UML

Présenté par Youssouf SAGAF
www.yosagaf.fr
contact@yosagaf.fr

Presco Consulting
163 Quai du Docteur Dervaux
92600 Asnières-Sur-Seine
Tel : 06 50 33 42 45

Ice break

Jeu : un mensonge et deux vérités

Dites deux vérités et une mensonge sur vous et l'audience se chargera de trouver le mensonge ainsi que les deux vérités.

Exemples :

- J'ai quatre enfants,
- J'ai gagné un concours de beauté une fois,
- Je suis allé en Chine,

Pour plus d'infos > *https://fr.wikihow.com/jouer-au-jeu-1-mensonge-et-2-vérité*

Objectifs

- Apprendre les concepts de base théoriques d’UML et leur représentation graphique.

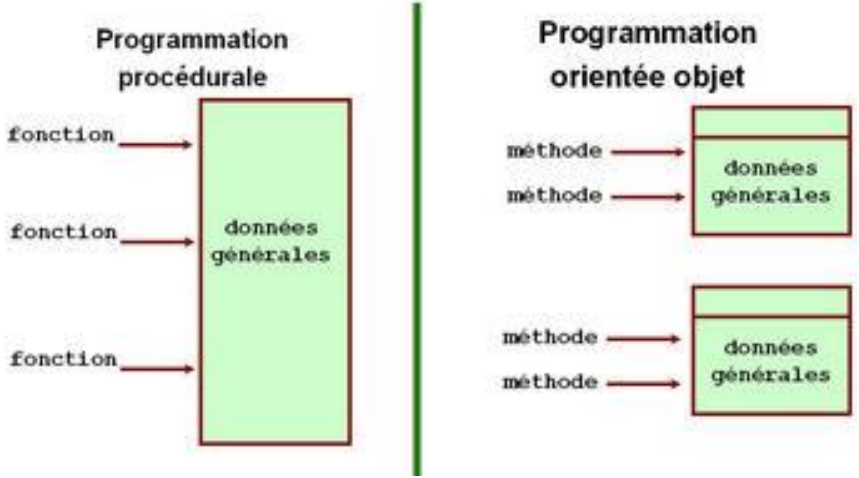
- Illustrer la modélisation logiciel à l’aide de certains diagrammes UML en s’appuyant sur des exemples et des exercices pratiques.

Plan

1. **Concepts de l’approche objet**
2. Modélisation UML
3. Modélisation objet élémentaire avec UML
    - Diagrammes de cas d’utilisation
    - Diagrammes de classes
    - Diagrammes d’objets
    - Diagrammes de séquences

Définition du paradigme orienté objet

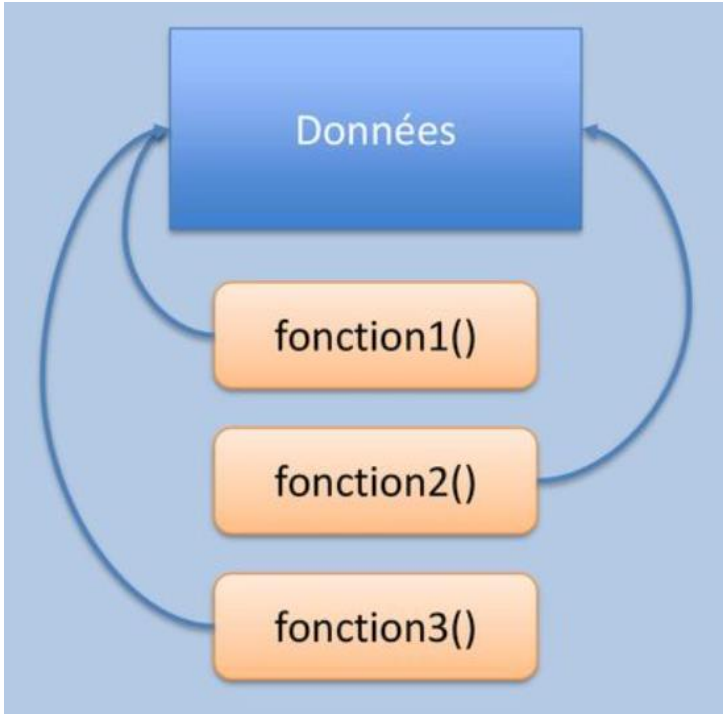
- La **programmation orientée objet** (POO) est un modèle de programmation informatique qui met en œuvre une conception basée sur les objets.
- La POO se différencie de la **programmation procédurale**, qui est basée sur l'utilisation des **procédures** et de la programmation fonctionnelle, qui elle, se repose entièrement sur le concept de **fonction**.



POO vs Programmation procédurale (1/2)

La programmation revient :

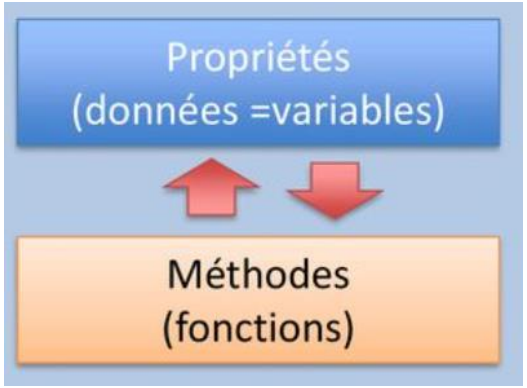
- Définir des variables
- Ecrire des procédures (fonctions) pour les manipuler sans associer explicitement les unes aux autres.
- Exécuter un programme se réduit à appeler ces procédures dans un ordre décrit par le séquence des instructions et en leur fournissant les données nécessaires à l'accomplissement de leurs tâches.



POO vs Programmation procédurale (2/2)

- Au fur et à mesure que les programmes écrits avec une approche procédurale grossissent, ils deviennent **fragiles** et mettent en danger les **données**.

- La **POO** c'est un peu comme le GOLF
  - Les concepts de base sont simples à apprendre mais :
    - Les utiliser tous et de manière élégante est beaucoup plus difficile.
  - 4 concepts de bases :
    - **Propriétés** (les variables) et **méthodes** (les fonctions)
    - Classe et objet
    - Encapsulation
    - Héritage

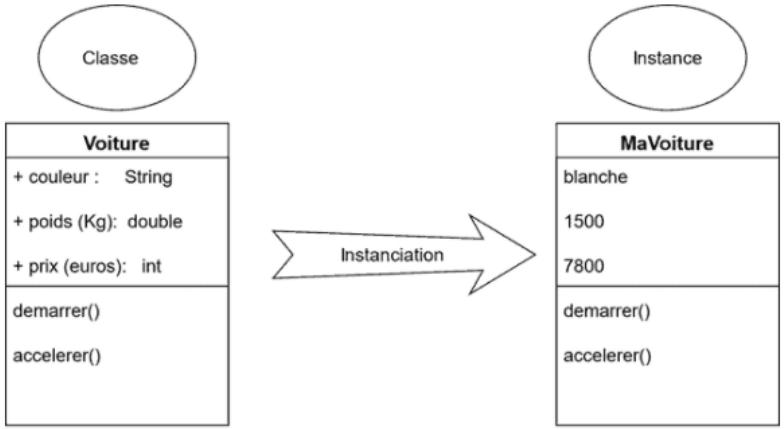


Avantage du paradigme POO

- Avantages du paradigme orienté objet :
  - **La modularité** : les objets forment des modules compacts regroupant des données et un ensemble d'opérations.
  - **L'abstraction** : les objets de la POO sont proches de celles du monde réel. Les concepts utilisés sont donc proches des abstractions familières que nous exploitons.
  - **Productivité et ré-utilisabilité** : plus l'application est complexe et plus l'approche POO est intéressante. Le niveau de réutilisabilité est supérieur à la programmation procédurale.
  - **Sûreté :** l'encapsulation et le typage des classes offrent une certaine robustesse aux applications.

