

# ANALYSE & MODÉLISATION

## I. Diagramme des cas d'utilisation

### Les éléments des *Cas d'utilisation*

1. Les cas d'utilisation : besoin des utilisateurs du système
2. Les acteurs et les composants du système
3. La définition du système

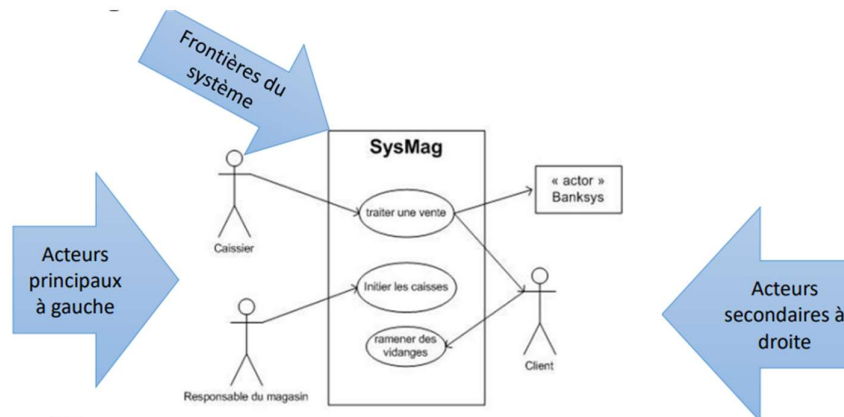
### Acteurs

== Représente un rôle joué par un user humain ou un autre système qui interagit directement avec le système étudié (message)

- **Acteur principal** == celui pour qui le cas d'utilisation produit un résultat observable.
- **Acteur secondaire** == ceux qui sont consultés pour fournir des infos complémentaires.

### • La représentation

On représente chaque cas dans un ovale + verbe à l'infinitif.



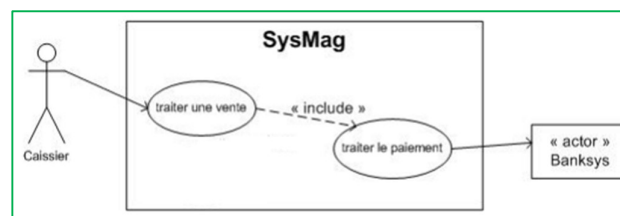
### • Les ≠ types de relations

#### A. Cas d'utilisation interne – « include »

== Un cas d'utilisat° utilise un autre cas d'utilisat° s'il fait appel à ce dernier comme à une sous-fonction. Quand un cas n'est pas directement relié à un acteur.

Le sens de la flèche : dans le sens du texte.

Ex : Cette inclusion induit qu'à chaque vente, il faut traiter le paiement !



/!\ La surcharge ! Il ne faut représenter que ce qui est intéressant et lorsque l'énoncé le justifie.

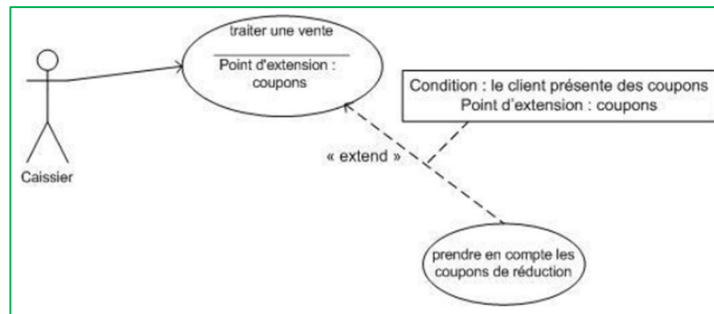
## B. Relation d'extension – « extend »

== Un cas d'utilisat° étend un autre cas d'utilisat° si dans ce dernier il est possible à un moment donné de réaliser le premier.

Le cas de base peut fonctionner tout seul, mais il peut également être complété par un autre, sous certaines conditions, et uniquement à certains points particuliers de son flot d'évènements.

Le sens de la flèche : dans le sens inverse du texte.

Ex : Cette inclusion induit qu'il n'y a pas toujours de coupons à prendre en compte dans une vente.



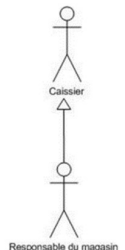
## C. Généralisation / Spécialisation

- Cas A : cas général
- Cas B : cas particulier de A
  - Le cas A est une généralisation du cas B
  - Le cas B est une spécialisation du cas A



⇒ La seule relation possible entre 2 acteurs == Généralisation/Spécialisation

- Un acteur A est une **généralisation** d'un acteur B si A peut être remplacé par B.
- Du coup, tous les cas d'utilisation accessibles à A le sont aussi à B, mais pas l'inverse.



## • Scénario

== succession particulière d'enchaînement d'actions, s'exécutant du début à la fin du cas d'utilisation. Un enchaînement ne contient ni branchement ni alternative.

== collection de scénarios possibles entre le système en construction et les acteurs externes.

### A. Scénario nominal

== celui dans lequel tout se passe bien : il conduit au bon déroulement du cas d'utilisation

Étapes	Actions
1) Demande de passer commande	
	2) Affiche la carte des pizzas
3) Choisit une pizza et complète la quantité	
Répète le point 3 pour toutes pizzas souhaitées	
4) Précise que sa commande est terminée	
	5) Affiche l'adresse de livraison
6) Confirme son adresse de livraison	

	7) Affiche toutes les pizzas de la commande et le montant total + demande de confirmation
8) Confirme sa commande	
	9) Enregistre la commande et l'attribue au client

### ➤ Présentation/Scénarios des alternatives / extensions

Étapes

Action de branchement

1a	Condition causant branchement à l'étape 1 du scénario
1a. 1	Étape 1 du scénario alternatif
1b	....

