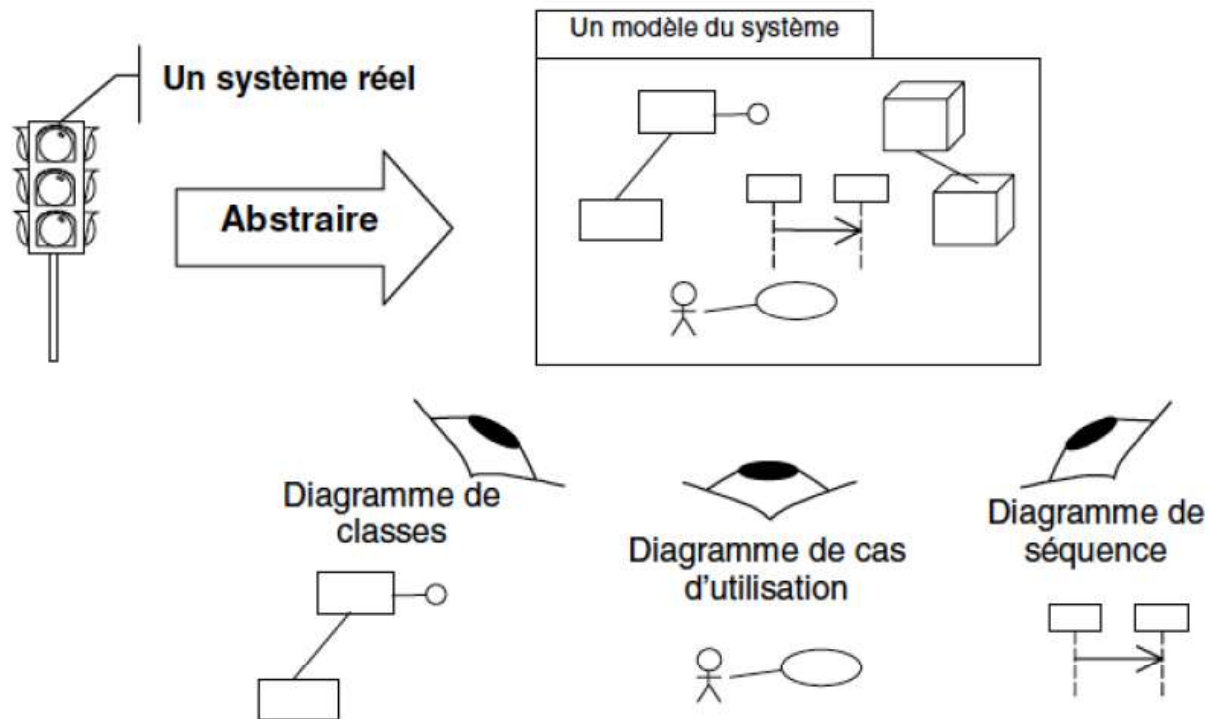
UML v2.0 date de 2005. Il s'agit d'une version majeure apportant des innovations radicales et étendant largement le champ d'application d'UML.

Différents types de diagrammes

Il existe 2 types de vues du systèmes qui comportent chacune leurs propres diagrammes (en tout 9 type de diagrammes)..

- les vues statiques
 - **diagrammes de cas d'utilisation**
 - **diagrammes d'objets**
 - **diagrammes de classes**
 - diagrammes de composants
 - diagrammes de déploiement
- les vues dynamiques
 - diagrammes de collaborations
 - **diagrammes de séquences**
 - diagrammes d'états-transitions
 - diagrammes d'activités

Diagramme UML : points de vue sur le système



[Image empruntée à Muller et Gaertner]

L'utilisation des diagrammes

- **Définition d'un diagramme** : un diagramme UML est une représentation graphique, qui s'intéresse à un aspect précis du modèle. L'UML permet de définir et de visualiser un modèle à l'aide de diagrammes.
- **Caractéristiques des diagrammes UML** : les diagrammes UML supportent l'abstraction. La structure des diagrammes et la notation graphique des éléments sont normalisés.

