# Génie Logiciel 2

## La modélisation UML

# Présentation

Nicolas HOAREAU - MAST au DII
nicolas.hoareau@univ-amu.fr

# Use Case Diagram

# Use Case Diagram

- Capture le comportement d'un système, d'un sous-système, d'une classe ou d'un composant tel qu'un utilisateur extérieur le voit.
- Il scinde la fonctionnalité du système en unités cohérentes
- Les cas d'utilisation permettent d'exprimer le besoin des utilisateurs d'un système

# Use Case Diagram

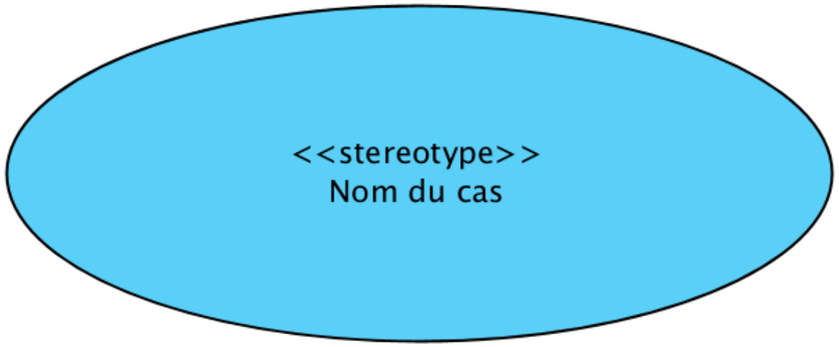
## L'acteur

# L'acteur

- L'idéalisation d'un rôle joué par une personne externe, un processus ou une chose qui interagit avec un système
- Il se représente par un petit bonhomme avec son nom (i.e. son rôle) inscrit dessous.

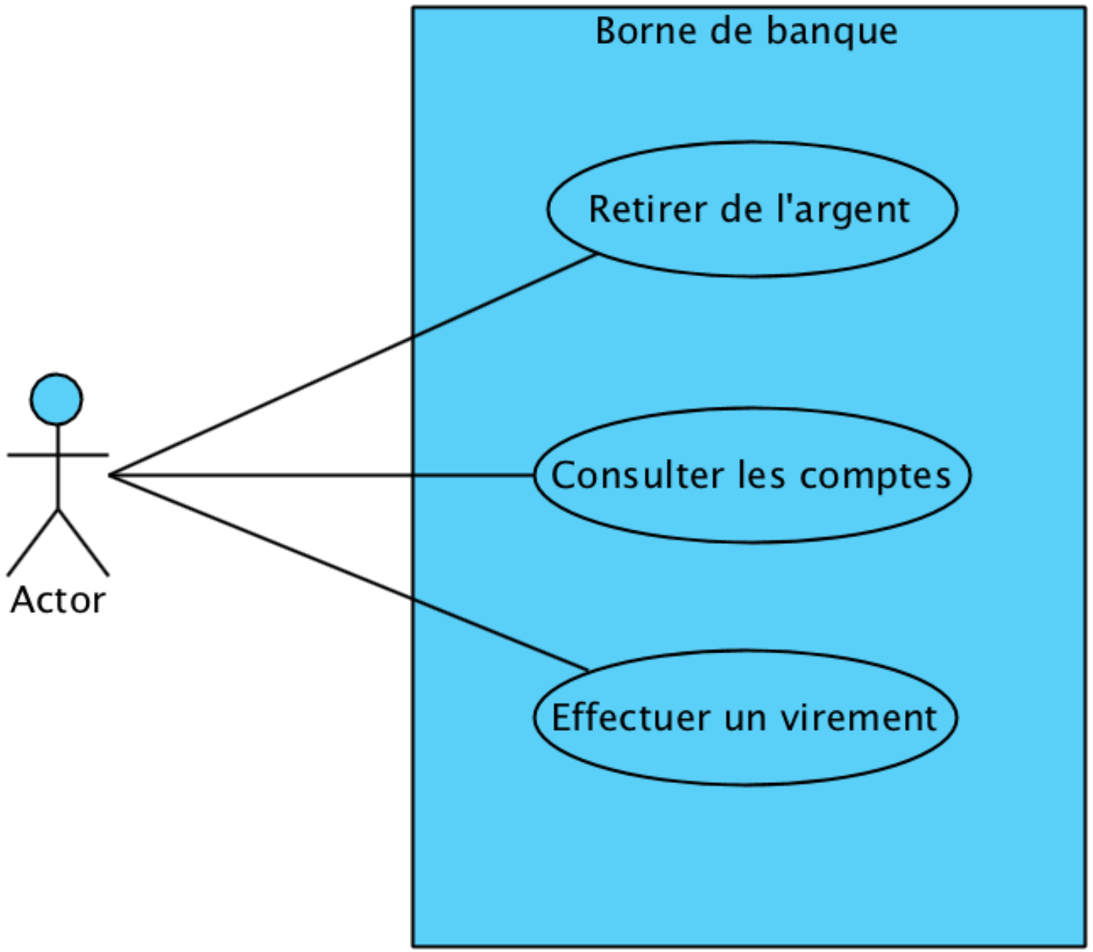
# Cas d'utilisation

- Un cas d'utilisation est une unité cohérente représentant une fonctionnalité visible de l'extérieur.
- Il réalise un service de bout en bout, avec un déclenchement, un déroulement et une fin, pour l'acteur qui l'initie.



# Système

Le nom du système figure à l'intérieur du cadre, en haut. Les acteurs sont à l'extérieur et les cas d'utilisation à l'intérieur.



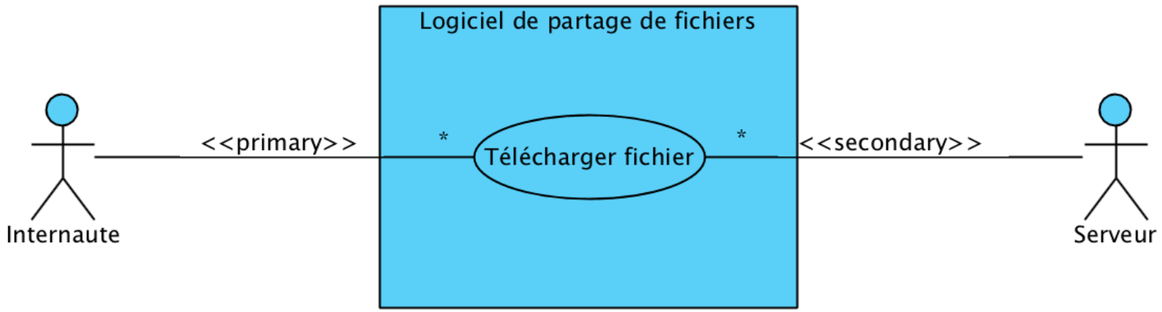
# Multiplicité

- Il est possible d'ajouter une multiplicité sur l'association du côté du cas d'utilisation. Le symbole * signifie plusieurs
- Exactement **n** s'écrit tout simplement **n**
- `n..m` signifie entre **n** et **m**

# Acteurs principaux et secondaires

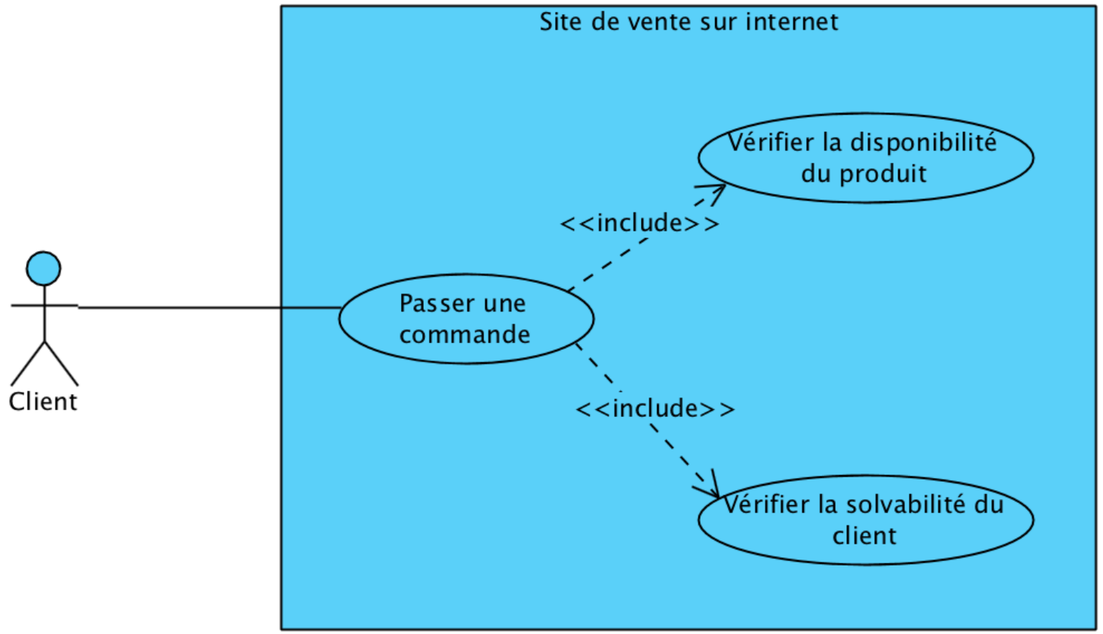
- Un acteur est qualifié de principal pour un cas d'utilisation lorsque ce cas rend service à cet acteur `<< primary >>`
- Un cas d'utilisation a au plus un acteur principal
- Les autres acteurs sont alors qualifiés de secondaires `<< secondary >>`
- Un acteur secondaire est sollicité pour des informations complémentaires.

