



Introduction à l'Orienté Objet :

Présentation du paradigme Objet

Intérêt de l'orienté objet

Etapas création d'applications OObjet

La Modélisation Object avec UML:

Notion de modélisation

Difficultés de la modélisation

Apport des méthodes de conception

Le Langage de modélisation UML

Modélisation élémentaire en UML:

Diagrammes de cas d'utilisation

Diagrammes de classes

Diagrammes d'objets/instances

Diagrammes de séquences

Auteur: Sarra KOUIDER

# Conception et Programmation Orientées Objet

---

## Initiation à la modélisation UML ( Cours 2 )



## Introduction à l'Orienté Objet :

Présentation du paradigme Objet  
Intérêt de l'orienté objet  
Etapas création d'applications OObject

## La Modélisation Object avec UML:

Notion de modélisation  
Difficultés de la modélisation  
Apport des méthodes de conception  
Le Langage de modélisation UML

## Modélisation élémentaire en UML:

Diagrammes de cas d'utilisation  
Diagrammes de classes  
Diagrammes d'objets/instances  
Diagrammes de séquences

Auteur: Sarra KOUIDER

# Conception et Programmation Orientées Objet

---

## Introduction à l'orienté Objet

## Que représente un objet dans le monde réel ?

### L'Objet dans le monde réel

Un **Objet** est une toute chose **concrète perceptible** à l'être humain par la vue et le toucher.

*Autrement défini :* un **Objet** est toute chose définie par son **utilisation**, sa **valeur**, et ses **attributs**.

Un objet voiture



**uneVoiture**

**Attributs :**

couleur = bleue  
poids = 979 kg  
puissance = 12 CV  
capacité carburant = 50 l  
conducteur = Dupont  
vitesse instantanée = 50 km/h

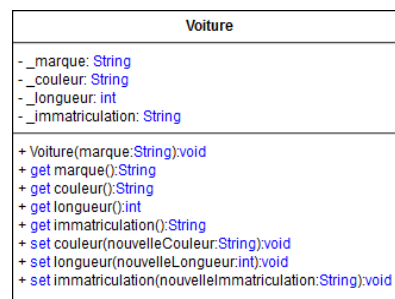
**Opérations :**

démarrer  
déplacer  
mettreEssence

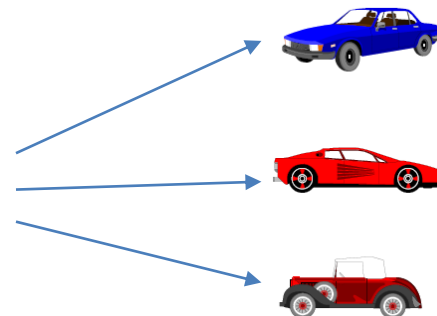
## Le paradigme Object dans le monde numérique

### La Programmation Orientée Objet (POO)

- ❖ La notion de programmation orientée objet à pour concepts clés: **les Classes** et **les Objets**.
- ❖ La notion de **classe** représente le **type** de l'objet, alors que le mot **objet** est **l'instance** physiquement réalisée de la classe, qui n'a d'existence que pendant l'exécution du programme.
- ❖ Un **Programme Orienté Object** est un **ensemble de classes** qui collaborent entre elles. Ces classes s'échangent des messages entre elles pour réaliser certaines tâches.

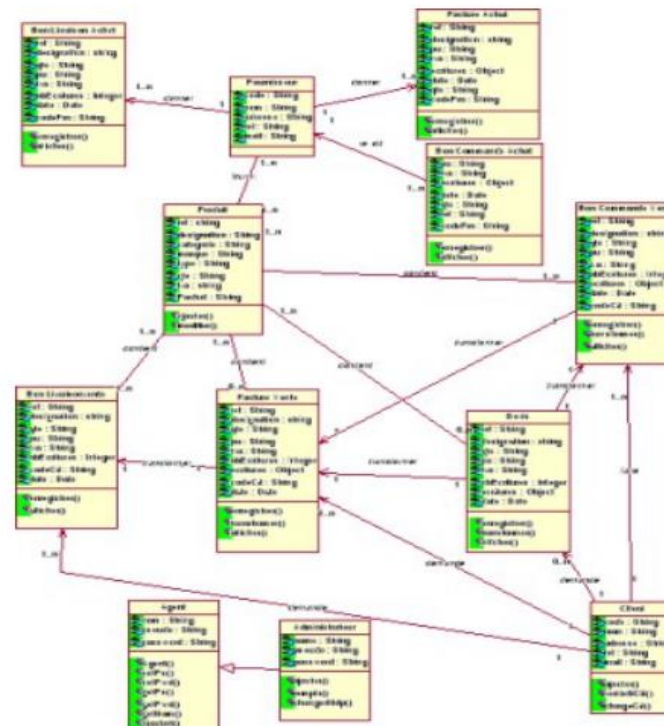


Classe



Instances

## Le paradigme Objet dans le monde numérique



## exemple d'un POO représenté en UML

## Pourquoi parler de la conception par objets ?

Succès des approches par objets (30 dernières années) :

- Décrire un système avec des représentations informatiques proches des entités du problème et de sa solution
- Avantages reconnus en termes de :
  - facilité du codage initial,
  - stabilité du logiciel construit car les objets manipulés sont plus stables que les fonctionnalités attendues,
  - aisance à réutiliser les artefacts existants et ...
  - à maintenir le logiciel, le corriger, le faire évoluer ;
- Fort développement dans les langages de conception, de programmation, les bases de données, les interfaces graphiques, les systèmes d'exploitation, etc.

Exemple de langages Orienté Objet : C++, .net, Java...

## Caractéristiques majeures d'un modèle orienté objet

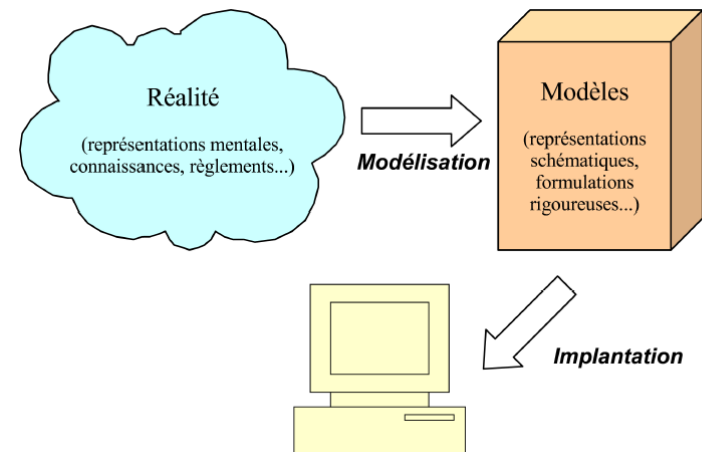
- **abstraction :**  
Ressortir les caractéristiques externes essentielles d'une entité pour la distinguer des autres.
- **encapsulation :**  
Cacher les détails qui ne font pas partie des caractéristiques essentielles d'une entité.
- **modularité :**  
Décomposer un programme en un ensemble de modules cohérents et faiblement couplés pouvant être compilés séparément.
- **hiérarchisation :**  
ranger ou ordonnancer les abstractions

### Attention

**si un modèle ne possède pas l'un de ces éléments, il n'est pas orienté objet.**

## Etapes de création d'une application orientée objet

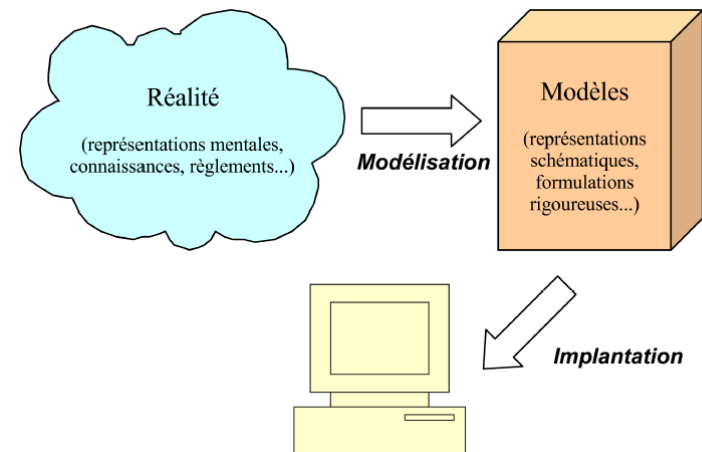
1. Utilisation d'un langage de modélisation pour la conception (UML)
2. Utilisation d'un langage de programmation pour l'implémentation (C#)





## Etapes de création d'une application orientée objet

1. Utilisation d'un langage de modélisation pour la conception (UML)
2. Utilisation d'un langage de programmation pour l'implémentation (C#)





Introduction à l'Orienté Objet :

Présentation du paradigme Objet

Intérêt de l'orienté objet

Etapas création d'applications OObjet

**La Modélisation Object avec UML:**

Notion de modélisation

Difficultés de la modélisation

Apport des méthodes de conception

Le Langage de modélisation UML

**Modélisation élémentaire en UML:**

Diagrammes de cas d'utilisation

Diagrammes de classes

Diagrammes d'objets/instances

Diagrammes de séquences

Auteur: Sarra KOUIDER

# Conception et Programmation Orientées Objet

---

## La modélisation Object avec UML

## Que veut on dire par la modélisation d'applications informatique ?

Première activité d'un informaticien face à un système à mettre en place.

Produire une représentation simplifiée du monde pour :

- accumuler et organiser des connaissances,
- décrire un problème,
- trouver et exprimer une solution,
- raisonner, calculer.

## Que veut on dire par la modélisation d'applications informatique ?

Il s'agit en particulier de résoudre le hiatus entre :

- le monde réel, complexe, en constante évolution, décrit de manière informelle et souvent ambiguë par les experts d'un domaine
- le monde informatique, où les langages sont codifiés de manière stricte et disposent d'une sémantique unique.

## Pourquoi modéliser une application avant toute chose ?

	Répartition effort dév.	Origine des erreurs	Coût de la maintenance
Définition des besoins	6%	56%	82%
Conception	5%	27%	13%
Codage	7%	7%	1%
Intégration Tests	15%	10%	4%
Maintenance	67%		

(Zeltovitz, De Marco)

## Poids de la maintenance d'une application informatique

## Difficultés de la modélisation

La modélisation est une tâche rendue difficile par différents aspects :

- spécifications parfois imprécises, incomplètes, ou incohérentes,
- taille et complexité des systèmes importantes et croissantes,
- évolution des besoins des utilisateurs,
- évolution de l'environnement technique (matériel et logiciel),
- des équipes à gérer plus grandes, avec des spécialisations techniques, nécessaires du fait de la taille des logiciels à construire, mais le travail est plus délicat à structurer.

## Apports des méthodes d'analyse et de conception

Guides structurant :

- organisation du travail en différentes phases (analyse, conception, codage, etc.) ou en niveaux d'abstraction (conceptuel, logique, physique),
- concepts fondateurs : par exemple les concepts de fonction, signal, état, objet, classe, etc.,
- représentations semi-formelles, documents, diagrammes, etc.

Langage de modélisation = formalisme de représentation qui facilite

- la communication
- l'organisation
- la vérification
- la génération d'une partie du code

## Vers l'ingénierie des modèles

- approche récente du génie logiciel
- faire face à l'évolution rapide des technologies
- modèles au centre du développement
- capitaliser les modèles d'une application
- déduire le code par transformations successives
- intégré dans la plupart des ateliers de construction



## Quesque c'est que l'UML ?

UML (Unified Modeling Language)

- langage de modélisation graphique
- issu en 1995 de la fusion de plusieurs méthodes à objets incluant OOSE (Jacobson), OOD (Booch), OMT (Rumbaugh)
- assurant une continuité des concepts depuis les phases amonts jusqu'à l'implémentation

incluant :

- les concepts des approches par objets : classe, instance, classification, etc.
- intégrant d'autres aspects : associations, fonctionnalités, événements, états, séquences, composants, patrons de collaboration, etc.

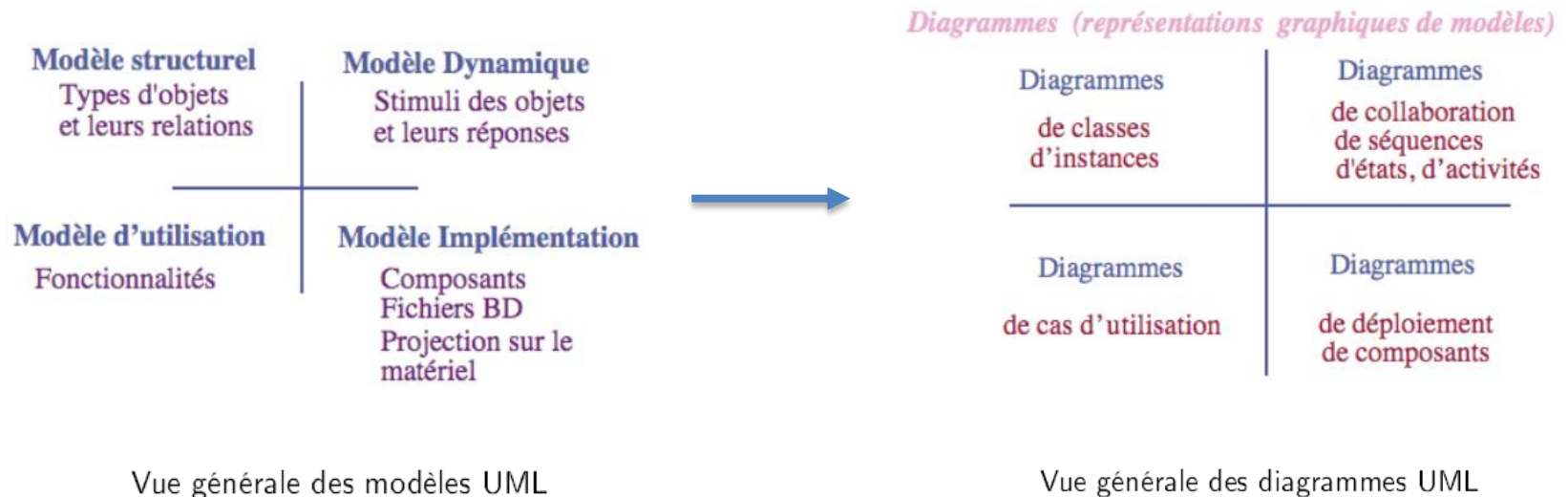
## C'est quoi un "Modèle UML" ?

