

# UML

---

# UML

# Sommaire

1. Introduction à UML
2. Diagramme de cas d'utilisation
3. Diagramme d'activité
4. Diagramme de séquence
5. Diagramme de classe
6. Diagramme d'objet
7. Diagramme de composant

# Introduction à UML

## Qu'est-ce que UML ?

Le Unified Modeling Language™ (UML®) est un langage de modélisation visuelle standard destiné à être utilisé pour :

- la **modélisation des processus** commerciaux et autres processus similaires
- **l'analyse, la conception et la mise en œuvre** de systèmes basés sur des logiciels

## A quoi sert-il ?

Il est utilisé pour : **décrire, spécifier, concevoir** et **documenter** les processus d'entreprise existants ou nouveaux, la structure et le comportement des artefacts des systèmes logiciels

Il est utilisé par **les analystes commerciaux, les architectes** et **les développeurs de logiciels**

## A propos d'UML

- La version actuelle d'UML est **UML 2.5**, publiée en **juin 2015**
- Les mises à jour sont gérées par le groupe de gestion des objets (**OMG™**) **OMG UML**
- Les premières versions d'UML ont été créées par les "**Three Amigos**"

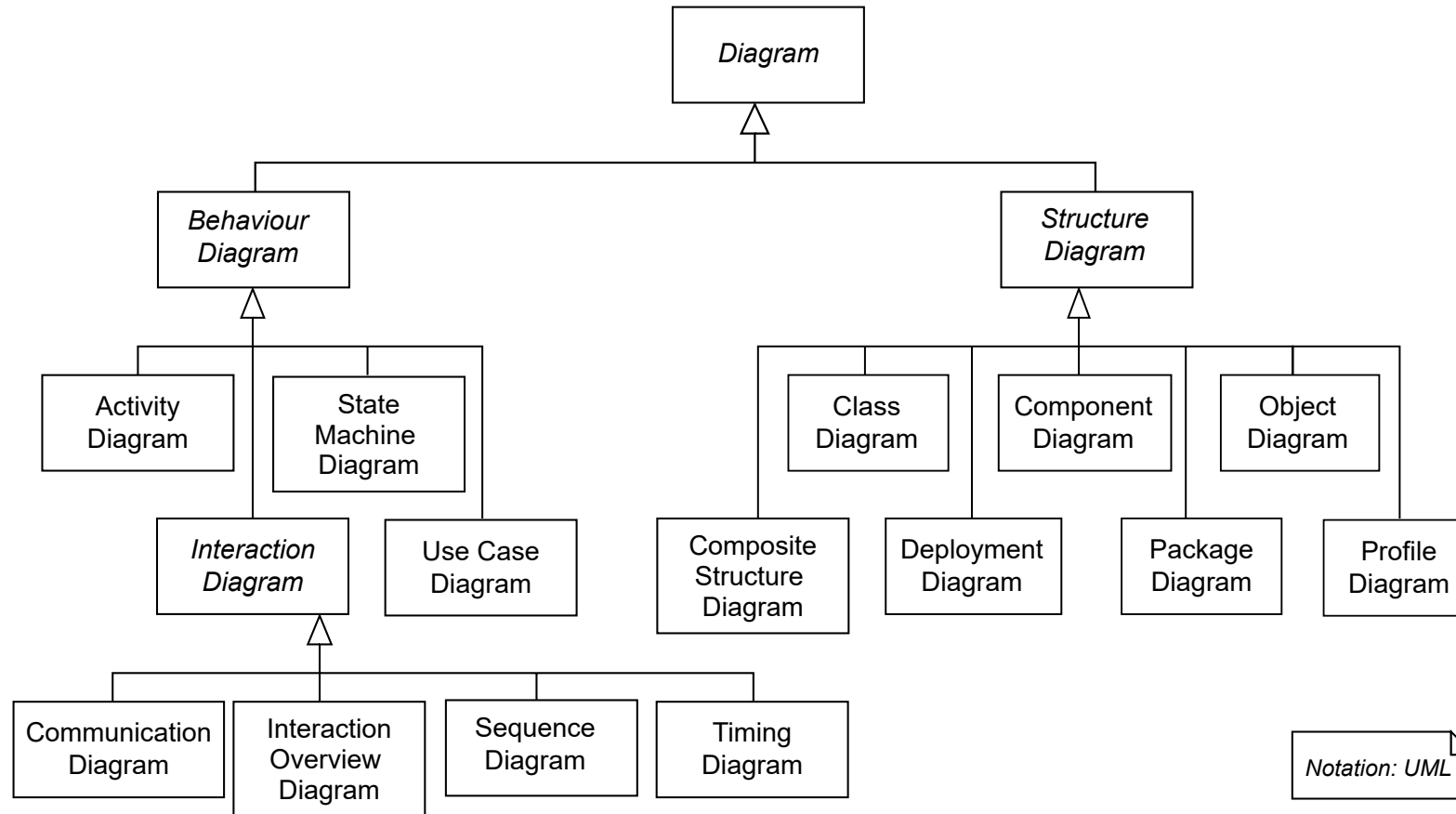


# Les familles de diagrammes

- La spécification UML définit deux types principaux de diagrammes UML :
  1. **Les diagrammes de structure** montrent la structure statique du système et de ses parties à différents niveaux d'abstraction et de mise en œuvre, ainsi que la manière dont ils sont liés les uns aux autres
  2. **Les diagrammes de comportement** montrent le comportement dynamique des objets d'un système, qui peut être décrit comme une série de changements apportés au système au fil du temps



# Liste des diagrammes UML



## Les outils recommandés

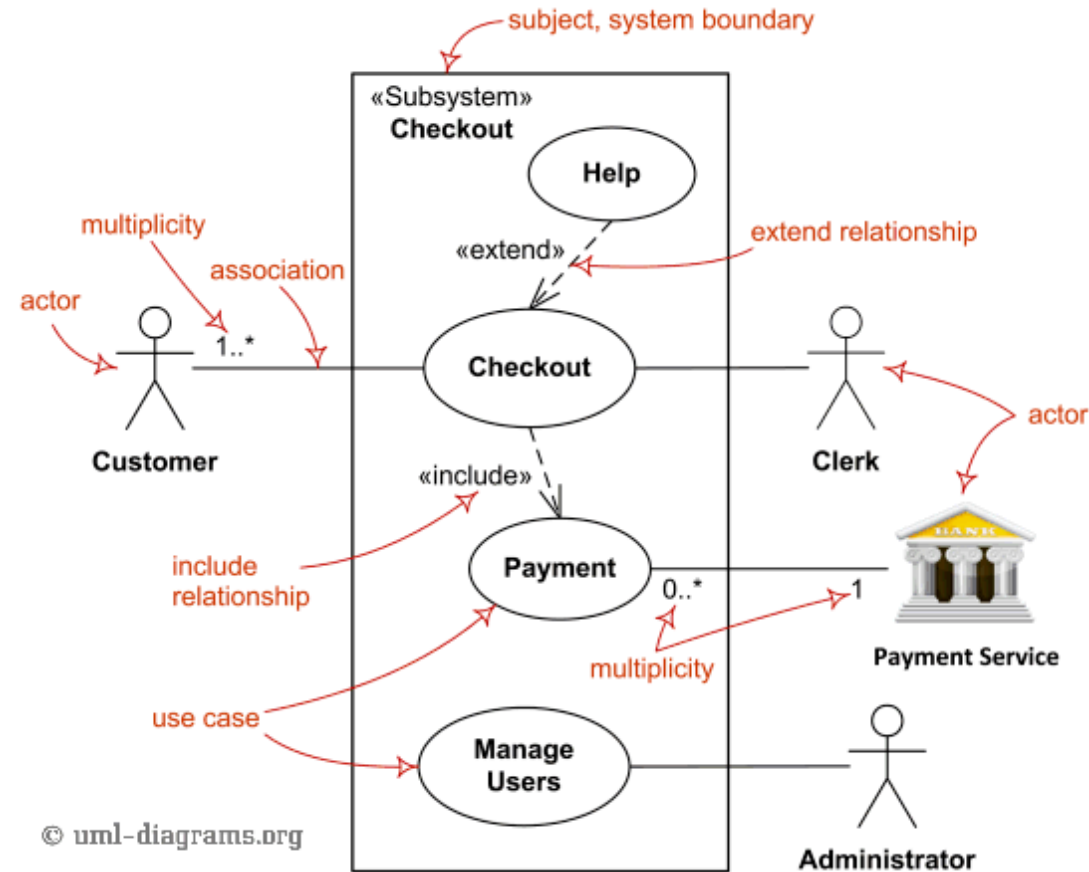
- [Draw.io](#): logiciel de diagramme en ligne gratuit
- [PlantText](#): outil en ligne qui génère des diagrammes à partir de la syntaxe PlantUML
- [Meirmaid.js](#): outil en ligne de text to diagram inspiré du markdown

# Diagramme de cas d'utilisation

# Définition

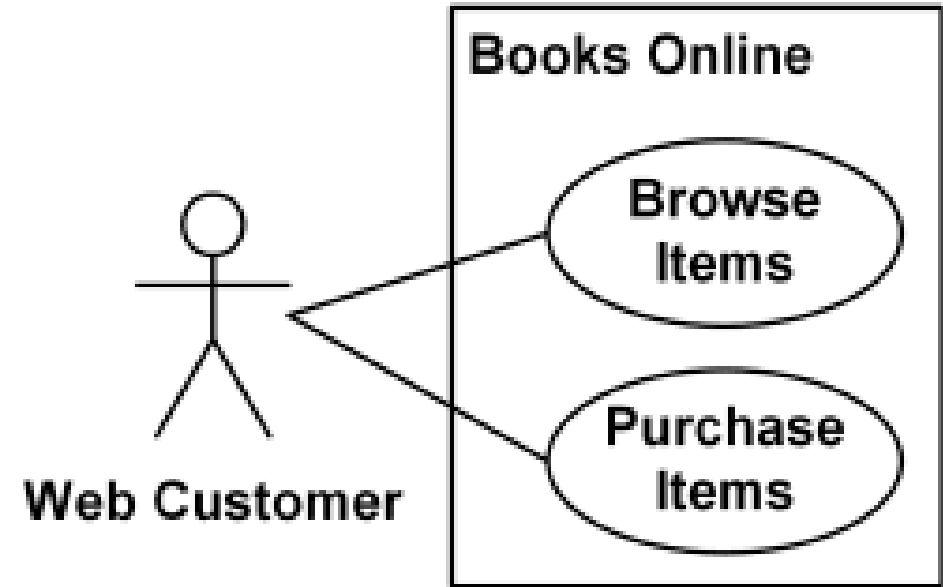
- Décrit un ensemble d'actions (**cas d'utilisation**) qu'un ou plusieurs systèmes (**sujet**) devraient ou peuvent effectuer en collaboration avec un ou plusieurs utilisateurs externes du système (**acteurs**) afin de fournir des résultats observables et utiles aux acteurs ou autres parties prenantes du ou des systèmes
- les diagrammes de cas d'utilisation sont une spécialisation des diagrammes de classes (*UML 2.4.1*)

# Diagramme de cas d'utilisation



# Sujet

- Le sujet d'un cas d'utilisation définit et représente les limites d'une entreprise, d'un système logiciel, d'un système physique
- Le sujet est représenté par un **rectangle** avec le nom du sujet situé en haut à gauche



## Stéréotype de sujet

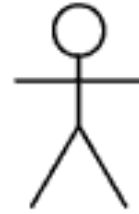
Les sujet peuvent caractériser un système, un sous-système, un composant ou un dispositif logiciel et/ou matériel, ex :

- Site web
- Système de paiement
- Distributeur automatique de billets (DAB)
- Terminal de point de vente (TPV)

