

# Plan

## 1 Introduction à la Modélisation Orientée Objet

## 2 **Modélisation objet élémentaire avec UML**

- Diagrammes de cas d'utilisation
- Diagrammes de classes
- Diagrammes d'objets
- Diagrammes de séquences

## 3 UML et méthodologie

## 4 Modélisation avancée avec UML

## 5 Bonnes pratiques de la modélisation objet

# 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**.



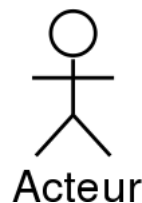
# Cas d'utilisation

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



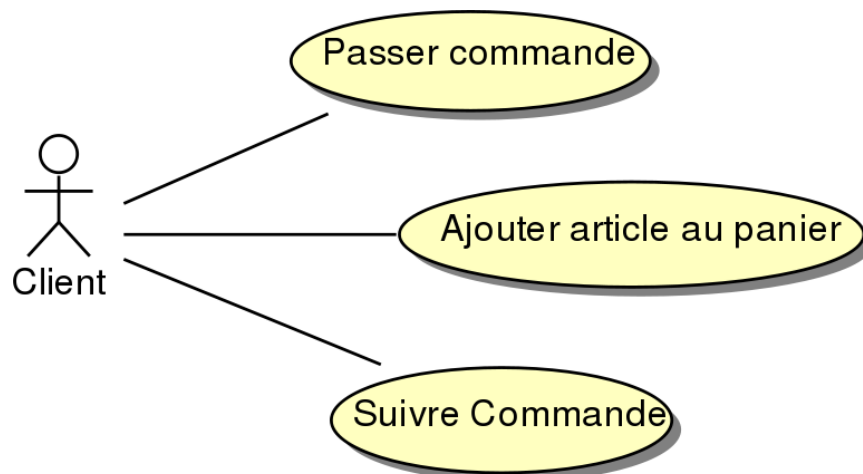
Cas d'utilisation

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



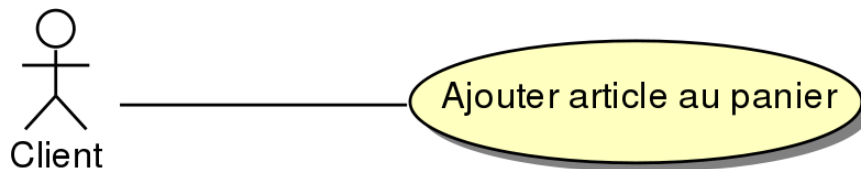
Un cas d'utilisation est l'expression d'un service réalisé de bout en bout, avec un déclenchement, un déroulement et une fin, pour l'acteur qui l'initie.

# Acteurs et cas d'utilisation



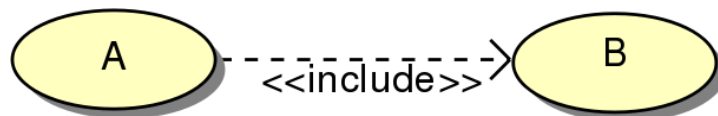
## Relations entre cas d'utilisation en acteurs

- 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

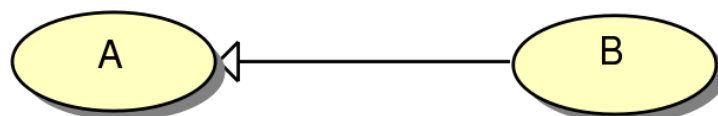
- **Inclusion** : le cas A inclut le cas B (B est une partie *obligatoire* de A).



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



- **Généralisation** : le cas A est une généralisation du cas du cas B (B est une sorte de A).