Plan

- Concepts de l'approche objet
- Modélisation UML
- **Modélisation objet élémentaire avec UML**
 - **Diagrammes de cas d'utilisation**
 - Diagrammes d'objets
 - Diagrammes de classe
 - Diagrammes de séquences

Modélisation des besoins

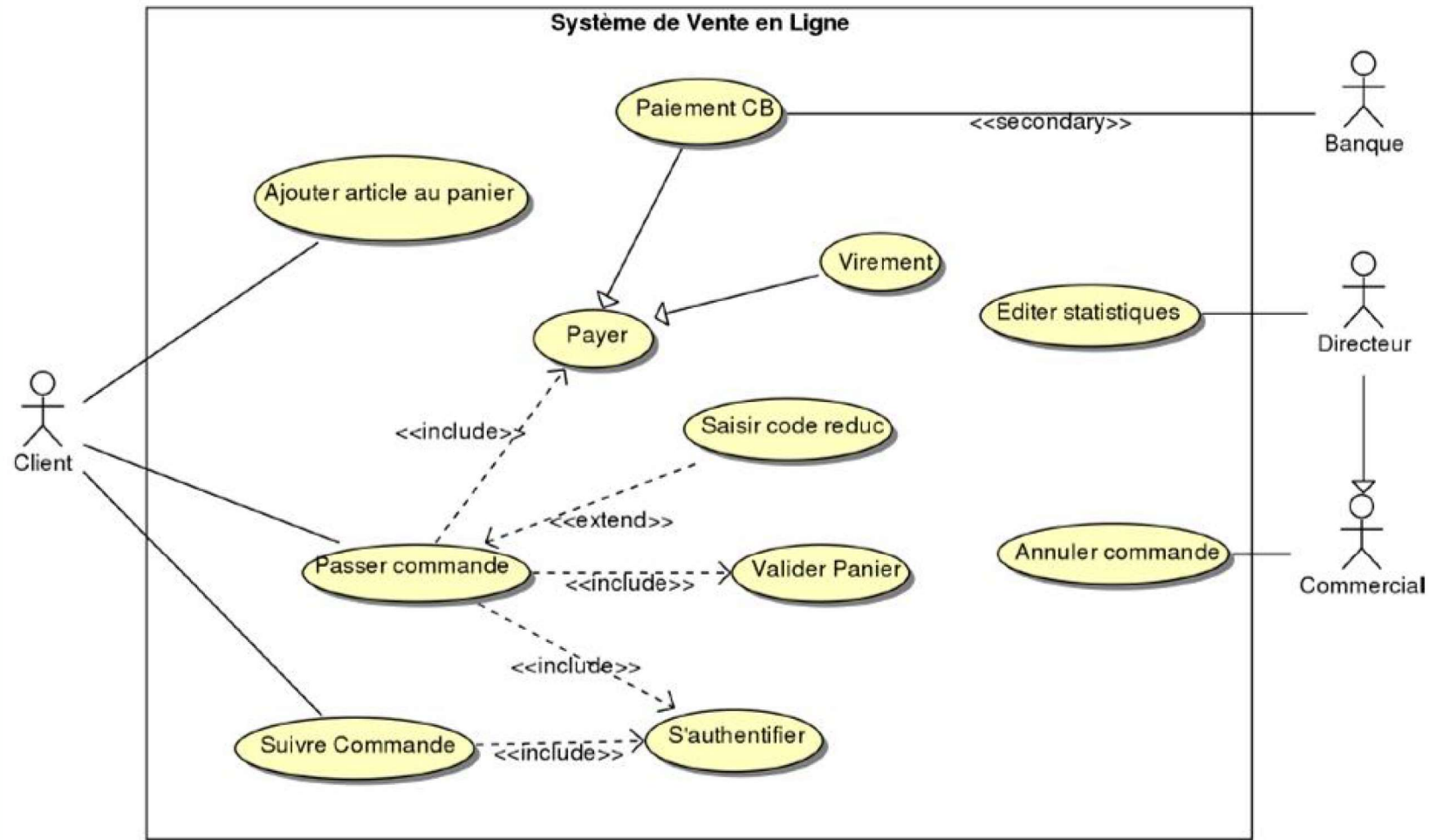
Avant de développer un système, il faut savoir **précisément** à QUOI il devra servir, cad à quels besoins il devra répondre.