Pour classifier ces différents types de sujet, UML nous fournit quelques stéréotypes: «Subsystem», «Process», «Service», «Component»

# Acteur

- Un acteur est une entité externe qui interagit avec le sujet
- Il peut être un humain, un système ou une entité utilisant le service



**Student**

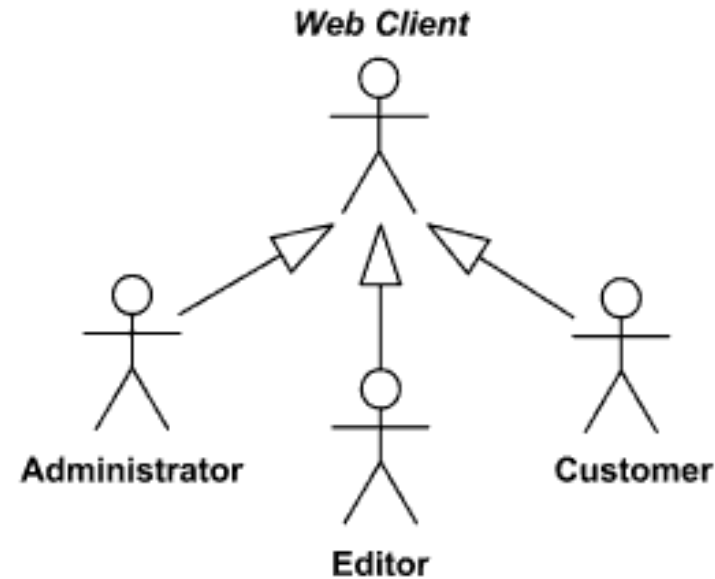


**Bank**



# Relations entre les acteurs

- Nous pouvons définir des acteurs abstraits ou concrets et les spécialiser en utilisant la relation de généralisation
- La généralisation entre acteurs est représentée par une ligne continue dirigée avec une grande pointe de flèche



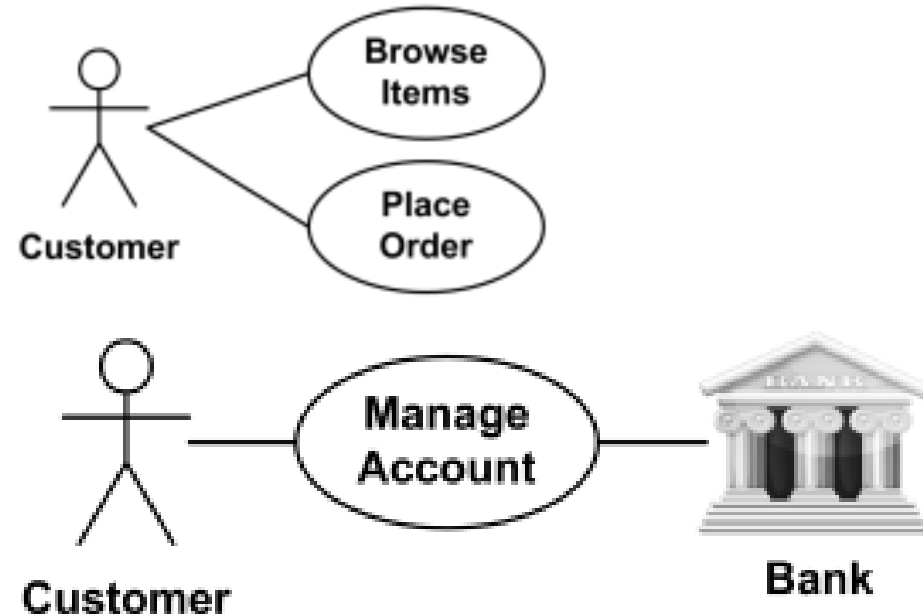
## Cas d'utilisation

- Le cas d'utilisation est généralement représenté par une ellipse contenant le nom du cas d'utilisation
- chaque cas d'utilisation doit avoir un nom



# Association

- Une association entre un acteur et un cas d'utilisation indique que l'acteur et le cas d'utilisation interagissent ou communiquent d'une manière ou d'une autre
- Un acteur peut être associé à un ou plusieurs cas d'utilisation



## "Extend" d'un cas d'utilisation

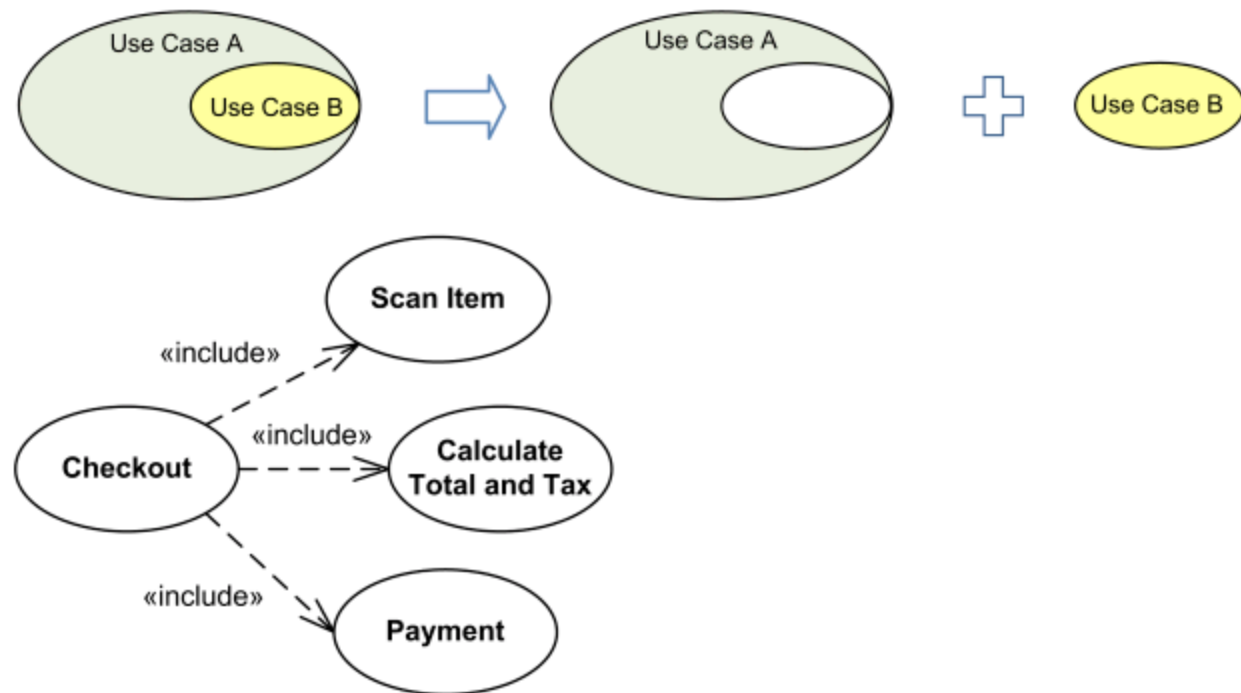
- La relation **Extend** modélise une extension possible d'un cas d'utilisation principal
- Elle indique qu'un cas d'utilisation secondaire peut se produire en plus du cas d'utilisation principal, sous certaines conditions



# "Include" d'un cas d'utilisation

**Include** permet de :

- simplifier un grand cas d'utilisation en le divisant en plusieurs cas d'utilisation
- extraire des parties communes des comportements de deux ou plusieurs cas d'utilisation



## La difficulté à modéliser un Use Case

Trouver le bon niveau de détail est difficile. Trop de détails peuvent rendre le diagramme compliqué, et trop peu peuvent le rendre inutile.

**Conséquence** : Un diagramme trop détaillé devient illisible et un diagramme trop simple ne capte pas les besoins réels.

**Solution** : Garder les cas d'utilisation **simples et compréhensibles**, en se concentrant sur les interactions principales. Les détails peuvent être décrits dans des textes explicatifs.

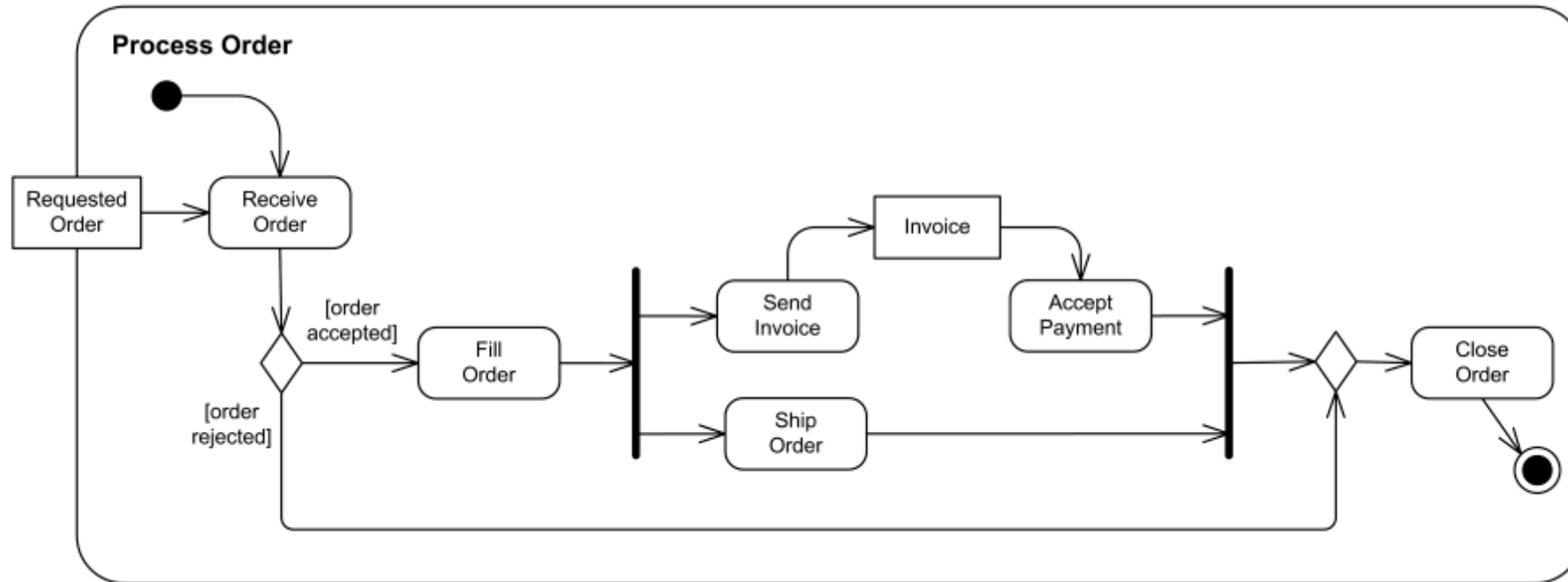
# Diagramme d'activité

# Définition

- Le diagramme d'activité décrit les **flux de contrôle** et de **données** entre **les actions, les décisions, les événements** et **les objets** impliqués dans **un processus**
- Un diagramme d'activité peut être utilisé pour décrire **les scénarios d'utilisation**, les cas de test, **les algorithmes** ou les workflows



# Diagramme d'activité



# Une activité

- Une activité représente un **ensemble d'actions coordonnées**.
- Le flux d'actions est modélisé par des nœuds d'activité **reliés** entre eux **par des liens**.
- Un nœud peut représenter **une action** (comme un calcul ou un appel de fonction), mais aussi des **éléments de contrôle du flux** comme la synchronisation, la prise de décision, ou la gestion de tâches parallèles.