### *Représentation abstraite d'une réalité*

Image simplifiée du monde réel selon un point de vue utile pour :

- comprendre et visualiser (en réduisant la complexité),
- communiquer (langage commun, nombre restreint de concepts),
- mémoriser les choix effectués,
- valider (contrôler la cohérence, simuler, tester).

## Types des Modèles UML existants





Introduction à l'Orienté Objet :

Présentation du paradigme Objet

Intérêt de l'orienté objet

Etapas création d'applications OObject

La Modélisation Object avec UML:

Notion de modélisation

Difficultés de la modélisation

Apport des méthodes de conception

Le Langage de modélisation UML

**Modélisation élémentaire en UML:**

Diagrammes de cas d'utilisation

Diagrammes de classes

Diagrammes d'objets/instances

Diagrammes de séquences

Auteur: Sarra KOUIDER

# Conception et Programmation Orientées Objet

---

## Modélisation élémentaire en UML

## 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**.

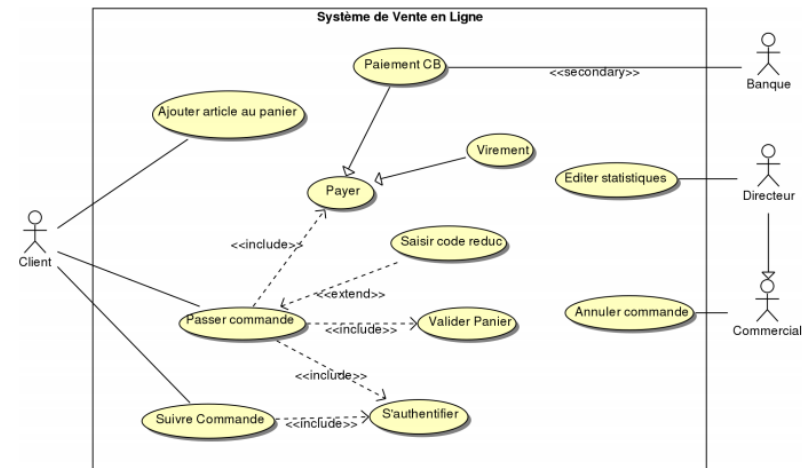
## Apport des diagrammes de cas d'utilisation

- Délimite le système et décrit la manière de l'utiliser
- Oriente le développement entier du système
- Présente :
  - Les fonctionnalités externes
  - le point de vue des utilisateurs
  - les interactions avec les acteurs extérieurs

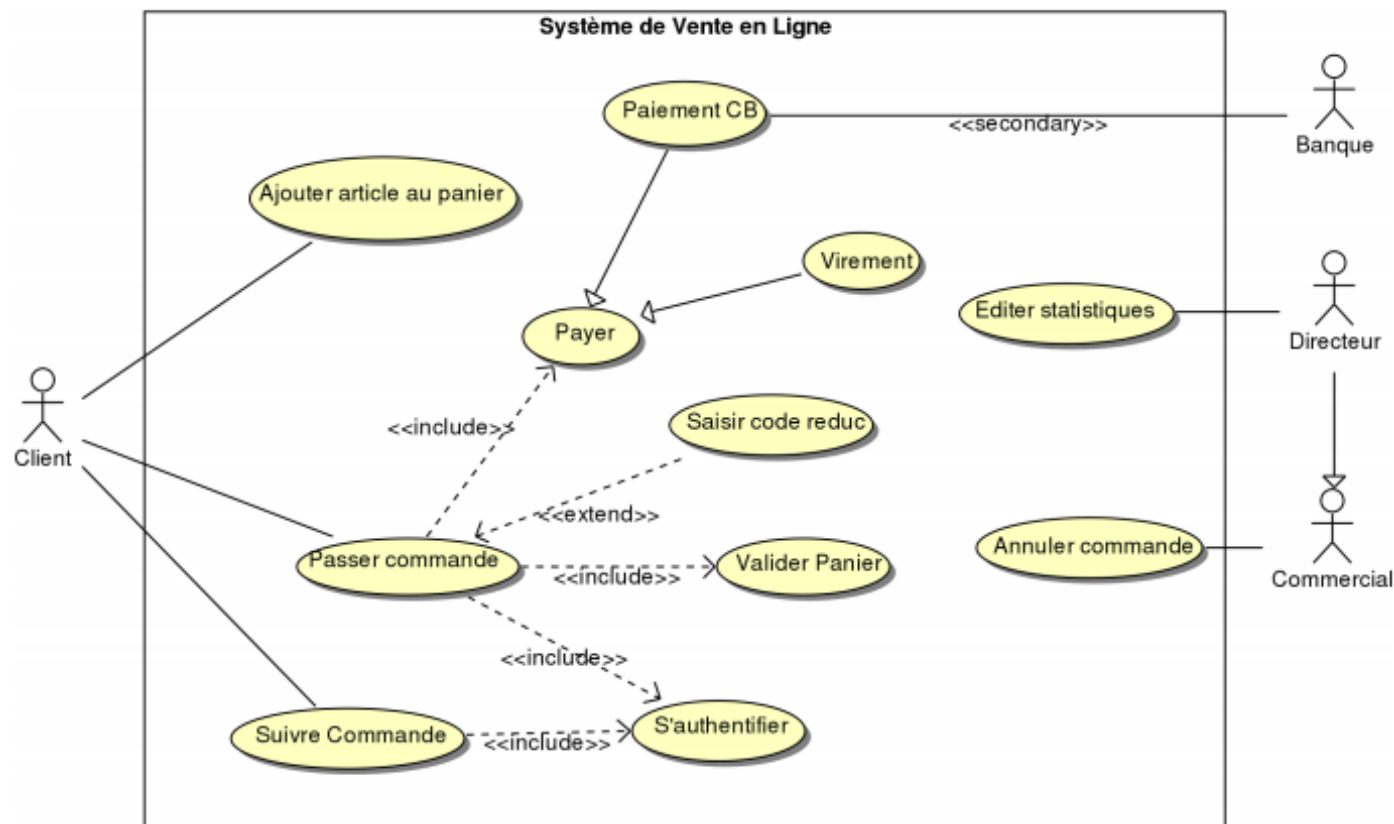
## Concepts majeurs du modèle d'utilisation

Concepts majeurs :

- frontière qui délimite le système
- acteurs
  - entité extérieure au système et interagissant avec
  - acteurs humains
  - acteurs machine (système extérieur communiquant avec le système étudié)
- cas d'utilisation (toute manière d'utiliser le système)
- relations




## Exemple d'un diagramme de cas d'utilisation





## Les éléments d'un diagramme de 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.



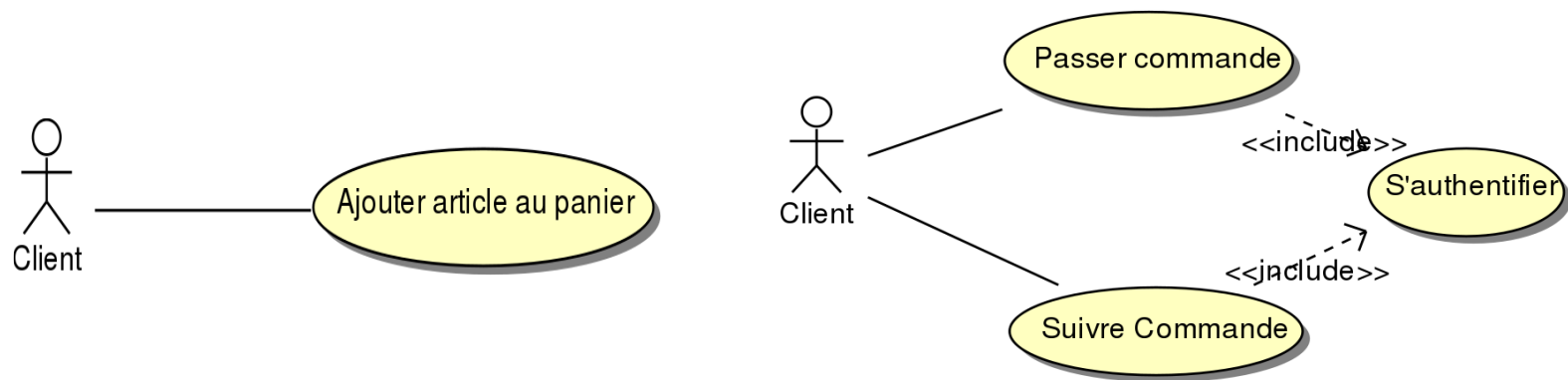
Un cas d'utilisation est l'expression d'un service réalisé de bout en bout, avec un déclenchement, un déroulement et une fin, pour l'acteur qui l'initie.

### Attention

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

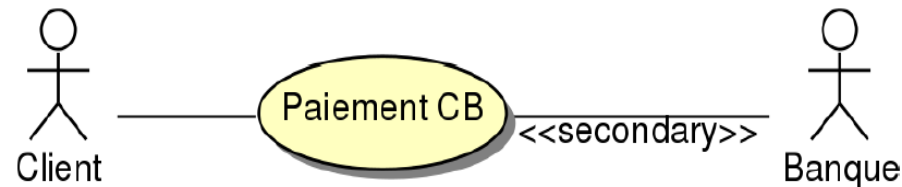
## Relations entre cas d'utilisation et acteurs

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



## Relations entre cas d'utilisation et acteurs

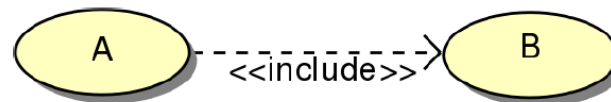
- L'acteur est dit **principal** pour un cas d'utilisation lorsque l'acteur est à l'initiative des échanges nécessaires pour réaliser le cas d'utilisation.



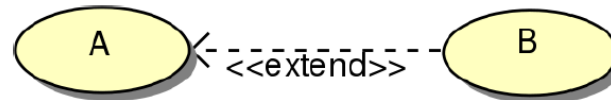
- Les acteurs **secondaires** sont sollicités par le système alors que le plus souvent, les acteurs principaux ont l'initiative des interactions.
  - Le plus souvent, les acteurs secondaires sont d'autres systèmes informatiques avec lesquels le système développé est inter-connecté.

## Relations entre cas d'utilisation

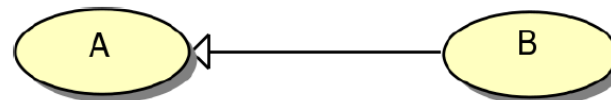
- **Inclusion** : le cas A inclut le cas B (B est une partie *obligatoire* de A).



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



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

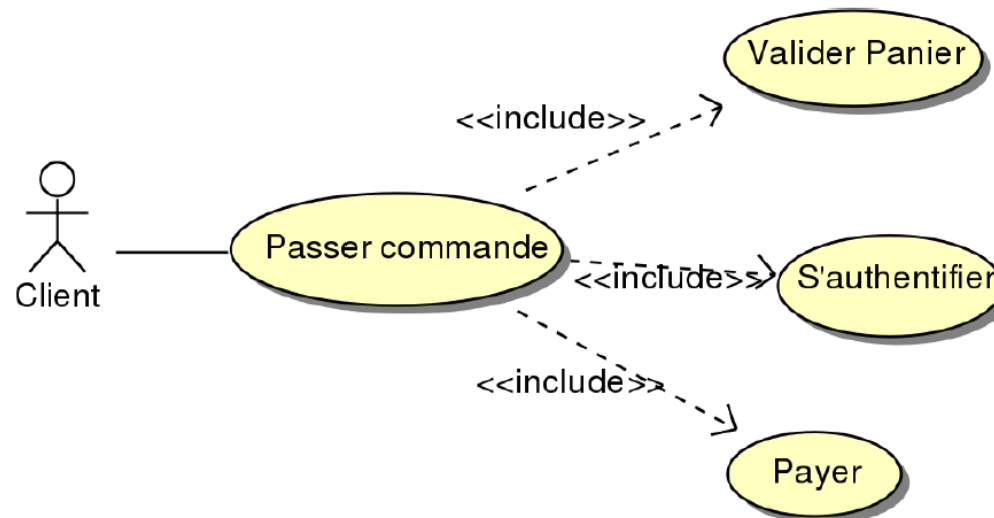


### Attention

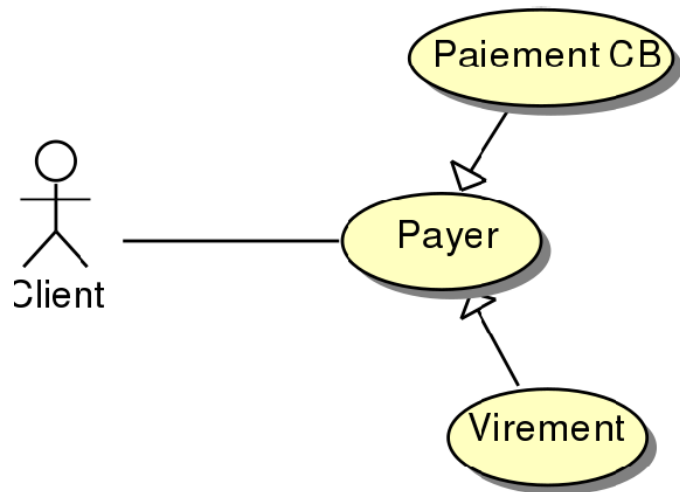
Le sens des flèches indique la dépendance, pas le sens de la relation d'inclusion.