# Représentation



# Relation d'inclusion

- Un cas A inclut un cas B si le comportement décrit par le cas A inclut le comportement du cas B : le cas A dépend de B
- Cette dépendance est symbolisée par le stéréotype `<< include >>`
- Une dépendance se représente par une flèche avec un trait pointillé



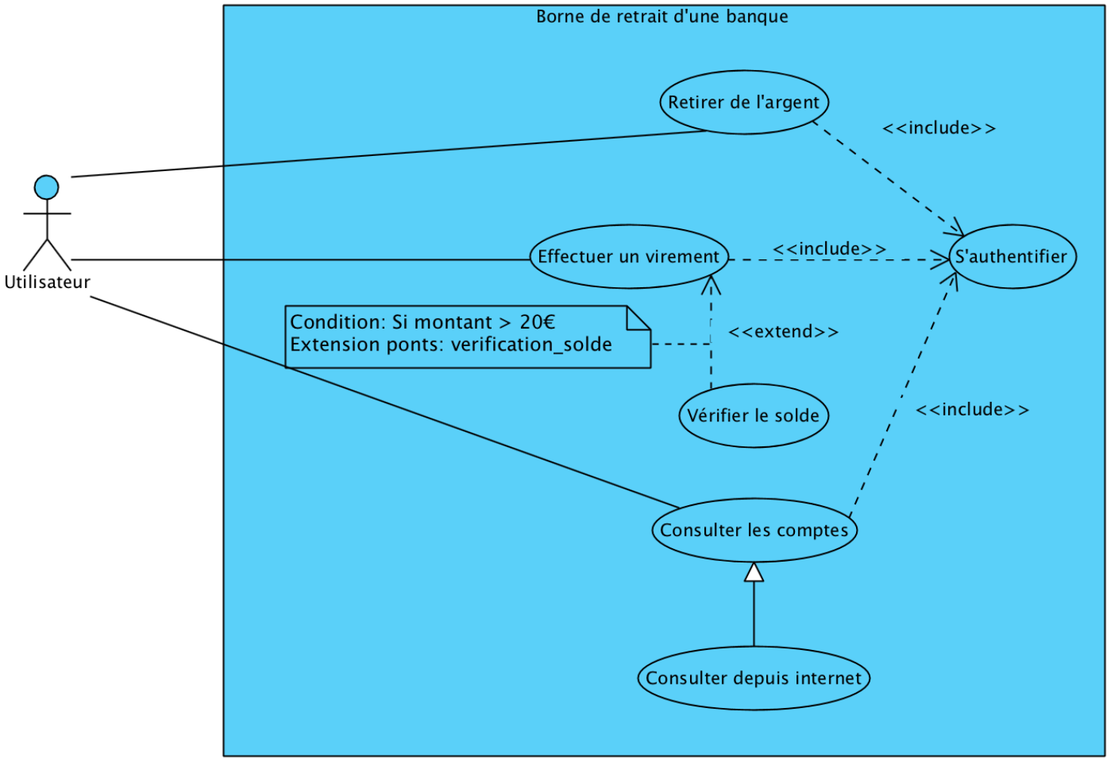
# Relation d'extension

- On dit qu'un cas d'utilisation A étend un cas d'utilisation B lorsque le cas d'utilisation A peut être appelé au cours de l'exécution du cas d'utilisation B.
- Contrairement à l'inclusion, l'extension est optionnelle
- Cette dépendance est symbolisée par le stéréotype << extend >>
- Une extension est souvent soumise à condition. La condition est exprimée sous la forme d'une note

# Relation de généralisation

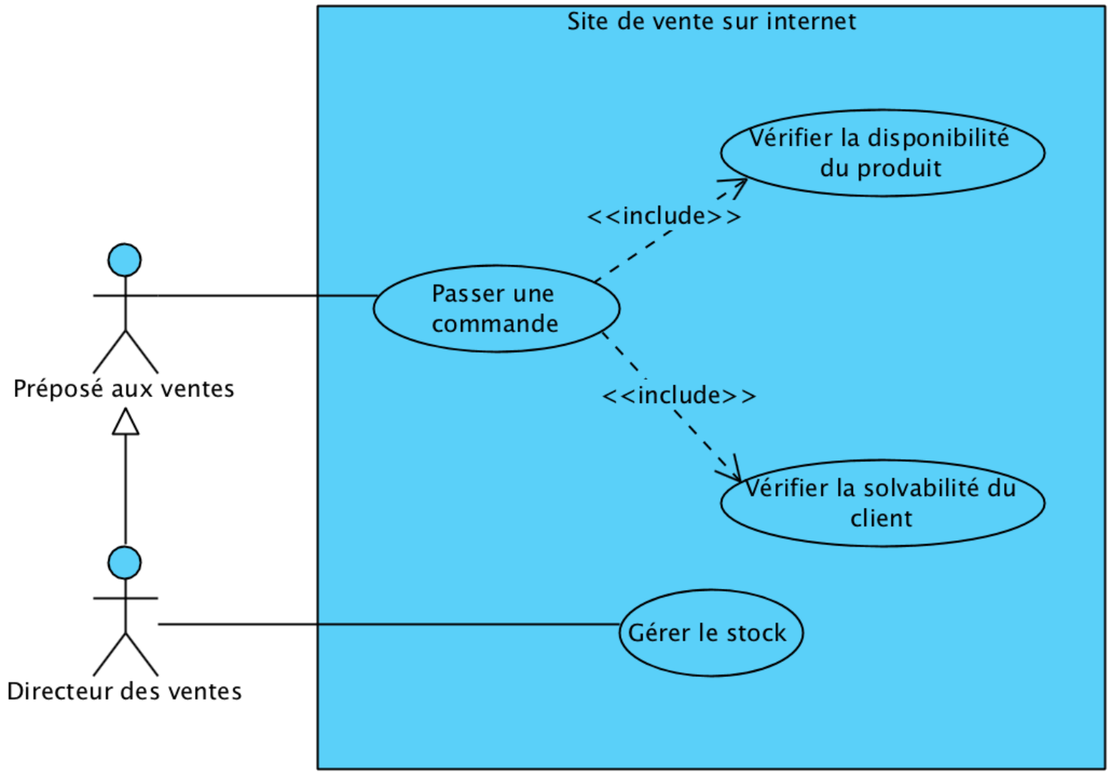
- Un cas A est une généralisation d'un cas B si B est un cas particulier de A
- Cette relation se traduit par le concept d'héritage dans les langages orientés objet.

# Représentation



# Relations entre acteurs

- La seule relation possible entre deux acteurs est la généralisation
- Un acteur A est une généralisation d'un acteur B si l'acteur A peut être substitué par l'acteur B.
- Dans ce cas, tous les cas d'utilisation accessibles à A le sont aussi à B, mais l'inverse n'est pas vrai.



# Class Diagram

# Class Diagram

- Le diagramme de classes est considéré comme le plus important de la modélisation orientée objet.
- Il est le seul obligatoire lors d'une telle modélisation.
- Le diagramme de classes montre la structure interne d'un système.
- Il permet de fournir une représentation abstraite des objets du système.

# Notions de classe et d'instance de classe

Une instance est une concrétisation d'un concept abstrait. Par exemple :

- La Ferrari Enzo qui se trouve dans votre garage est une instance du concept abstrait Automobile ;
- l'amitié qui lie Jean et Marie est une instance du concept abstrait Amitié ;

# Notions de classe et d'instance de classe

Une classe est un concept abstrait représentant des éléments :

- des éléments concrets (ex. : des avions),
- des éléments abstraits (ex. : des commandes).
- des composants d'une application (ex. : les boutons).
- des structures informatiques (ex. : des tables de hachage).
- des éléments comportementaux (ex. : des tâches).

# Caractéristiques d'une classe

- Une classe définit un jeu d'objets dotés de caractéristiques communes
- Les caractéristiques d'un objet permettent de spécifier son état et son comportement

# Caractéristiques d'une classe

## État d'un objet

- Ce sont les attributs et généralement les terminaisons d'associations
- Les attributs sont utilisés pour des valeurs de données pures
- Les associations sont utilisées pour connecter les classes
- Les propriétés ont valeurs quand la classe est instanciée

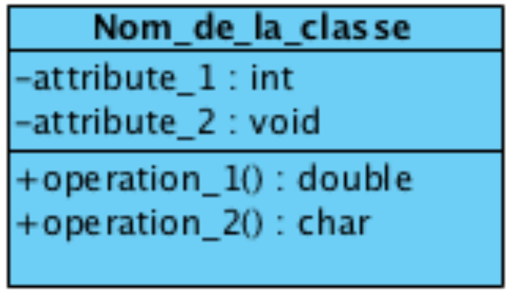
# Caractéristiques d'une classe

Comportement d'un objet :

- Les opérations décrivent les éléments individuels d'un comportement que l'on peut invoquer.
- Une opération est la spécification d'une méthode.
- L'implémentation d'une méthode est appelée méthode.
- Les attributs, les terminaisons et les méthodes constituent les caractéristiques d'une classe.