## Description d'une activité

L'activité contient des nœuds d'activité qui peuvent être:

- action
- objet
- contrôle

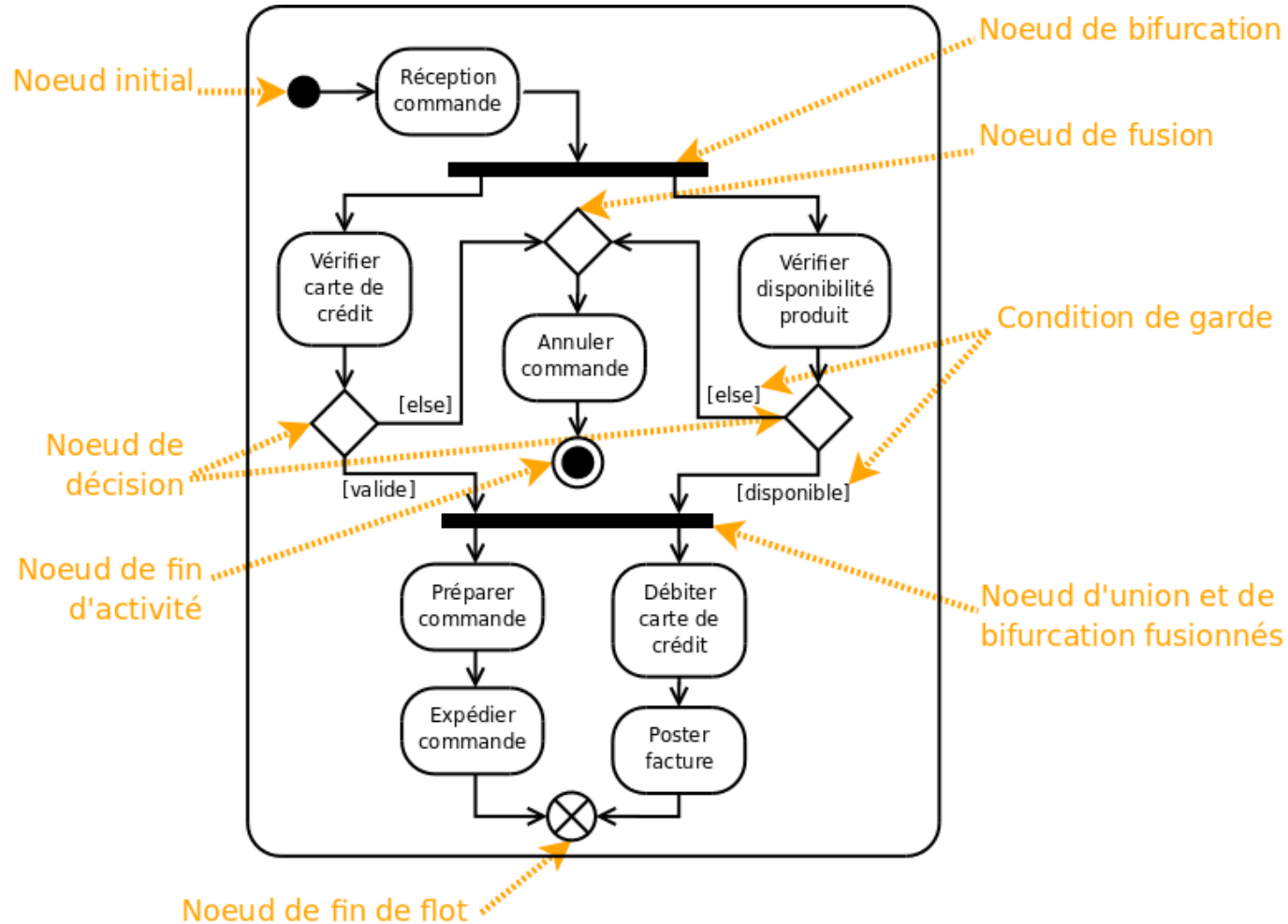
L'activité peut être représentée sous la forme d'un **rectangle** aux **angles arrondis**, avec le nom de l'activité dans le coin supérieur gauche et les nœuds et les bords de l'activité à l'intérieur de la bordure.

# Action

- L'action est un élément nommé qui représente une étape atomique unique au sein de l'activité
- Les actions sont notées sous forme de rectangles à coins ronds
- Une nomme une action par un verbe d'action



# Les nœuds des contrôles d'activité



## Nœud de départ et de fin d'activité

- Le de départ symbolise le début de l'activité, que l'on représente par un cercle noir



- Le de fin symbolise l'étape finale de l'activité, que l'on représente par un cercle noir avec un contour



## Nœud de fin de flux

- Le nœud de fin de flux est un nœud de contrôle qui met fin à un flux.
- Il détruit tous les jetons qui lui parviennent, mais n'a aucun effet sur les autres flux de l'activité.
- La notation du nœud final de flux est un petit cercle avec un X à l'intérieur.



# Différence entre fin d'activité et de flux

## 1. Le nœud de fin d'activité:

Quand le flux atteint ce nœud, tous les autres flux en cours dans l'activité s'arrêtent.

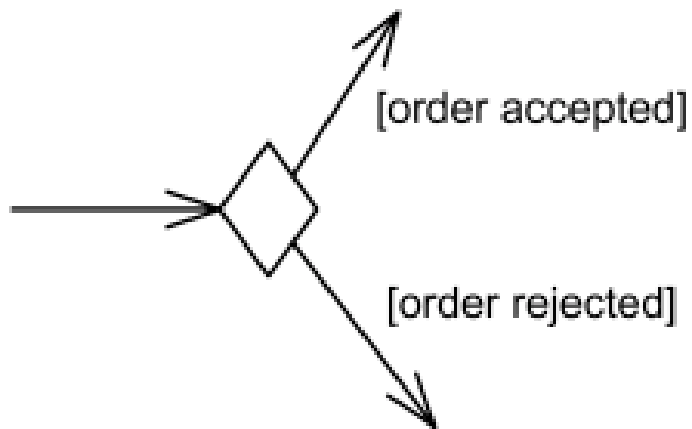
## 2. Le nœud de fin de flux:

Quand le flux atteint ce nœud, les autres flux peuvent continuer à s'exécuter. Il est utilisé pour arrêter un **chemin spécifique**.



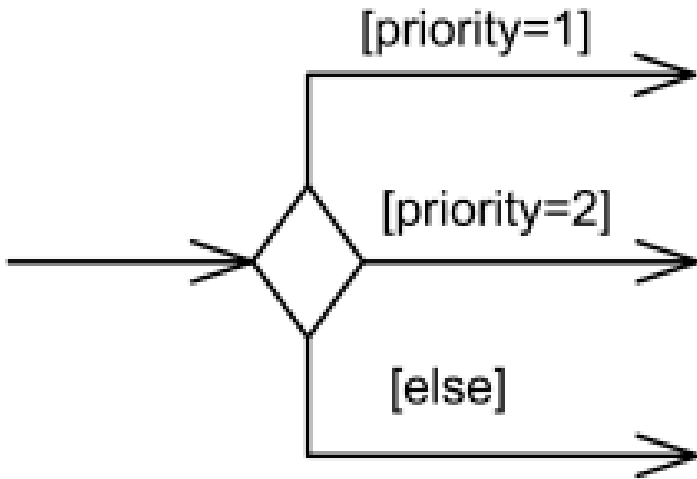
## Nœud de décision

- Embranchement conditionnel dans le flux, qui est représenté par un losange
- Il comporte **une** ou **deux entrées** et **au moins deux sorties**
- Le chemin choisi dépend des "guard" notées entre `[]` sur les sorties



## Nœud de décision à plusieurs sorties

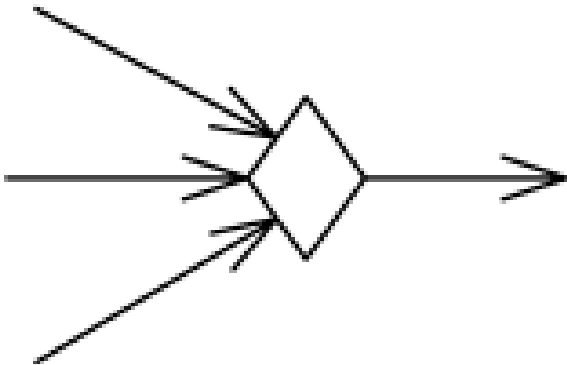
- La modélisation doit être réalisé de sorte qu'un jeton en entrée ne doit prendre qu'une seule sortie sur un nœud de décision
- Une seule clause `else` peut être indiquée en tant que sortie



## Nœud de fusion

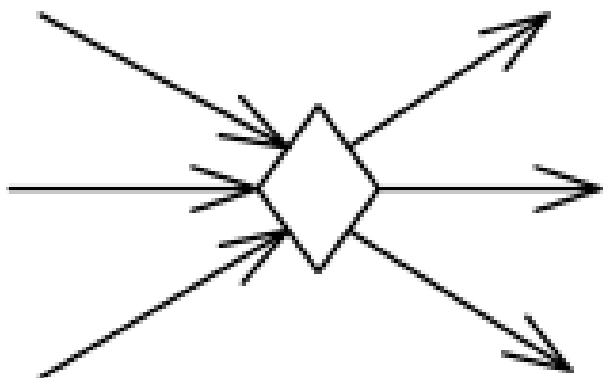
Ce nœud de contrôle rassemble plusieurs flux alternatifs entrants pour accepter **un seul flux sortant**.

Toutes les flux entrant et sortant d'un nœud de fusion doivent être des **flux d'objets** ou des **flux de contrôle**.



## Nœud de décision et fusion combiné

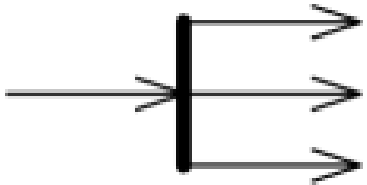
Les fonctionnalités du **nœud de fusion** et du **nœud de décision** peuvent être combinées en un seul symbol:



## Nœud de fork

Le nœud de bifurcation (fork) est un nœud de contrôle qui comporte un bord entrant et plusieurs bords sortants et est utilisé pour **diviser le flux entrant en plusieurs flux simultanés**.

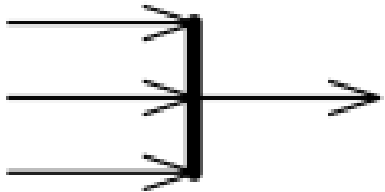
Les nœuds de fork sont introduits pour prendre en charge le **parallélisme** dans les activités.



## Nœud d'union

Le nœud d'union (join) est un nœud de contrôle qui comporte **plusieurs arêtes entrantes** et **une arête sortante** et est utilisé pour **synchroniser les flux simultanés entrants**.

Les nœuds de jointure sont introduits pour prendre en charge le parallélisme dans les activités.



## Nœud d'objet

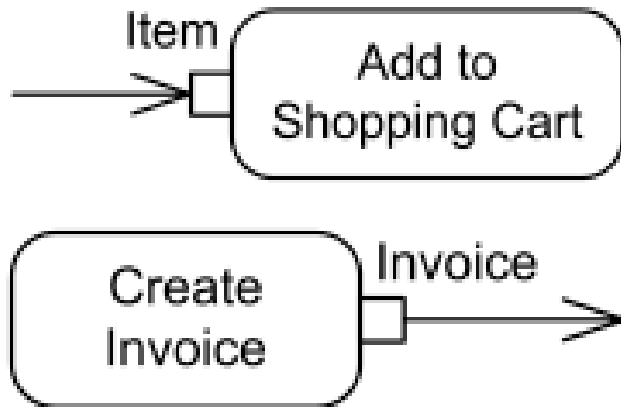
Un nœud d'objet est un élément abstrait qui permet de définir le flux des objets dans une activité.