- **Modéliser les besoins** permet de :
 - faire l'inventaire des fonctionnalités attendues ;
 - organiser les besoins entre eux, de manière à faire apparaître des relations (réutilisations possibles, ...).
- Avec UML, on modélise les besoins au moyen de **diagrammes de cas d'utilisation**.

Diagramme de cas d'utilisation (use case)

- Les cas d'utilisation ou use cases permettent de structurer les besoins des utilisateurs et les objectives correspondants d'un système.
- Les cas d'utilisation identifient les utilisateurs du systèmes (acteurs) et leurs interactions avec le système.
- Permettent de classer les acteurs et structurer les objectifs du système:
 - Définissent le contour du système à modéliser en précisant le but à atteindre
 - Permettent d'identifier les fonctionnalités principales du système

Exemple de diagramme de cas d'utilisation

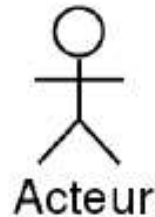


Cas d'utilisation

- Un **cas d'utilisation** est un service rendu à l'utilisateur, il implique des séries d'actions plus élémentaires.

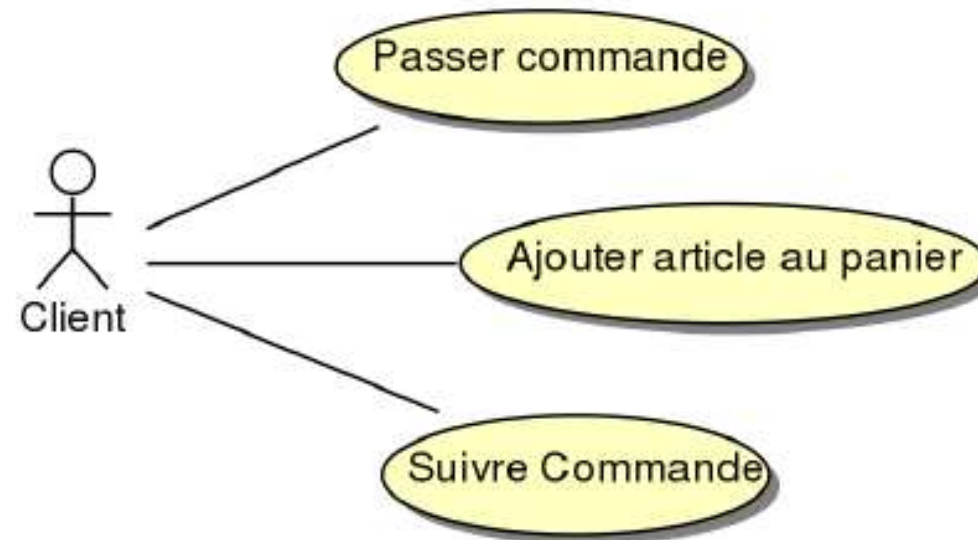


- Un **acteur** est une entité extérieure au système modélisé, et qui interagit avec lui.



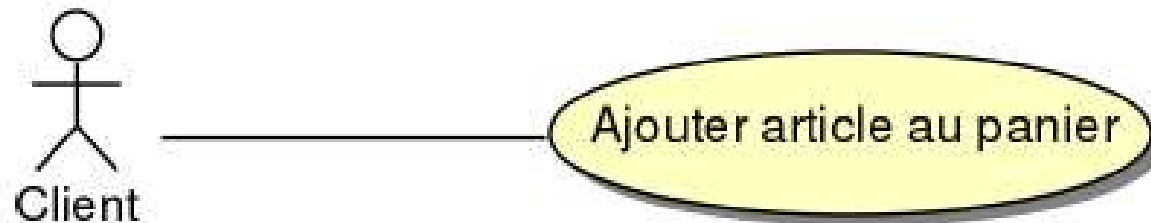
Un **cas d'utilisation** est l'expression d'un service réalisé de bout en bout, avec un déclenchement et un déroulement.

Acteurs et cas d'utilisation



Relations entre cas d'utilisation et acteurs

- Les acteurs impliqués dans un cas d'utilisation lui sont liés par une **association**.
- Un acteur peut utiliser plusieurs fois le même cas d'utilisation.