# Représentation

- Une classe est un classeur
- Elle est représentée par un rectangle divisé en trois à cinq compartiments
- Le premier indique le nom de la classe
- Le deuxième ses attributs
- Et le troisième ses opérations



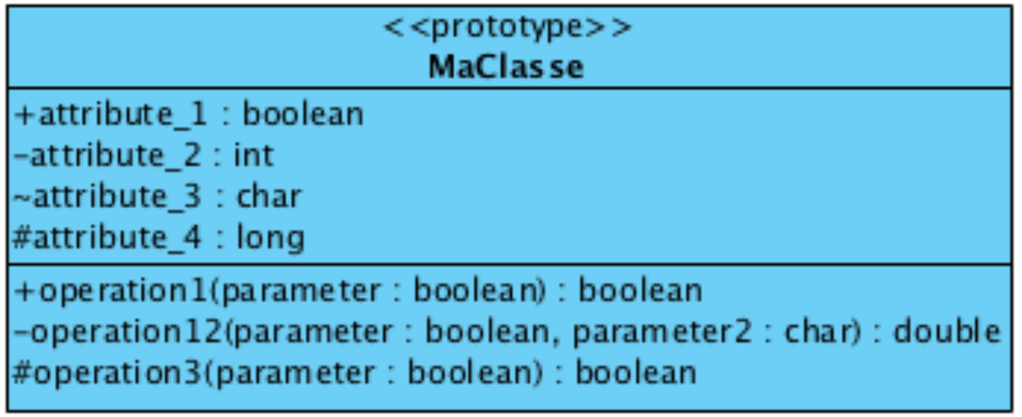
# Encapsulation, visibilité, interface

– privé

+ publique

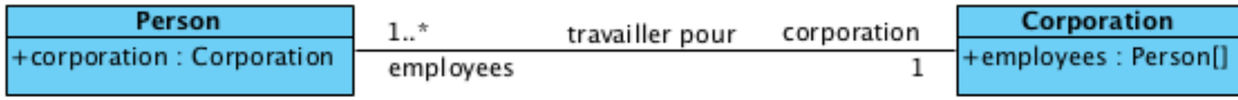
# protégé (accessible uniquement dans les sous-classes)

~ paquet (accessible uniquement dans le paquet)



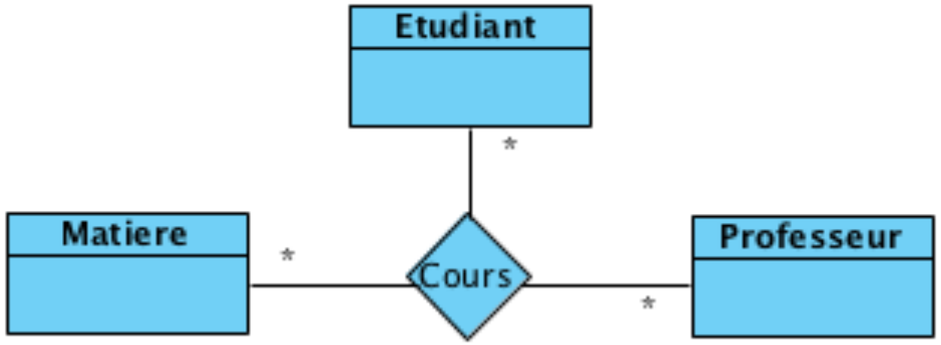
# Relation d'association

Association : lien entre les classes associées



# Relation d'association

Association n-aire



# Terminaison d'association

Une terminaison d'association est une propriété

- Nom
- Visibilité
- Multiplicité
- Navigabilité

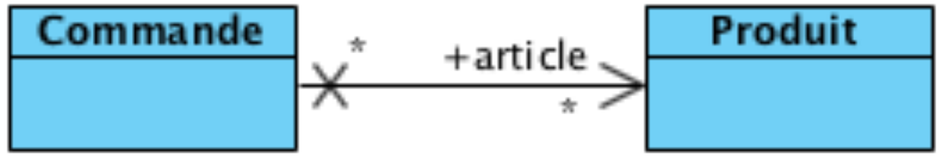
# Multiplicité

La multiplicité associée à une terminaison déclare le nombre d'objets susceptibles d'occuper la position définie par la terminaison d'association.

- exactement un : 1 ou 1..1 ;
- plusieurs : * ou 0..* ;
- au moins un : 1..* ;
- de un à six : 1..6.

# Navigabilité

La navigabilité indique s'il est possible de traverser une association. On représente graphiquement la navigabilité par une flèche du côté de la terminaison navigable et on empêche la navigabilité par une croix du côté de la terminaison non navigable. Par défaut, une association est navigable dans les deux sens.

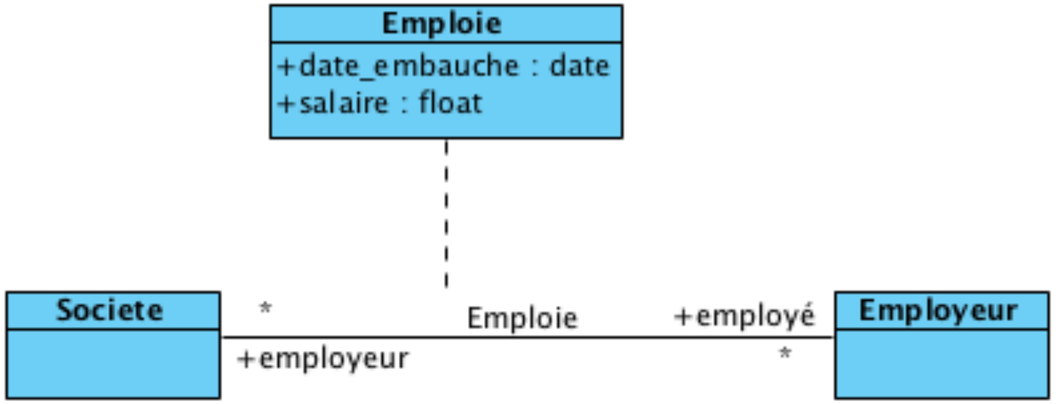


# Classe-association

- Parfois, une association doit posséder des propriétés
- Une classe-association possède les caractéristiques des associations et des classes : elle se connecte à deux ou plusieurs classes et possède également des attributs et des opérations.
- Une classe-association est caractérisée par un trait discontinu entre la classe et l'association qu'elle représente.

# Représentation

Classe-association



# Agrégation et composition

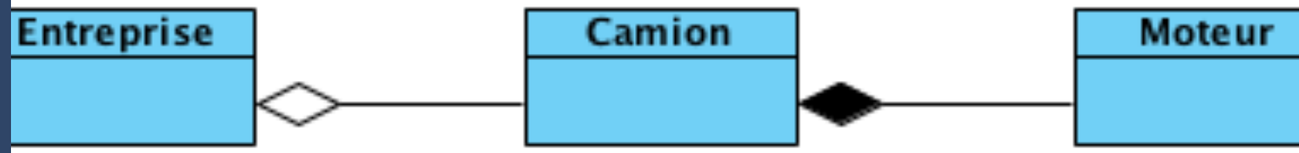
- Une agrégation est une association qui représente une relation d'inclusion structurelle ou comportementale d'un élément dans un ensemble.
- Graphiquement, on ajoute un losange vide du côté de l'agrégat

# Agrégation et composition

- La composition, également appelée agrégation composite, décrit une contenance structurelle entre instances.
- Ainsi, la destruction de l'objet composite implique la destruction de ses composants.
- la multiplicité du côté composite ne doit pas être supérieure à 1 (i.e. 1 ou 0..1).
- On ajoute un losange plein à côté de l'agrégat