## Relations entre cas d'utilisation

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



## La généralisation



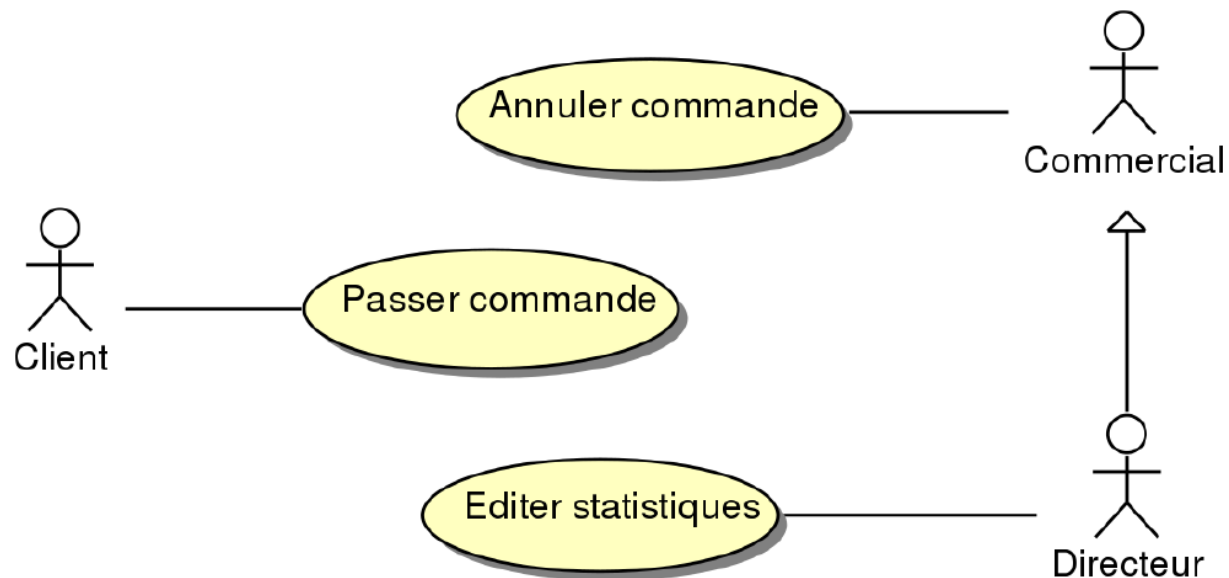
- Un virement est un cas particulier de paiement.

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.

## La généralisation entre les acteurs

- Une seule relation possible : la **généralisation**.



## 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 et limite 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.



Introduction à l'Orienté Objet :

Présentation du paradigme Objet

Intérêt de l'orienté objet

Etapes création d'applications OObject

La Modélisation Object avec UML:

Notion de modélisation

Difficultés de la modélisation

Apport des méthodes de conception

Le Langage de modélisation UML

Modélisation élémentaire en UML:

Diagrammes de cas d'utilisation

**Diagrammes de classes**

Diagrammes d'objets/instances

Diagrammes de séquences

Auteur: Sarra KOUIDER

# Conception et Programmation Orientées Objet

---

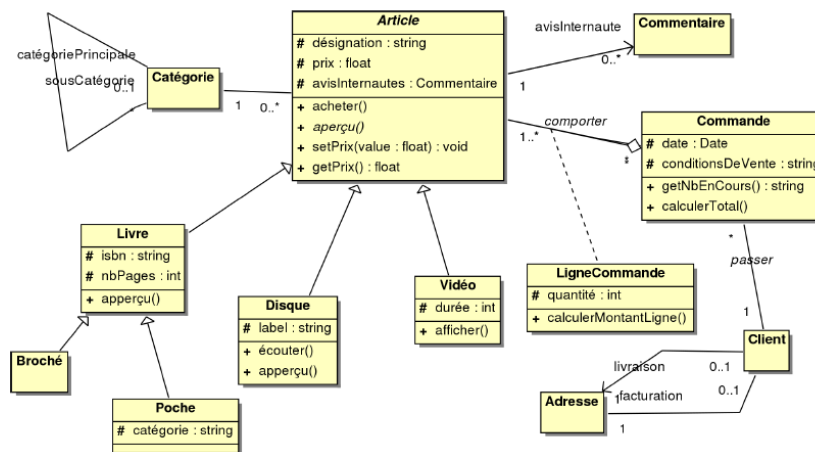
## Diagramme de Classes

## Quesque c'est qu'un diagramme de Classes ?

### Définition d'un diagramme de Classes

- ❖ Un diagramme de classes est la représentation d'un ensemble de **classes**, d'**interfaces** et de **paquetages** ainsi que leurs **relations**, et permettant de spécifier **la structure statique d'un système**.
- ❖ Un diagramme de classes **fait abstraction** des aspects **dynamiques** et **temporels**.

exemple d'un diagramme de Classes

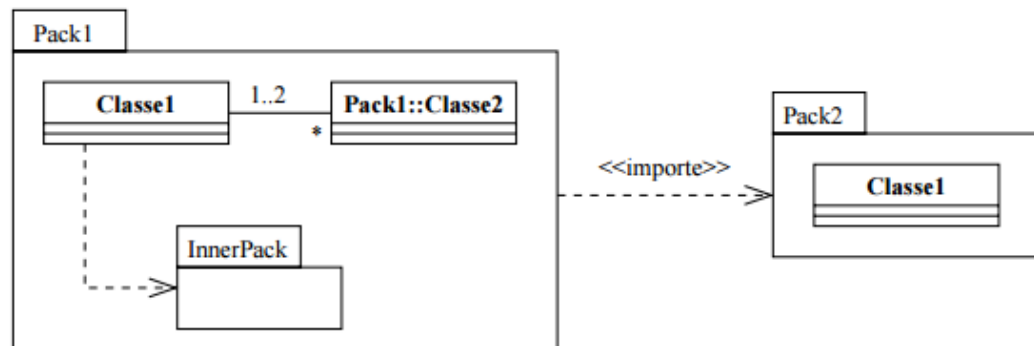


## A quoi sert un diagramme de Classes ?

- Lors de l'analyse et de la conception:
  - Définitions formelles des objets qui composent le système à partir des **cas d'utilisation** et des **diagrammes d'interaction** (séquences et collaboration).
  - Bases conceptuelles pour les **diagrammes d'état-transition**, de **déploiement**, ...
- Lors de l'implantation :
  - Génération automatique des structures statiques du système (classes, relations, ...).

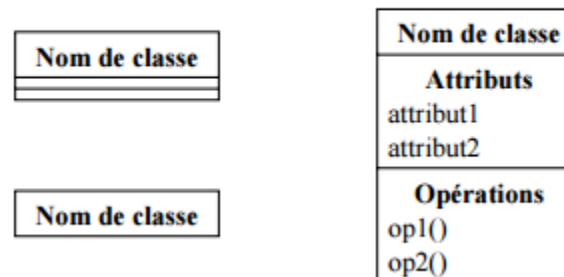
## Éléments d'un diagramme de classes : Les Paquetages

- **Définition** : Mécanisme de partitionnement des modèles et de regroupement des éléments de modélisation.
- Chaque paquetage peut contenir un ensemble de diagrammes et/ou de paquetages.
- Chaque élément d'un paquetage possède un nom **unique** dans ce paquetage.
- Possibilité de définir des relations entre paquetages (dépendances, cf. plus loin).



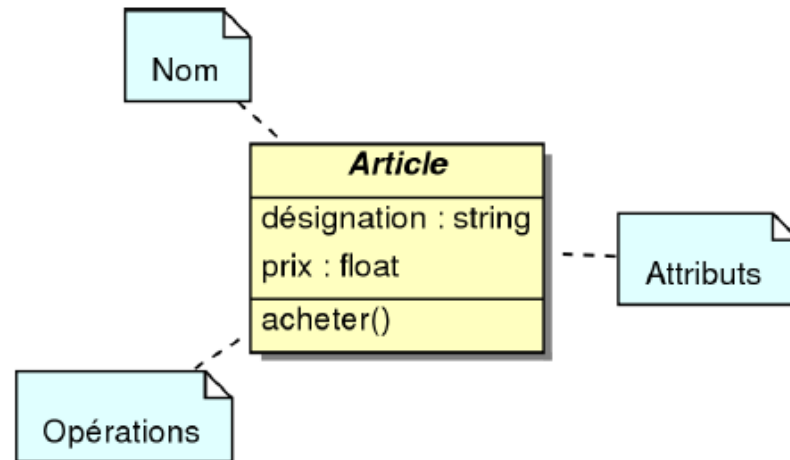
## Éléments d'un diagramme de classes : Les Classes

- **Définition** : description d'un ensemble d'objets partageant les mêmes attributs, opérations, méthodes, relations et sémantiques.
- Généralement, en fonction de l'objectif du diagramme, elle est décrite par : un nom (**obligatoire**), des attributs, des opérations, des exceptions, ...
- Un nom de classe est **unique** au sein du paquetage
- Représentation UML :



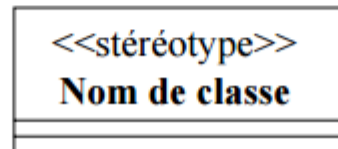
## Éléments d'un diagramme de classes : Les Classes

- Une classe est composée d'un **nom**, d'**attributs** et d'**opérations**.
- Selon l'avancement de la modélisation, ces informations ne sont pas forcément toutes connues.
- D'autres compartiments peuvent être ajoutés : responsabilités, exceptions, etc.



## Éléments d'un diagramme de classes : Les stéréotypes

- Un stéréotype permet d'**étendre** les classes déjà existantes en leur donnant une **signification sémantique différente**.
- Si la classe A est un stéréotype de la classe B, alors A se comporte **comme** B tout en ayant une signification sémantique différente.
- Mécanisme proche de la généralisation/spécialisation sauf qu'il permet le changement de sémantique.
- Représentation UML :





## Éléments d'un diagramme de classes : Les stéréotypes

- Quelques stéréotypes prédéfinis :

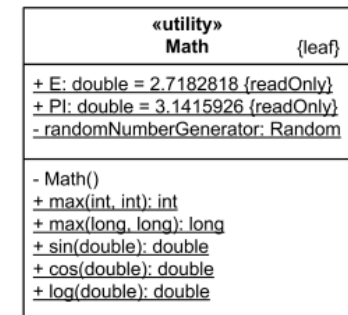
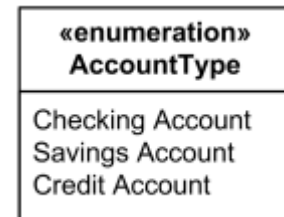
«**énumération**» : classe définissant un ensemble d'identificateurs formant le domaine de valeur d'un type.

«**utilitaire**» : classe réduite au concept de module et qui ne peut être instanciée.

«**acteur**» : classe modélisant un ensemble de rôles joués par un acteur.

«**interface**» : classe contenant uniquement une description des opérations visibles.

«**exception**» : classe modélisant un cas particulier de signal : les exceptions.



## Éléments d'un diagramme de classes : **Les Attributs**

- **Définition** : propriété définie par un **nom**, un **type** et éventuellement une **valeur initiale**.
- **Syntaxe UML** :  
`[ visibilité ] nom_attribut [ multiplicité ] :  
type_attribut [ = valeur_initiale ]`
  - **visibilité** : cf. plus loin.
  - **nom\_attribut** : identificateur de l'attribut, **unique** au sein de la classe.
  - **multiplicité** : l'attribut représente un ensemble de valeurs;  
exemple de tableau: `Parents [ 1..2 ] : Personne.`
  - **valeur\_initiale** : valeur prise par l'attribut lors de l'instanciation de la classe (valeur en concordance avec le type de l'attribut).

## Éléments d'un diagramme de classes : Les Opérations

- **Définition** : spécification du **comportement** des instances de la classe.
- Cinq catégories d'opérations :
  - les **constructeurs** qui créent les objets,
  - les **destructeurs** qui détruisent les objets,
  - les **sélecteurs** (opérations de consultation) qui renvoient tout ou partie de l'état d'un objet,
  - les **modificateurs** qui changent tout ou partie de l'état d'un objet,
  - les **itérateurs** qui visitent l'état d'un objet ou le contenu d'une structure de données contenant des objets.

## Éléments d'un diagramme de classes : **Les Opérations**

- Syntaxe UML :

`[ visibilité ] nom_opération ( [ arguments ] ) :  
type_retourné propriétés`

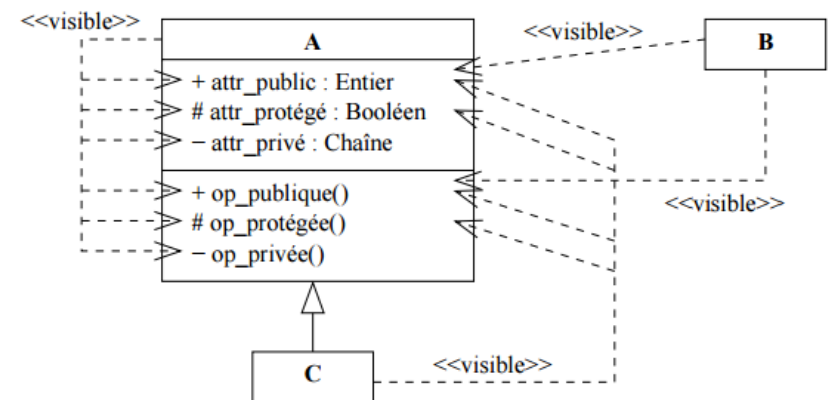
- visibilité : cf. plus loin.
- nom\_opération : identificateur de l'opération, **unique** au sein de la classe.
- type\_retourné : type de la valeur retournée par l'opération; si omis l'opération ne retourne aucune valeur.

## Éléments d'un diagramme de classes : Les arguments des opérations

- arguments : description des valeurs nécessaire à l'opération.
- Syntaxe UML :  
[ direction ] nom\_argument : type\_argument [ =  
valeur\_par\_défaut ] [ , autres\_arguments ]
  - valeur\_par\_défaut : valeur prise par l'argument si aucune valeur n'est donnée lors de l'utilisation de cette opération.
  - direction : UML définit trois directions pour les arguments
    - \* in (par défaut) : paramètre en entrée seule et non modifié par l'exécution de l'opération,
    - \* out : paramètre en sortie seule; l'appelant peut ainsi récupérer des informations,
    - \* inout : paramètre en entrée-sortie; l'argument est passé à l'opération et modifiable.