## 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.

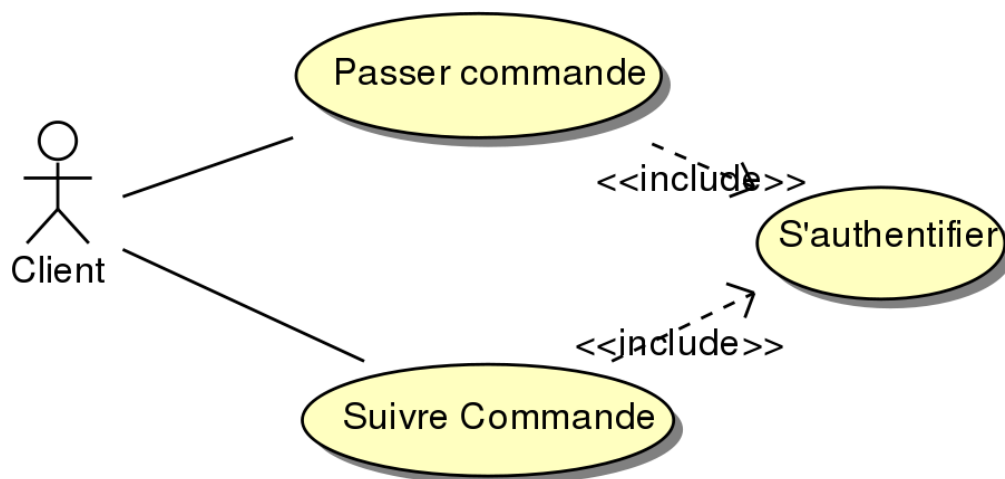
### Attention

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



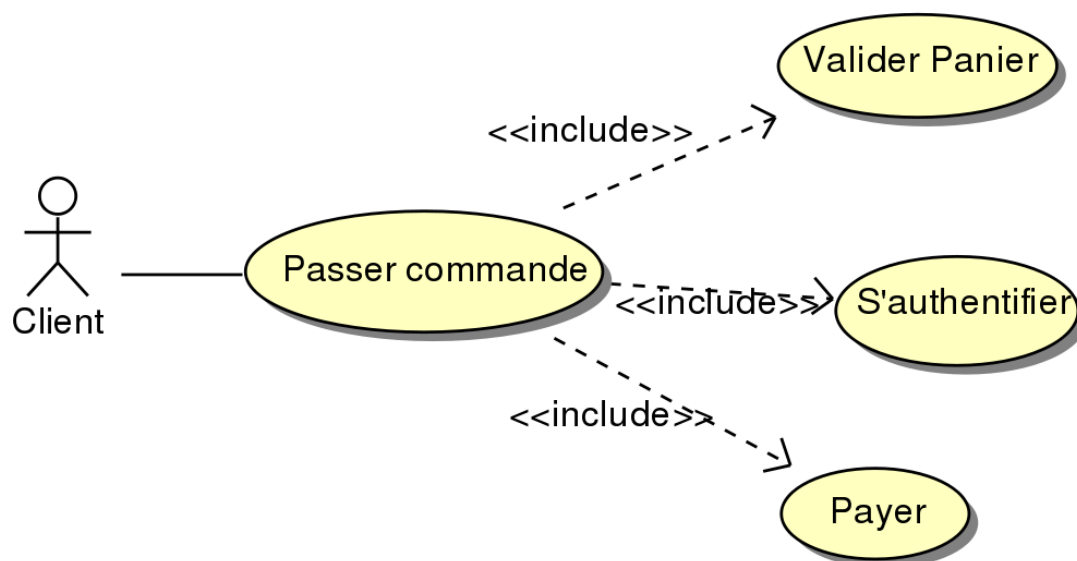
# Réutilisabilité avec les inclusions et les extensions