# Agrégation et composition

Représentation



# Généralisation et Héritage

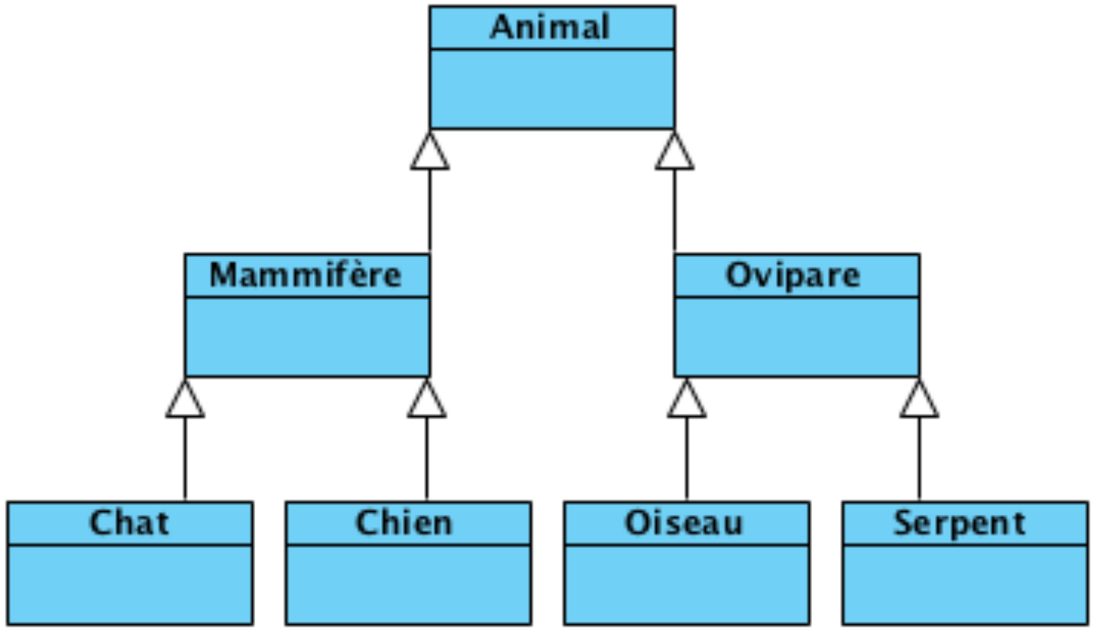
- La classe enfant possède toutes les caractéristiques de ses parents
- Une classe enfant peut redéfinir une ou plusieurs méthodes de la classe parent.
- Toutes les associations de la classe parent s'appliquent aux classes dérivées

# Généralisation et Héritage

- Une instance d'une classe peut être utilisée partout où une instance de sa classe parent est attendue.
- une classe peut avoir plusieurs parents, on parle alors d'héritage multiple

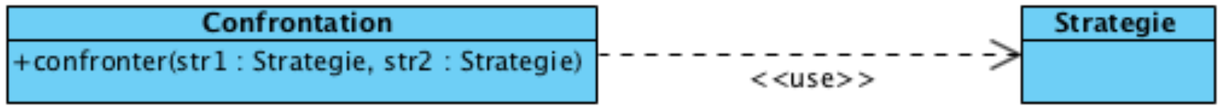
# Généralisation et Héritage

Représentation



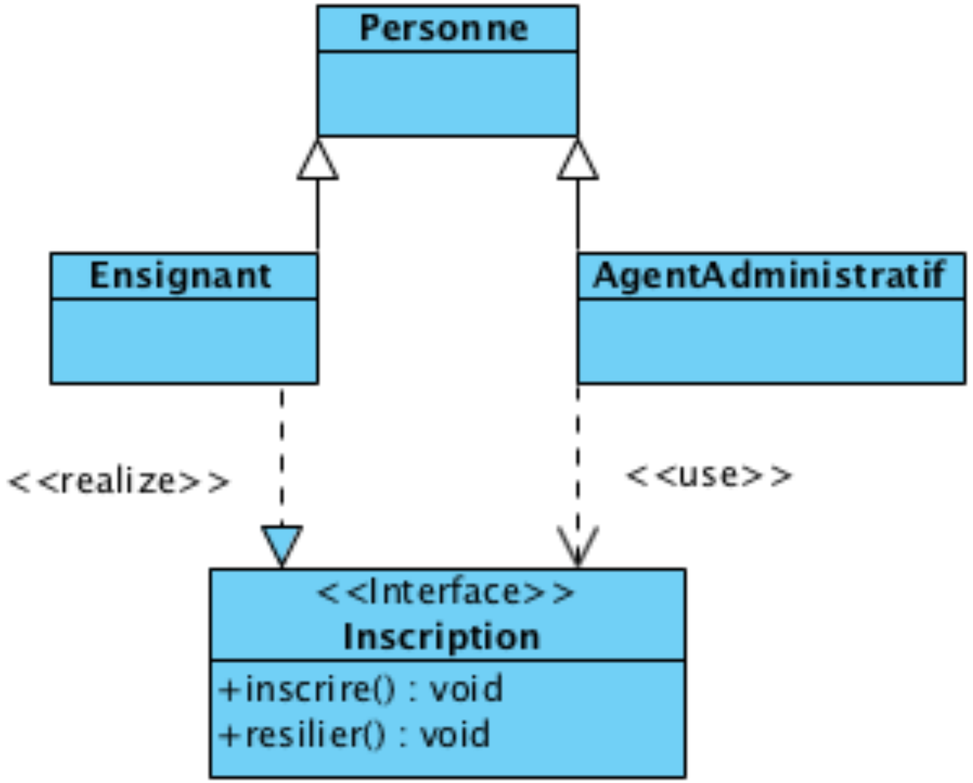
# Dépendance

- Une dépendance est une relation unidirectionnelle exprimant une dépendance sémantique entre des éléments du modèle.
- Elle est représentée par un trait discontinu orienté
- Elle indique que la modification de la cible peut impliquer une modification de la source.



# Interface

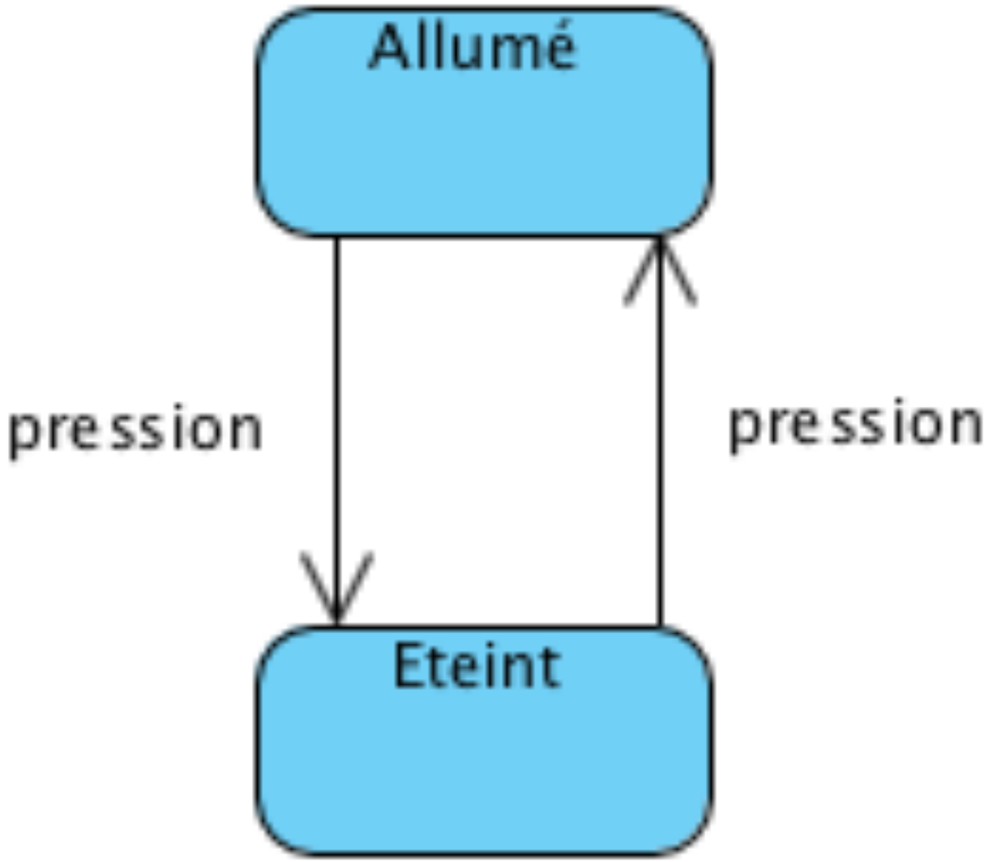
- Le rôle de ce classeur, stéréotypé << interface >>, est de regrouper un ensemble de propriétés et d'opérations assurant un service cohérent.
- L'objectif est de diminuer le couplage entre deux classeurs.



# State Machine Diagram

# State Machine Diagram

- Modélise les états possibles du système et les transitions entre ces états
- Généralement pour une seule classe
- Suffisamment précis pour générer du code
- Facile à interpréter et implémenter

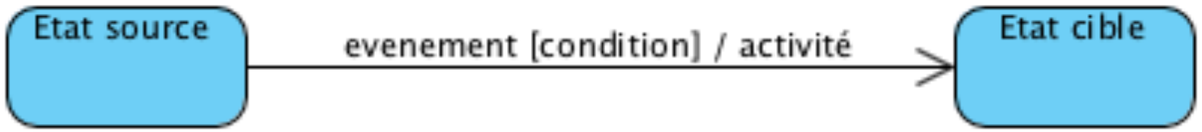


# État initial et final

- L'état initial est un pseudoétat qui indique l'état de départ, par défaut, lorsque le diagramme d'états-transitions, ou l'état enveloppant, est invoqué. Lorsqu'un objet est créé, il entre dans l'état initial.
- L'état final est un pseudoétat qui indique que le diagramme d'états-transitions, ou l'état enveloppant, est terminé.