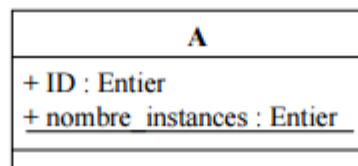
## Visibilité des éléments d'un diagramme de classes

- UML définit trois niveaux de visibilité :
  - public : l'élément est visible pour tous les clients/utilisateurs de la classe, noté par **+**.
  - protégé : l'élément est visible pour les sous-classes de la classe, noté par **#**.
  - privé : l'élément est visible pour la classe seule, noté par **-**.



## Portée des éléments d'un diagramme de classes

- UML définit deux niveaux de portée :
  - portée d'instance** (par défaut) : les éléments sont valides pour d'une seule instance de la classe; les éléments n'ont aucune existence en dehors de l'instance.
  - portée de la classe** : les éléments sont toujours valides; ils ne sont pas attachés à une instance particulière mais à une classe.
- Syntaxe UML : un élément ayant une portée de classe est souligné  
+ Parents [ 1..2 ] : Personne



## Les associations dans un diagramme de classes

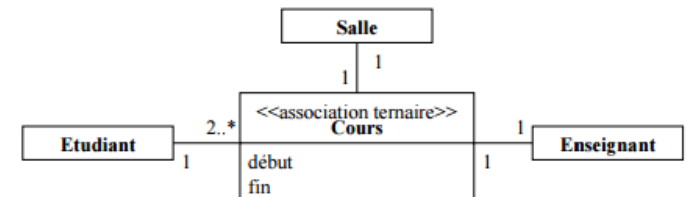
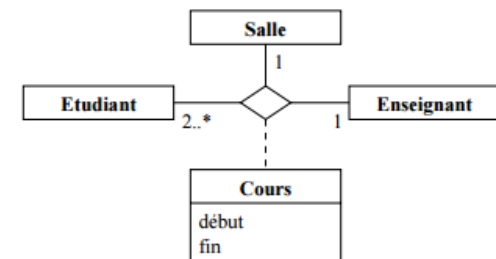
- **Définition** : relation entre au moins deux classes qui entraînent des connexions entre leurs instances.
- **Représentation UML** : trait reliant les deux classes en relation.





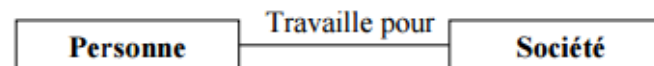
## Les associations : Arité

- Les associations ont le plus souvent une arité **binaire** : deux classes en relation.
- Représentation UML des associations d'arité supérieure (n-aire) : losange.
- Traduire une association d'arité n-aire en un ensemble d'associations binaires.
- Note : la difficulté de trouver un nom différent pour chaque extrémité d'une association n-aire est souvent le signe d'une association d'arité inférieure.

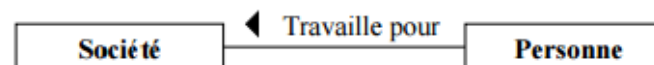


## Les associations : **Nommage**

- Les associations peuvent être nommées c.-à-d. identifiées par un texte unique décrivant la sémantique de l'association.

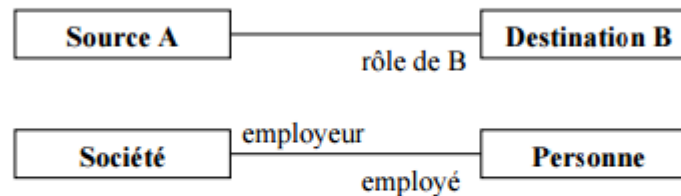


- Note : utiliser une **forme verbale** active (“travaille pour”) ou passive (“employé par”).
- Si ambiguïté, indiquer le sens de lecture avec les signes ◀ ou ▶ (par défaut lecture de gauche à droite).



## Les associations : rôle des extrémités

- Les extrémités des associations peuvent être qualifiées par des rôles.
- Un rôle indique comment une classe *Source* voit une classe *Destination*.



- Le rôle est un **pseudo-attribut** de la classe source (utilisé comme un attribut).
- Note : ne pas utiliser à la fois le nommage d'une association et les rôles des extrémités de la même association.

## Les associations : **Multiplicité**

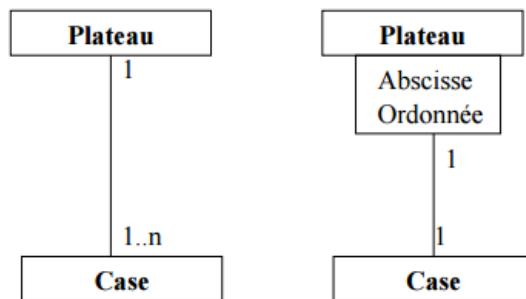
- **Définition** : la multiplicité précise le nombre d'instances pouvant être liées par une extrémité d'association à une instance pour chaque autre extrémité d'association.
- Sous sa forme générale, elle s'exprime par une suite d'intervalles disjoints sur l'ensemble des entiers naturels.
- Sous-estimer la multiplicité des associations réduit la flexibilité du modèle, la sur-estimer conduit à des développements inefficaces.



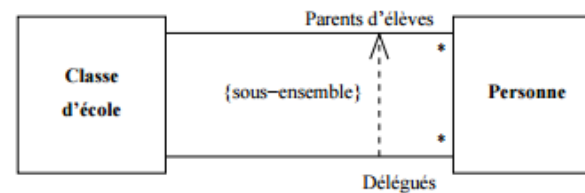
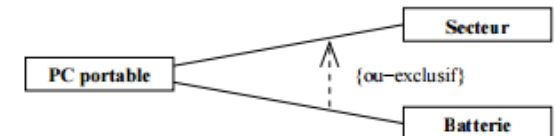
- Multiplicités définies par UML :

1	un et un seul (multiplicité par défaut)
0..1	zéro ou un
N	exactement N
M..N	de M à N
*	zéro ou plus, c.-à-d. de 0 à $+\infty$
0..*	zéro ou plus, c.-à-d. de 0 à $+\infty$
1..*	un ou plus, c.-à-d. de 1 à $+\infty$

## Les associations : **qualification et contrainte**



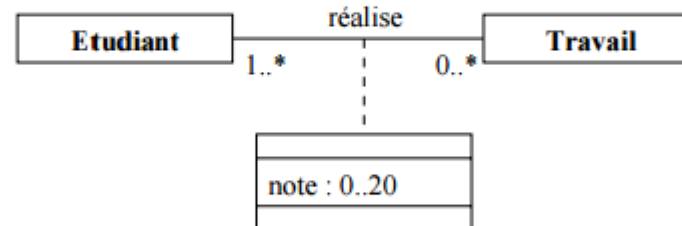
Association : qualification



Association : contraintes

## Les associations : **classe-association**

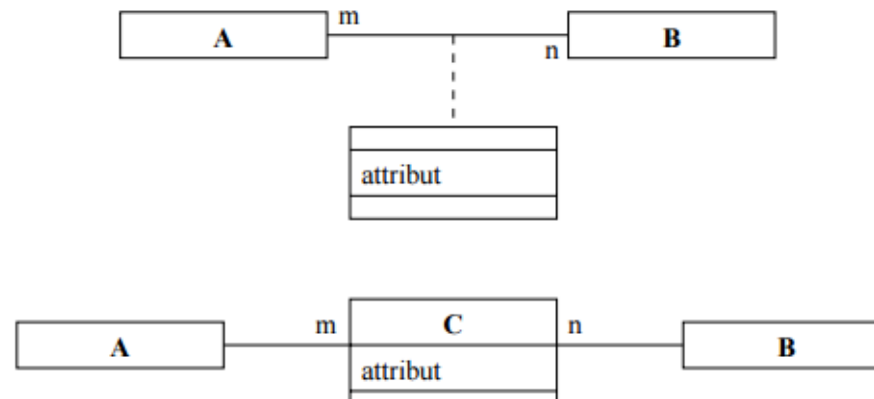
- Si une association possède des propriétés ou des opérations, il est possible de la qualifier à l'aide d'une **classe-association**.
- Une classe-association possède les mêmes caractéristiques que les associations et les classes.



- Une classe-association qui ne participe pas à d'autres relations avec d'autres classes peut ne pas porter de nom.

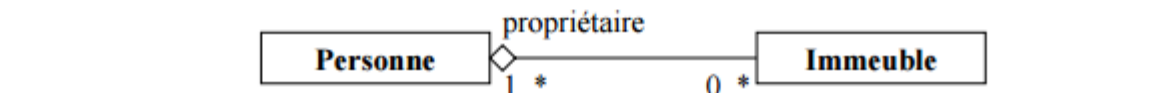
## Les associations : **classe-association**

- Lors de la conception, une classe-association peut être remplacée par une classe intermédiaire.



## Les associations : **Aggrégation**

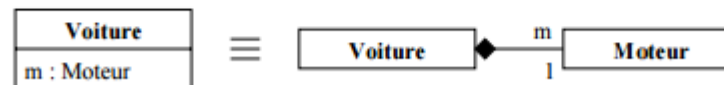
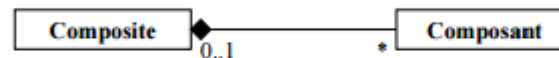
- **Définition** : forme spéciale d'association exprimant une relation de composition entre agrégats (part-whole).
- Association irréflexive, antisymétrique dans laquelle une des extrémités joue un rôle prédominant, pouvant être récursive.
- Permet de modéliser les contraintes d'intégrité et de désigner les agrégats comme garant de ces contraintes.
- A travers une aggrégation, il est possible de représenter :
  - la propagation des valeurs d'attributs d'une classe vers l'autre;
  - une action sur une classe qui implique une action sur une autre classe (ex: copie profonde);
  - une subordination des objets d'une classe à ceux d'une autre.





## Les associations : **Composition**

- Cas particulier d'aggrégation. La classe ayant le rôle prédominant est la classe **composite** ou classe conteneur.
- La composition implique :
  - la durée de vie des composants est la même que celle du composite.
  - la multiplicité du côté du composite prend ses valeurs dans 0 ou 1.
- La composition et les attributs sont sémantiquement équivalents.



## Les associations : **Aggrégation vs Composition**

La composition est aussi dite « agrégation forte ».

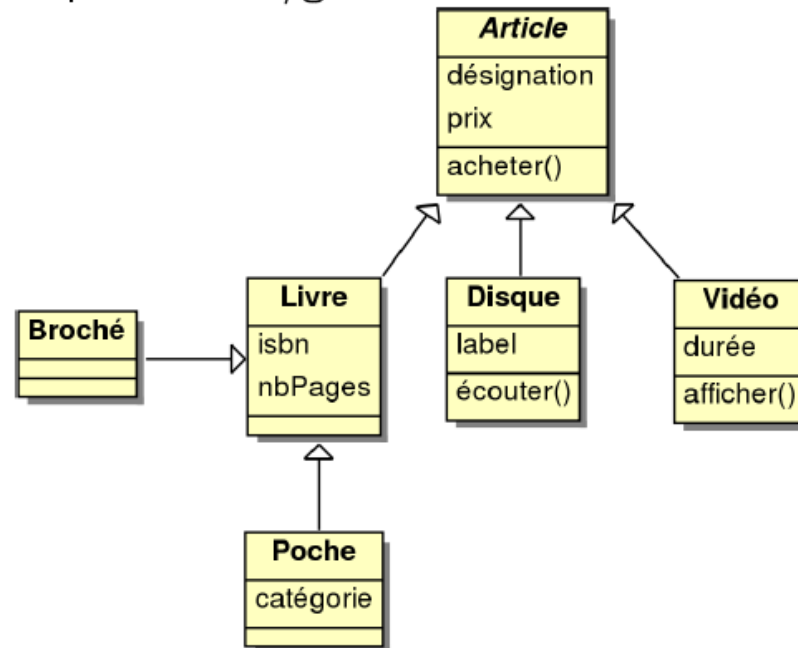
- Pour décider de mettre une composition plutôt qu'une agrégation, on doit se poser les questions suivantes :
  - Est-ce que la destruction de l'objet composite (du tout) implique nécessairement la destruction des objets composants (les parties) ? C'est le cas si les composants n'ont pas d'autonomie vis-à-vis des composites.
  - Lorsque l'on copie le composite, doit-on aussi copier les composants, ou est-ce qu'on peut les « réutiliser », auquel cas un composant peut faire partie de plusieurs composites ?

Si on répond par l'affirmative à ces deux questions, on doit utiliser une composition.

## L'Héritage

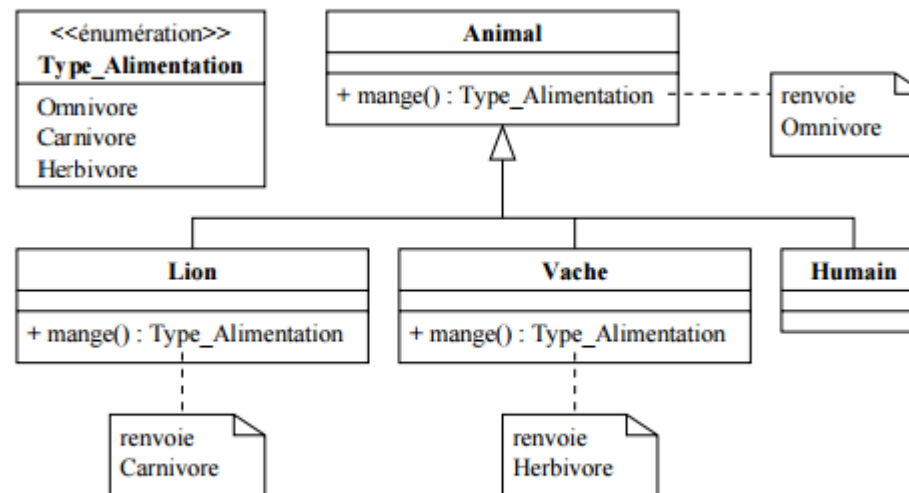
- L'héritage une relation de spécialisation/généralisation.