### ➔ Alternatives

3-8 [Le client abandonne] Echec du cas d'utilisation
3a [Pizza choisie non disponible]
3.a.1 Le système affiche une erreur
Branchement en 3

### ➔ Extensions

6.a [Le client ne confirme pas son adresse]
6.a.1. Le client introduit une nouvelle adresse
6.a.2. Le système vérifie le format de l'adresse
6.a.3 Le système enregistre la nouvelles adresse
Branchement en 5/7




## • Modèle de description des cas d'utilisations

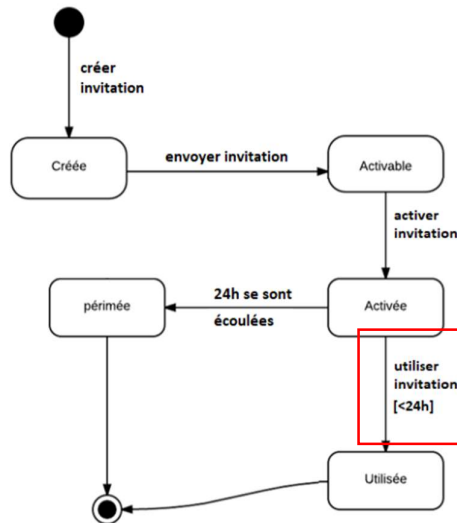
Nom du cas d'utilisation	Commencer par un verbe
Objectif	Intention principale du cas d'utilisation
Acteur principal	Celui qui fait appel au système pour obtenir un service
Parties prenantes et intérêts	Intervenants qui s'intéressent à ce use case
Pré conditions	Définition de ce qui est vrai avant le début du scénario. Testable
État après réussite	Définition de ce qui est vrai lorsque le cas d'utilisation se termine avec succès
État après échec	Définition de ce qui est vrai lorsque le cas d'utilisation ne se termine pas avec succès
Trigger	Événement déclencheur

Nom du cas d'utilisation	Traiter une vente
Objectif	
Acteur principal	Caissier
Parties prenantes et intérêts	Banksys
Pré conditions	La caisse a été initiée par le responsable de magasin, <b>le caissier est authentifié.</b>
État après réussite	La vente est enregistrée et le ticket imprimé. Les stocks ont été modifiés. Le paiement s'est déroulé avec succès.
État après échec	La vente n'est pas enregistrée. Le paiement n'a pas été effectué. Le stock n'est pas modifié.
Trigger	Le caissier initie la vente (lorsque le client se présente à la caisse mais ceci n'est pas détectable dans le système). <b>Le trigger est l'action n°1 du scénario nominal</b>

## II. Diagrammes d'états

### • La représentation

État	
État initial	
État final	



- Un administrateur crée une invitation.
- Un administrateur envoie une invitation à un client potentiel.
- Le client peut l'activer.
- Lorsqu'une invitation est activée, elle doit être utilisée endéans les 24 heures.
- Dans le cas contraire, elle est périmée.

⇒ À tout instant, **un objet se trouve dans un certain état** et le système se comportera d'une façon spécifique en réponse aux évènements se produisant

### • Transitions

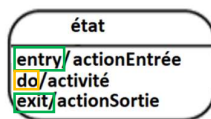
Événement [garde] / action

= changement d'état considéré comme instantané.

= **connexion unidirectionnelle** == une relation orientée entre 2 états telle qu'un passage de l'un à l'autre est possible.

- Évènement : ce qui déclenche la transaction
- [garde] : condit° qui doit être vérifiée pour que l'évènement déclenche la transition.
- Action : opération (ex : méthode en Java) membre de la classe de l'objet qui reçoit l'évènement.

### • Évènement internes



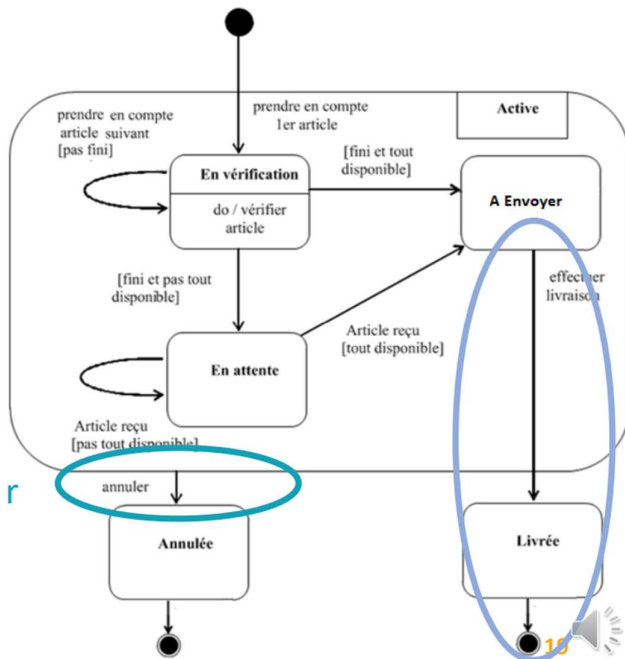
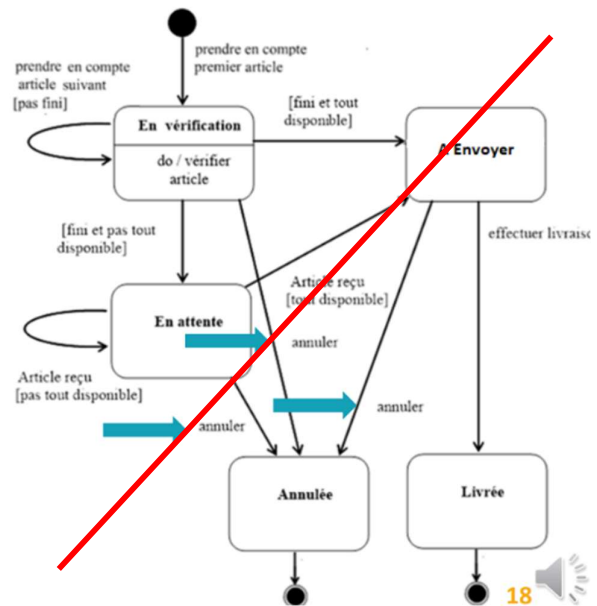
- **Action** == courte et atomique. Ne peut pas être interrompue.
  - ➔ **Déclenchée par un évènement**.
    - Soit se produit de façon **instantanée** lors d'une transition (lors de l'entrée/sortie d'un certain état).
    - Soit à l'intérieur de l'état lorsqu'un évènement apparaît.
- **Activité** = lorsqu'on est dans un certain état, on peut exécuter une activité récurrente.
  - ➔ **Qlq chose qui se déroule tant qu'un objet se trouve dans un certain état**.
    - Une activité a une durée
    - À la fin de l'activité, le système peut effectuer un changement d'état.
    - Les transitions peuvent aussi se produire lorsque l'activité est en cours. Celle-ci se termine alr et le changement d'état d'effectue.