Ils sont utilisés pour **montrer** comment les objets sont **créés**, **modifiés**, ou **utilisés** par différentes actions dans le flux d'activité.

## Les pin

Une pin est un **nœud d'objet** pour les **entrées** et les **sorties** aux actions.

Les pins aident à clarifier le flux des données et à rendre le diagramme plus lisible en montrant explicitement où et comment les objets sont manipulés

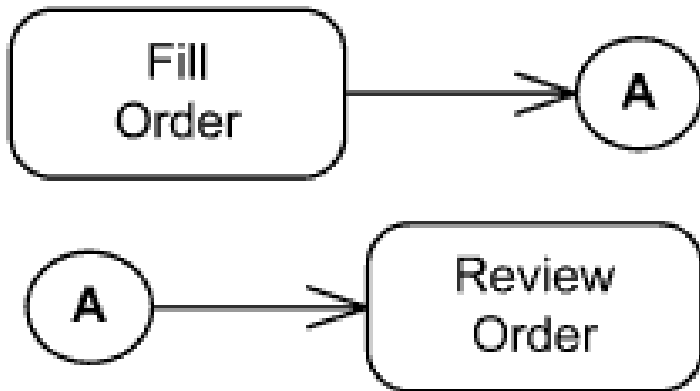




## Les connecteurs

Une limite d'activité peut être noté à l'aide d'un **connecteur**, qui est un petit cercle avec un nom à l'intérieur.

Les connecteurs sont généralement utilisés pour **éviter de dessiner de longue connexion**. Ils sont purement esthétique.

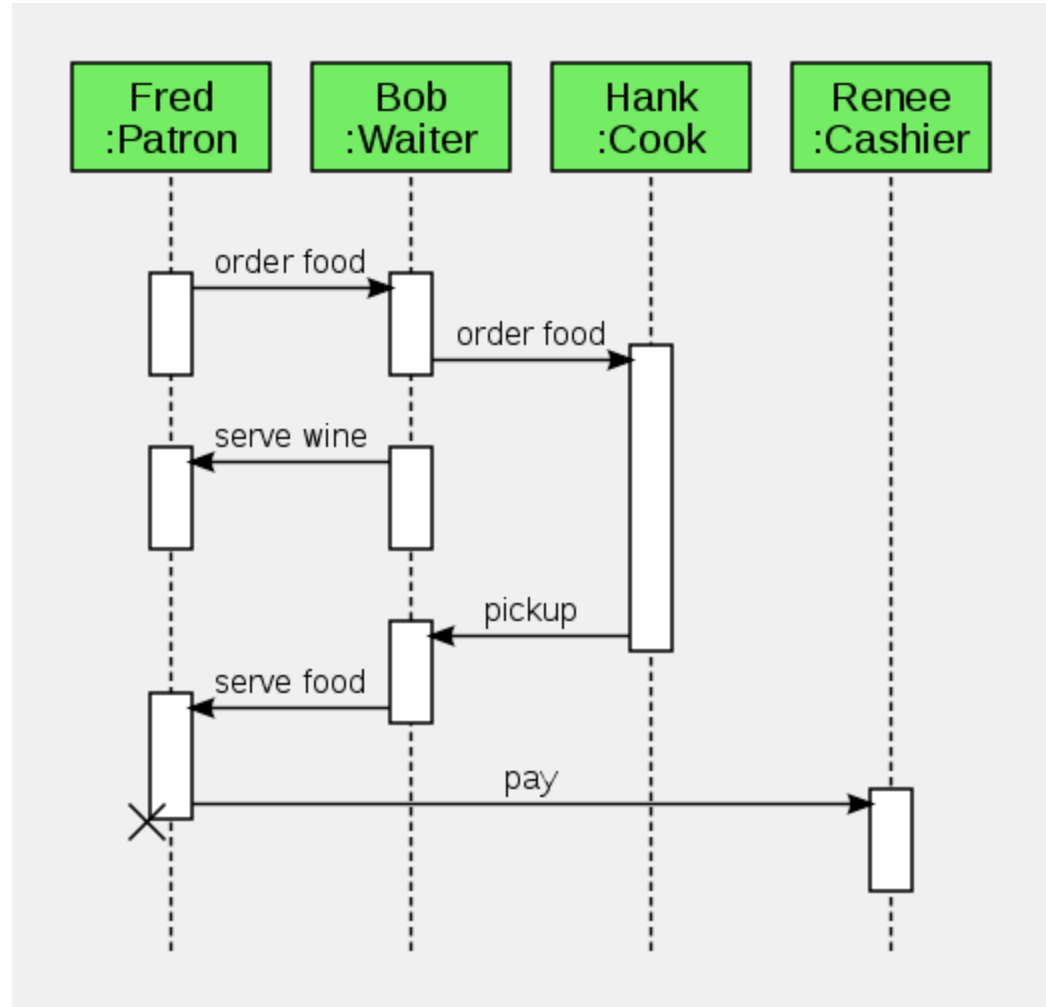


# Diagramme de séquence

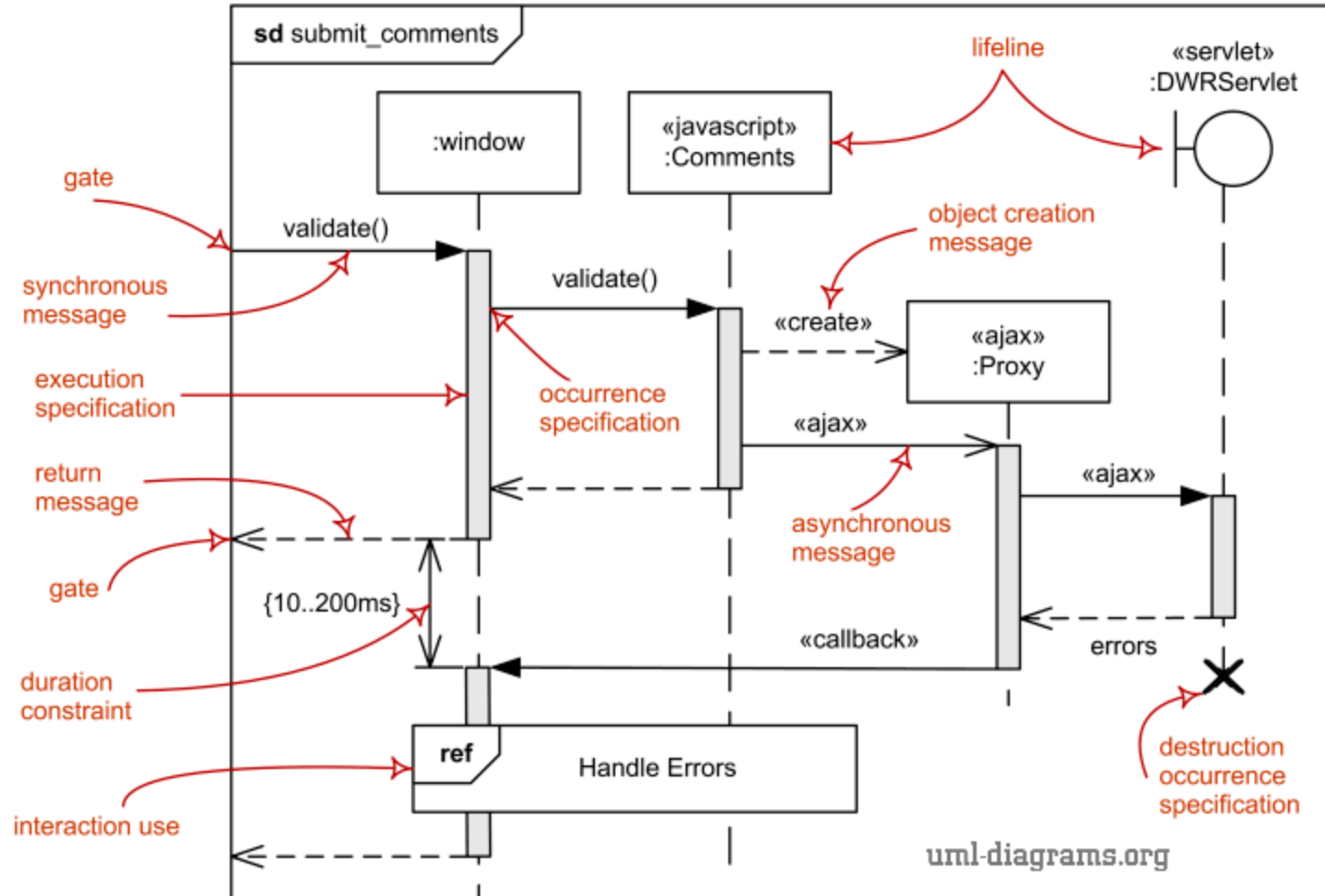
# Définition

- Les diagrammes de séquences sont la représentation graphique des interactions entre **les acteurs** et **le système** selon un **ordre chronologique** dans la formulation UML
- Le diagramme de séquence permet de montrer les **interactions d'objets** dans le cadre d'un scénario d'un diagramme **des cas d'utilisation**
- Le but est de décrire comment se déroulent les interactions entre les acteurs ou objets

# Diagramme de séquence

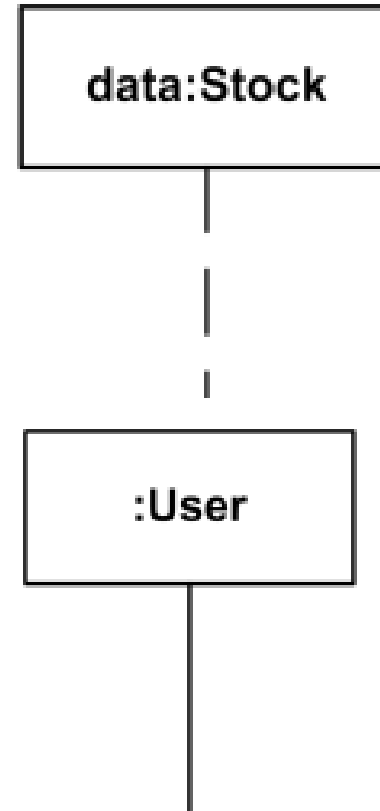


# Elements d'un diagramme de séquence



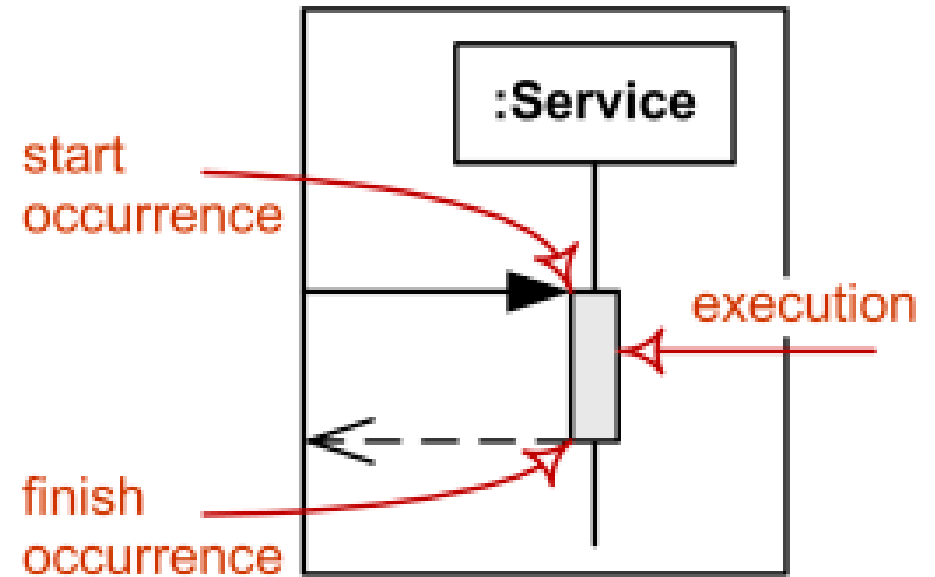
# Objet

- Représente une classe ou un objet en langage UML
- Le symbole objet montre comment un objet va se comporter dans le contexte du système
- Un objet peut être anonyme