# Transition externe

- Une transition externe est une transition qui modifie l'état actif.
- Il s'agit du type de transition le plus répandu. Elle est représentée par une flèche allant de l'état source vers l'état cible.

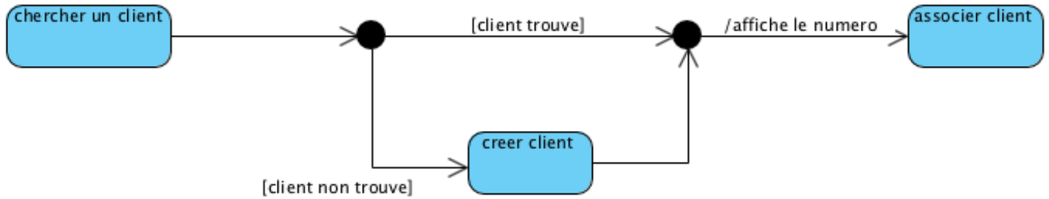


# Point de choix

- Il est possible de représenter des alternatives pour le franchissement d'une transition.
- Les points de jonction (représentés par un petit cercle plein)
- Les points de décision (représentés par un losange).

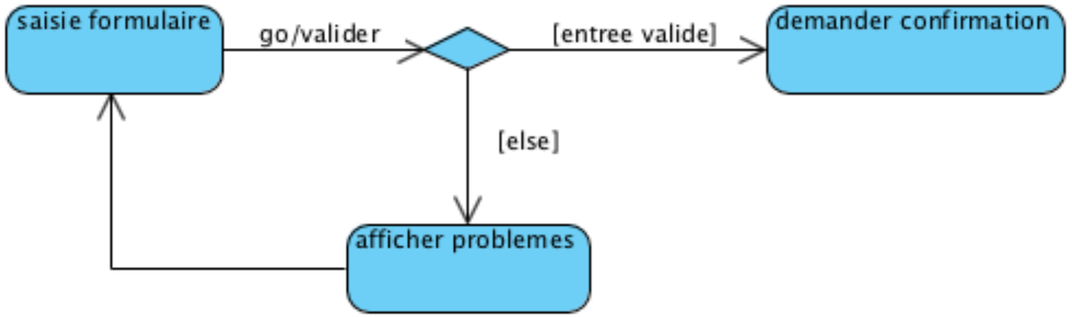
# Point de jonction

- L'objectif étant d'aboutir à une notation plus compacte ou plus lisible des chemins alternatifs.
- Un point de jonction peut avoir plusieurs segments de transition entrante et plusieurs segments de transition sortante



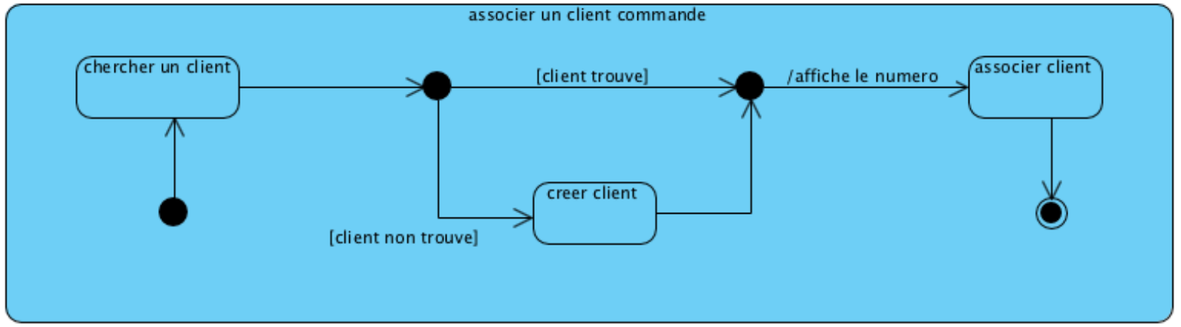
# Point de décision

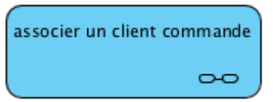
- Un point de décision possède une entrée et au moins deux sorties.
- Contrairement à un point de jonction, les gardes situées après le point de décision sont évaluées au moment où il est atteint.
- Il est possible d'utiliser une garde particulière, notée `[else]`



# Etat composite

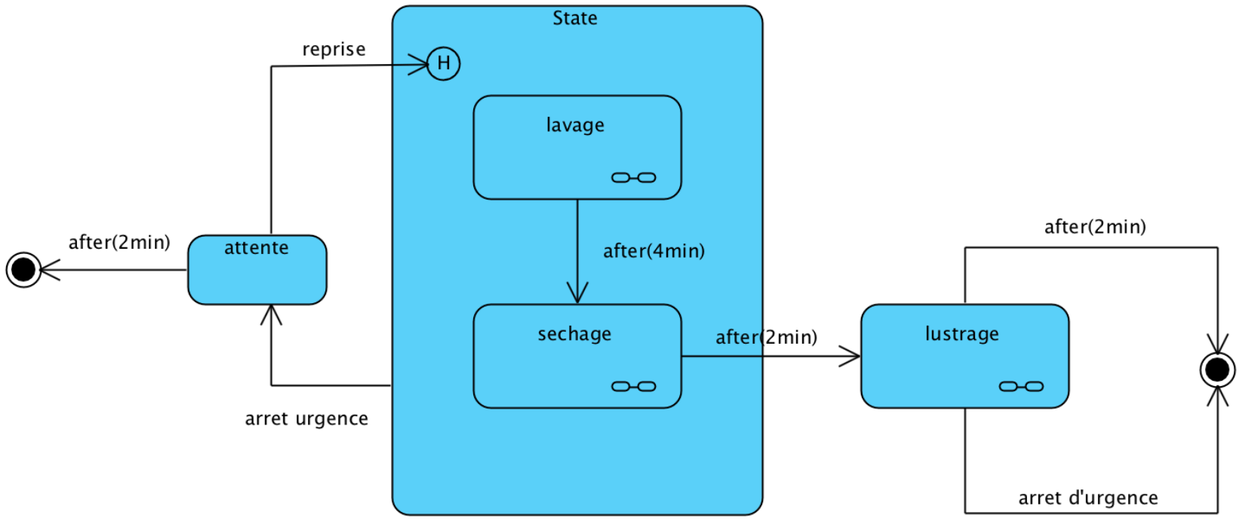
- Un état composite est un état décomposé en régions contenant chacune un ou plusieurs sous-états.



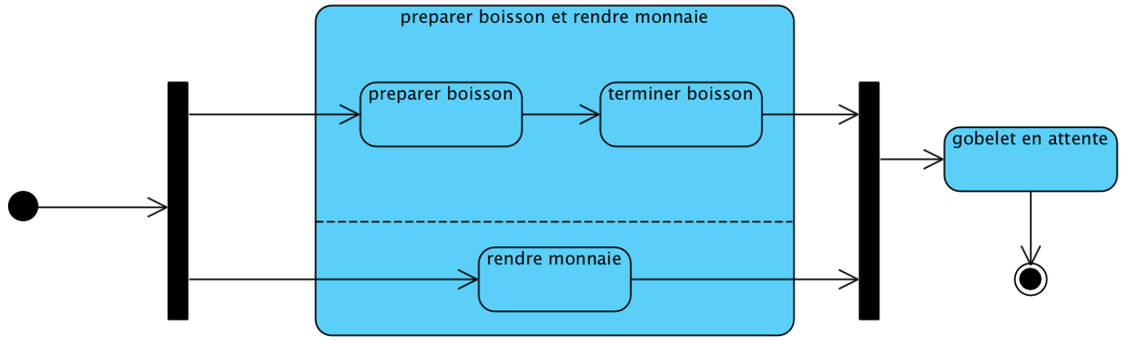
# Etat Historique

- Un état historique, également qualifié d'état historique plat, est un pseudoétat qui mémorise le dernier sous-état actif d'un état composite.
- Graphiquement, il est représenté par un cercle contenant un H.



# Concurrence

- Permet de décrire efficacement les mécanismes concurrents
- Chaque région représentant un flot d'exécution.
- Les différentes régions sont séparées par un trait horizontal en pointillé
- Les transitions complexes sont représentées par une barre épaisse.



# Activity Diagram

# Activity Diagram

- Les diagrammes d'activités permettent de mettre l'accent sur les traitements.
- La modélisation du cheminement de flots de contrôle et de flots de données.
- Représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation.
- Proches des diagrammes d'états-transitions

# Action

- Une action est le plus petit traitement
- Une action a une incidence sur l'état du système ou en extrait une information.
- La notion d'action est à rapprocher de la notion d'instruction élémentaire d'un langage de programmation

# Action

## call operation

L'action call operation correspond à l'invocation d'une opération sur un objet de manière synchrone ou asynchrone.

# Action

## call behavior

L'action call behavior est une variante de l'action call operation car elle invoque directement une activité plutôt qu'une opération.