## • Super état

== Au lieu d'avoir 3 transitions, on en fait plus qu'1 → évite un diagramme surchargée.

Exemple : On peut voir ici qu'une commande peut être annulée dans 3 états : - En vérification

- A envoyer
- En attente



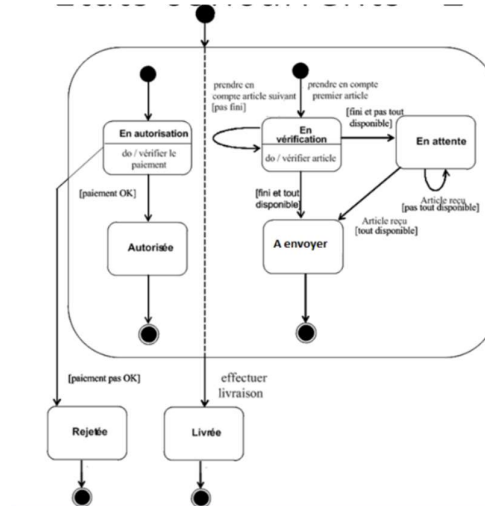
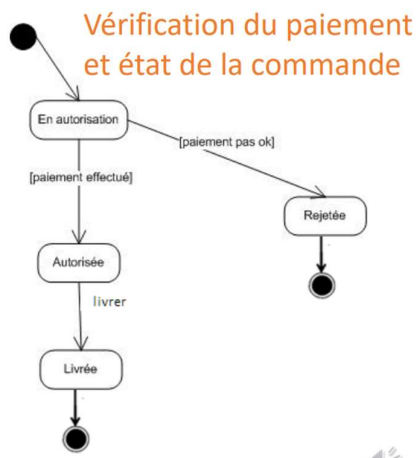
Voilà le concept du super-état. Il englobe tous les états à partir duquel la commande peut être annulée.

Il reste qu'une seule transition « annuler » à partir du super état.

/!\ On remarque que la transition « à envoyer » à « Livrée » ne part pas du super état mais de l'état « à envoyer »

## • État concurrent

Imaginons que la demande puisse être envoyée pcq ts les articles sont disponible/en stock. Mais on va aussi vérifier que la commande a été payée.



Quand la commande arrive aux deux états finaux des diagrammes internes, elle peut passer à l'état « Livrée ».

Entre-temps, les deux diagrammes peuvent se dérouler dans n'importe quel ordre.

21

## • Historique

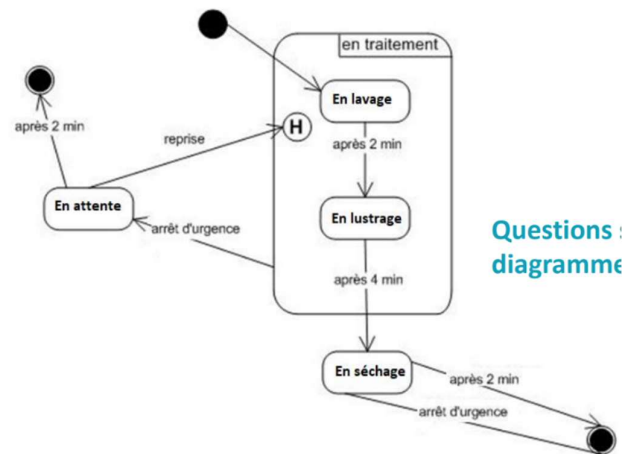
⇒ Une région d'un super-état peut contenir un indicateur d'historique



Prenons un exemple : Imaginons on va au car-Wash , dans un premier temps, on passe par l'étape de lavage, après 2min, à l'étape de lustrage, etc...

Imaginons il y a eu un arrêt d'urgence, dans ce cas, on arrive à l'état « attente », on arrête tout.




Lorsque le problème est réglé on peut reprendre le processus, mais où reprendre ? Pour savoir où on en était exactement, on place un indicateur d'historique qui nous permettra de savoir où on en était dans le processus.