Le concept de classe

- Le premier concept de la POO est la classe qui est une structure abstraite décrivant les objets du monde réel sous deux angles.
  - ses propriétés (ses caractéristiques) et ses méthodes (les actions qu'elle peut effectuer ou son comportement).
- La classe est une sorte de moule ou de modèle.
- Les objets sont construits à partir de ce moule (classe) par un processus appelé **instanciation**.
- Toutes les instances de classe s'appellent des objets.



Le concept d’objet (1/2)

- Un **objet** représente une entité du monde réel qui se caractérise par un ensemble de **propriétés (attributs)**, des **états significatifs** et un **comportement**.
- Le **comportement** d’un objet est caractérisé par l’ensemble des opérations qu’il peut exécuter en réaction aux message provenant des autres objets.

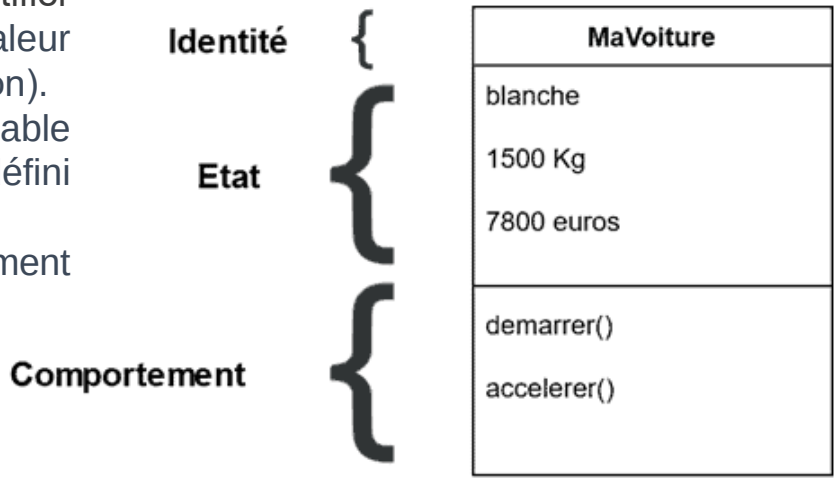
- **Exemple** : Considérons l’employé Durand, n°1245, embauché en tant que ingénieur dans le site N.

- Cet objet est caractérisé par la liste de ses attributs et son état est représenté par les valeurs de ses attributs :
  - n° employé : 1245
  - nom : Durand
  - qualification : ingénieur
  - Lieux de travail : site N

- Son comportement est caractérise par les opérations qu’il peut exécuter :
  - entrer dans l’organisme
  - changer de qualification
  - changer de lieu de travail
  - sortir de l’organisme

Le concept d'objet (2/2)

- Un **objet** est une instance de classe.
- Pour faire le parallèle avec le monde réel, l'objet c'est un peu comme une maison bâtit sur la base d'un plan particulier. Tant que les architectes se réfèrent à ce plan, ils produiront toujours les mêmes maisons.
- Un objet est caractérise par :
  - **une identité** : l'identité doit permettre d'identifier sans ambiguïté l'objet. Chaque objet a une valeur par défaut (lorsqu'elle est indiqué à l'instanciation).
  - **des méthodes** : chaque objet est capable d'exécuter les actions ou le comportement défini dans la classe.
  - Ces actions sont traduites en POO concrètement sous forme de ***méthodes***.



Encapsulation

- Les attributs et les méthodes d'un objet qui constituent sa structure interne ne sont pas en général pas accessibles aux autres objets : c'est le principe de l'encapsulation.
- Par exemple, pour pouvoir modifier la couleur d'une voiture, il faudra lui ajouter une méthode publique, **changerCouleur,** qui s'occupera de changer la valeur de son attribut **couleur**.
- Les autres objets, n'ont ainsi plus besoin de savoir comment changer la couleur de la voiture, ils se contentent d'appeler la méthode **changerCouleur**.

Plan

1. Concepts de l’approche objet
2. **Modélisation UML**
3. Modélisation objet élémentaire avec UML
    - Diagrammes de cas d’utilisation
    - Diagrammes de classes
    - Diagrammes d’objets
    - Diagrammes de séquences

Bibliographie — UML

- UML 2.0, guide de référence
  - James Rumbaugh, Ivar Jacobson, Grady Booch
  - Edition Campus Press (2005)
- UML 2.0
  - Benoit Charoux, Aomar Osmani, Yann Thierry-Mieg
  - Edition Pearson, Education France (2008)
- UML 2.0 Superstructure et UML 2.0 Insfrastructure
  - OMG (Object Management Group)
  - www.uml.org (2004)

Matériel et logiciel

- Systèmes informatiques :
  - **80 % de logiciels**
  - 20 % de matériel
- Depuis quelques années, la fabrication du matériel est assurée par quelques fabricants seulement.
  - le matériel est relativement fiable
  - le marché est standardisé

Les problèmes liés à l’informatique sont essentiellement des problèmes logiciel.

La *"crise du logiciel"*

- Étude sur 8 380 projets (Standish Group, 1995) :
  - succès : 16 %;
  - problématique : 53 % (budget ou délais non respectés, défaut de fonctionnalités) ;
  - échec : 31 % (abandonné).

Le taux de succès décroît avec la taille des projets et la taille des entreprises.

- Génie Logiciel (Software Engineering) :
  - comment faire des logiciels de qualité ?
  - qu'attend-on d'un logiciel ?
  - quels sont les critères de qualité ?

Critère de qualité d’un logiciel

- Utilité
  - Adéquation entre le logiciel et les besoins des utilisateurs.
- Utilisabilité
- Fiabilité
- Interopérabilité
  - Interactions avec d’autres logiciels.
- Performance
- Portabilité
- Réutilisabilité
- Facilité de maintenance
  - Un logiciel ne s’use pourtant, la maintenance absorbe une très grosse partie des efforts de développement.

Cycle de vie d’un logiciel

La qualité du processus de fabrication garantie de la qualité du produit.

- Pour obtenir un logiciel de qualité, il faut en maîtriser le processus d'élaboration.
  - La vie d'un logiciel est composée de différentes étapes.
  - La succession de ces étapes forme le cycle de vie du logiciel.
  - Il faut contrôler la succession de ces différentes étapes.

Étapes du développement d’un logiciel

1. Étude de faisabilité
2. Spécification
    - Déterminer les fonctionnalités du logiciel.
3. Conception
    - Déterminer la façon dont le logiciel fournit les différentes fonctionnalités recherchées.
4. Codage
5. Tests
    - Essayer le logiciel sur des données d’exemple pour s’assurer qu’il fonctionne correctement.
6. Maintenance

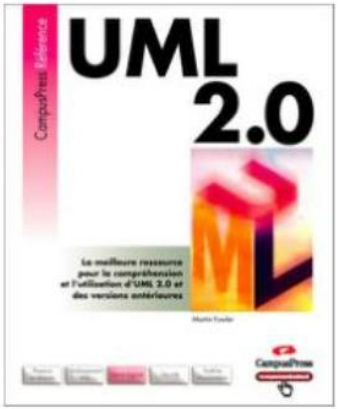
Modèle

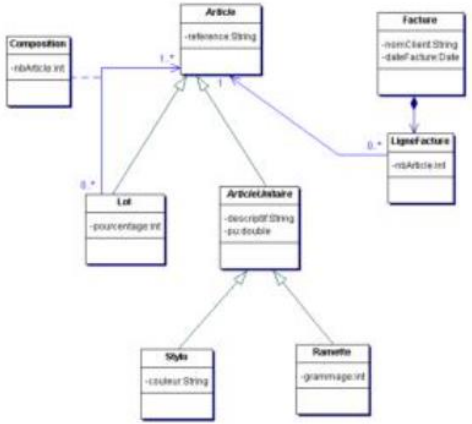
Modèle = objet conçu et construit (artefact)

- Un modèle est une représentation abstraite de la réalité qui exclut certains détails du monde réel.
  - il permet de réduire la complexité d'un phénomène en éliminant les détails qui n'influencent pas son comportement de manière significative.
  - il reflète ce que le concepteur croit important pour la compréhension et la prédiction du phénomène modélisé, les limites du phénomène modélisé dépendent des objectifs du modèle.