- Les éléments spécialisés héritent de la structure et du comportement des éléments plus généraux (attributs et opérations)
- **Exemple :** Par héritage d'Article, un livre a d'office un prix, une désignation et une opération acheter(), sans qu'il soit nécessaire de le préciser



## Polymorphisme

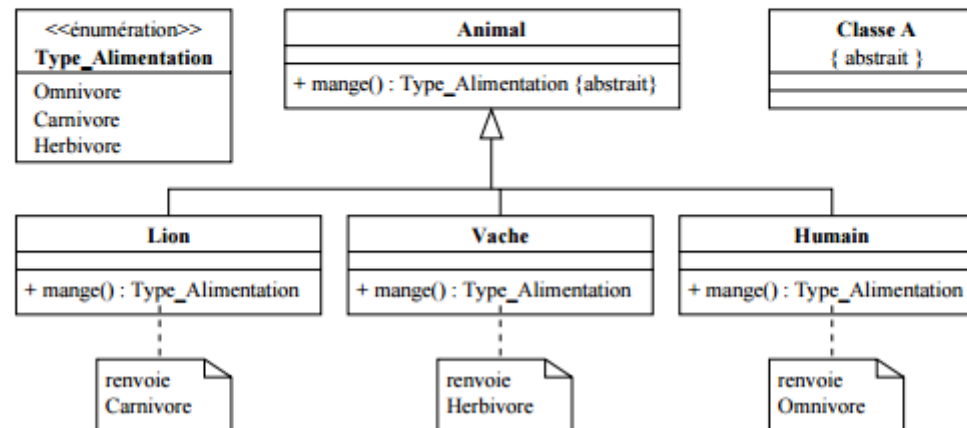
- **Définition :** mécanisme permettant à une classe fille la spécialisation d'opérations.



- L'exécution de l'opération `mange()` dépendra du type réel de l'objet :  
`Lion::mange()` pour un **Lion** et `Animal::mange()` pour un **Humain**.

## Classe abstraite

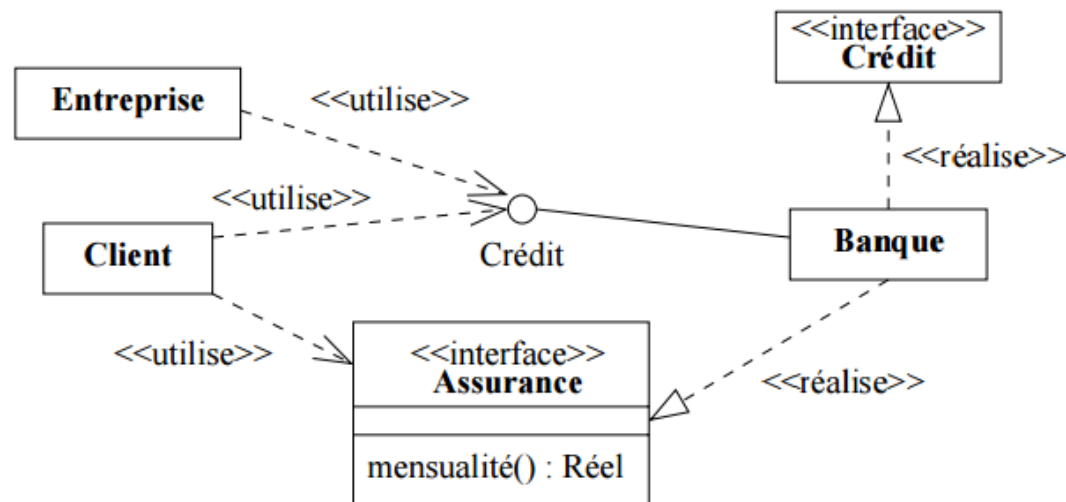
- **Définition** : classe **non instanciable** définissant au moins un mécanisme général instanciable par des classes filles.
- **Représentation UML** : définition de la classe avec la propriété {abstrait} ou si une de ses opérations (héritée ou non) possède la propriété {abstrait = vrai}.



## Interface

- **Définition** : description d'un ensemble d'opérations utilisées pour spécifier un service offert par une classe.
- Ne contient ni attribut, ni association, ni implémentation des opérations (les opérations sont abstraites).
- Une classe réalisant une interface doit :
  - soit implémenter les opérations de l'interface,
  - soit définir les opérations de l'interface comme des opérations abstraites (implémentables par les classes filles).
- Représentation UML :
  - classe ayant le stéréotype «interface», ou par un cercle pour faire référence à l'interface utilisée dans la classe,
  - flèche d'héritage en pointillés pour la réalisation d'une interface par une classe,
  - flèche de dépendance en pointillés pour l'utilisation.

## Interface





Introduction à l'Orienté Objet :

Présentation du paradigme Objet

Intérêt de l'orienté objet

Etapas création d'applications OObjet

La Modélisation Object avec UML:

Notion de modélisation

Difficultés de la modélisation

Apport des méthodes de conception

Le Langage de modélisation UML

Modélisation élémentaire en UML:

Diagrammes de cas d'utilisation

Diagrammes de classes

**Diagrammes d'objets/instances**

Diagrammes de séquences

Auteur: Sarra KOUIDER

# Conception et Programmation Orientées Objet

---

**Diagrammes d'objets/instances**



## Quesque c'est qu'un diagramme d'objets/instances ?

### Définition d'un diagramme d'Objets

- ❖ Un diagramme d'objets est la représentation d'un **ensemble d'objets** et de **liens**, exprimant la **structure statique** du système.
- ❖ Un diagramme d'objets est **une instance d'un diagramme de classes**. Il permet **d'illustrer l'état** d'un système à **un moment donnée**.



Diagramme de Classes

Diagramme d'objets

## A quoi sert un diagramme d'objets/instances ?

- Le **diagramme d'objets** représente les objets d'un système à un instant donné. Il permet de :
  - Illustrer le modèle de classes (en montrant un exemple qui explique le modèle) ;
  - Préciser certains aspects du système (en mettant en évidence des détails imperceptibles dans le diagramme de classes) ;
  - Exprimer une exception (en modélisant des cas particuliers, des connaissances non généralisables. . .).



Diagramme de Classes

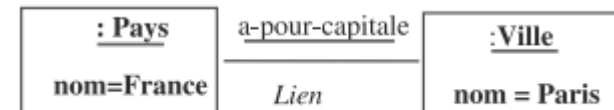


Diagramme d'objets

## A quoi sert un diagramme d'objets/instances ?

- Le **diagramme d'objets** représente les objets d'un système à un instant donné. Il permet de :
  - Illustrer le modèle de classes (en montrant un exemple qui explique le modèle) ;
  - Préciser certains aspects du système (en mettant en évidence des détails imperceptibles dans le diagramme de classes) ;
  - Exprimer une exception (en modélisant des cas particuliers, des connaissances non généralisables. . . ).

Le diagramme de classes modélise des *règles* et  
le diagramme d'objets modélise des *faits*.

## Caractéristiques d'un diagramme d'objets ?

- Un diagramme d'objets est composé :
  - d'objets (instances de classes),
  - de liens (instances d'associations).
- La notation des diagrammes d'objets **est dérivée** de celle des diagrammes de classes.



Diagramme de Classes

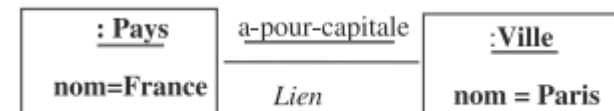
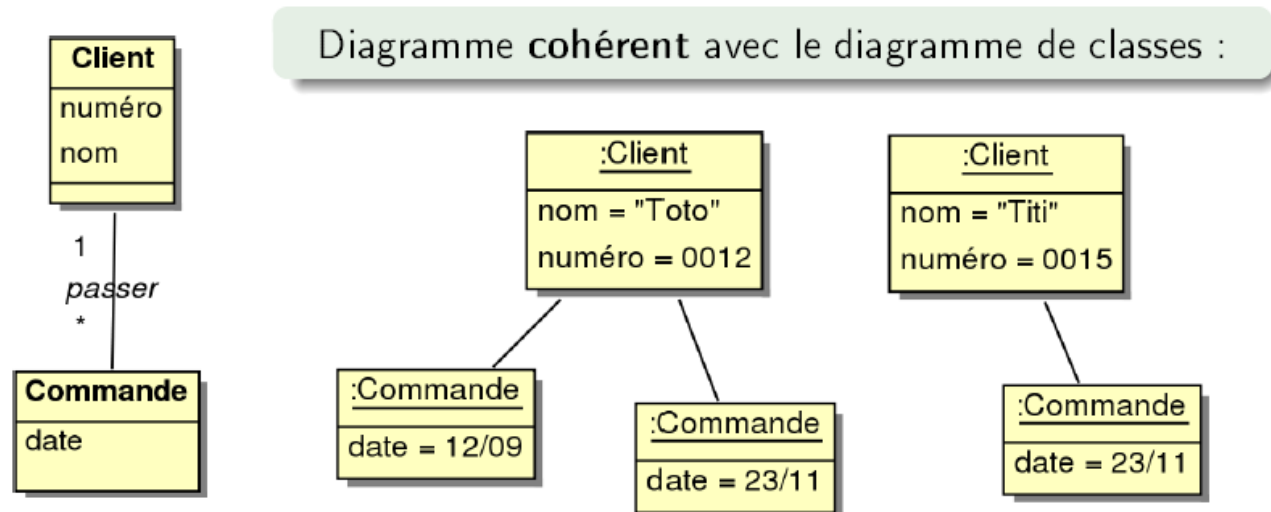


Diagramme d'objets

## Caractéristiques d'un diagramme d'objets ?

- Le diagramme de classes **contraint** la structure et les liens entre les objets.



## Représentation des objets en UML

- Un objet est une instance d'une classe : il représente "l'état" d'une classe à un instant précis.

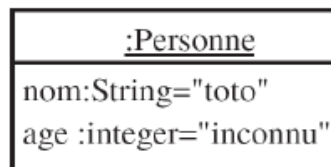
- Représentation UML :

nom de l'objet

nom de l'objet:Classe

:Classe

- Les instances peuvent être **anonymes** (a,c,d), **nommées** (b,f), **orphelines** (e), **multiples** (d) ou **stéréotypées** (g).



(a)



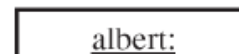
(b)



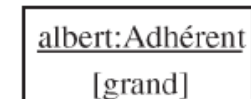
(c)



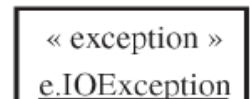
(d)



(e)



(f)



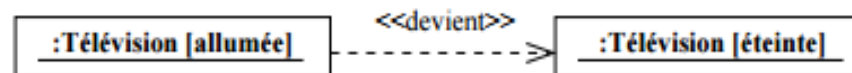
(g)

Exemples d'objets en UML

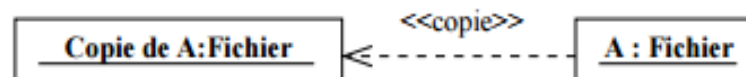
## Représentation des objets en UML

- Possibilité de modéliser les changements d'états des objets

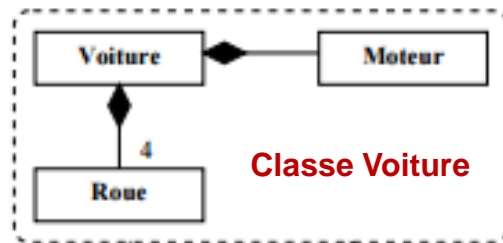
:



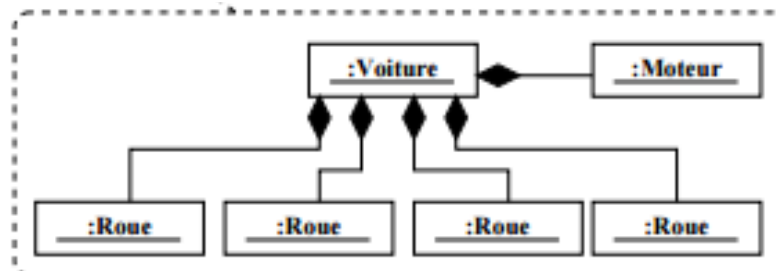
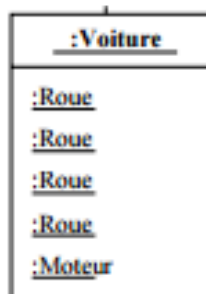
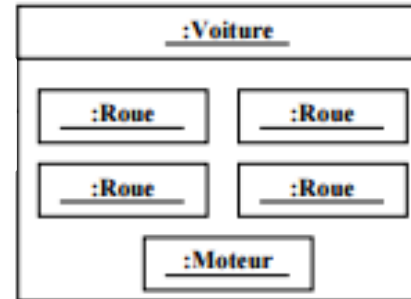
- Possibilité d'utiliser des liens stéréotypés (exemple : la copie d'objets)



## Représentation d'un objet composite en UML



Classe Voiture



3 dénotations possibles pour représenter une objet composite



## La représentation des liens dans un diagramme d'objets

- Les objets sont reliés par des instances d'associations : les liens.
- Un lien représente une relation entre objets **à un instant donné**.