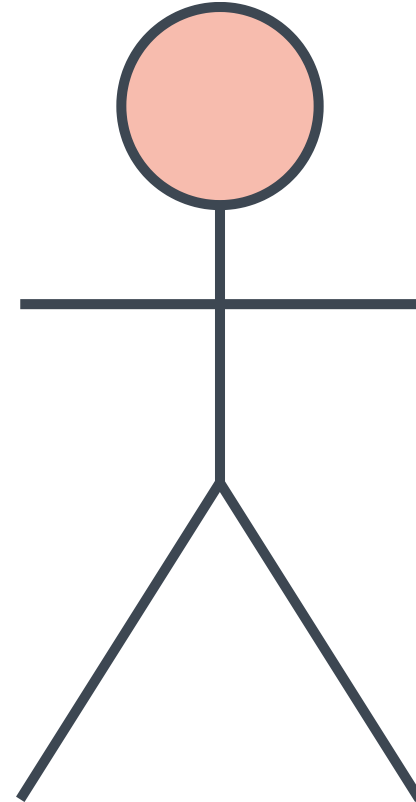
## Boîte d'activation

- Représente le temps nécessaire pour qu'un objet accomplisse une tâche
- Plus la tâche nécessite de temps, plus la boîte d'activation est longue



# Acteur

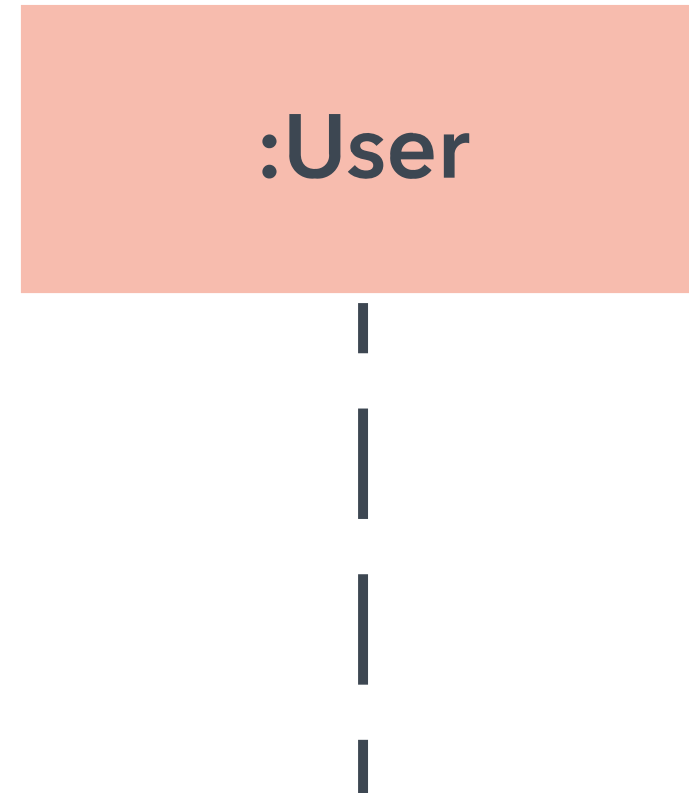
Montre les entités qui interagissent avec le système ou qui sont extérieures à lui








## Ligne de vie

- Représente le passage du temps qui se prolonge vers le bas.
- Cette ligne verticale en pointillés montre les événements séquentiels affectant un objet au cours du processus schématisé



## Messages courants

- Message synchrone : 
- Message asynchrone : 
- Réponse (synchrone ou asynchrone) : 
- Symbole de suppression d'objet :



## Fragments combinés

- Un fragment combiné est représenté par **un rectangle** qui englobe une partie du diagramme de séquence.
- Il contient des opérandes d'interaction et est défini par un opérateur d'interaction.
- Les opérateurs d'interaction déterminent le type de logique appliquée, comme les conditions (if-else), les boucles (for, while), ou les parallélismes.

## Opérateur d'interaction: loop

L'opérateur d'interaction loop signifie que le fragment combiné représente une **boucle**.

L'opérande boucle sera répété **un certain nombre de fois**.

Loop

[Condition]

## Opérateur d'interaction: alternative

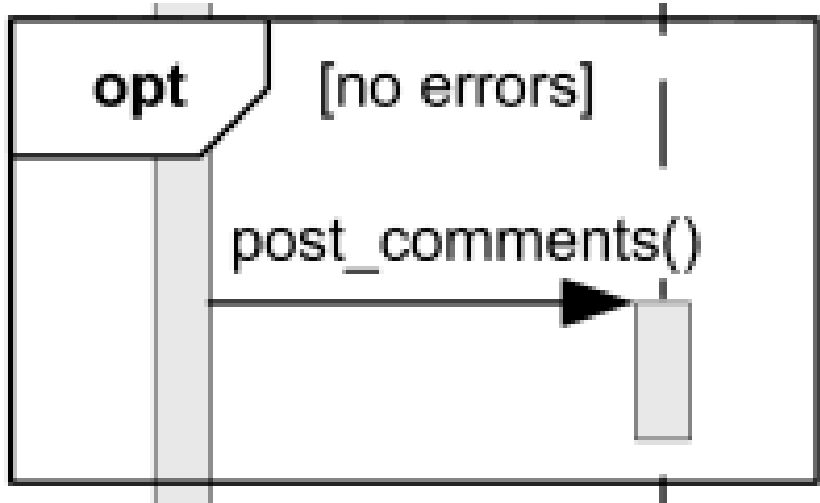
L'opérateur d'interaction **alt** signifie que le fragment combiné représente un choix ou des alternatives de comportement.

L'un des opérandes au plus sera choisi.

L'opérande choisi doit avoir une expression explicite ou implicite qui est évalué.



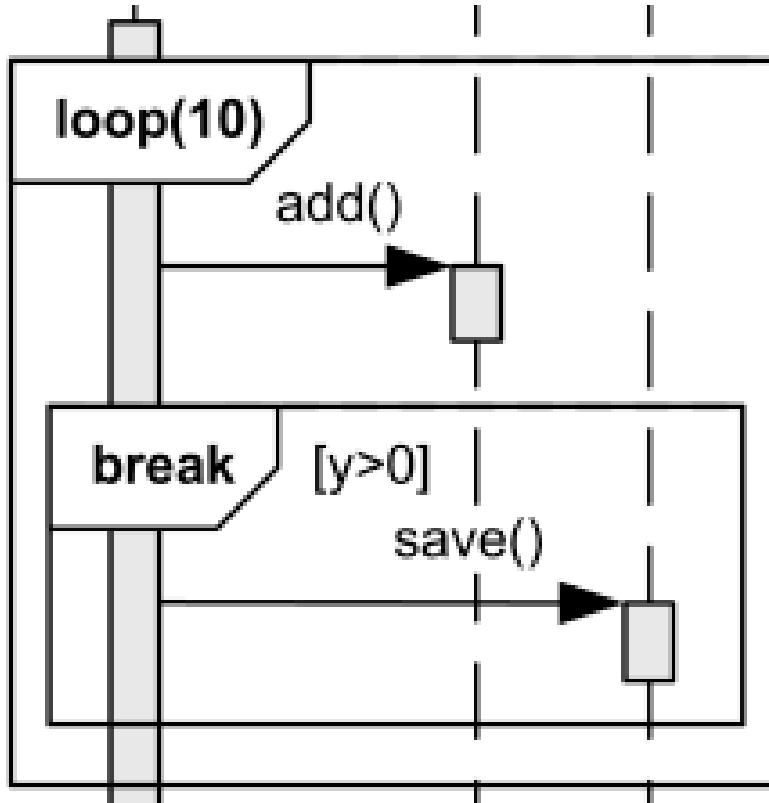
## Opérateur d'interaction: option



L'opérateur d'interaction `opt` signifie que le fragment combiné représente un choix de comportement où soit **l'opérande (unique) se produit**, soit **rien ne se produit**.

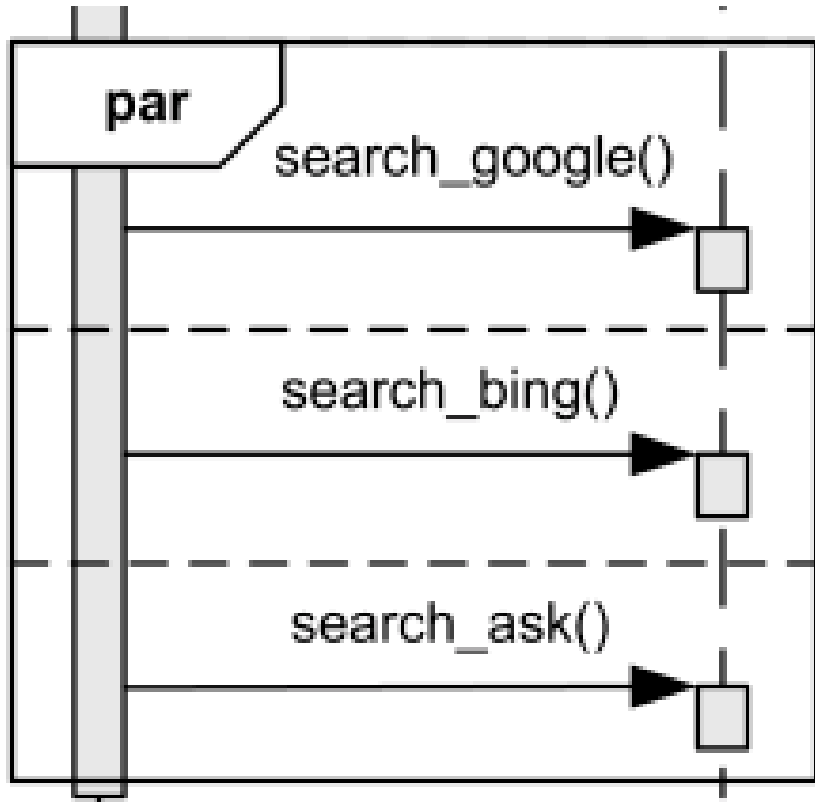
L'opérateur option est similaire à l'opérateur alt sans le second opérande.

## Opérateur d'interaction: break



L'opérateur d'interaction break représente un **scénario de rupture** ou **d'exception** qui est exécuté à la place du reste du fragment d'interaction qui l'entoure.

# Opérateur d'interaction: **par**



L'opérateur d'interaction **par** définit **l'exécution potentiellement parallèle** des comportements des opérandes du fragment combiné.



# Diagramme de classe

## Définition

Le diagramme de classes est un diagramme qui montre la **structure du système** conçu au niveau des **classes** et des **interfaces**, ainsi que leurs caractéristiques, **contraintes** et **relations** - associations, généralisations, dépendances, etc.



## Classificateur (ou classifier)

Dans un diagramme de classes, un classificateur est un concept abstrait qui représente un **ensemble d'instances partageant des caractéristiques communes**.

Les classificateurs courants incluent les classes, les interfaces, les types de données et les composants.

Dans les diagrammes, les classificateurs sont généralement représentés par des **rectangles** avec le **nom** du classificateur.

# Classe

Une classe est représentée par un rectangle séparé en trois parties :

SearchService
engine: SearchEngine query: SearchRequest
search()

- la première partie contient le **nom de la classe**
- la seconde contient les **attributs de la classe**
- la dernière contient les **opérations de la classe**

## Définition des attributs

La syntaxe pour définir les attributs dans une classe se fait de la manière suivante:

`visibilité nom : type = valeur initiale {propriétés}`

Les éléments obligatoire sont la visibilité, le nom et le type de l'attribut.