Concevoir un modèle UML

- Avant de construire un bâtiment, il nous faut un plan.
- En programmation objet, il faut élaborer le schéma UML avant de commande le codage.





UML — Unified Modeling Language

- Au milieu des années 90, les auteurs de Booch, OOSE et OMT ont décidé de créer un langage de modélisation unifié avec pour objectifs :
  - modéliser un système des concepts à l'exécutable, en utilisant les techniques orientée objet ;
  - réduire la complexité de la modélisation ;
  - utilisable par l'homme comme par la machine :
    - représentations graphiques mais disposant de qualités formelles suffisantes pour être traduites automatiquement en code source ;
    - ces représentations ne disposent cependant pas de qualités formelles suffisantes pour justifier d'aussi bonnes propriétés mathématiques que des langages de spécification formelle (Z, VDM...).
- Officiellement UML est né en 1994.

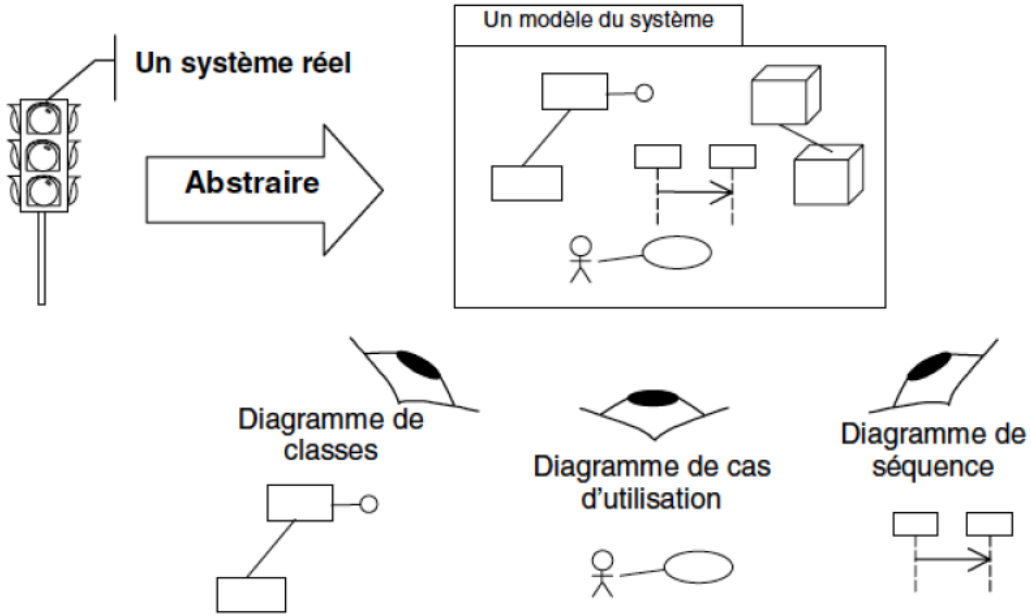
UML v2.0 date de 2005. Il s'agit d'une version majeure apportant des innovations radicales et étendant largement le champ d'application d'UML.

Différents types de diagrammes

Il existe 2 types de vues du systèmes qui comportent chacune leurs propres diagrammes (en tout 9 type de diagrammes)..

- les vues statiques
  - **diagrammes de cas d'utilisation**
  - **diagrammes d'objets**
  - **diagrammes de classes**
  - diagrammes de composants
  - diagrammes de déploiement
- les vues dynamiques
  - diagrammes de collaborations
  - **diagrammes de séquences**
  - diagrammes d'états-transitions
  - diagrammes d'activités

Diagramme UML : points de vue sur le système



[Image empruntée à Muller et Gaertner]

L’utilisation des diagrammes

- **Définition d’un diagramme** : un diagramme UML est une représentation graphique, qui s’intéresse à un aspect précis du modèle. L’UML permet de définir et de visualiser un modèle à l’aide de diagrammes.
- **Caractéristiques des diagrammes UML** : les diagrammes UML supportent l’abstraction. La structure des diagrammes et la notation graphique des éléments sont normalisés.

Plan

- Concepts de l’approche objet
- Modélisation UML
- **Modélisation objet élémentaire avec UML**
  - **Diagrammes de cas d’utilisation**
  - Diagrammes d’objets
  - Diagrammes de classe
  - Diagrammes de séquences

Modélisation des besoins

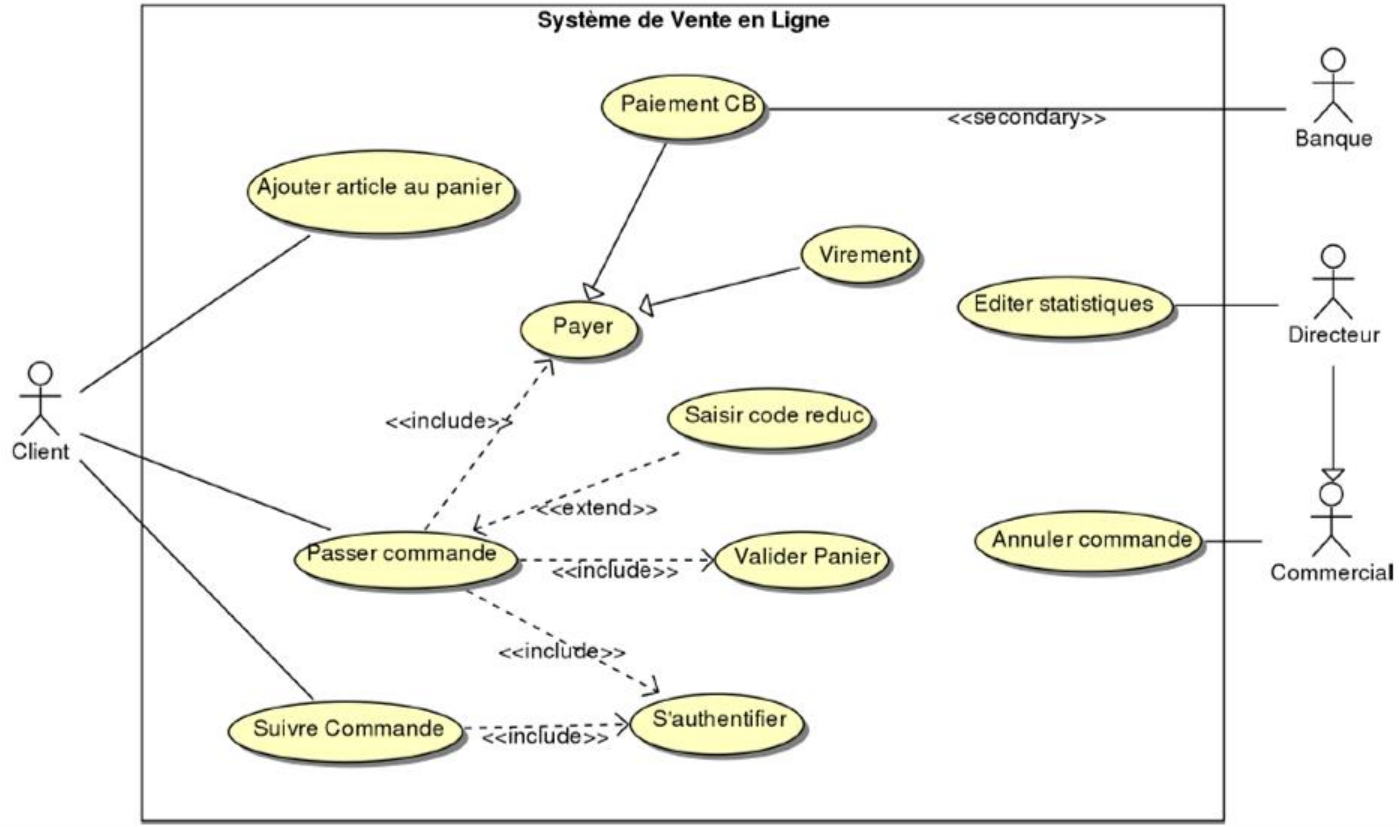
Avant de développer un système, il faut savoir **précisément** à QUOI il devra servir, cad à quels besoins il devra répondre.