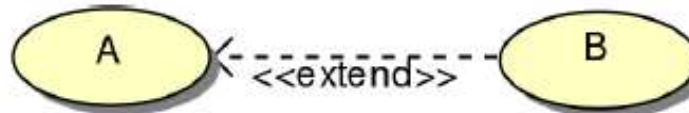


Relations entre cas d'utilisation

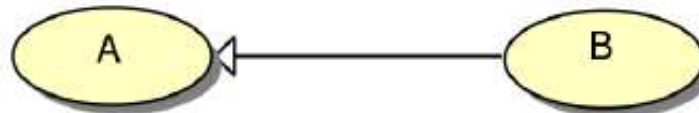
- **Inclusion** : le cas A inclut le cas B (*B est une partie obligatoire de A*).



- **Extension** : le cas B étend le cas A (*A est une partie optionnelle de B*)



- **Généralisation** : le cas A est une généralisation du cas B (*B est une sorte de A ou le cas d'utilisation enfant B est une spécialisation du cas d'utilisation parent*)



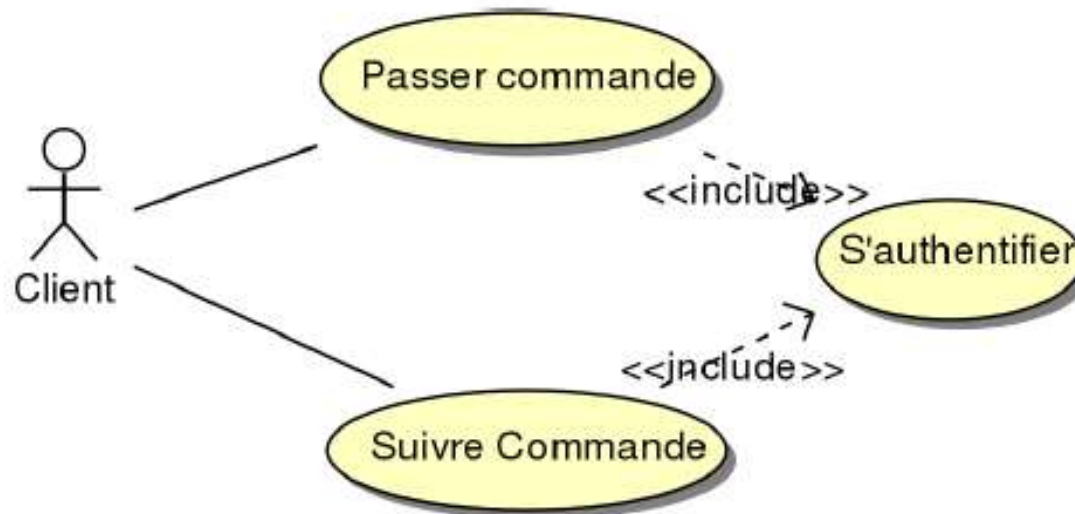
Dépendances d'inclusion et d'extension

- Les inclusions et les extensions sont représentées par des **dépendances**.
 - lorsqu'un cas B inclut un cas A, B dépend de A.
 - lorsqu'un cas B étend un cas A, B dépend aussi de A.
 - on note toujours la dépendance par une flèche pointillée B --> A qui se lit <<B dépend de A >>.
- Lorsqu'un élément A dépend d'un élément B, toute modification de B sera susceptible d'avoir un impact sur A.
- Les <<**include**>> et les <<**extend**>> sont des stéréotypes (entre guillemets) des relations de dépendance.

Attention : Le sens des flèches indique la dépendance, pas le sens de la relation d'inclusion.