Questions :  
diagramme

### III. Diagramme d'activités

- La représentation

Activité	
Activité initiale	
Activité finale	

Verbe ou substantif

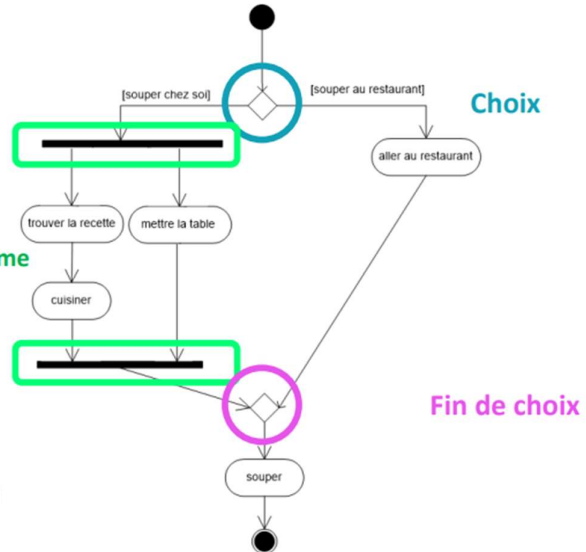


Transition (déclenchée dès que l'activité source est terminée)



Garde = condition de transition

Parallélisme

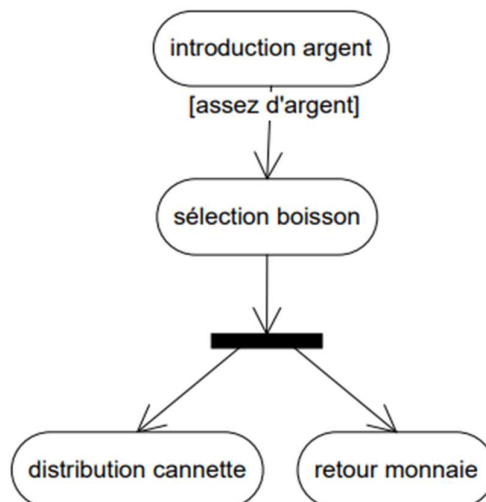


- 3 types de barres de synchronisation

- A. Synchronisation sortante/disjonctive

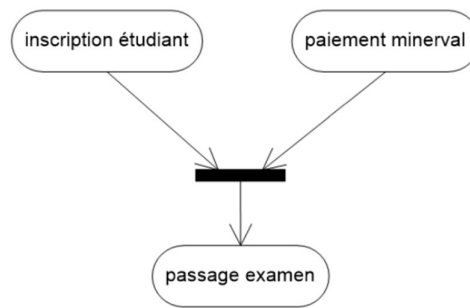
== activités pouvant se dérouler en parallèle.

== déclenchement simultanée de plusieurs transitions



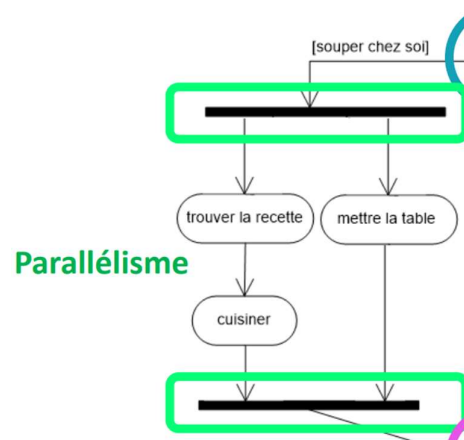
## B. Synchronisation entrante/conjonctive