# Action

## send

Cette action crée un message et le transmet à un objet cible, où elle peut déclencher un comportement. Il s'agit d'un appel asynchrone (i.e. qui ne bloque pas l'objet appelant) bien adapté à l'envoi de signaux (send signal).

# Action

## accept event

L'exécution de cette action bloque l'exécution en cours jusqu'à la réception du type d'événement spécifié, qui généralement est un signal. Cette action est utilisée pour la réception de signaux asynchrones.

# Action

## accept call

Il s'agit d'une variante de l'action accept event pour les appels synchrones.

# Action

## reply

Cette action permet de transmettre un message en réponse à la réception d'une action de type accept call.

# Action

## create

Cette action permet d'instancier un objet.

# Action

## destroy

Cette action permet de détruire un objet.

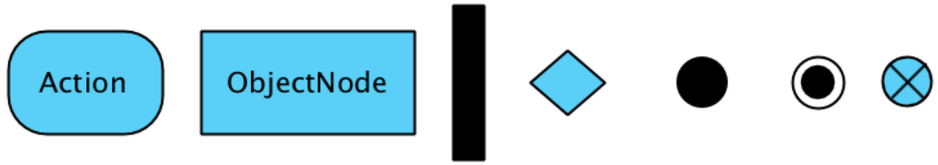
# Action

## raise exception

Cette action permet de lever explicitement une exception.

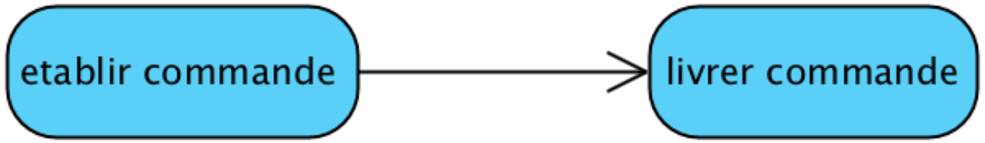
# Activity diagram

- Le nœud représentant une action
- un nœud objet
- un nœud de décision ou de fusion
- un nœud de bifurcation ou d'union
- un nœud initial
- un nœud final
- nœud final de flot.



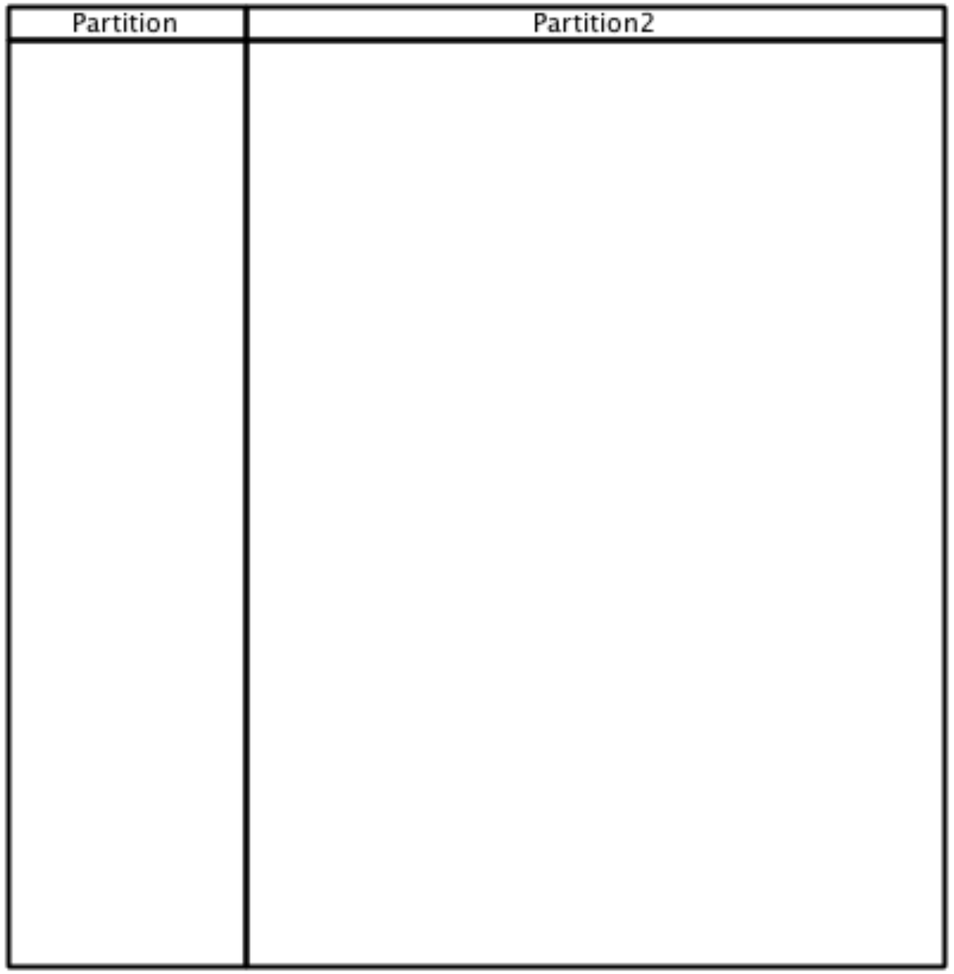
# Transition

- Le passage d'une activité vers une autre est matérialisé par une transition.
- Graphiquement les transitions sont représentées par des flèches en traits pleins



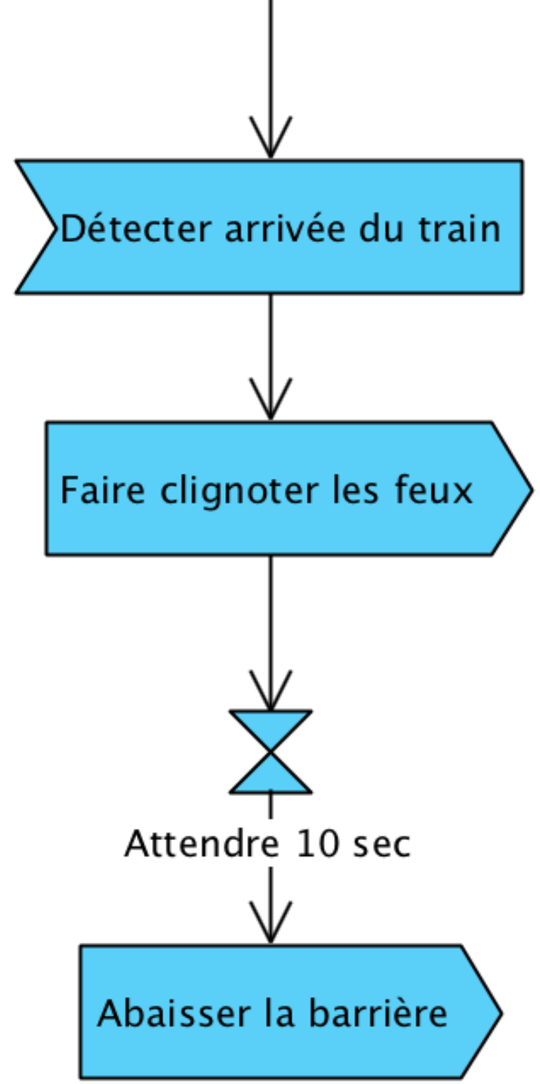
# Les partitions

- Les partitions, souvent appelées swimlane
- Permettent d'organiser les nœuds d'activités dans un diagramme d'activités en opérant des regroupements
- Correspondent souvent à des unités d'organisation du modèle.



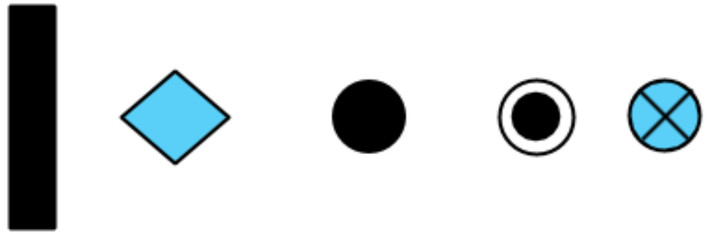
# Nœud d'action

- Un nœud d'action est un nœud d'activité exécutable
- Les actions sont généralement liées à des opérations qui sont directement invoquées
- Certaines actions de communication ont une notation spéciale



# Nœud de contrôle

Un nœud de contrôle est un nœud d'activité abstrait utilisé pour coordonner les flots entre les nœuds d'une activité.



# Nœud de contrôle

## Nœud initial

Un nœud initial est un nœud de contrôle à partir duquel le flot débute lorsque l'activité enveloppante est invoquée. Une activité peut avoir plusieurs nœuds initiaux. Un nœud initial possède un arc sortant et pas d'arc entrant.

# Nœud de contrôle

## Nœud de fin d'activité

Lorsque l'un des arcs d'un nœud de fin d'activité est activé
l'exécution de l'activité enveloppante s'achève

# Nœud de contrôle

## Nœud de fin de flot

Lorsqu'un flot d'exécution atteint un nœud de fin de flot, le flot en question est terminé, mais cette fin de flot n'a aucune incidence sur les autres flots actifs de l'activité enveloppante.