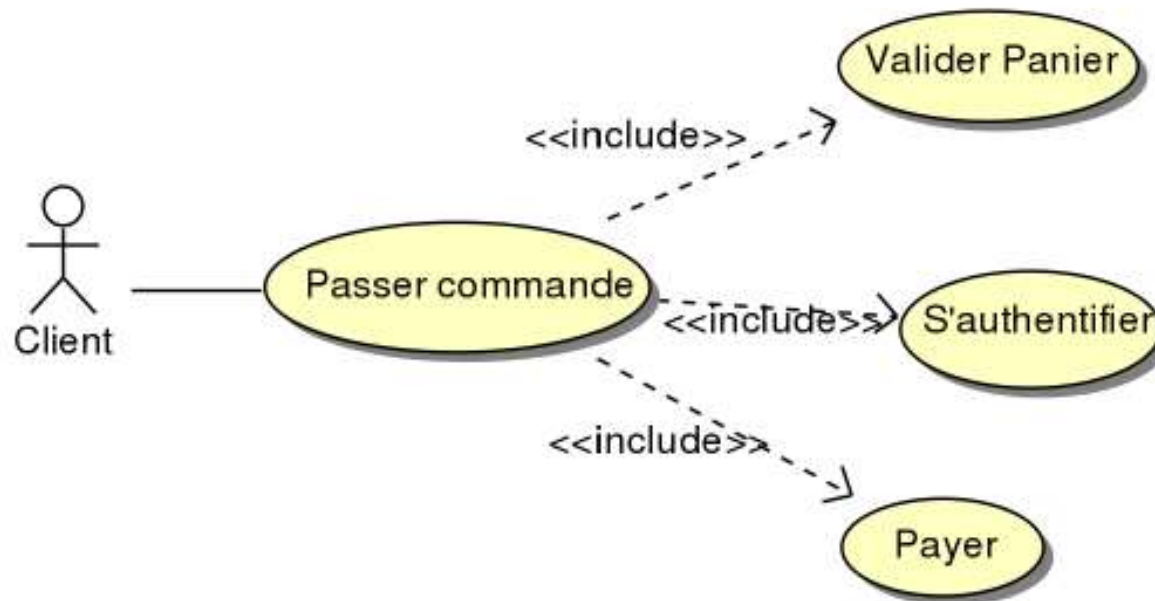
Réutilisabilité avec les inclusions et les extensions

Les relations entre cas permettent la réutilisabilité du cas **s'authentifier** : il sera inutile de développer plusieurs fois un module d'authentification.



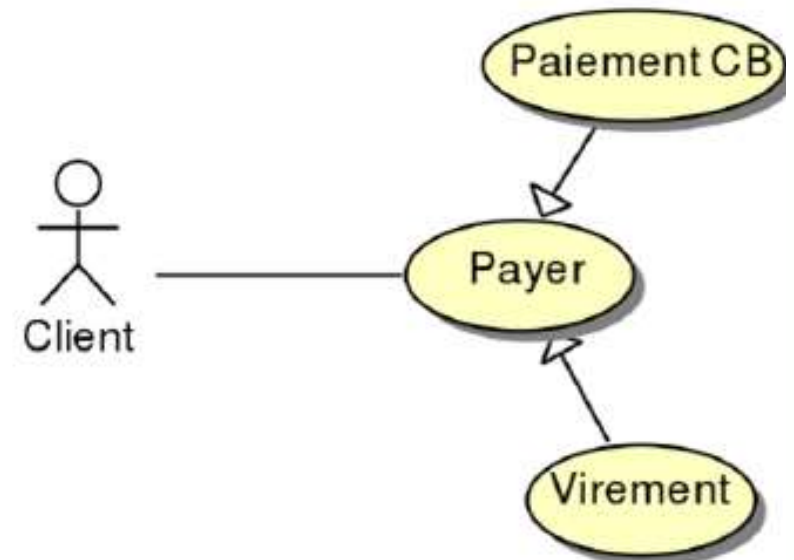
Décomposition grâce aux inclusions et aux extensions

- Quand un cas est trop complexe (faisant intervenir un trop grand nombre d'actions), on peut procéder à sa décomposition en cas plus simples.