=> les activités doivent se synchroniser avant de continuer le flot d'activités



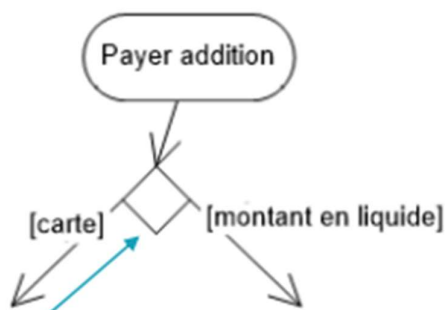
## C. Parallélisme

=> activités exécutées dans n'importe quel ordre et même en alternance



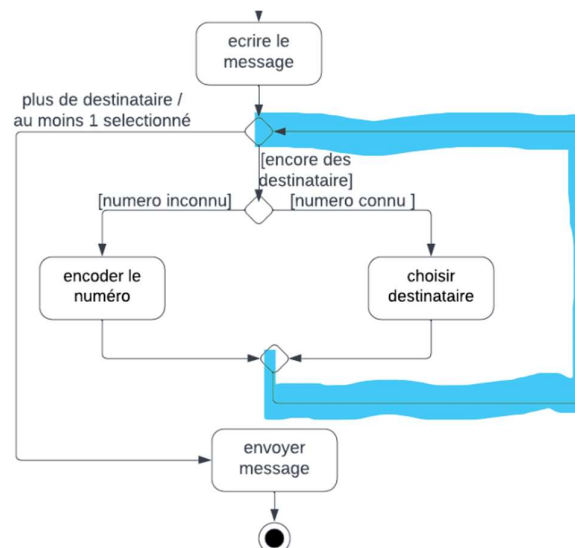
## • Nœud de décision

**Représentation** : un losange avec 1 seule transition entrante et au moins 2 transitions sortantes.



## Itération

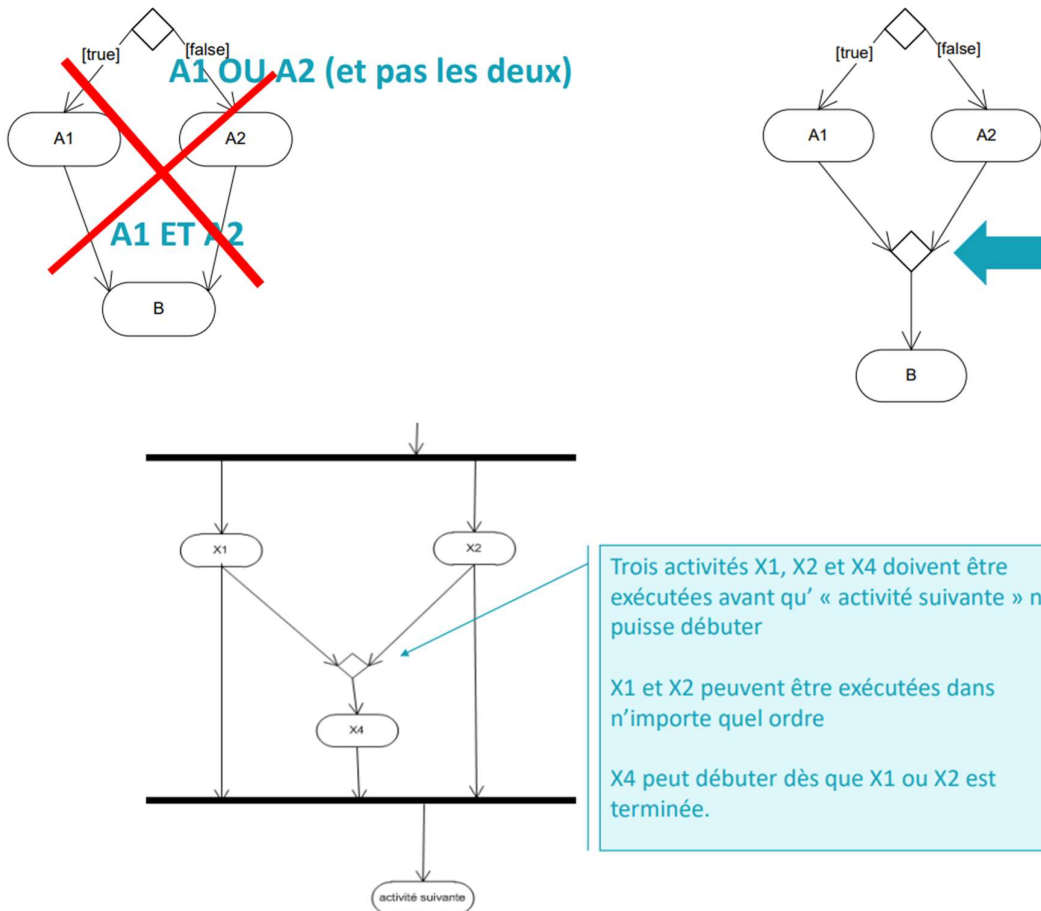
=> en gros c'est une boucle, et ça revient au nœud de décision





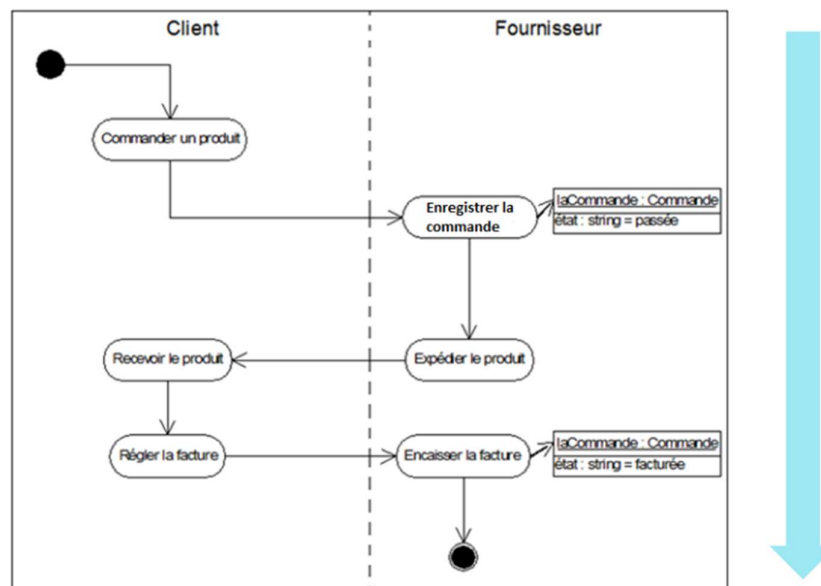
- **Nœud de fusion**

== après un nœud de décision, il faut rassembler plusieurs transitions entrantes en **1 seule** transition sortante.



- **Couloirs d'activités**

== Pour organiser un diagramme d'activités selon ≠ responsables des actions représentées.



## IV. Diagramme d'interactions

= décrire l'évolut° (== cmt les objets interagissent via messages) dans une utilisat° particulière.

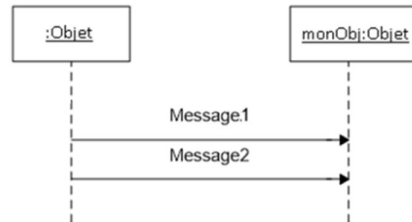
## V. Diagramme de séquences

== un des diagramme d'interaction les + utilisé. Permet de décrire l'évolution du système en présentant les interactions entre objets de manière chronologique.

⇒ Concerne les objets et non les classes : ils montrent les messages qui sont envoyés d'un objet à un autre.

### • La représentation

- Axe vertical : temps qui s'écoule (de haut en bas)
- Axe horizontal : objets concernés



#### 2 types de messages

- **Signal**
- Appel**

### • L'appel

== invocation synchrone d'une opération (*en gros c'est comme un appel de méthode en Java*).