# Nœud de contrôle

## Nœud de décision

Un nœud de décision est un nœud de contrôle qui permet de faire un choix entre plusieurs flots sortants. Il possède un arc entrant et plusieurs arcs sortants. Ces derniers sont généralement accompagnés de conditions de garde pour conditionner le choix.

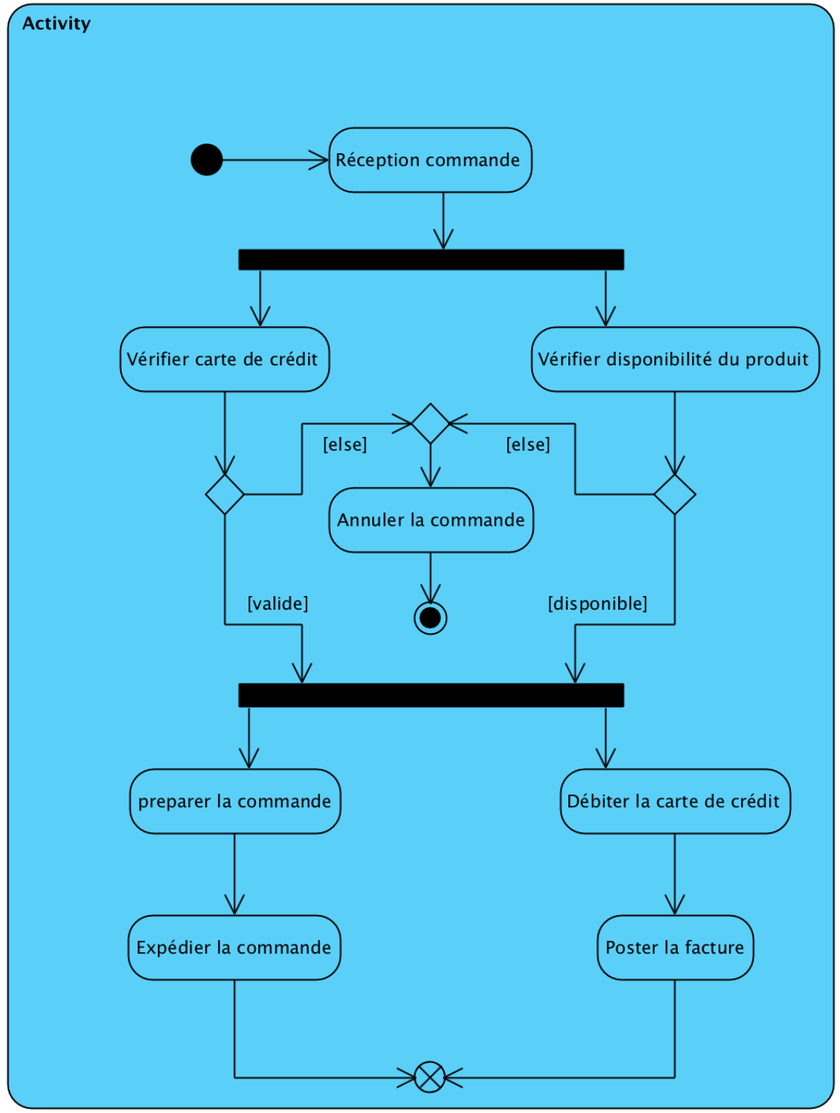
# Nœud de contrôle

## Nœud de bifurcation

Un nœud de bifurcation, également appelé nœud de débranchement est un nœud de contrôle qui sépare un flot en plusieurs flots concurrents.

# Activity diagram

## Représentation



# Sequence Diagram

# Sequence Diagram

- Les principales informations contenues sont les messages échangés entre les lignes de vie, présentés dans un ordre chronologique.
- Ainsi, contrairement au diagramme de communication, le temps y est représenté explicitement par une dimension (la dimension verticale) et s'écoule de haut en bas

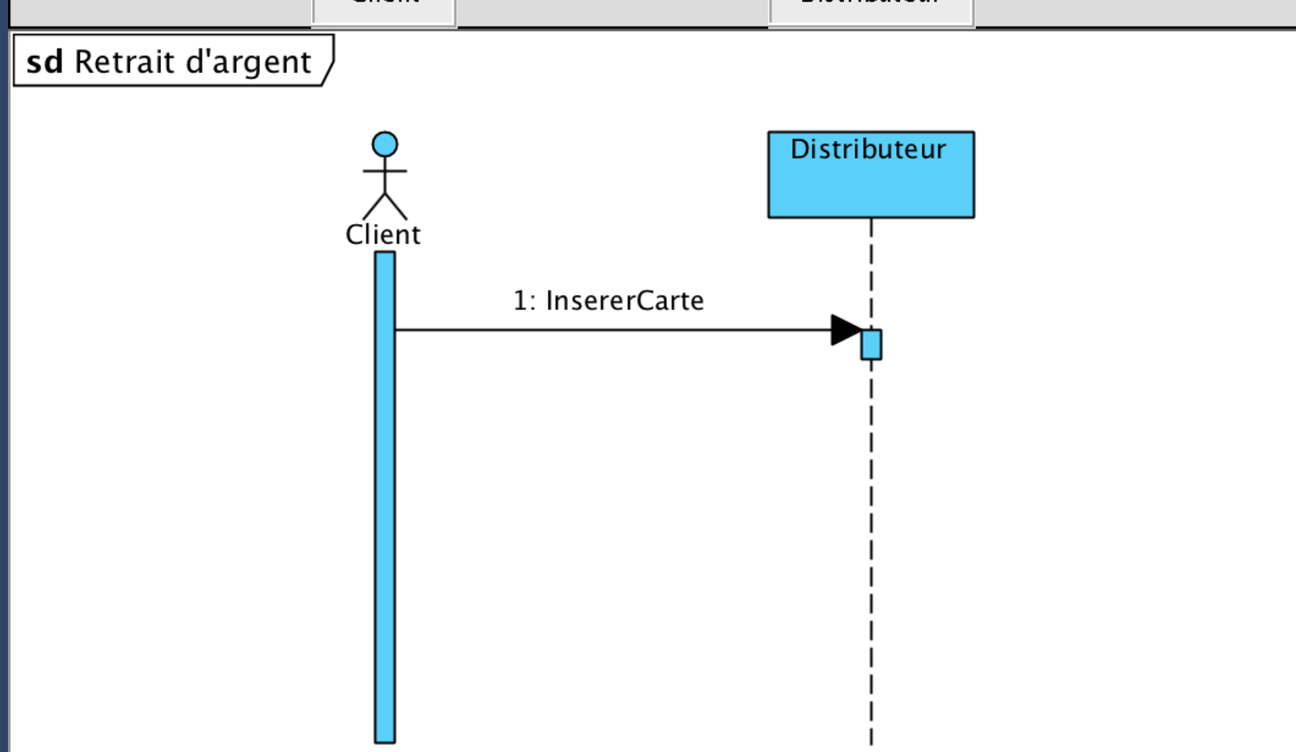
# Ligne de vie

- Une ligne de vie représente un participant à une interaction (objet ou acteur).
- Une ligne de vie se représente par un rectangle, auquel est accroché une ligne verticale pointillée