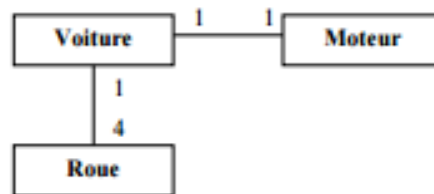


Diagramme de classes

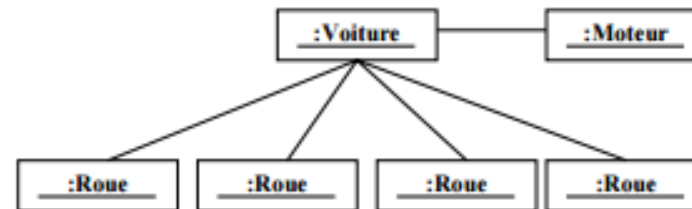


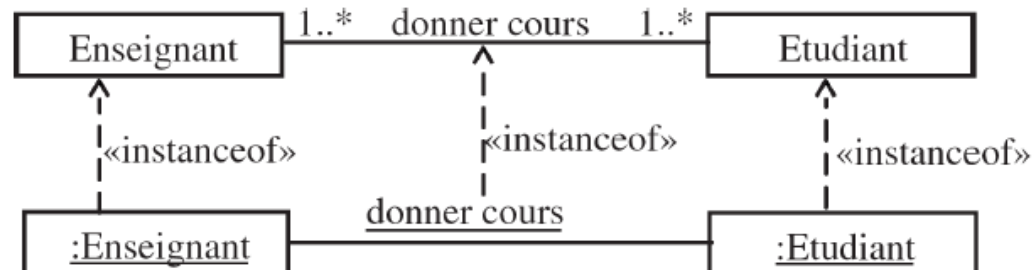
Diagramme d'objets

### Attention

Naturellement, on ne représente pas les multiplicités qui n'ont aucun sens au niveau des objets.

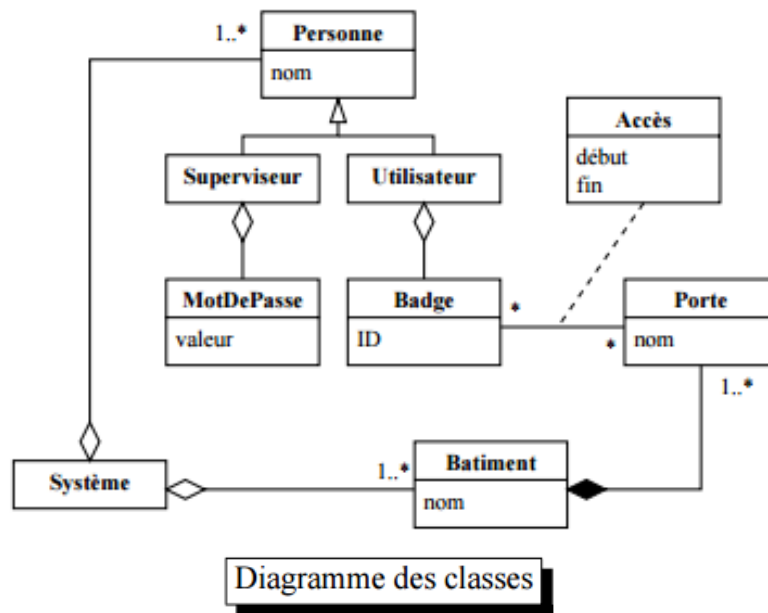
## Relation de dépendance d'instanciation

- La relation de **dépendance d'instanciation** (stéréotypée) décrit la relation entre un classeur et ses instances.
- Elle relie, en particulier, les associations aux liens et les classes aux objets.



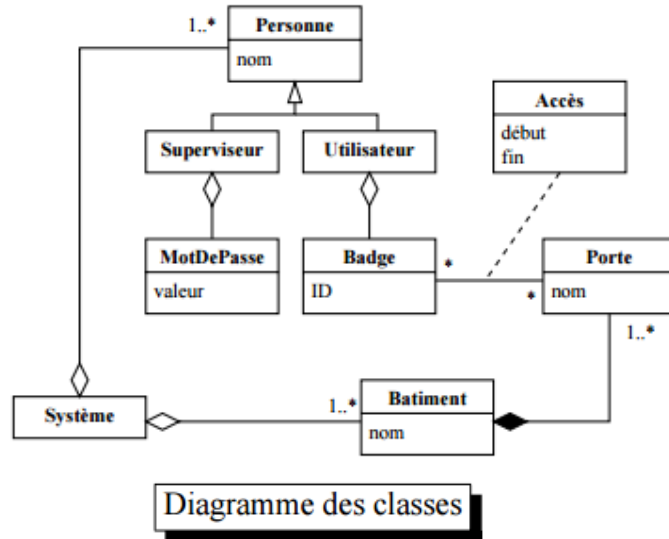
## Exemple d'un système de sécurité

- Il s'agit d'un système de sécurité limitant les accès à des parties d'un édifice à l'aide de cartes magnétiques.

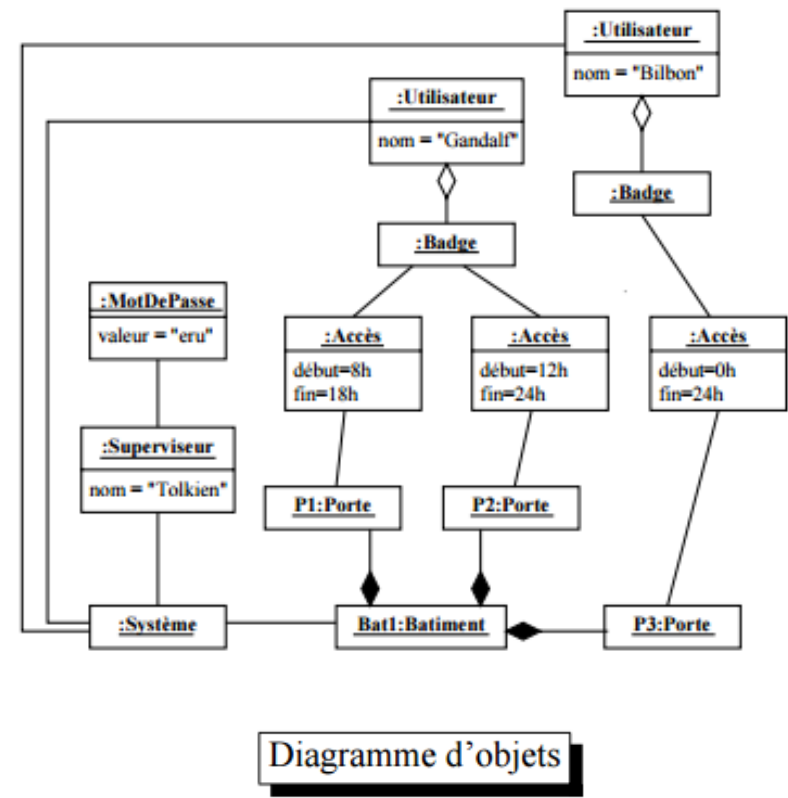


- Le système gère un seul bâtiment contenant trois portes.
- Le système peut être géré par une personne nommée Tolkien.
- Deux utilisateurs peuvent accéder au bâtiment :
  - Gandalf a accès à la première (8h-18h) et seconde porte (12h-24h)
  - Bilbon a accès à la troisième porte toute la journée.

- Le système gère un seul bâtiment contenant trois portes.
- Le système peut être géré par une personne nommée Tolkien.
- Deux utilisateurs peuvent accéder au bâtiment :
  - Gandalf a accès à la première (8h-18h) et seconde porte (12h-24h)
  - Bilbon a accès à la troisième porte toute la journée.



## Exemple d'un système de sécurité





Introduction à l'Orienté Objet :

Présentation du paradigme Objet

Intérêt de l'orienté objet

Etapas création d'applications OObjet

La Modélisation Object avec UML:

Notion de modélisation

Difficultés de la modélisation

Apport des méthodes de conception

Le Langage de modélisation UML

Modélisation élémentaire en UML:

Diagrammes de cas d'utilisation

Diagrammes de classes

Diagrammes d'objets/instances

**Diagrammes de séquences**

Auteur: Sarra KOUIDER

# Conception et Programmation Orientées Objet

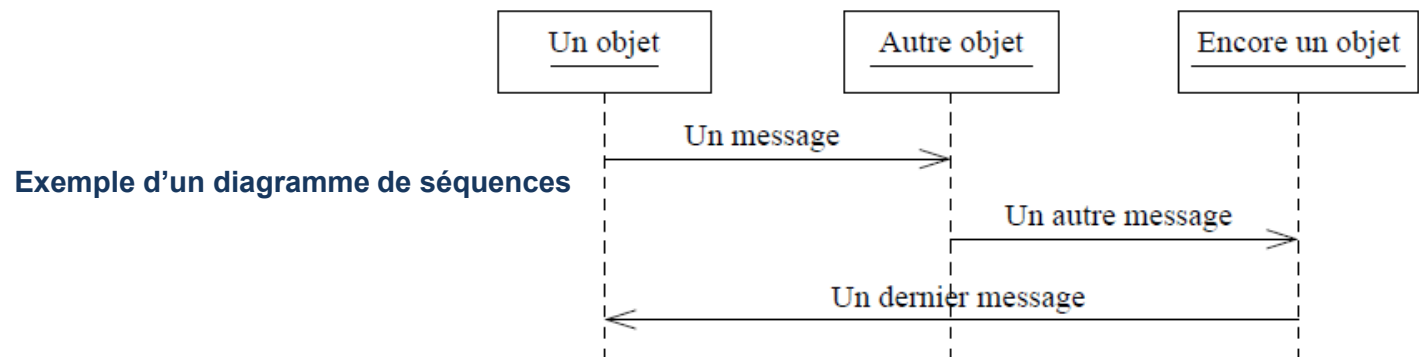
---

## Diagrammes de séquences

## Quesque c'est qu'un diagramme de séquences ?

### Définition d'un diagramme de séquence

- ❖ Les diagrammes de séquences sont des **diagrammes d'interaction** qui décrivent **l'ordre des interactions** entre les **objets** qui composent le système.
- ❖ Le diagramme de séquences est une représentation du système se concentrant sur **la séquence** des interactions d'un **point de vue temporel** (représentation **dynamique** du système).



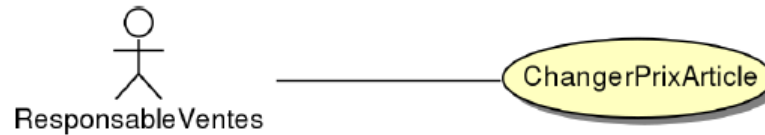
## Quel est l'intérêt d'un diagramme de séquences ?

- Les diagrammes de séquence permettent de représenter les interactions entre des instances particulières. Un diagramme met en jeu :
  - les objets intervenant dans l'interaction (acteurs ou objets appartenant au système)
  - la description de l'interaction (messages)
  - les interactions entre les intervenants (diagramme de séquences)
- Le diagramme de séquence permet d'insister sur la chronologie des interactions : le temps s'écoule grosso modo du haut vers le bas.

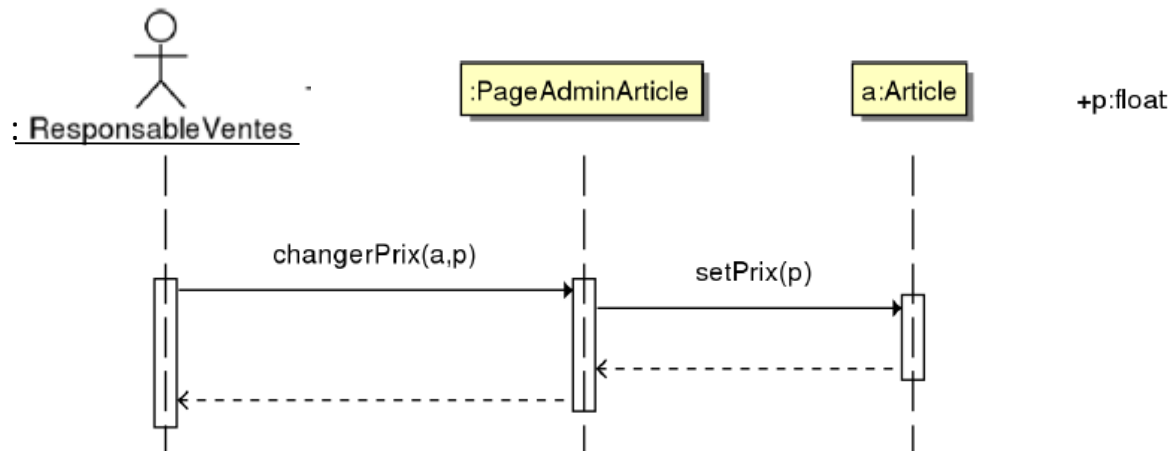
## Quel est l'intérêt d'un diagramme de séquences ?