- **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**.

Diagramme de cas d’utilisation (use case)

- Les cas d’utilisation ou use cases permettent de structurer les besoins des utilisateurs et les objectives correspondants d’un système.
- Les cas d’utilisation identifient les utilisateurs du systèmes (acteurs) et leurs interactions avec le système.
- Permettent de classer les acteurs et structurer les objectifs du système:
  - Définissent le contour du système à modéliser en précisant le but à atteindre
  - Permettent d’identifier les fonctionnalités principales du système

Exemple de diagramme de cas d'utilisation



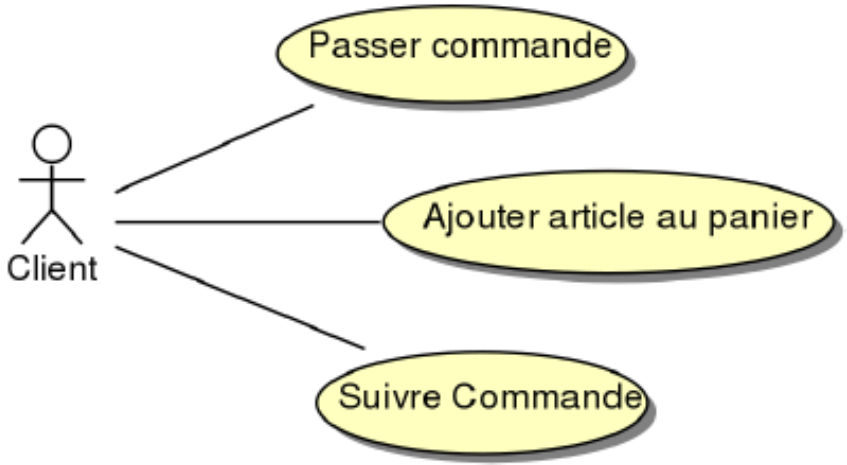
Cas d’utilisation

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

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

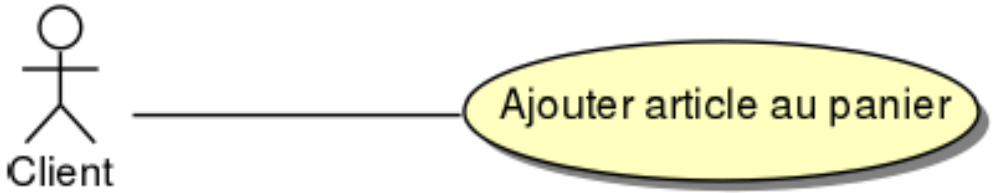
Un **cas d'utilisation** est l'expression d'un service réalisé de bout en bout, avec un déclenchement et un déroulement.

Acteurs et cas d’utilisation



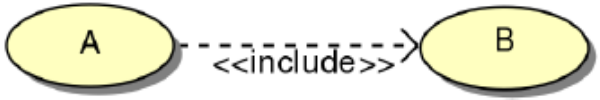
Relations entre cas d’utilisation et acteurs

- 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.

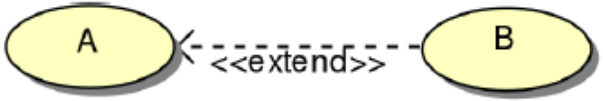


Relations entre cas d'utilisation

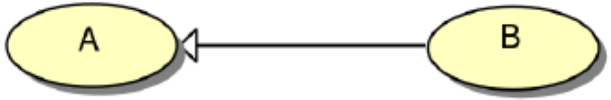
- **Inclusion** : le cas A inclut le cas B (*B* est une partie *obligatoire* de *A*).



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



- **Généralisation :** le cas A est une généralisation du cas B (B est une sorte de A ou le cas d'utilisation enfant B est une spécialisation du cas d'utilisation parent)



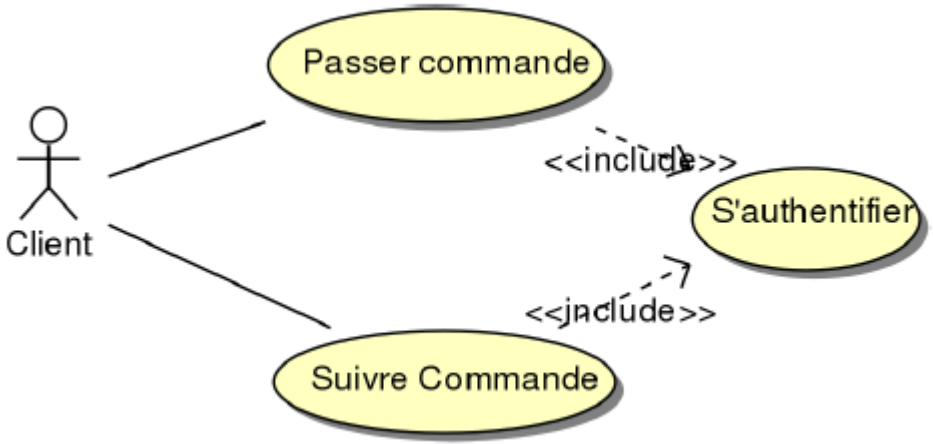
Dépendances d'inclusion et d'extension

- Les inclusions et les extensions sont représentées par des **dépendances**.
  - lorsqu'un cas B inclut un cas A, B dépend de A.
  - lorsqu'un cas B étend un cas A, B dépend aussi de A.
  - on note toujours la dépendance par une flèche pointillée B --> A qui se lit <<B dépend de A >>.
- Lorsqu'un élément A dépend d'un élément B, toute modification de B sera susceptible d'avoir un impact sur A.
- Les <<***include***>> et les <<***extend***>> sont des stéréotypes (entre guillemets) des relations de dépendance.

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

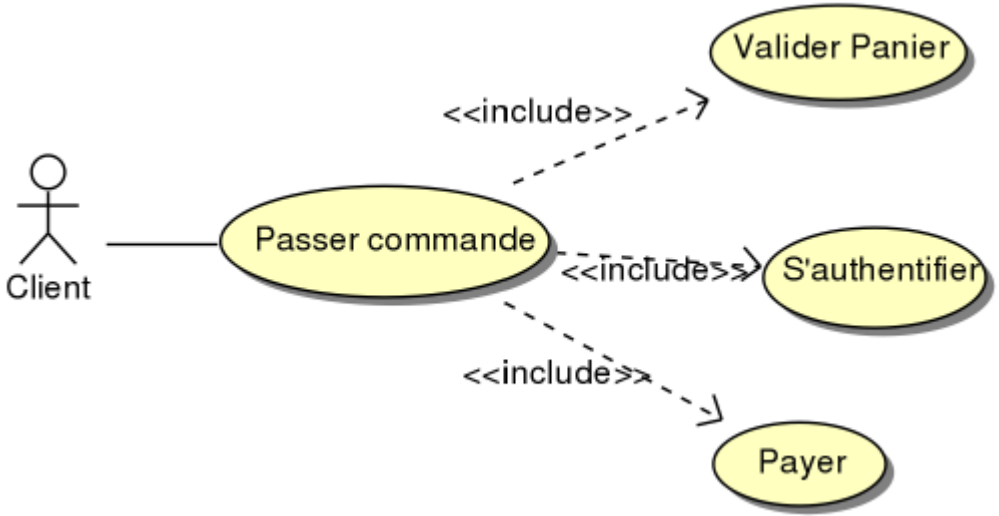
Réutilisabilité avec les inclusions et les extensions