Exemple:

```
- nom : String = "Inconnu" {readonly}  
+ age : int = 0  
# adresse : String
```

# Définition des opérations

La définition des opérations se fait de la manière suivante:

visibilité nom (<paramètres>) : type de retour {propriétés}

Exemple:

```
+ calculerSurface(longueur: double, largeur: double) : double  
- validerSaisie(saisie: String) : boolean  
# traiterCommande(idCommande: int) : void
```

# Visibilité

La visibilité permet de restreindre l'utilisation d'un élément nommé, soit dans les espaces de noms, soit dans l'accès à l'élément

## Visibilité:

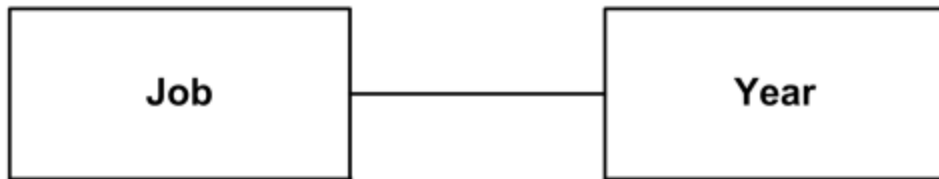
- public : +
- package : ~
- protégé : #
- privé : -

SQLStatement
+executeQuery(sql: String): ResultSet #isPoolable(): Boolean ~getQueryTimeout(): int -clearWarnings()



# Association

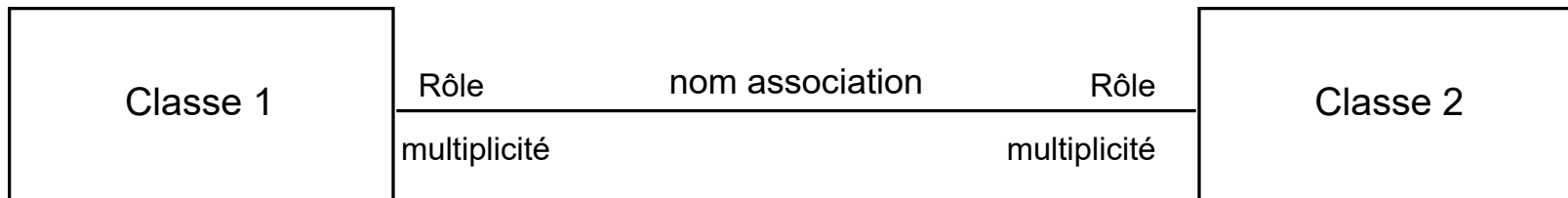
L'association est une **relation entre les classes** qui est utilisée pour montrer que les instances de classes peuvent être soit **liées les unes aux autres**, soit combinées logiquement ou physiquement dans une certaine agrégation



## Nommer les associations

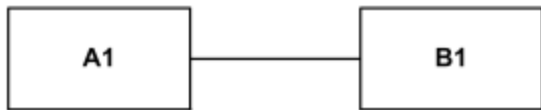
Le diagramme de classe nous permet de donner plus de contexte concernant les associations.

On peut nommer l'association ou encore préciser la multiplicité de chaque côté de la relation.

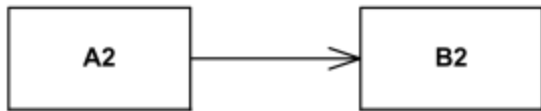


# Navigabilité

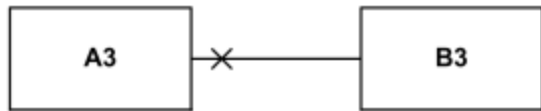
La navigabilité dans une association entre classes signifie que vous pouvez accéder aux instances d'une classe à partir des instances de l'autre classe.



: navigabilité non spécifiée



: A2 non spécifié, B2 navigable à partir de A2.

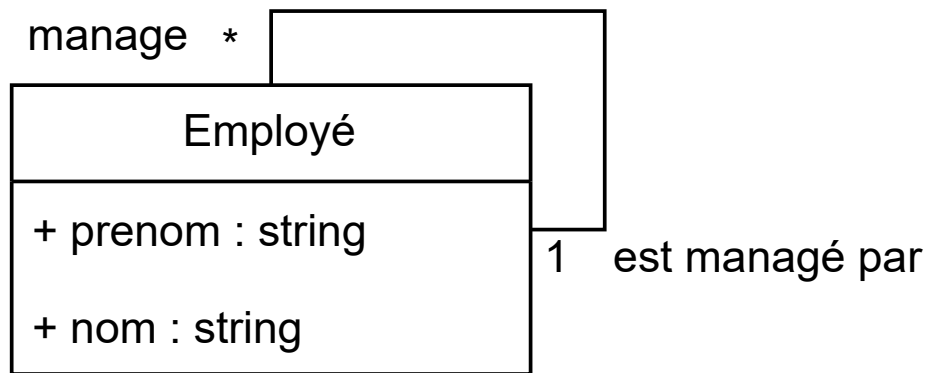


A3 n'est pas navigable à partir de B3, tandis que B3 a une navigabilité non spécifiée.

# Association reflexive

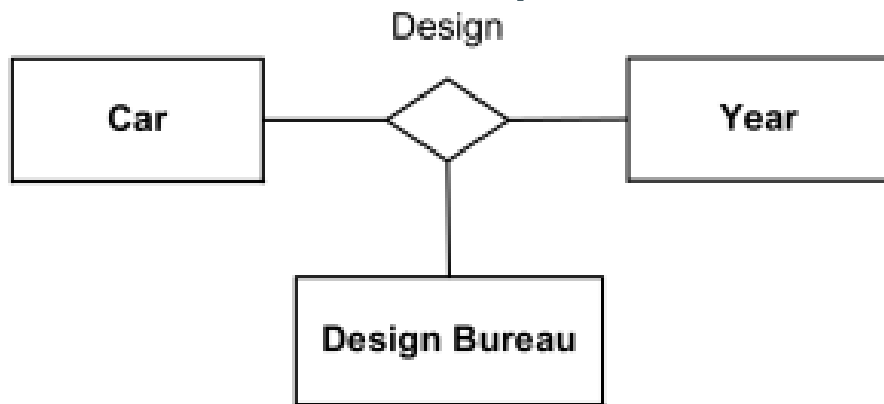
Une association réflexive est une association qui associe une classe avec elle-même.

L'explicitation des associations est souvent utile dans le cas des associations réflexives non symétrique (ou chaque objet ne joue pas le même rôle).



## Association N-aire

Une association n-aire (avec **plus de deux extrémités**) est représentée par un losange connecté par des lignes continues à chaque classe impliquée. Cela permet de montrer clairement les relations entre plusieurs classes.



*Exemple d'association ternaire*

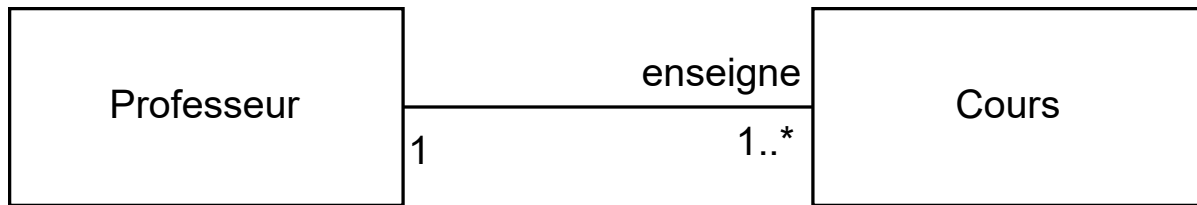
# Multiplicité

La multiplicité est une définition de la cardinalité d'une collection d'éléments

Multiplicité	Raccourci	Cardinalité
0..0	0	La collection est vide
0..1		Une ou aucune instance
1..1	1	Exactement une instance
0..*	*	0 ou plusieurs instances
1..*		Au moins une instance
5..5	5	Exactement 5 instances
m..n		Au minimum m mais au maximum n instances

# Interpréter les cardinalités

Les cardinalités se lisent à l'extrémité de la ligne d'association en partant de la classe. Voici un exemple:

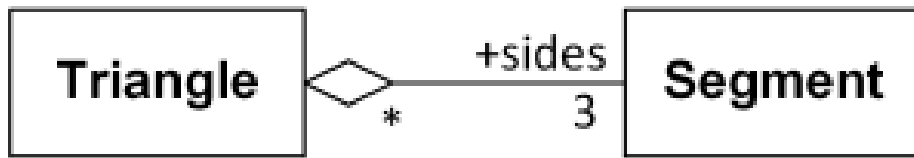


**Professeur** : Un professeur peut enseigner plusieurs cours.

**Cours** : Chaque cours est enseigné par un seul professeur.

# Agrégation

L'agrégation est une forme "faible" d'agrégation qui est utilisée lorsqu'une instance partielle est indépendante de l'objet composite



Une agrégation peut exprimer :

- qu'une classe (un "élément") fait partie d'une autre ("l'agrégat")
- qu'un changement d'état d'une classe, entraîne un changement d'état d'une autre
- qu'une action sur une classe, entraîne une action sur une autre



# Composition

La composition est une forme "forte" d'agrégation qui présente les caractéristiques suivantes

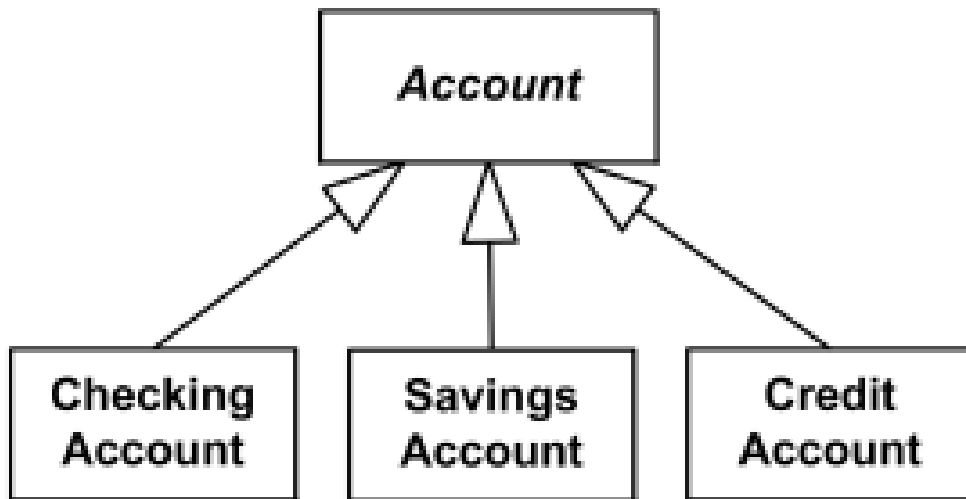
- il s'agit d'une association binaire
- il s'agit d'une relation entre un tout et une partie
- si l'agrégat est détruit, ses composants le sont aussi



# Généralisation

Une relation de généralisation est une relation dans laquelle un élément de modèle (l'enfant) est basé sur un autre élément de modèle (le parent).

L'instance enfant est une instance indirecte du parent, ce qui permet généralement de dire "La classe enfant est une sorte de parent".