**Attention Chaque cas d'utilisation donne lieu à un diagramme de séquence**

- Cas d'utilisation :

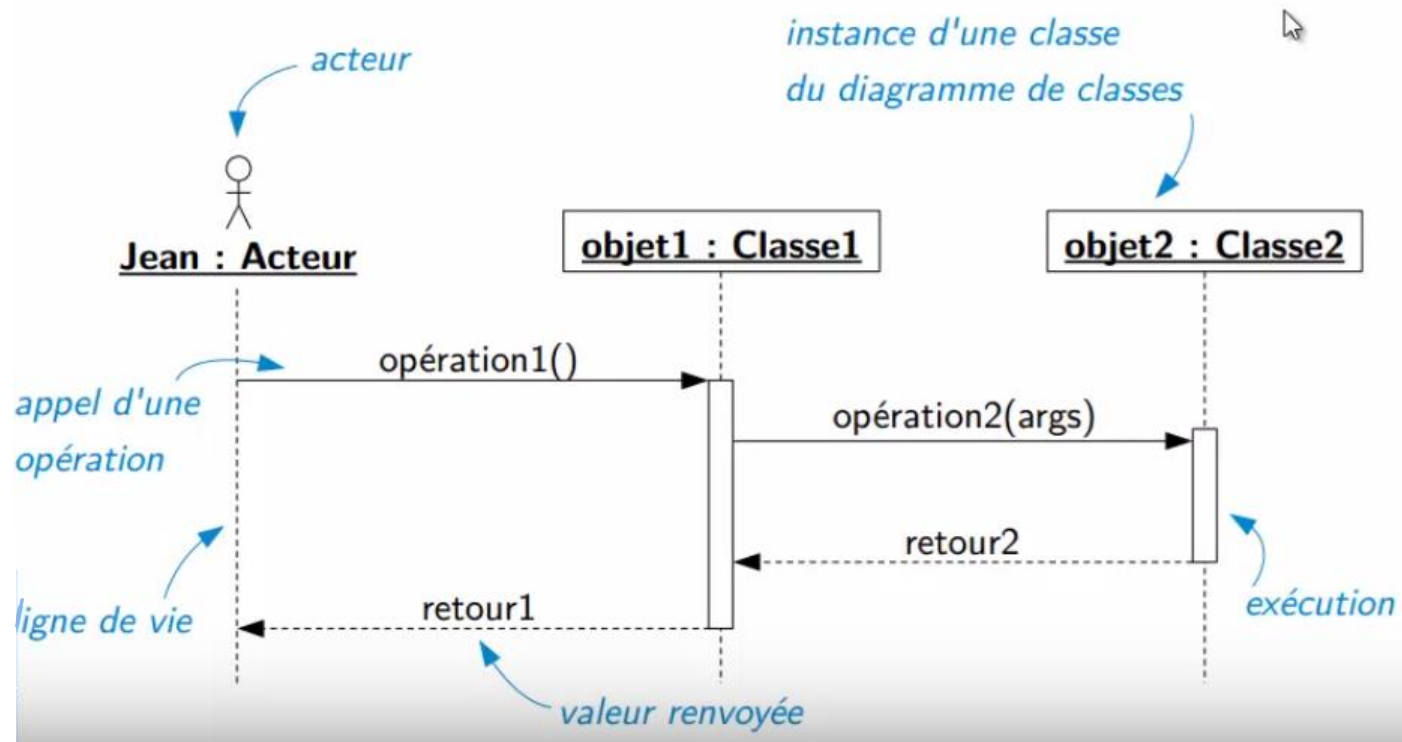


- Diagramme de séquences correspondant :



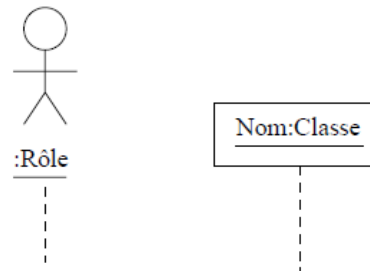


## Les éléments de bases d'un diagramme de séquences



## Les éléments de bases : Les objets

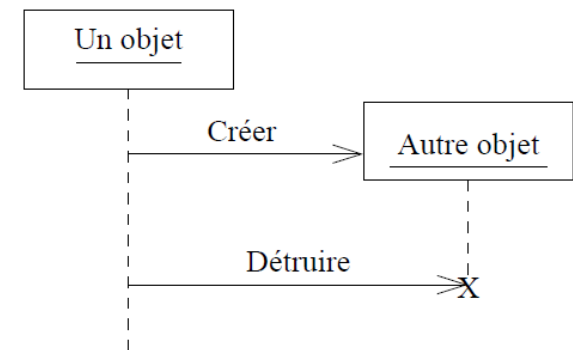
- Dans UML, les objets sont représentés comme suit :



- Le nom de l'objet est composé de son **rôle** (rôle ou nom) et/ou du nom de la classe instanciée (classe).
- Le nom est souligné pour indiquer qu'il s'agit d'une instance.

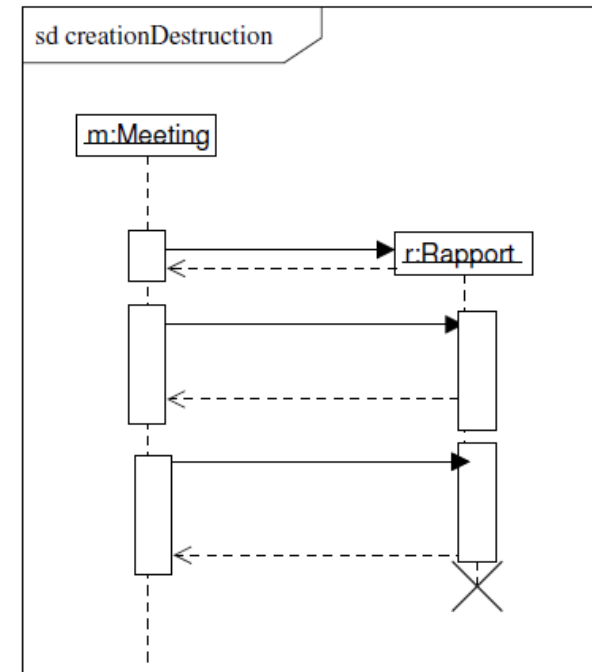
## Les éléments de bases : La ligne de vie des objets

- Elle est représentée par une ligne verticale en dessous des objets.
- Elle représente la période de temps durant laquelle l'objet "existe".
- **Création** d'un objet : un message pointe sur le symbole de l'objet.
- **Destruction** d'un objet : sa ligne de vie se termine par une croix en trait épais (X).



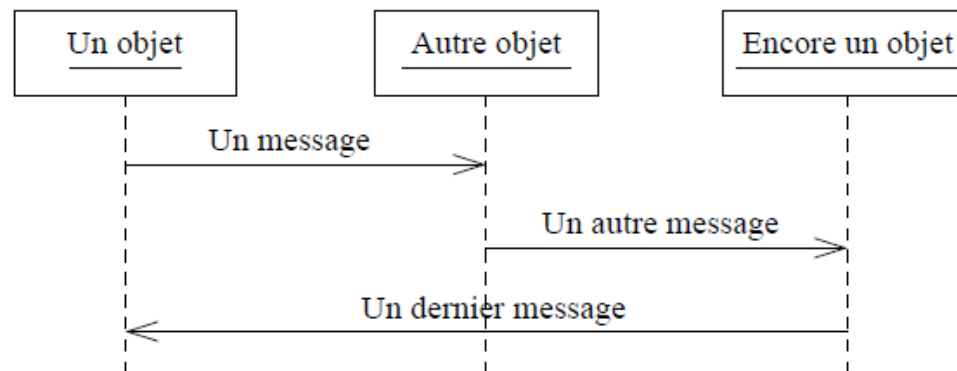
## Les éléments de bases : La ligne de vie des objets

- À chaque instance est associée une ligne de vie, qui représente la vie de l'objet.
- Les événements survenant sur une ligne de vie (réception de message ou envoi de message) sont ordonnés chronologiquement.
- La ligne de vie est représentée par une ligne pointillée quand l'instance est inactive, et par une boîte blanche ou grisée quand l'instance est active.
- Quand une instance est détruite, on stoppe la ligne de vie par une croix.



## Les éléments de bases : Les messages

- Les objets communiquent en échangeant des messages représentés sous forme de flèches.
- La dimension verticale représente l'écoulement du temps.
- Les messages sont étiquetés par le nom de l'opération ou du signal invoqué.



## Les éléments de bases : Les messages

- Les étiquettes décrivent les messages auxquels elles sont attachées.
- Syntaxe générale: `([attribut =] signal-ou-NomOperation [(liste-arguments)])[: valeur-retour]) | *`  
où la syntaxe pour un argument est :  
`([nomParam =] valeur-argument) | (attribut = nomParamOut [: valeurArgument]) | -`  
  - \* signifie : n'importe quel type de message
  - signifie : paramètre indéfini

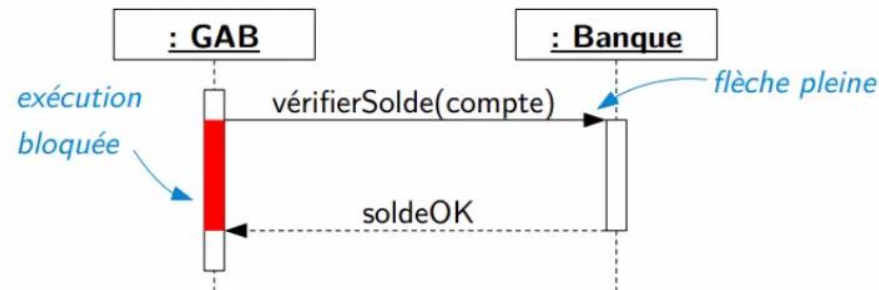
### Exemples :

- getAge()
- setAge(age=15)
- getAge() :12
- setAge(-)
- age=getAge() :12

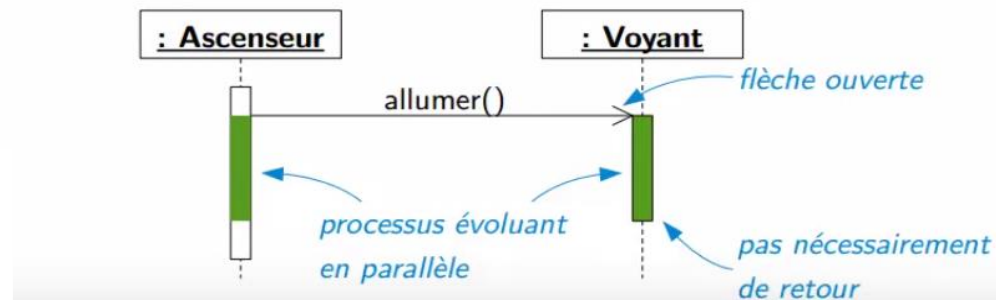


## Les éléments de bases : Types de messages

Message synchrone : Émetteur bloqué en attente du retour



Message asynchrone : Émetteur non bloqué, continue son exécution



## Les éléments de bases : **Types de messages**

- Un message **synchrone** bloque l'expéditeur jusqu'à la réponse du destinataire. Le flot de contrôle passe de l'émetteur au récepteur.
  - Typiquement : appel de méthode
    - Si un objet A invoque une méthode d'un objet B, A reste bloqué tant que B n'a pas terminé.

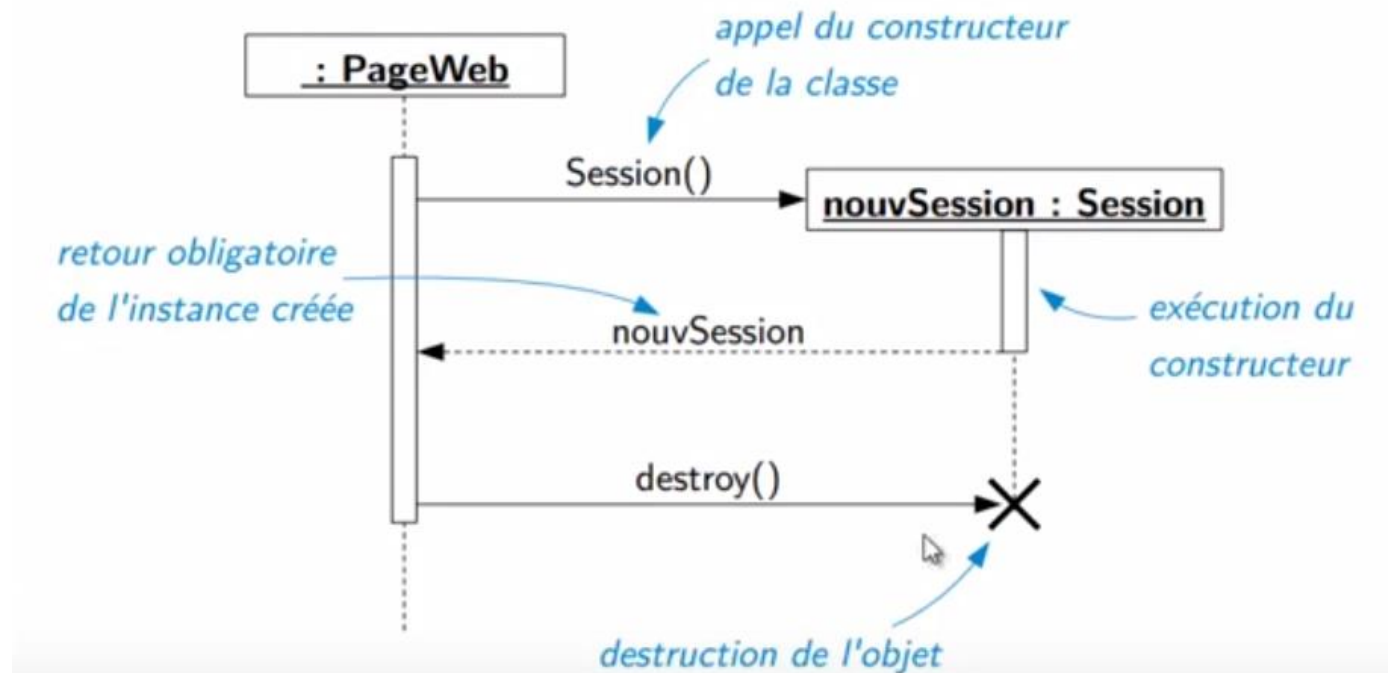


- Un message **asynchrone** n'est pas bloquant pour l'expéditeur. Le message envoyé peut être pris en compte par le récepteur à tout moment ou ignoré.
  - Typiquement : envoi de signal (voir stéréotype de classe « signal »).