Généralisation

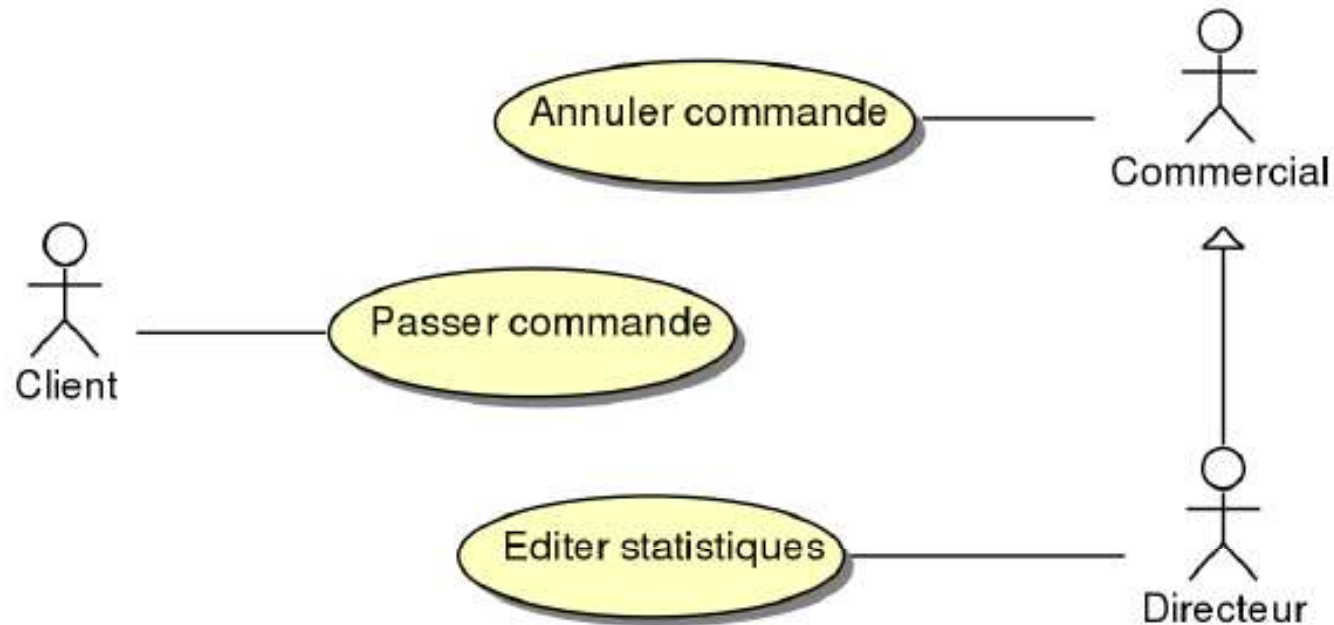
- Un virement est un cas particulier de paiement.
- Un virement **une sorte de** paiement



- La flèche pointe vers l'élément général
- Cette relation de généralisation/spécialisation est présente dans la plupart des diagrammes UML et se traduit par le concept d'héritage dans les langages orientés objet.

Relations entre acteurs

- Une seule relation possible : la généralisation.



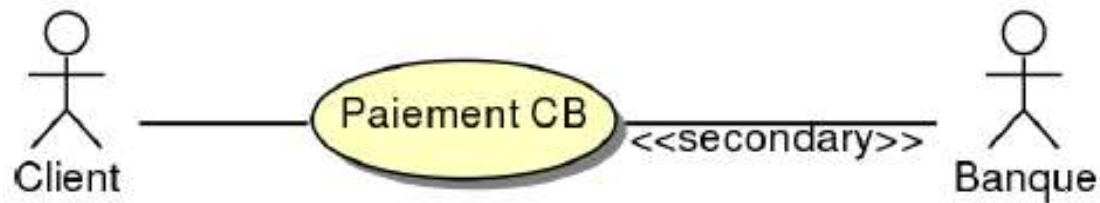
Identification des acteurs

- Les principaux acteurs sont les utilisateurs du système
 - une même personne physique peut être représentée par plusieurs acteurs si elle a plusieurs rôles.
 - si plusieurs personnes jouent le même rôle vis-à-vis du système, elles seront représentées par un seul acteur.
- En plus des utilisateurs, les acteurs peuvent être :
 - des périphériques manipulés par le système (imprimantes, ..);
 - des logiciels déjà disponibles à intégrer dans le projet;
 - des systèmes informatiques externes au système mais qui interagissent avec lui, etc.
- Pour faciliter la recherche des acteurs, on se fonde sur les frontières du système.

Attention : un acteur correspond à un rôle et non pas à une personne physique.

Acteurs principaux et secondaires

- L'acteur est dit principal pour un cas d'utilisation lorsque l'acteur est à l'initiative des échanges nécessaires pour réaliser le cas d'utilisation.