# Interface

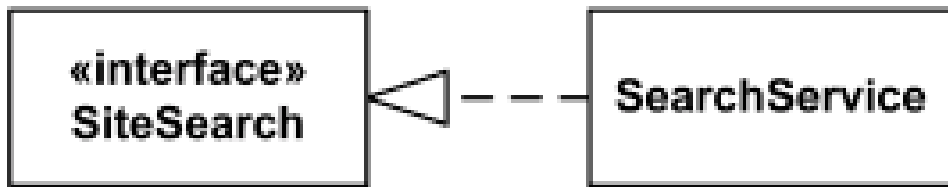
- Une interface spécifie un **contrat**
- Toute classe qui **implémente** l'interface doit remplir ce contrat et fournit donc les services décrits par le contrat
- Les interfaces étant des déclarations, elles ne sont pas instanciables

«interface»  
SiteSearch

# Implémentation d'une interface

L'implémentation d'une interface est une relation de réalisation spécialisée entre une classe et une interface.

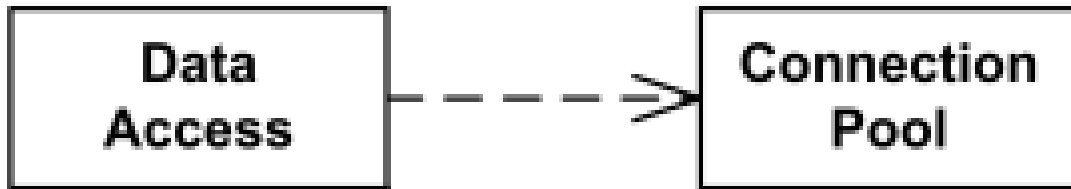
Cette relation signifie que la classe est conforme au contrat spécifié par l'interface.



## Les dépendances

La dépendance signifie une relation fournisseur/client entre des éléments de modèle où **la modification du fournisseur peut avoir un impact sur les éléments de modèle du client.**

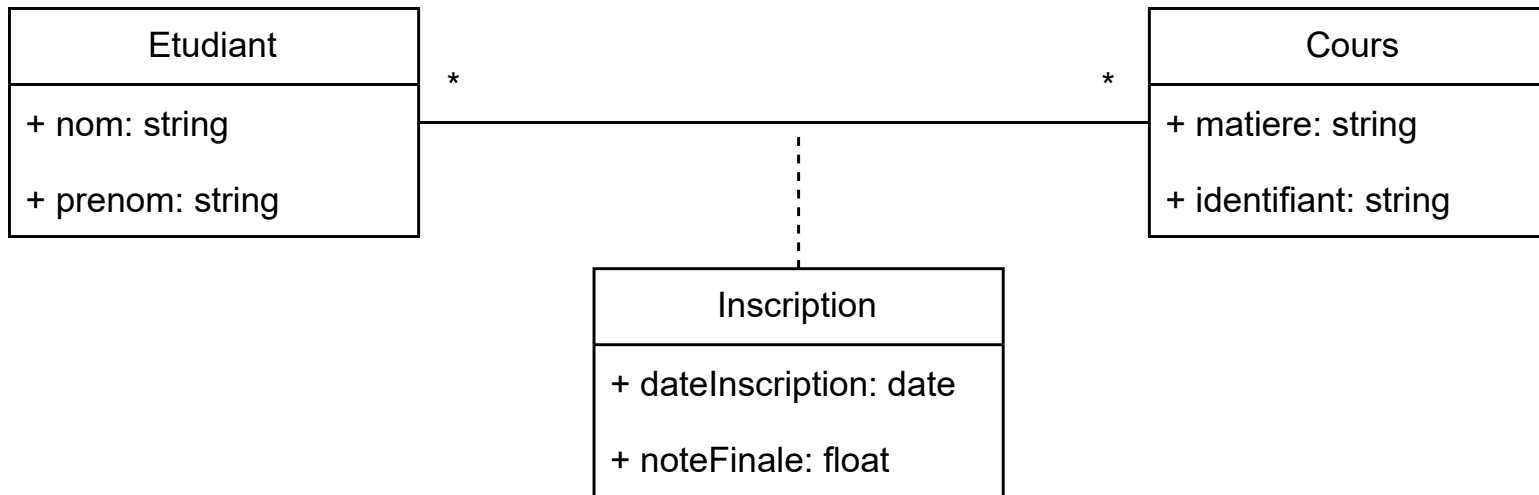
Une dépendance implique que la sémantique du client n'est pas complète sans le fournisseur.



# Classe d'association

Une classe d'association est utilisée pour représenter une relation entre deux autres classes qui possède ses propres attributs et opérations.

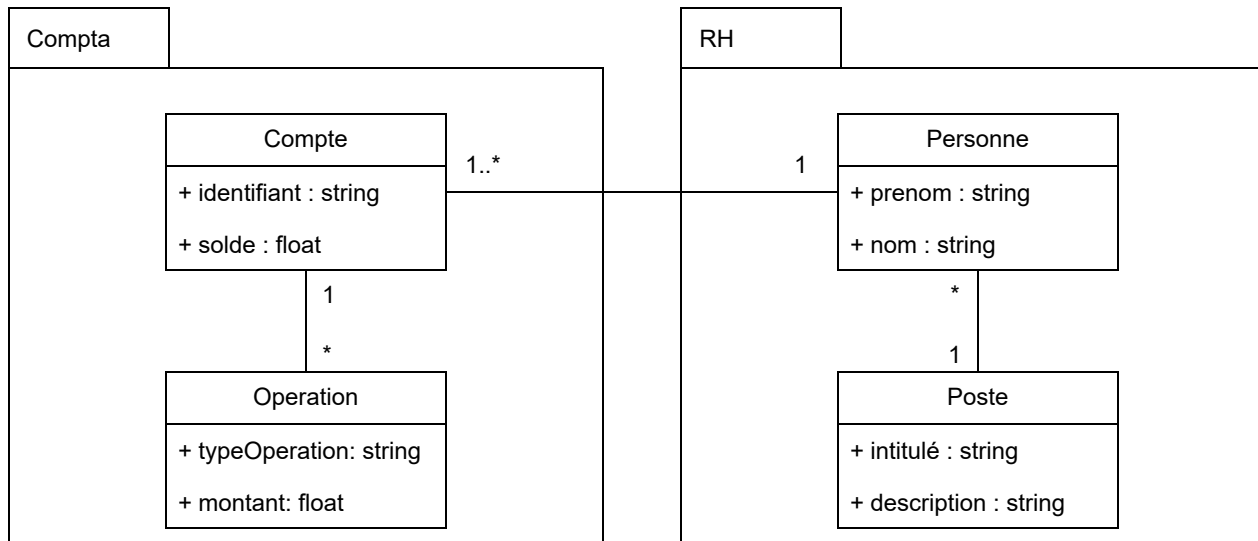
Une classe d'association permet de modéliser des informations supplémentaires sur la relation elle-même.



# Les packages

Les paquetages (package) sont des éléments servant à organiser un modèle.

Ils sont particulièrement utiles dès que le modèle comporte de nombreuses classes et que celles-ci peuvent être triées selon plusieurs aspects structurants.



# Diagramme d'objet



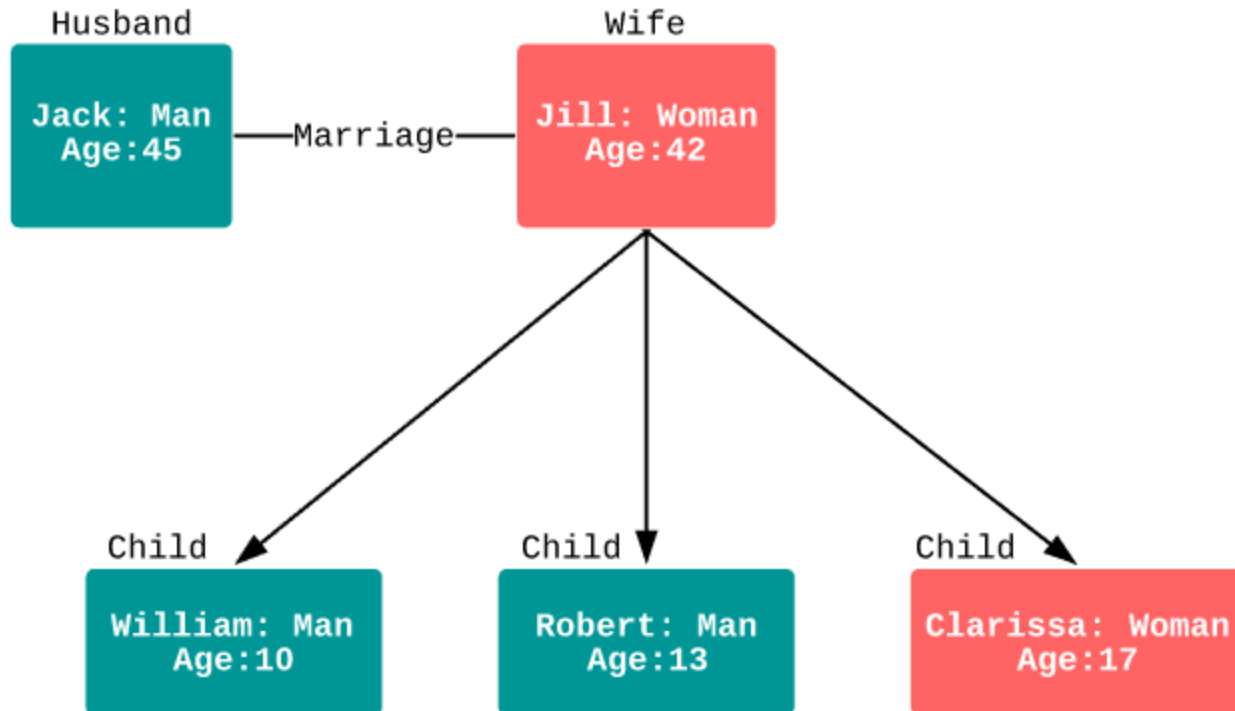
## Définition

Un diagramme d'objets représente une **instance spécifique** d'un diagramme de classes à un **moment précis**. Dans sa représentation visuelle, il est très similaire à un diagramme de classes.

Il se concentre sur les attributs d'un ensemble d'objets et sur **la façon dont ils interagissent** les uns avec les autres.

On peut généralement le voir comme une capture d'écran du système en fonctionnement à un instant T.

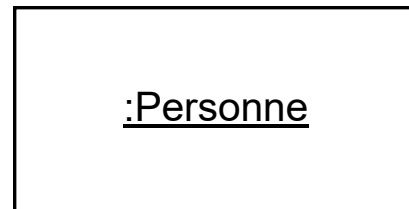
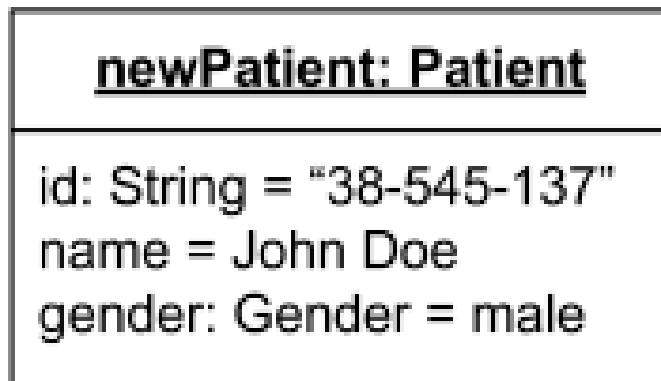
# Diagramme d'objet



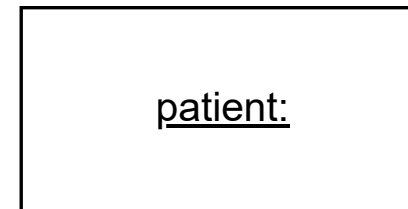
# Définition d'un objet

Un **objet** est une instance spécifique d'une classe, c'est-à-dire un exemple réel avec des valeurs d'attributs définies.