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



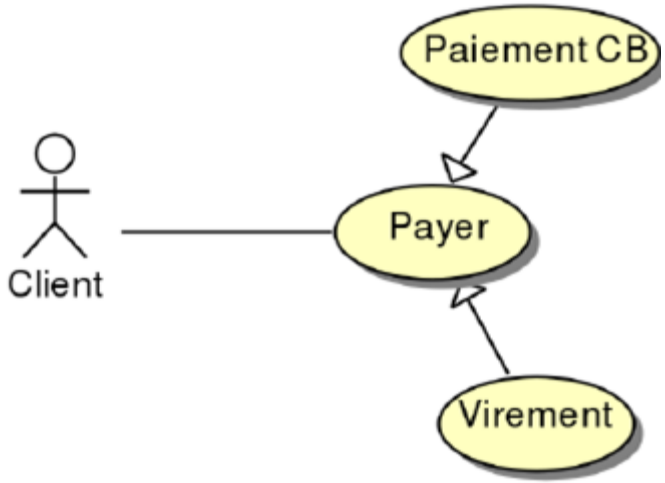
Décomposition grâce aux inclusions et aux extensions

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



Généralisation

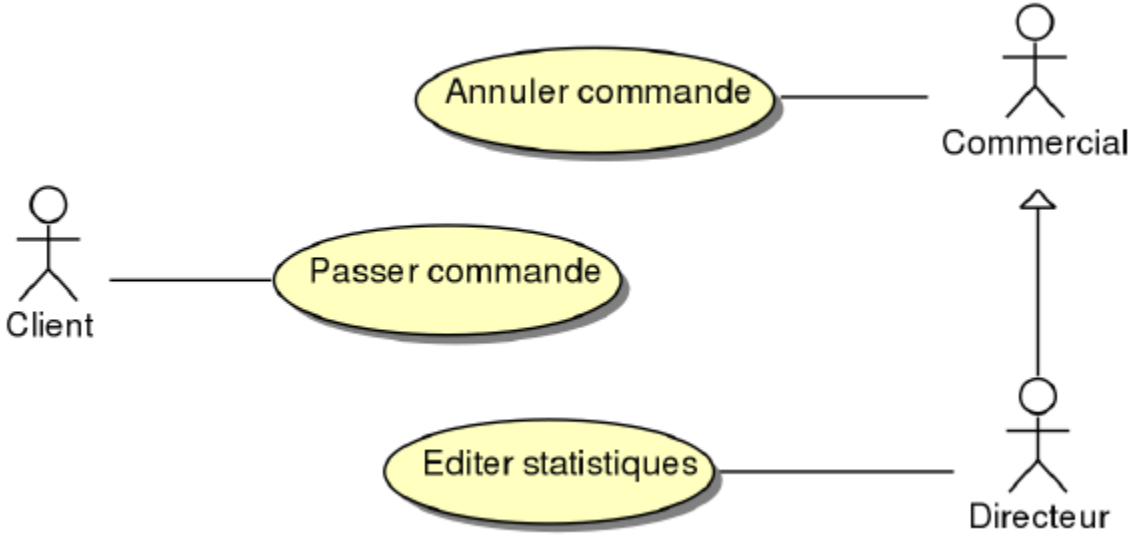
- Un virement est un cas particulier de paiement.
- Un virement **une sorte de** paiement



- La flèche pointe vers l'élément général
- Cette relation de généralisation/spécialisation est présente dans la plupart des diagrammes UML et se traduit par le concept d'héritage dans les langages orientés objet.

Relations entre acteurs

- Une seule relation possible : la généralisation.



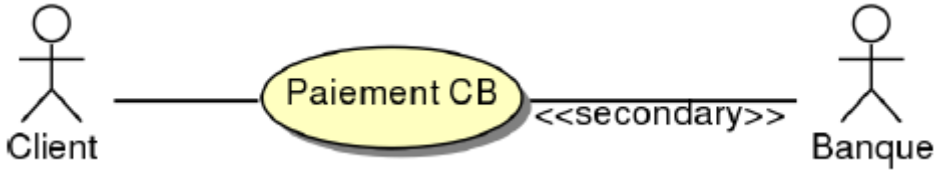
Identification des acteurs

- Les principaux acteurs sont les utilisateurs du système
  - une même personne physique peut être représentée par plusieurs acteurs si elle a plusieurs rôles.
  - si plusieurs personnes jouent le même rôle vis-à-vis du système, elles seront représentées par un seul acteur.
- En plus des utilisateurs, les acteurs peuvent être :
  - des périphériques manipulés par le système (imprimantes, ..);
  - des logiciels déjà disponibles à intégrer dans le projet;
  - des systèmes informatiques externes au système mais qui interagissent avec lui, etc.
- Pour faciliter la recherche des acteurs, on se fonde sur les frontières du système.

Attention : un acteur correspond à un rôle et non pas à une personne physique.

Acteurs principaux et sécondaires

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



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

Recenser les cas d'utilisation

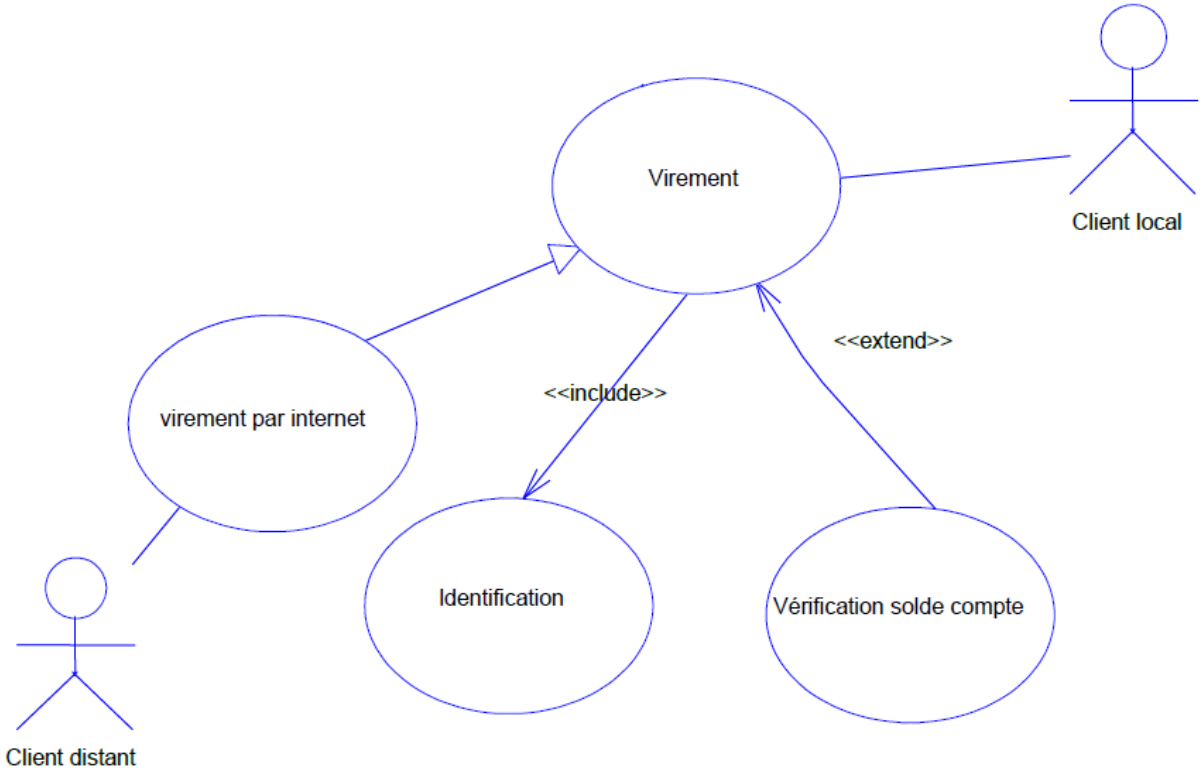
- Il n'y a pas une manière mécanique et totalement objective de repérer les cas d'utilisation.
  - il faut se placer du point de vue de chaque acteur et déterminer comment il se sert du système, dans quels cas il l'utilise, et à quelles fonctionnalités il doit avoir accès.
  - il faut éviter les redondances et limiter le nombre de cas en se situant au bon niveau d'abstraction (par exemple, ne pas réduire un cas à une seule action).
- Il ne faut pas faire apparaître les détails des cas d'utilisation, mais il faut rester au niveau des grandes fonctions du système.