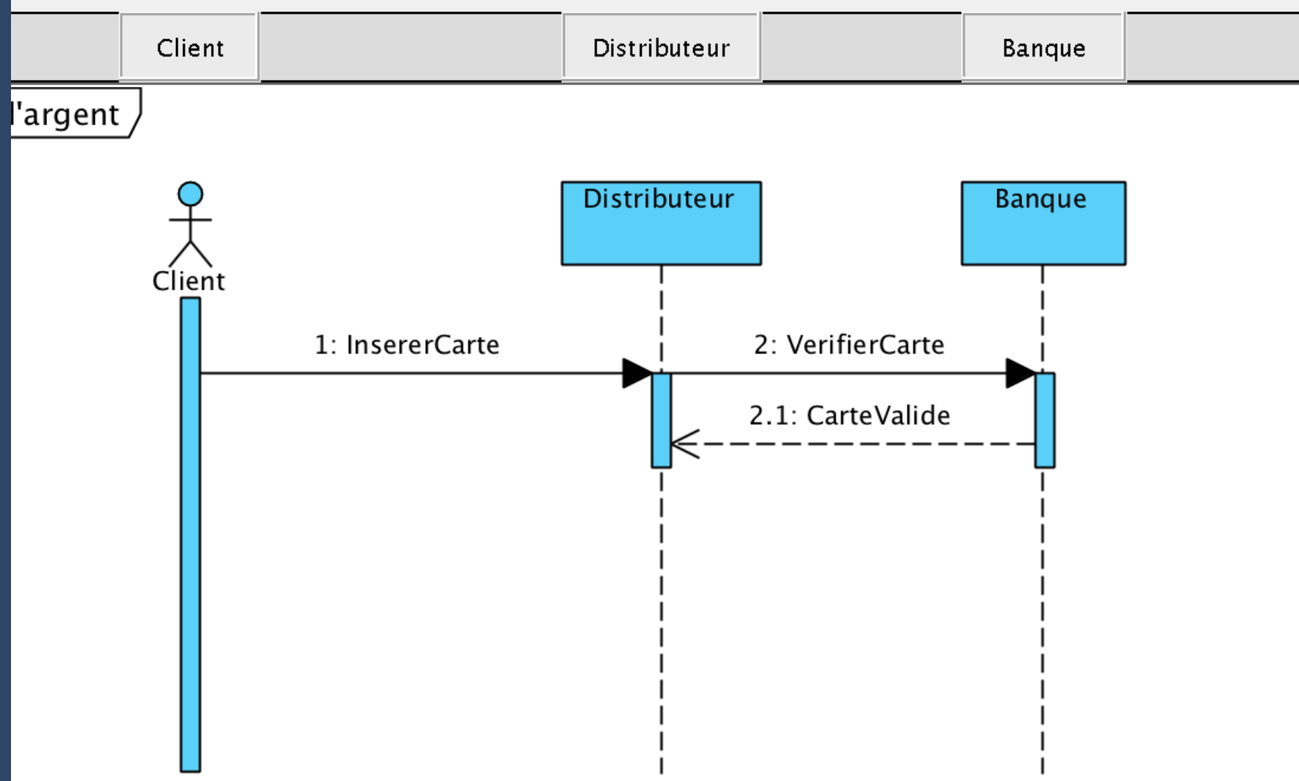
# Messages

Un message définit une communication particulière entre des lignes de vie. Plusieurs types de messages existent :

- l'envoi d'un signal
- l'invocation d'une opération
- la création ou la destruction d'une instance

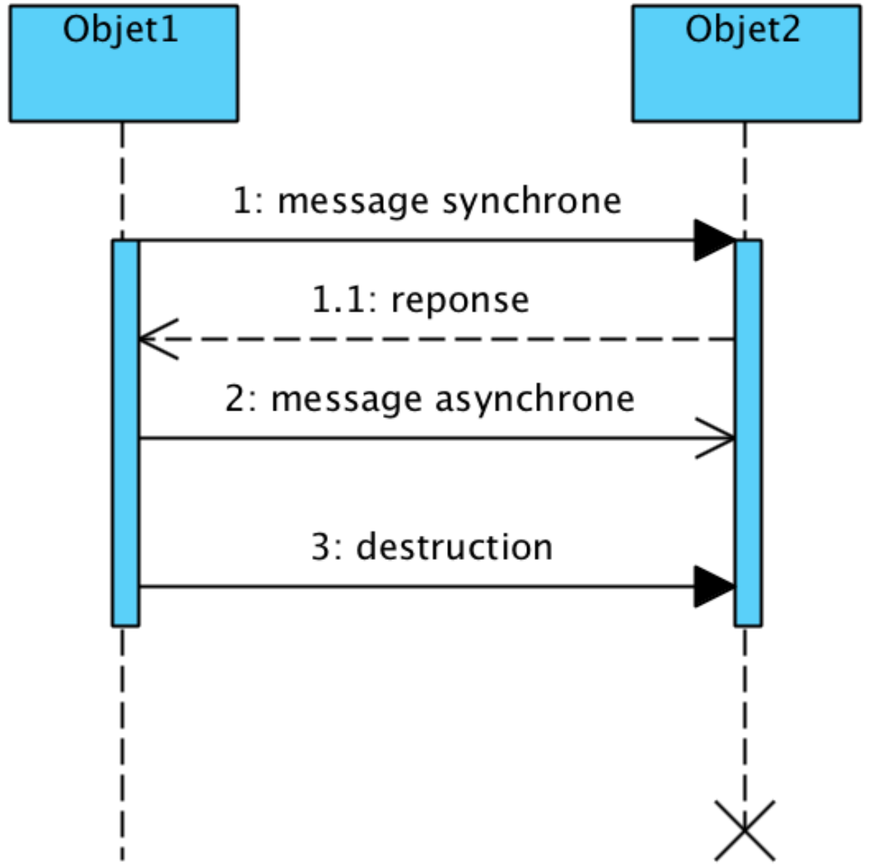


# Réponses


Client
Distributeur
Banque
l'argent
Client
Distributeur
Banque
1: InsererCarte
2: VerifierCarte
2.1: CarteValide

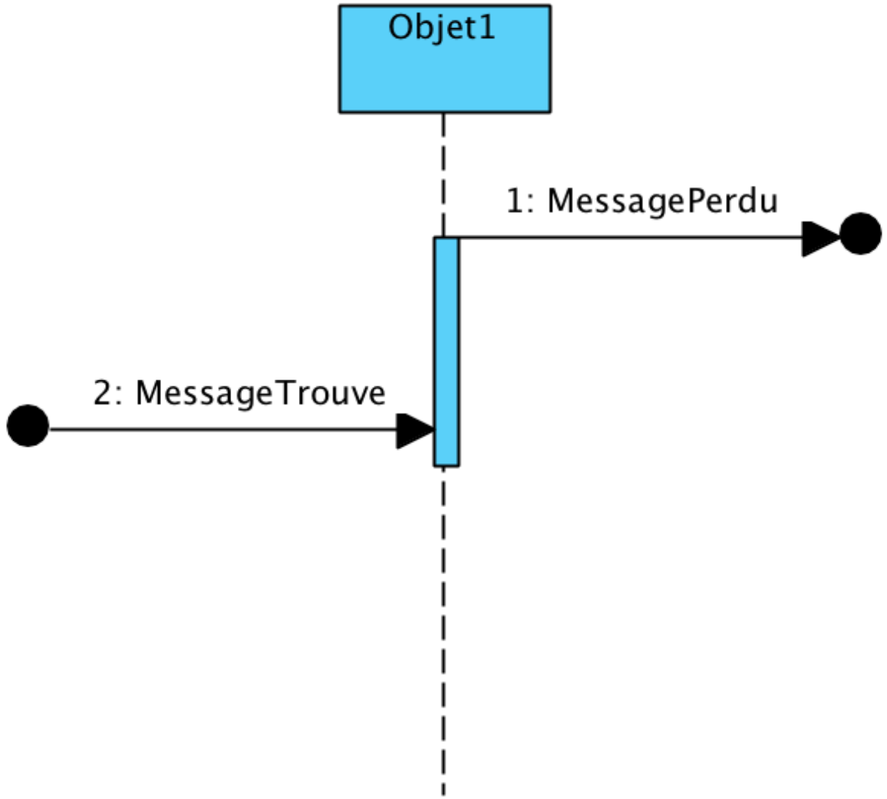
# Destruction

- La destruction d'un objet est matérialisée par une croix qui marque la fin de la ligne de vie de l'objet.
- La destruction d'un objet n'est pas nécessairement consécutive à la réception d'un message.



# Message perdu et trouvé

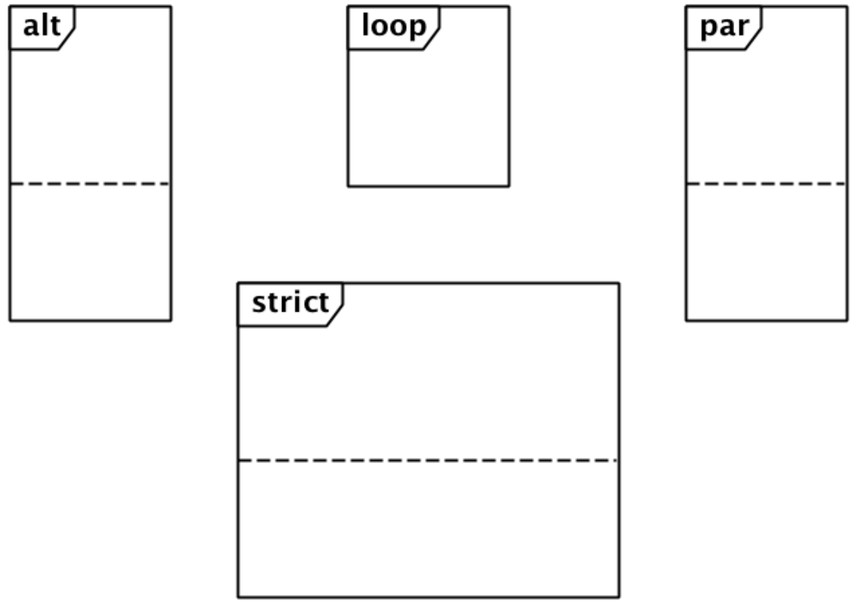
- Un message perdu est tel que l'événement d'envoi est connu, mais pas l'événement de réception.
- Un message trouvé est tel que l'événement de réception est connu, mais pas l'événement d'émission.



# Opérateurs

- les opérateurs de choix et de boucle : alternative, option, break et loop
- les opérateurs contrôlant l'envoi en parallèle de messages : parallel et critical region
- les opérateurs contrôlant l'envoi de messages : ignore, consider, assertion et negative
- les opérateurs fixant l'ordre d'envoi des messages : weak

# Opérateurs



# Création d'objet

- Un message peut être à l'origine de la création d'un objet
- Cela se représente par un flèche pointillée vers l'objet créé.