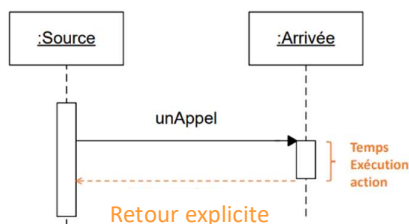
**Synchrone** == réponse attendue du récepteur ; l'émetteur perd le contrôle

### • Le signal

== communication asynchrone explicite (*en gros c'est comme au thread en Java*).

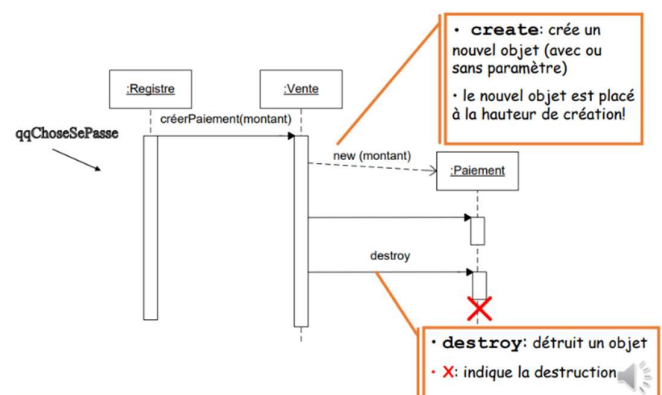
**Asynchrone** == aucune réponse attendue ; l'émetteur garde le contrôle

### • Barre d'activation/Retour

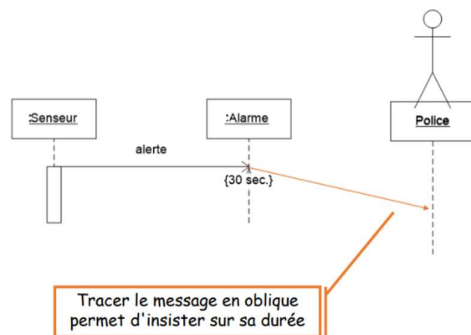


En gros mettons la valeur de retour tout le temps comme ça on ne loupe rien (Mais en fait faut la mettre que quand on a besoin de la valeur).

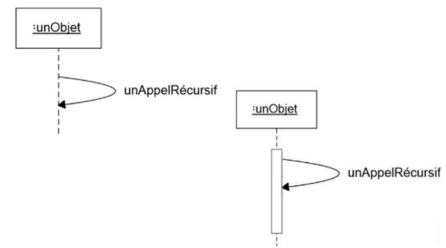
### • Création / Destruction



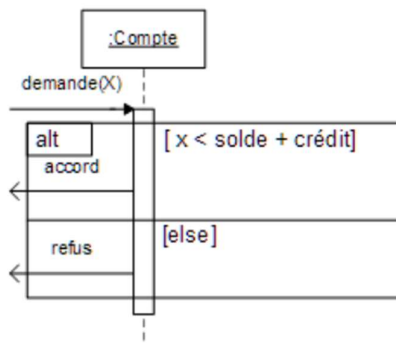
- Contraintes de temps



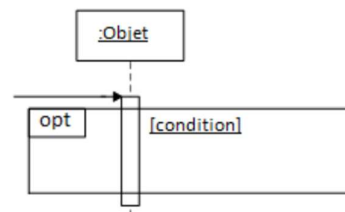
- Appel récursif



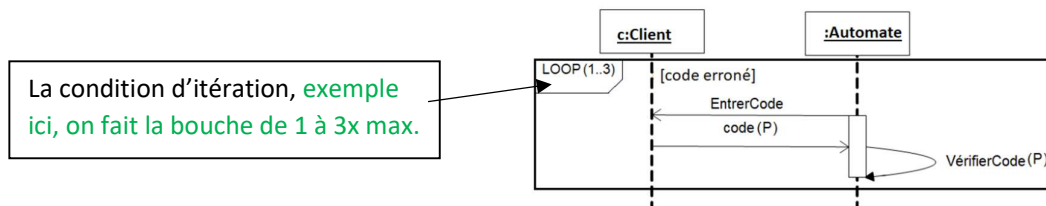
- Alternative (if...else)



- Option



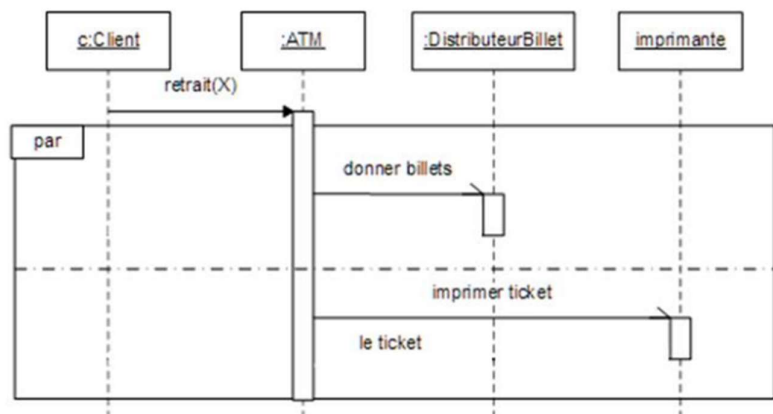
- Itération



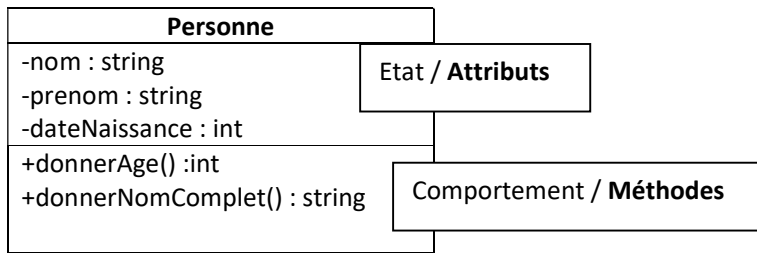
- Parallélisme

Rectangle divisée horizontalement en pointillée.