Trouver le bon niveau de détail pour les cas d'utilisation est un problème difficile qui nécessite de l'expérience.

Description des cas d’utilisation

- Le diagramme de cas d'utilisation décrit les grandes fonctions d'un système du point de vue des acteurs, mais n'expose pas de façon détaillée le dialogue entre les acteurs et les cas d'utilisation.
- Un simple nom est tout à fait insuffisant pour décrire un cas d’utilisation.
- Chaque cas d'utilisation doit être documenté pour qu'il n'y ait aucune ambiguïté concernant son déroulement et ce qu'il recouvre précisément.

Exemple pratique de cas d’utilisation



EXO1 — Diagramme de cas d’utilisation UML : Identification des acteurs et de cas d’utilisation simple

On considère une borne de chargement de titre de transport Navigo. Les utilisateurs peuvent charger leur Navigo et les techniciens peuvent intervenir en cas de panne.

Question 1 : Pour charger son titre, le client dépose son titre et suit les instructions indiquées. Quel est l’acteur et l’action dans ce cas d’utilisation.

Question 2 : Jojo, dont le métier est technicien au sein de la RATP, veut charger le navigo de son fils. Pour modéliser cette activité de Jojo, doit-on définir un nouvel acteur ? Comment modélise-t-on ça ?

Question 3 : Lorsque Jojo vient avec ses outils pour réparer la borne en cas de panne, est-il considéré comme un nouvel acteur ? Comment modélise-t-on cela ?

Question 3 : Certains agent de la RATP qui ne sont pas des techniciens sont aussi qualifiés pour opérer des opérations de maintenance en plus des opérations habituelles des techniciens telles que le remplacement de certains pièces et produits. Ils sont donc techniciens en plus d'être agents. Comment modéliser cela ?

EXO2 — Diagramme de cas d'utilisation UML : Caisse enregistreuse

Le déroulement normal d'utilisation d'une caisse enregistreuse est le suivant :

1. Un client arrive à la caisse
2. Le caissier enregistre le numéro d'identification de chaque article, ainsi que la quantité si celle-ci est supérieur à 1.
3. La caisse affiche chaque article et son libellé
4. Lorsque tous les articles ont été enregistrés, le caissier signale la fin de la vente
5. La caisse affiche le total des achats
6. Le client choisit son mode de paiement
    a. **Liquide** : le caissier encaisse l'argent et la caisse indique le montant éventuel à rendre au client
    b. **Chèque** : le caissier note l'identité du client et la caisse enregistre le montant sur le chèque
    c. **Carte de crédit** : le terminal bancaire transmet la demande à un centre d'autorisation multibanque
7. La caisse enregistre la vente et imprime un ticket
8. Le caissier transmet le ticket imprimé au client
9. Un client peut présenter un ticket de réduction avant le paiement. Lorsque le paiement est terminé, la caisse transmet les informations relatives aux articles vendus au système de gestion des stocks.
10. Tous les matins, le responsable du magasin initialise les caisses pour la journée.

Question : Donnez un diagramme de cas d'utilisation pour la caisse enregistreuse.

Installation d’outil de modélisation : StarUML

- Accéder au site web officiel de StarUML à l’adresse suivante : https://staruml.io/
- Cliquer sur le bouton **download** pour télécharger
- Installer la version téléchargée. Il faut avoir les droits d’administration pour pouvoir l’installer.
- Un guide intérressant du logicel est disponible en version pdf à l’adresse > https://inf1410.teluq.ca/files/2020/11/INF1410_GuideStarUML-DiagrammeUML_VT.pdf

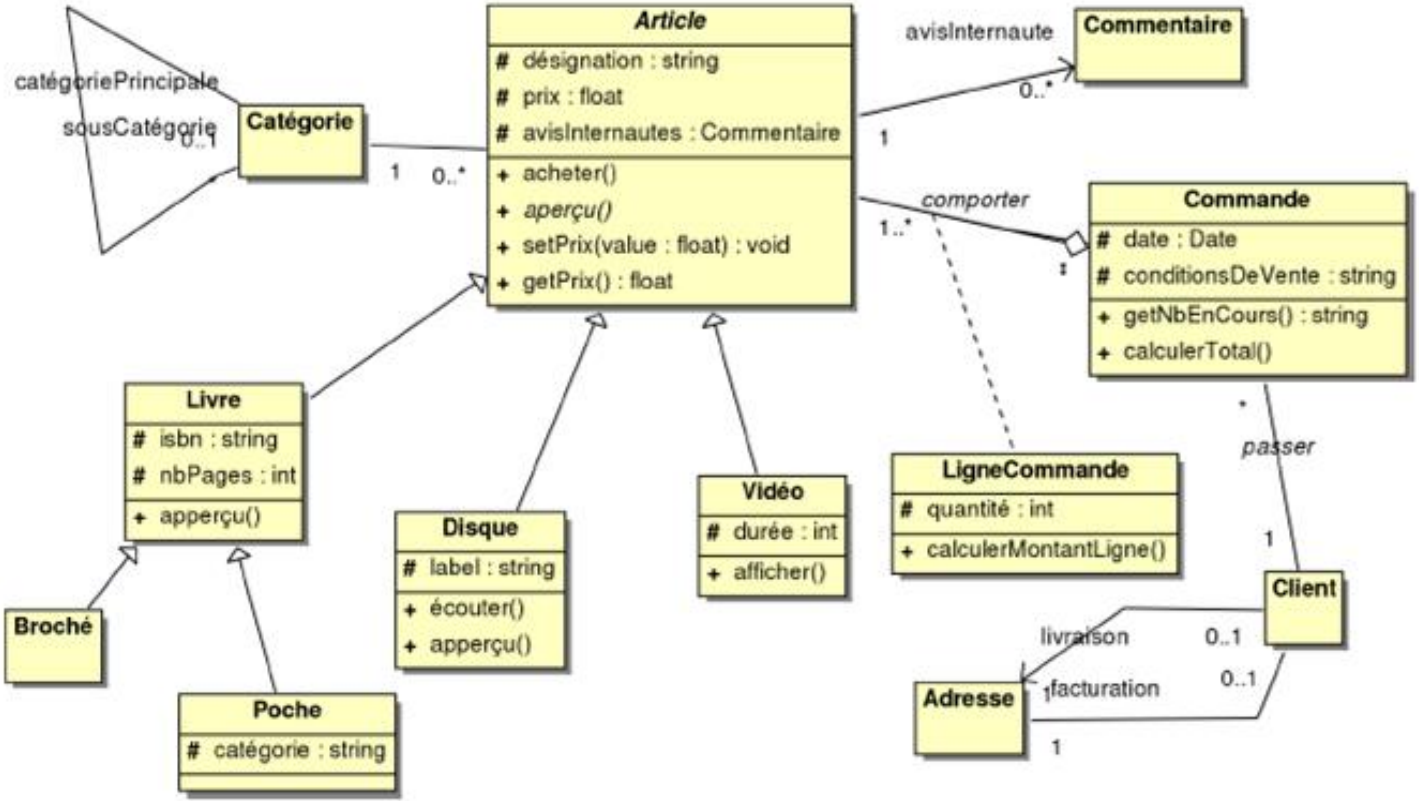
Plan

- Concepts de l’approche objet
- Modélisation UML
- **Modélisation objet élémentaire avec UML**
  - Diagrammes de cas d’utilisation
  - **Diagrammes de classe**
  - Diagrammes d’objets
  - Diagrammes de séquences

Diagramme de classes

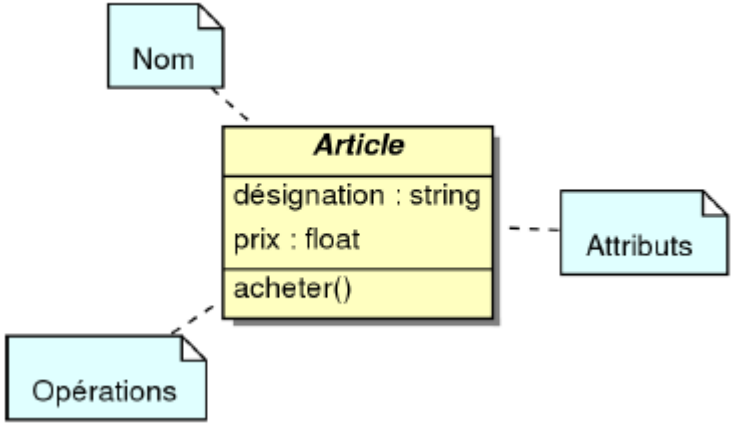
- Les **diagrammes de cas d'utilisation** modélisent à QUOI sert le système.
- Le système est composé d'objets qui interagissent entre eux et avec les acteurs pour réaliser ces cas d'utilisation.
- Les **diagrammes de classes** permettent de spécifier la structure et les liens entre les objets dont le système est composé.