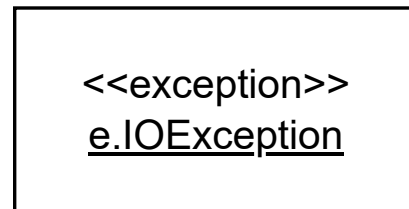
Un objet reprend la structure de classe en lui affectant des valeurs.



objet anonyme



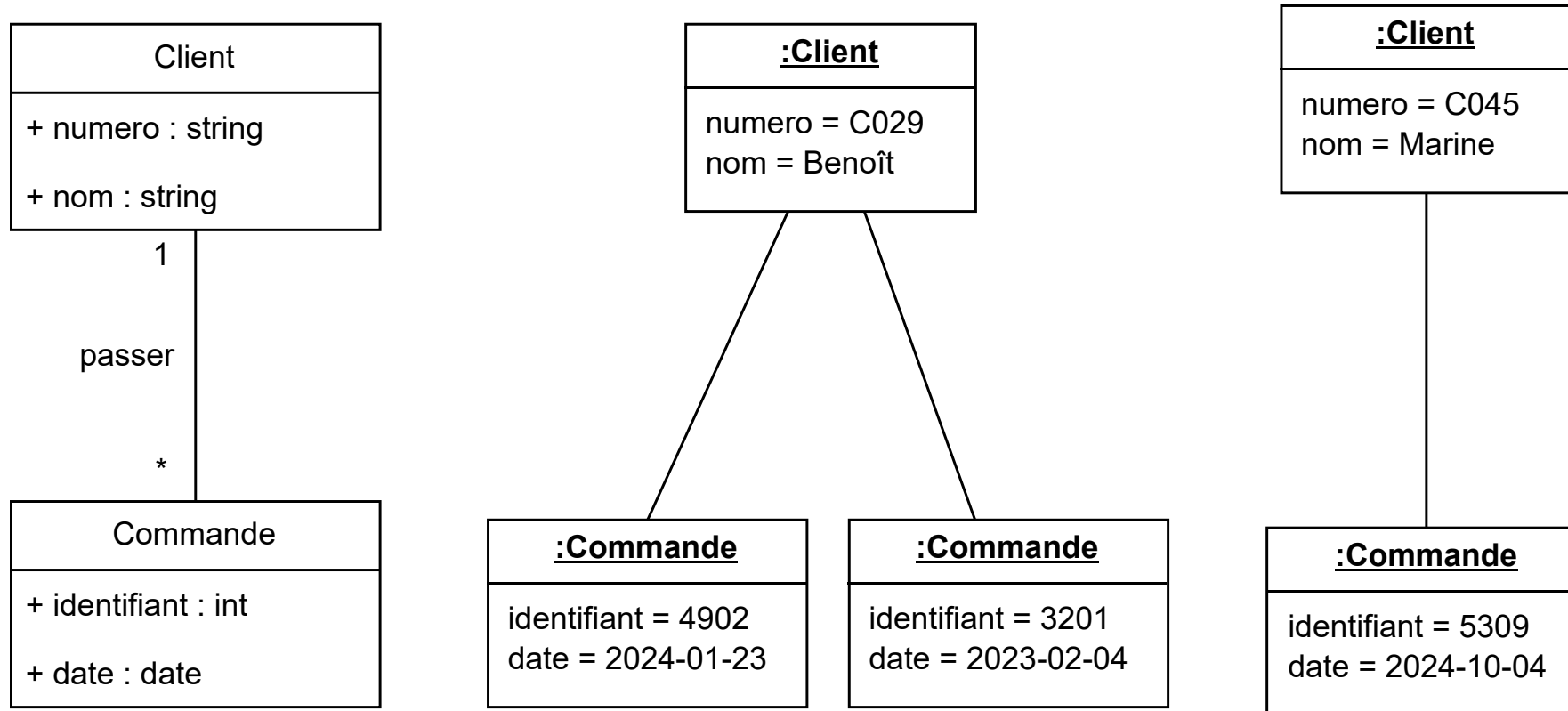
objet orphelin



objet stéréotypé

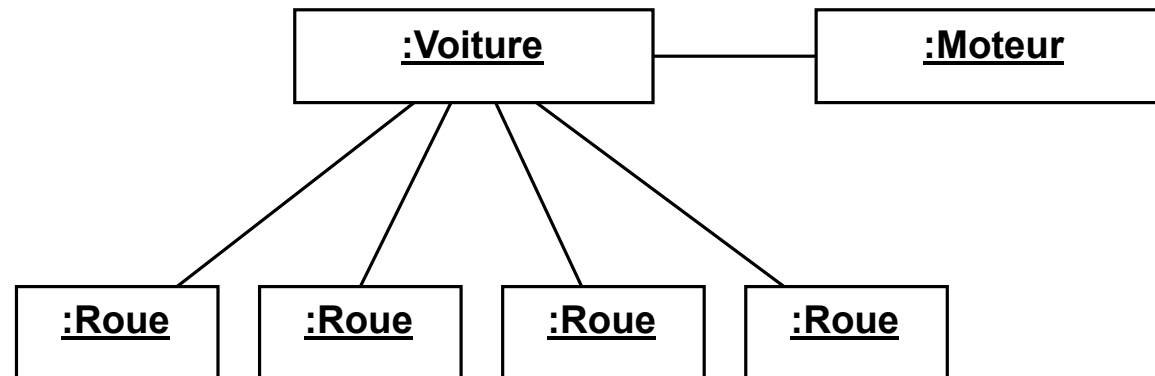
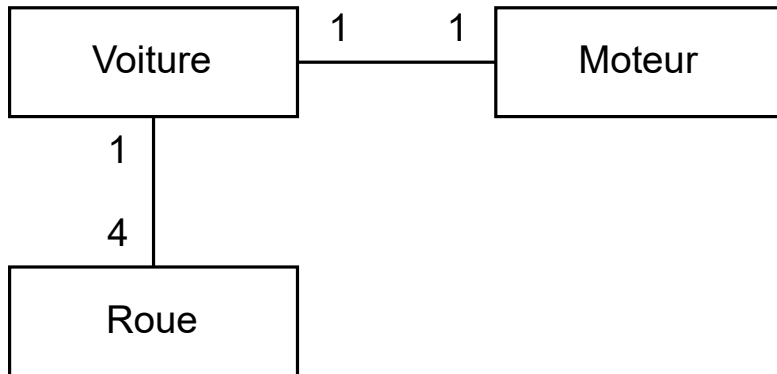
# Diagramme de classe et diagramme d'objet

Le diagramme de classe **contraint** la structure et les liens du diagramme d'objet pour s'assurer de la **cohérence** du système.



# Les liens

- Les objets sont reliés par des instances d'associations : **les liens**
- Un lien représente une relation entre objets a un instant donné
- **!** la multiplicité des extrémités des des liens est toujours de 1



# Diagramme de composants

# Introduction au Diagramme de Composants

- Un diagramme de composants est un diagramme UML utilisé pour **visualiser les composants logiciels** d'un système et leurs **interactions**.
- Il montre la **structure statique** d'un système en mettant en évidence ses composants, leurs interfaces, et les relations entre eux.
- Ce diagramme permet de mettre en évidence les dépendances entre les composants (**qui utilise quoi**).

## Définitions Clés

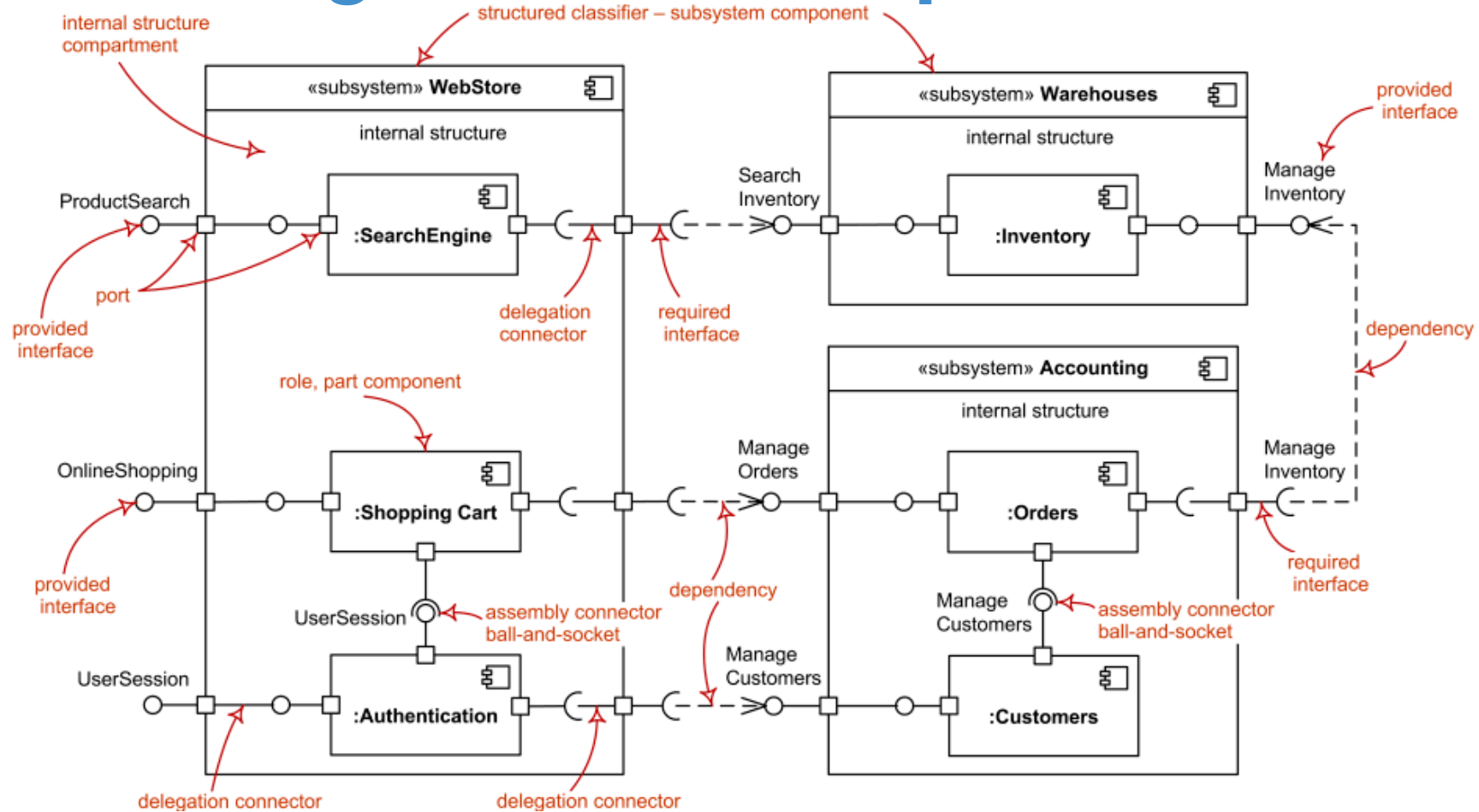
- **Composant** : Un module autonome qui réalise une fonction spécifique. Peut être du code, une bibliothèque ou un service.
- **Interface** : Un contrat ou point de communication qu'un composant expose pour interagir avec d'autres composants.
- **Connexion** : Une ligne reliant deux composants, symbolisant une dépendance ou une interaction.



# Rôle dans l'Architecture Logicielle

- Le diagramme de composants aide à **documenter** et **planifier** l'architecture d'un système.
- Il est crucial pour comprendre :
  - Les **dépendances**.
  - Les **points de communication**.
  - La **modularité** du système.

# Exemple de diagramme de composant



**Merci pour votre attention**

**Des questions ?**