Tous les traitements qui peuvent avoir lieu en //, sont chacun isolée entre ces lignes en pointillée.



## VI. Diagramme de classes



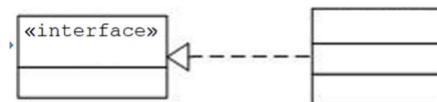
### • Classe abstraite

- ⇒ Une classe dont l'implémentation n'est pas complète.
- ⇒ Une classe qui n'est pas destinée à être instanciée (== initialiser) .
- ⇒ Toute classe qui contient une méthode abstraite



### • Interfaces

== une classe dans laquelle aucune méthode n'est implémentée, et où les champs ne sont pas indiqués.



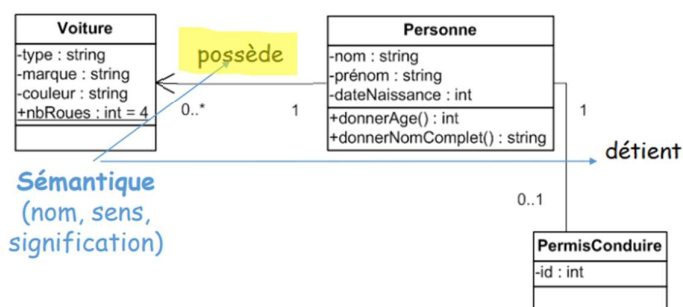
- ⇒ L'interface et toutes ses méthodes sont abstraites.

### • 3 types de relation :

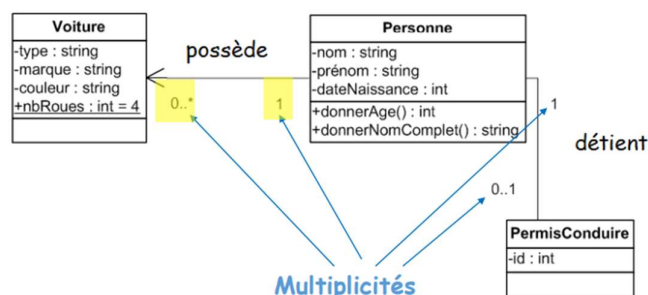
- Association.
- Agrégation / composition.
- Généralisation / spécialisation

### • Association

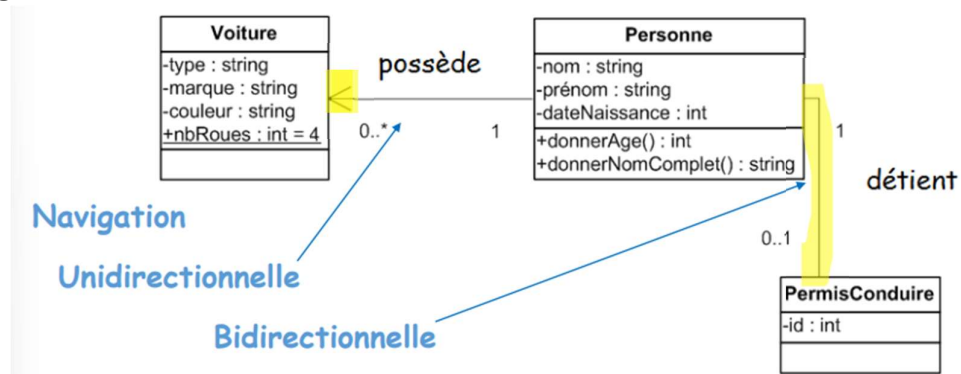
➤ **Sémantique** (décrit la signification de l'association)



➤ **Multiplicités** (décrit le nbre de fois qu'une instance peut participer à une relation)



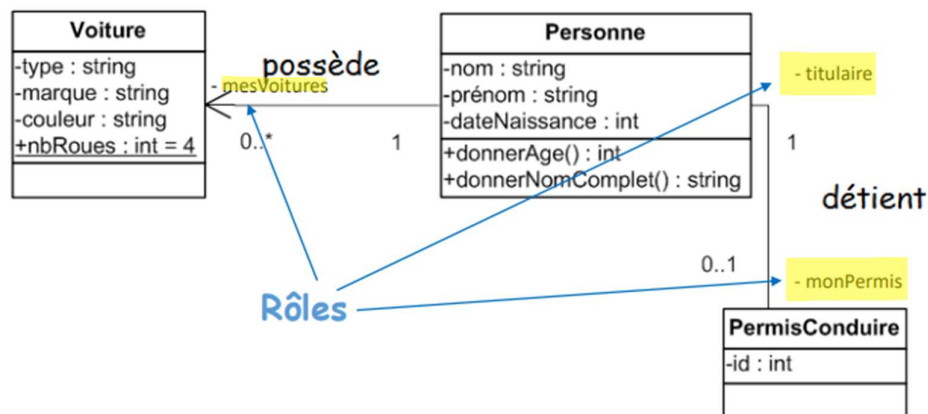
➤ **Navigation** (permet de limiter les responsabilités)



⇒ **Unidirectionnelle** == 1 classe prend la responsabilité dans l'association.

⇒ **Bidirectionnelle** == Les 2 classes prennent la responsabilité dans l'association.

➤ **Rôles** (décrit la fonction jouée par chaque classe dans l'association)



Unidirectionnelle : Une **Personne** connaîtra la liste des voitures qu'elle possède. C'est la **Personne** qui va maintenir l'association.