Exemple de diagramme de classe



Classes et objets

- Une **classe** est la description d'un ensemble d'objets ayant une sémantique, des attributs, des méthodes et des relations en commun. Elle spécifie l'ensemble des caractéristiques qui composent des objets de même type.



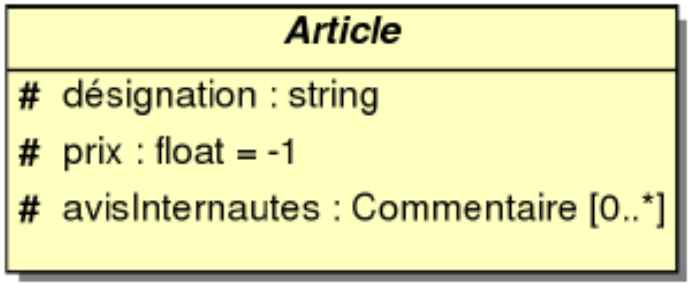
- Une classe est composée d'un **nom**, d'**attributs** et d'**opérations.**
- Selon l'avancement de la modélisation, les informations ne sont pas forcément toutes connues.

Propriétés : attributs et opérations

- Les attributs et les opérations sont les propriétés d'une classe. Leur nom commence par une minuscule.
  - Un **attribut** décrit une donnée de la classe.
    - Les types des attributs et leurs initialisations ainsi que les modificateurs d'accès peuvent être précisés dans le modèle
    - Les attributs prennent des valeurs lorsque la classe est instanciée : ils sont en quelque sorte des variables attachées aux objets.
  - Une **opération** est un service offert par la classe (un traitement que les objets correspondant peuvent effectuer).

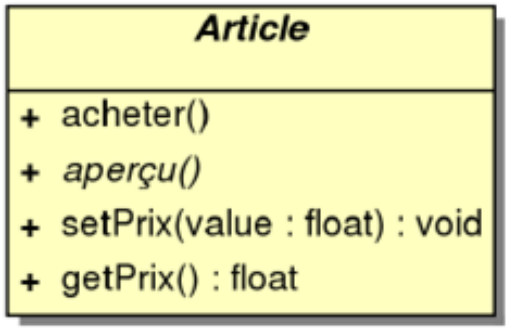
Compartiment des attributs

- Un attribut peut être initialisé et sa visibilité est définie lors de sa déclaration.
- UML définit 3 niveaux de visibilité pour les opérations :
  - **public** (+) : l'opération est visible pour tous les clients de classes
  - **protégé** (#) : l'opération est visible pour les sous-classe de la classe
  - **privé** (-) : l'opération n'est visible que par les objets de classe dans laquelle elle est déclarée



Compartiment des opérations

- Une opération est dénie par son ainsi que par les types de ses paramètres et le type de sa valeur de retour.

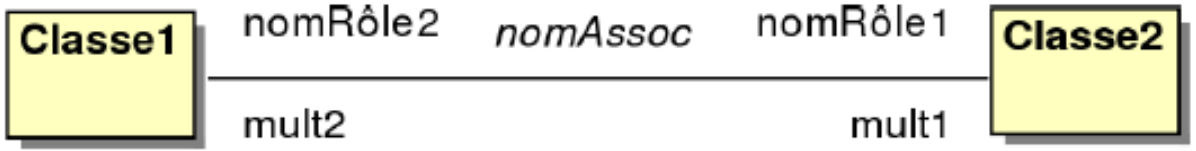


Relations entre classes

- Une relation d'**héritage** est une relation de généralisation/spécialisation permettant l'abstraction.
- Une **dépendance** est une relation unidirectionnelle exprimant une dépendance sémantique entre les éléments du modèle (flèche ouverte pointillée).
- Une **association** représente une relation sémantique entre les objets d'une classe.
- Une relation d'**agrégation** décrit une relation de contenance ou de composition.

Association

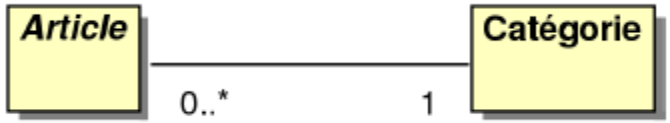
- Une association est une relation structurelle entre objets.
  - Une association est souvent utilisée pour représenter les liens possible entre objets de classes données.
  - Elle est représentée par un trait entre classes.



1	un et un seul
0..1	Zéro ou un
m..n	De m à n (entier)
*	Plusieurs
0..*	De zéro à plusieurs
1..*	d'un à plusieurs

Multiplicités des associations

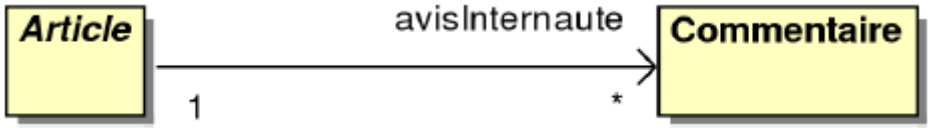
- La notion de **multiplicité** permet le contrôle du nombre d'objets intervenant dans chaque instance d'une association.
  - **Exemple** : un article n'appartient qu'à une seule catégorie (1) ; une catégorie concerne plus de 0 articles, sans maximum (*).



- La syntaxe est *MultMin*..*MultMax*.
  - <<*>> à la place de MultMax signifie <<plusieurs>> sans préciser de nombre.
  - <<n..n>> se note aussi << n>> , et << 0..*>> se note << *>> .

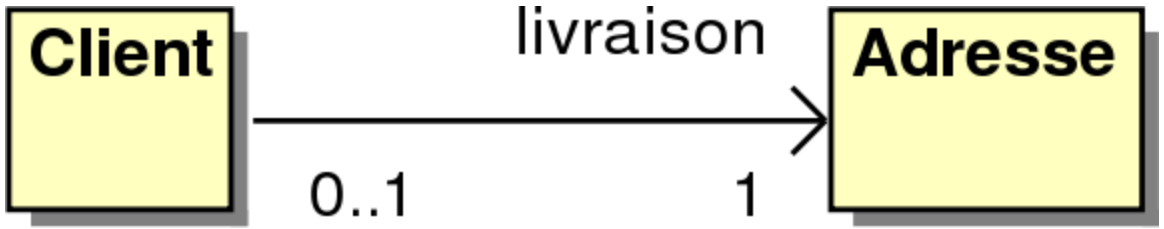
Navigabilité d'une association

- La navigabilité permet de spécifier dans quel(s) sens il est possible de traverser l'association à l'exécution.
- On restreint la navigabilité d'une association à un seul sens à l'aide d'une flèche.