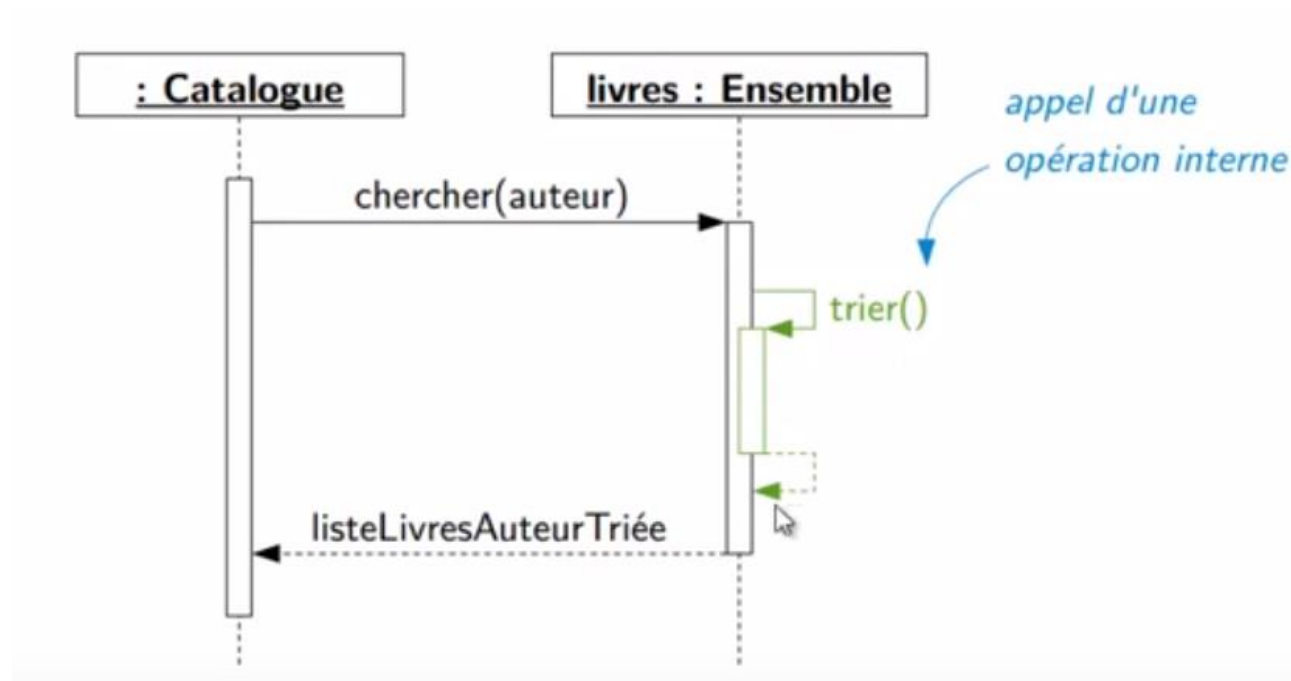




## Différents Messages: **Message de Création et destruction d'objet**



## Différents Messages: **Message réflexif**

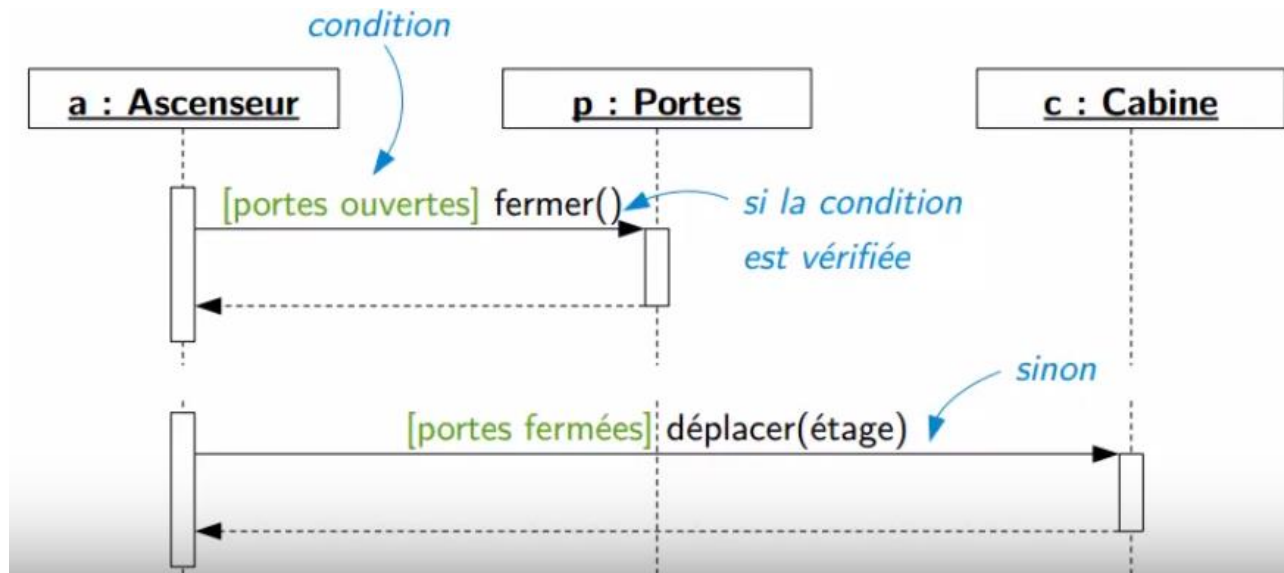


## Différents Messages: **Message Alternatif**

**Principe** : Condition à l'envoi d'un message

**Notation** :

- Deux diagrammes

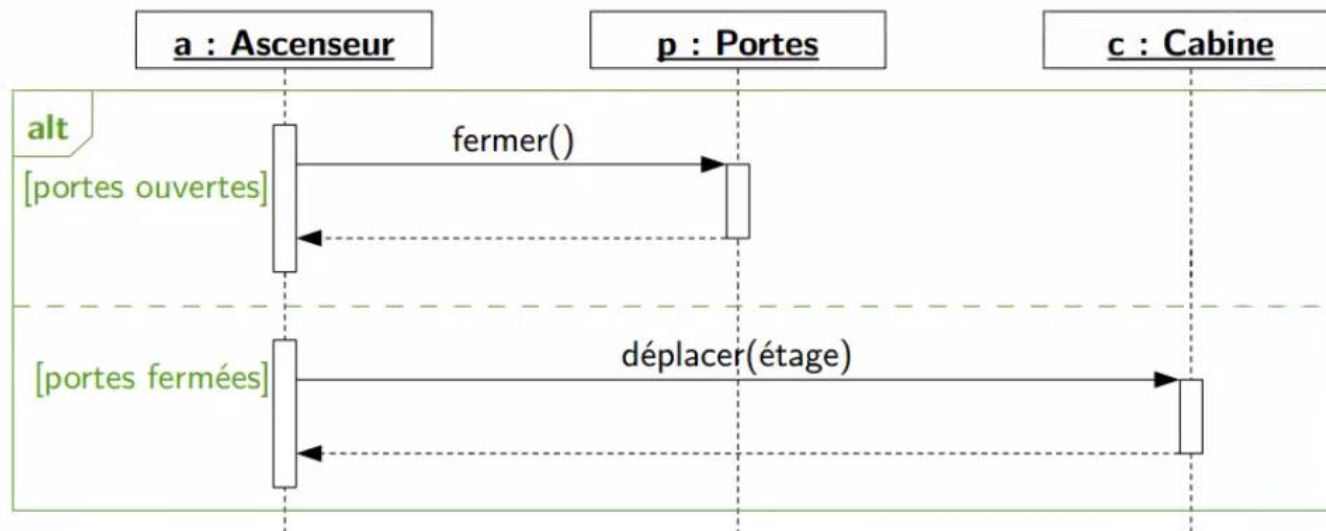


## Différents Messages: **Message Alternatif**

Principe : Condition à l'envoi d'un message

Notation :

- Deux diagrammes
- Bloc d'alternative **alt**

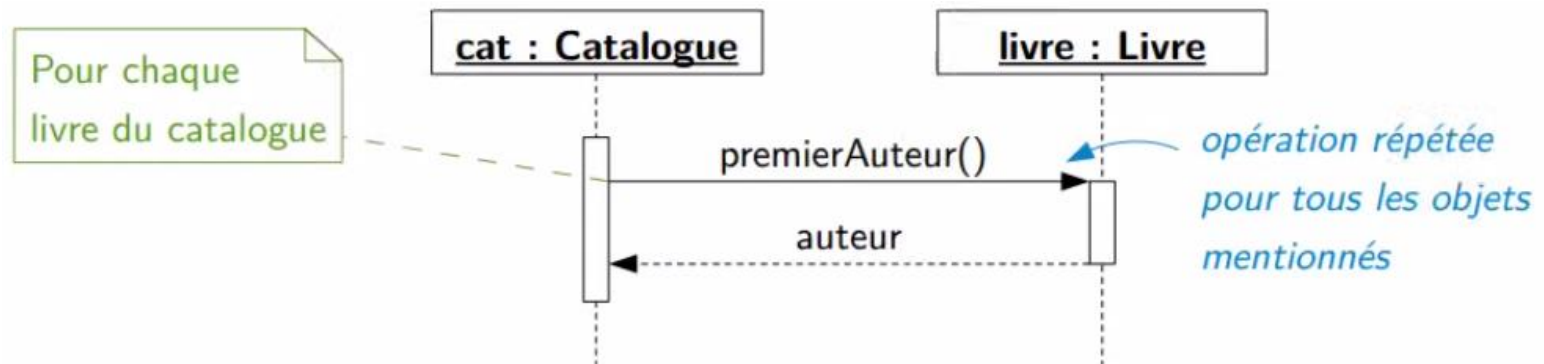


## Différents Messages: Les Boucles

Principe : Répéter un enchaînement de messages

Notation :

- Note

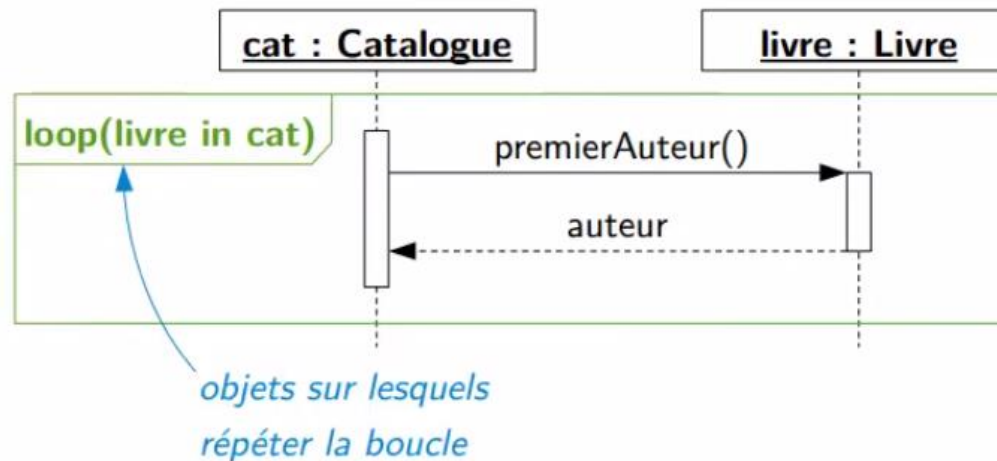


## Différents Messages: **Les Boucles**

**Principe** : Répéter un enchaînement de messages

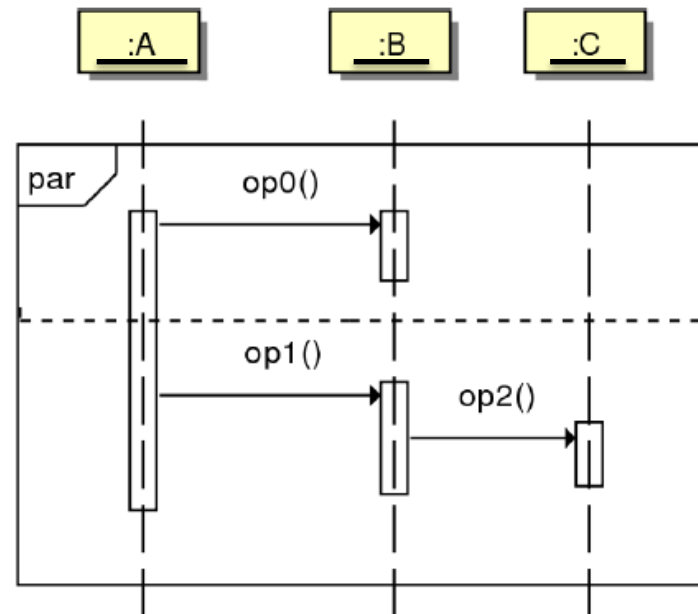
**Notation** :

- Note
- Bloc de boucle **loop**

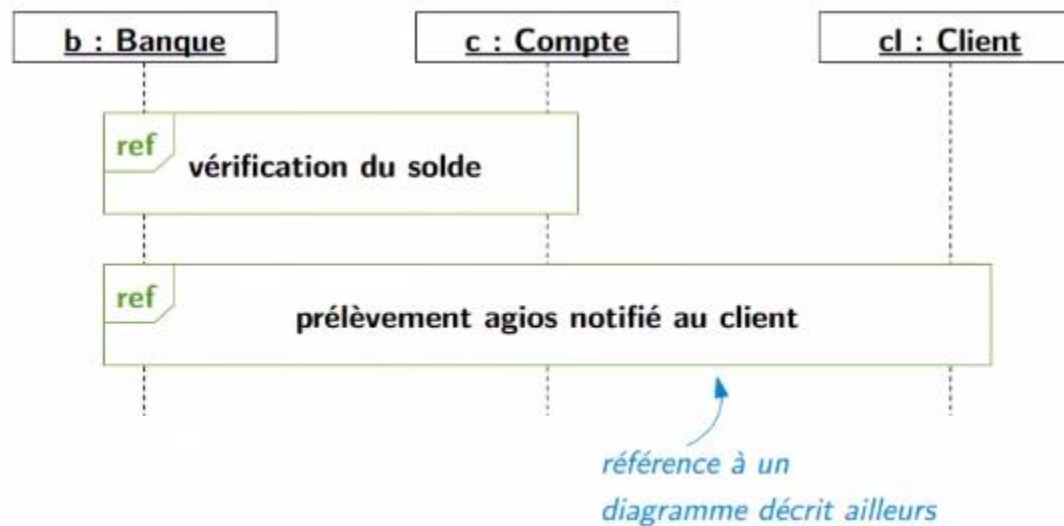


## Différents Messages: **Les traitements parallèles**

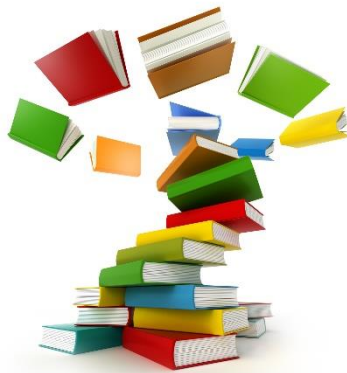
- L'opérateur **par** permet d'envoyer des messages en parallèle.
- Ce qui se passe de part et d'autre de la ligne pointillée est indépendant.



## Différents Messages: Les références vers d'autres diagrammes







*Merci*

---

*pour votre attention*