- Les acteurs secondaires sont sollicités par le système alors que le plus souvent, les acteurs principaux ont l'initiative des interactions.
 - le plus souvent, les acteurs secondaires sont d'autres systèmes informatiques avec lesquels le système développé est inter-connecté.

Recenser les cas d'utilisation

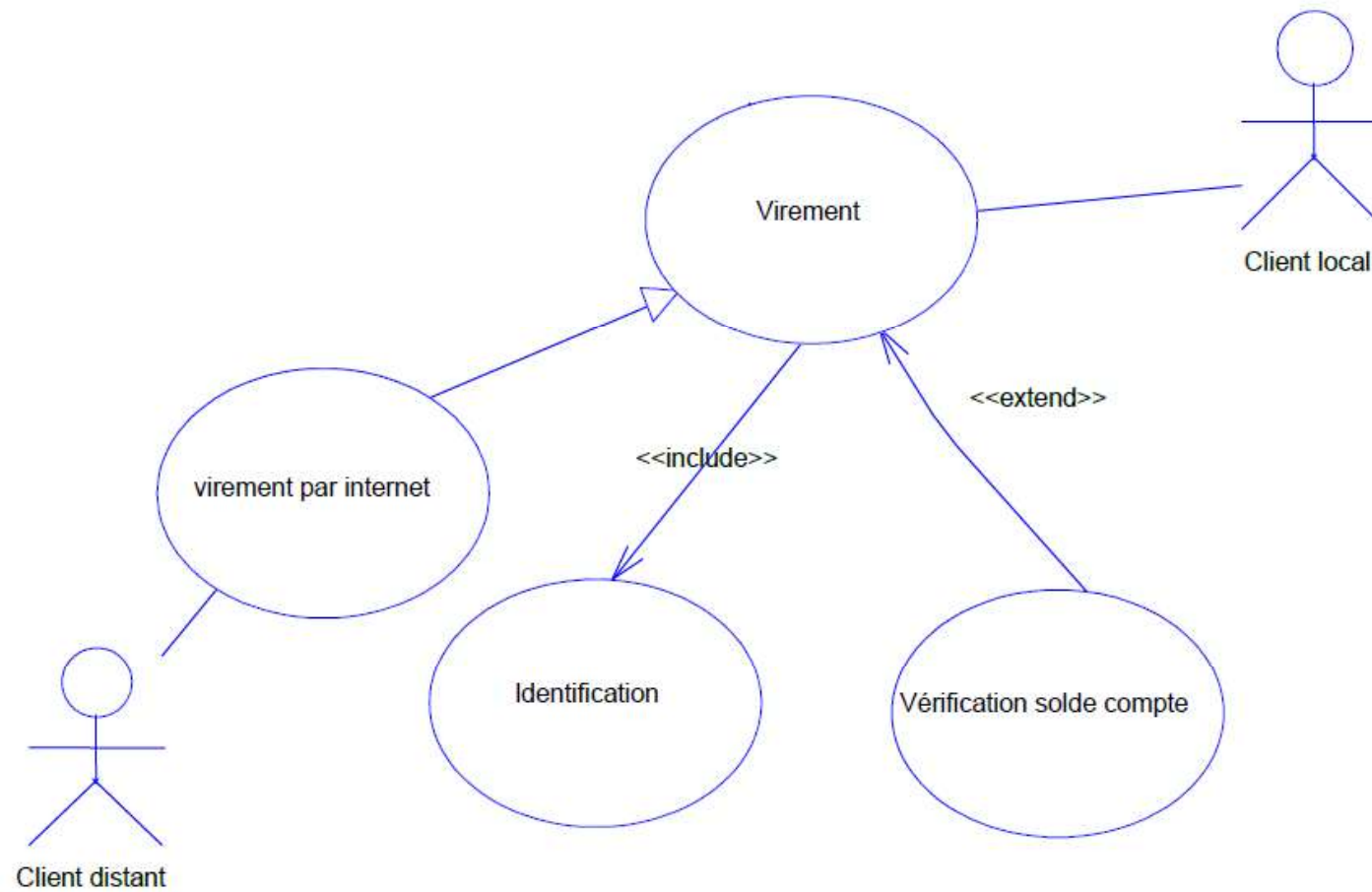
- Il n'y a pas une manière mécanique et totalement objective de repérer les cas d'utilisation.
 - il faut se placer du point de vue de chaque acteur et déterminer comment il se sert du système, dans quels cas il l'utilise, et à quelles fonctionnalités il doit avoir accès.
 - il faut éviter les redondances et limiter le nombre de cas en se situant au bon niveau d'abstraction (par exemple, ne pas réduire un cas à une seule action).
- Il ne faut pas faire apparaître les détails des cas d'utilisation, mais il faut rester au niveau des grandes fonctions du système.