Bidirectionnelle : la **Personne** connaîtra son permis, et le **PermisConduire** connaîtra le titulaire qui le détient. Les 2 classes participeront à la responsabilité de maintenir l'association entre elles.

• **Classe – Association**

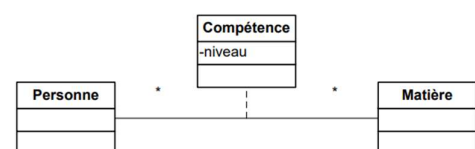
== utiliser 1 classe pour représenter 1 association.

➤ **Type 1**

On peut ajouter des attributs/opération dans cette classe-associat°.

On voit ici une associat° entre **Personne** et **Matière**. Une **Personne** a des compétences dans 1 **Matière** et on peut au niveau de la **Compétence** définir un niveau.

La **Compétence** = une classe – association entre **Personne** et **Matière**.

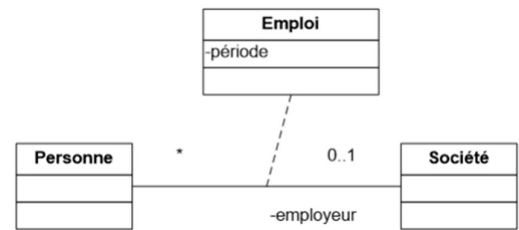


## ➤ Type 2

On pourrait dire également qu'1 **Personne** travaille pour une **Société**.

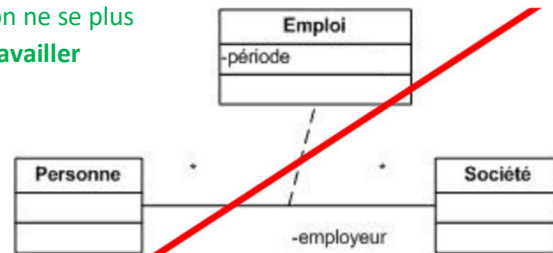
Donc on a **Personne**, **Société**, et une classe-associat° **Emploi** dans laquelle on désire préciser la période pendant laquelle la personne est employée.

La période dépend/précise l'association **Personne - Société**



## ➤ Type 3

Mnt on considère qu'une personne a plusieurs emplois dans sa vie, le fait qu'une personne pourrait retravailler plusieurs fois pour une même société (*donc il taff pour la société A, puis société B, etc, puis reviens travailler à la société A*) ➔ la classe – association ne se plus possible pcq ça **restreint une personne à ne pouvoir travailler qu'une seule fois pour une société**.

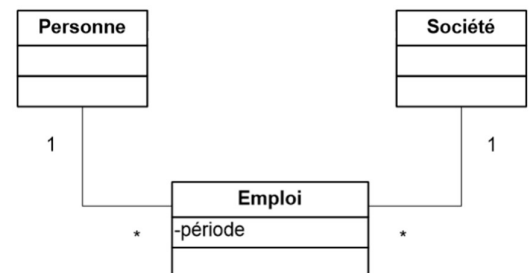


**Solution :** Utiliser une classe **Emploi** à part entière.

Donc on a 1 relation entre **Personne** et ses **Emploi** de 1 à \* (*1 Personne peut avoir plusieurs Emploi*).

1 relation entre **Société** et **Emploi** de 1 à \* (*1 Société à plusieurs employés, 1 Emploi se fait dans 1 seule Société à la fois*).

⇒ Permet à 1 **Personne** d'avoir plusieurs **Emploi** à ≠ moment dans la même **Société**.



## • Agrégation (Inclusion d'1 élément dans 1 ensemble)

Le **Point** est parti de **Polygone**. Le **Polygone** = agrégat.

On met le **losange** de ce côté de l'association pour montrer que le **Polygone** est l'agrégat de **Point**



## • Composition (Forme bcp + forte d'agrégation)

➔ Si la voiture est supprimée, le châssis l'est aussi. Le Châssis ne peut pas vivre si la Voiture ne vit pas.

➔ Du côté de l'agregation/composite, on a absolument une multiplicité de 1.



Les parties vivent et meurent avec le tout

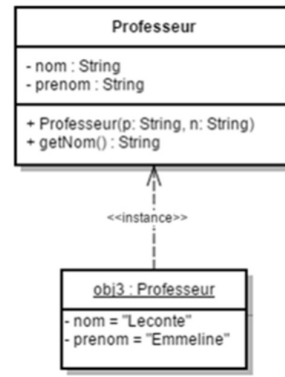
## • Classification / Instanciation

Un objet est une instance d'une classe.

L'**instanciation** d'une classe A = la création d'objet de type A.

La **classification** réfère à la relation entre un objet et son type.

Un objet possède un type d'une certaine classe.



## • OCL (Object Constraint Language)

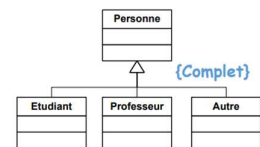
=> façon d'écrire entre {}, toutes les contraintes qu'on veut venir ajouter sur 1 diagramme.

Ex de {Complet}, dans une école, une **Personne** est soit **Etudiant** soit **Professeur** soit **Autre**.

Ex de {Incomplet}, un **Sportif**, on dit qu'il y a **Nageur**, **Danseur**, **Boxeur**, on ne précise pas Autres, il existe encore pleins d'autres sortent de sportif donc c'est une spécialisation incomplète.

## OCL exemple Spécialisation complète/incomplète

• **Complète** : l'ensemble des classes spécialisant la classe mère est complet. Il n'existe pas d'autres classes la spécialisant.



• **Incomplète** : inverse de complète.