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

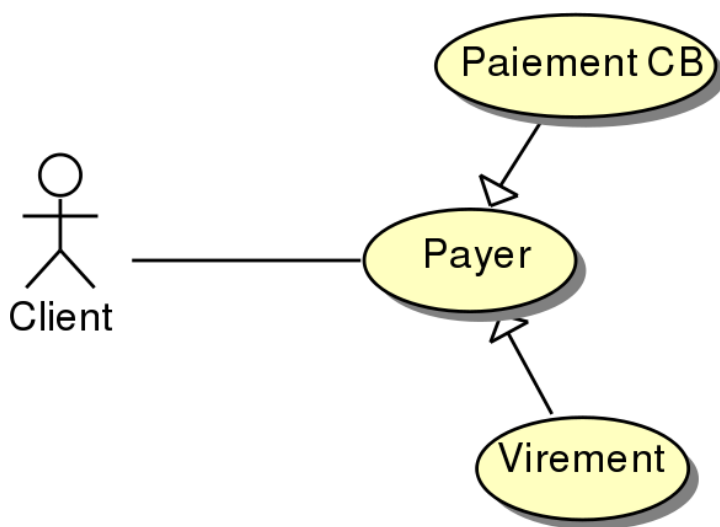


# Décomposition grâce aux inclusions et aux extensions

- 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



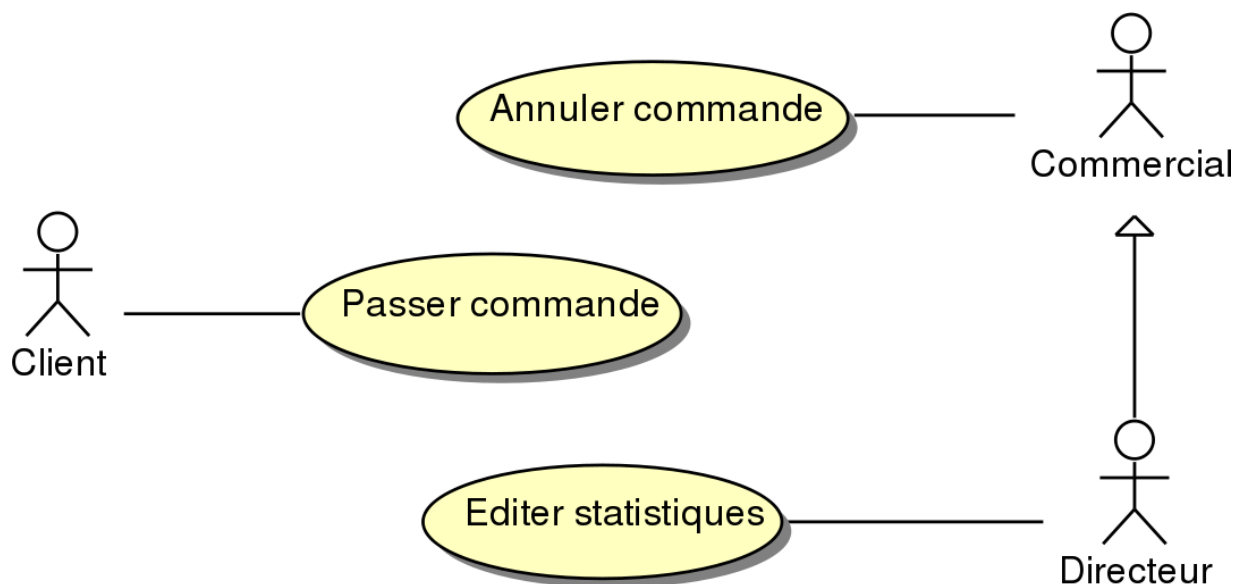
- Un virement est un cas particulier de paiement.

Un virement est **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.

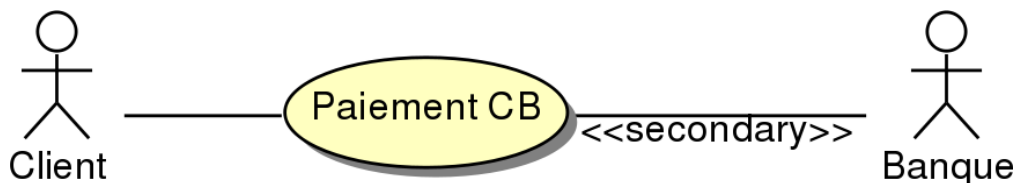
## Attention

Un acteur correspond à un **rôle**, pas à une personne physique.

- 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.

## 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.



# Description textuelle

- **Identification :**
  - **Nom du cas** : Payer CB
  - **Objectif** : Détailler les étapes permettant à client de payer par carte bancaire
  - **Acteurs** : Client, Banque (secondaire)
  - **Date** : 18/09
  - **Responsables** : Toto
  - **Version** : 1.0

# Description textuelle

- **Séquencements :**

- Le cas d'utilisation commence lorsqu'un client demande le paiement par carte bancaire

- **Pré-conditions**

- Le client a validé sa commande

- **Enchaînement nominal**

- ① Le client saisit les informations de sa carte bancaire
- ② Le système vérifie que le numéro de CB est correct
- ③ Le système vérifie la carte auprès du système bancaire
- ④ Le système demande au système bancaire de débiter le client
- ⑤ Le système notifie le client du bon déroulement de la transaction

- **Enchaînements alternatifs**

- ① En (2) : si le numéro est incorrect, le client est averti de l'erreur, et invité à recommencer
- ② En (3) : si les informations sont erronées, elles sont re-demandées au client

- **Post-conditions**

- La commande est validée
- Le compte de l'entreprise est crédité

# Description textuelle

- **Rubriques optionnelles**
  - **Contraintes non fonctionnelles :**
    - Fiabilité : les accès doivent être sécurisés
    - Confidentialité : les informations concernant le client ne doivent pas être divulgués
  - **Contraintes liées à l'interface homme-machine :**
    - Toujours demander la validation des opérations bancaires

# A vous de jouer

- **Élaborez le diagramme des cas d'utilisation d'une agence de voyage. Pour organiser un voyage, l'agent doit réserver au client une chambre d'hôtel, lui réserver un billet d'avion ou de train, lui réserver un taxi pour venir de l'aéroport ou de la gare et enfin, lui établir une facture. Certains clients peuvent demander une facture plus détaillée**