Trouver le bon niveau de détail pour les cas d'utilisation est un problème difficile qui nécessite de l'expérience.

Description des cas d'utilisation

- Le diagramme de cas d'utilisation décrit les grandes fonctions d'un système du point de vue des acteurs, mais n'expose pas de façon détaillée le dialogue entre les acteurs et les cas d'utilisation.
- Un simple nom est tout à fait insuffisant pour décrire un cas d'utilisation.
- Chaque cas d'utilisation doit être documenté pour qu'il n'y ait aucune ambiguïté concernant son déroulement et ce qu'il recouvre précisément.

Exemple pratique de cas d'utilisation



EXO1 — Diagramme de cas d'utilisation UML : Identification des acteurs et de cas d'utilisation simple

On considère une borne de chargement de titre de transport Navigo. Les utilisateurs peuvent charger leur Navigo et les techniciens peuvent intervenir en cas de panne.

Question 1 : Pour charger son titre, le client dépose son titre et suit les instructions indiquées. Quel est l'acteur et l'action dans ce cas d'utilisation.

Question 2 : Jojo, dont le métier est technicien au sein de la RATP, veut charger le navigo de son fils. Pour modéliser cette activité de Jojo, doit-on définir un nouvel acteur ? Comment modélise-t-on ça ?

Question 3 : Lorsque Jojo vient avec ses outils pour réparer la borne en cas de panne, est-il considéré comme un nouvel acteur ? Comment modélise-t-on cela ?

Question 3 : Certains agent de la RATP qui ne sont pas des techniciens sont aussi qualifiés pour opérer des opérations de maintenance en plus des opérations habituelles des techniciens telles que le remplacement de certaines pièces et produits. Ils sont donc techniciens en plus d'être agents. Comment modéliser cela ?

EXO2 — Diagramme de cas d'utilisation UML :

Caisse enregistreuse

Le déroulement normal d'utilisation d'une caisse enregistreuse est le suivant :

1. Un client arrive à la caisse
2. Le caissier enregistre le numéro d'identification de chaque article, ainsi que la quantité si celle-ci est supérieur à 1.
3. La caisse affiche chaque article et son libellé
4. Lorsque tous les articles ont été enregistrés, le caissier signale la fin de la vente
5. La caisse affiche le total des achats
6. Le client choisit son mode de paiement
 - a. **Liquide** : le caissier encaisse l'argent et la caisse indique le montant éventuel à rendre au client
 - b. **Chèque** : le caissier note l'identité du client et la caisse enregistre le montant sur le chèque
 - c. **Carte de crédit** : le terminal bancaire transmet la demande à un centre d'autorisation multibanque
7. La caisse enregistre la vente et imprime un ticket
8. Le caissier transmet le ticket imprimé au client
9. Un client peut présenter un ticket de réduction avant le paiement. Lorsque le paiement est terminé, la caisse transmet les informations relatives aux articles vendus au système de gestion des stocks.
10. Tous les matins, le responsable du magasin initialise les caisses pour la journée.

Question : Donnez un diagramme de cas d'utilisation pour la caisse enregistreuse.

Installation d'outil de modélisation : StarUML

- Accéder au site web officiel de StarUML à l'adresse suivante :
<https://staruml.io/>
- Cliquer sur le bouton **download** pour télécharger
- Installer la version téléchargée. Il faut avoir les droits d'administration pour pouvoir l'installer.
- Un guide intéressant du logiciel est disponible en version pdf à l'adresse >
https://inf1410.teluq.ca/files/2020/11/INF1410_GuideStarUML-DiagrammeUML_VT.pdf