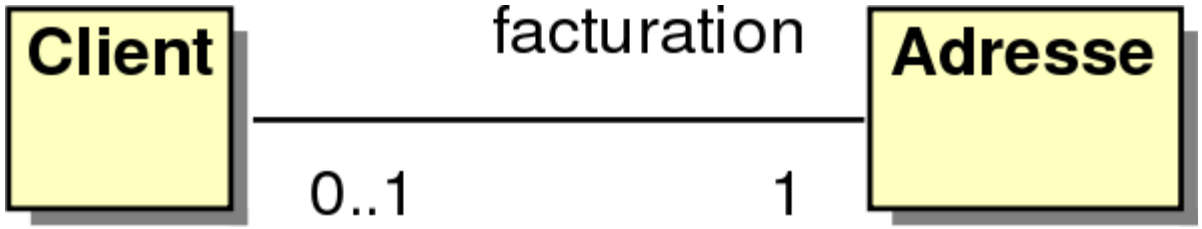


  - **Exemple** : Connaissant un article on connaît les commentaires, maispas l'inverse.
- On peut aussi représenter les associations navigables dans un seul sens par des attributs.
  - **Exemple** : En ajoutant un attribut avisInternaute de classe
  - Commentaire à la place de l'association.

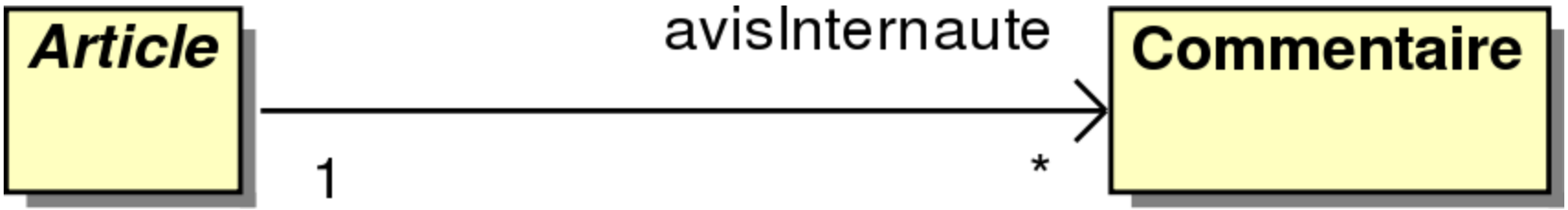
Association unidirectionnelle de 1 vers 1



Association bidirectionnelle de 1 vers 1

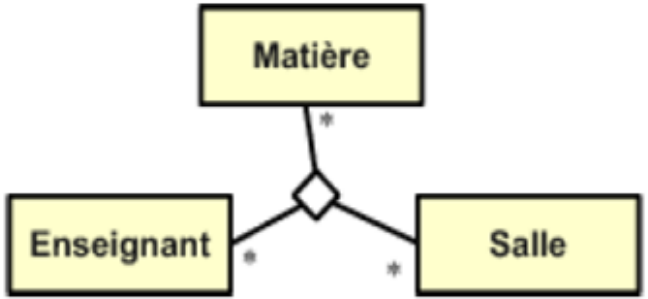


Association unidirectionnelle de 1 vers *



Association n-aire

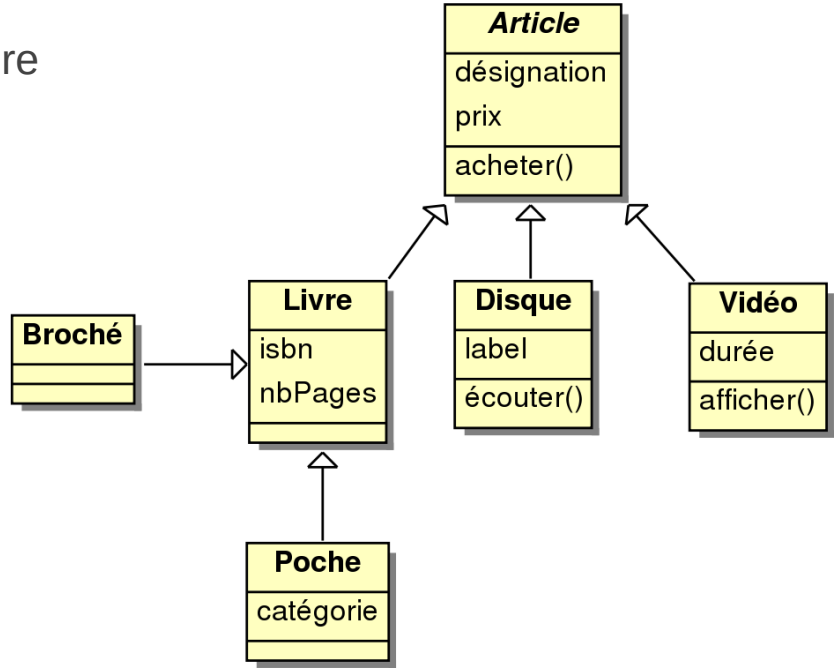
- Une **association n-aire** lie plus de deux classes.
  - notation avec un losange central pouvant éventuellement accueillir une classe-association.
  - la multiplicité de chaque classe s'applique à une instance du losange.
  - les associations n-aires sont peu fréquente et concernent surtout les cas où les multiplicités sont toutes <<*>>. Dans la plupart des cas, on utilisera plus avantageusement des classes-associations ou plusieurs relations binaires.



Relation d’héritage

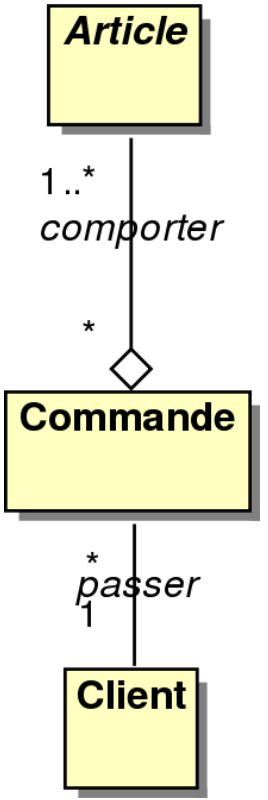
- L'héritage une relation de spécialisation/généralisation.
- Les éléments spécialisés héritent de la structure et du comportement des éléments plus généraux (attributs et opérations)

Exemple : Par héritage d'Article, un livre a un prix, une désignation et une opération acheter(), sans qu'il soit nécessaire de le préciser.



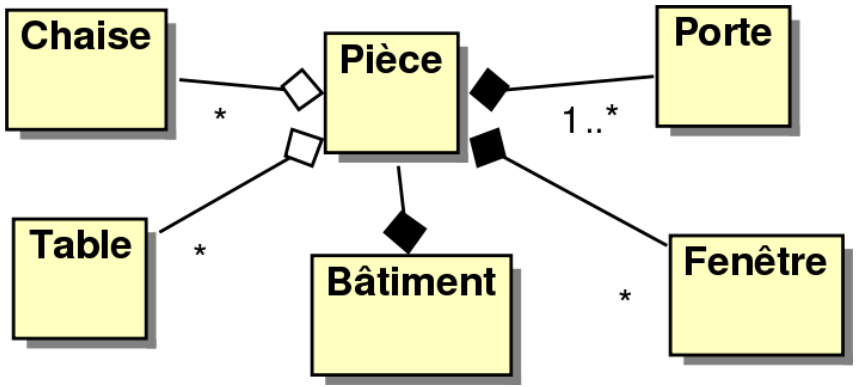
Association de type agrégation

- Une agrégation est une forme particulière d'association. Elle représente la relation d'inclusion d'un élément dans un ensemble.
- On représente l'agrégation par l'ajout d'un losange vide du côté de l'agrégat.
- Une agrégation dénote une relation d'un ensemble à ses parties. L'ensemble est l'**agrégat** et la partie l'**agrégé**.



Association de type composition

- La relation de composition décrit une contenance structurelle entre instances. On utilise un losange plein.
- La destruction et la copie de l'objet composite (l'ensemble) impliquent respectivement la destruction ou la copie de ses composants (les parties).
- Une instance de la partie n'appartient jamais à plus d'une instance de l'élément composite.



Composition et agrégation

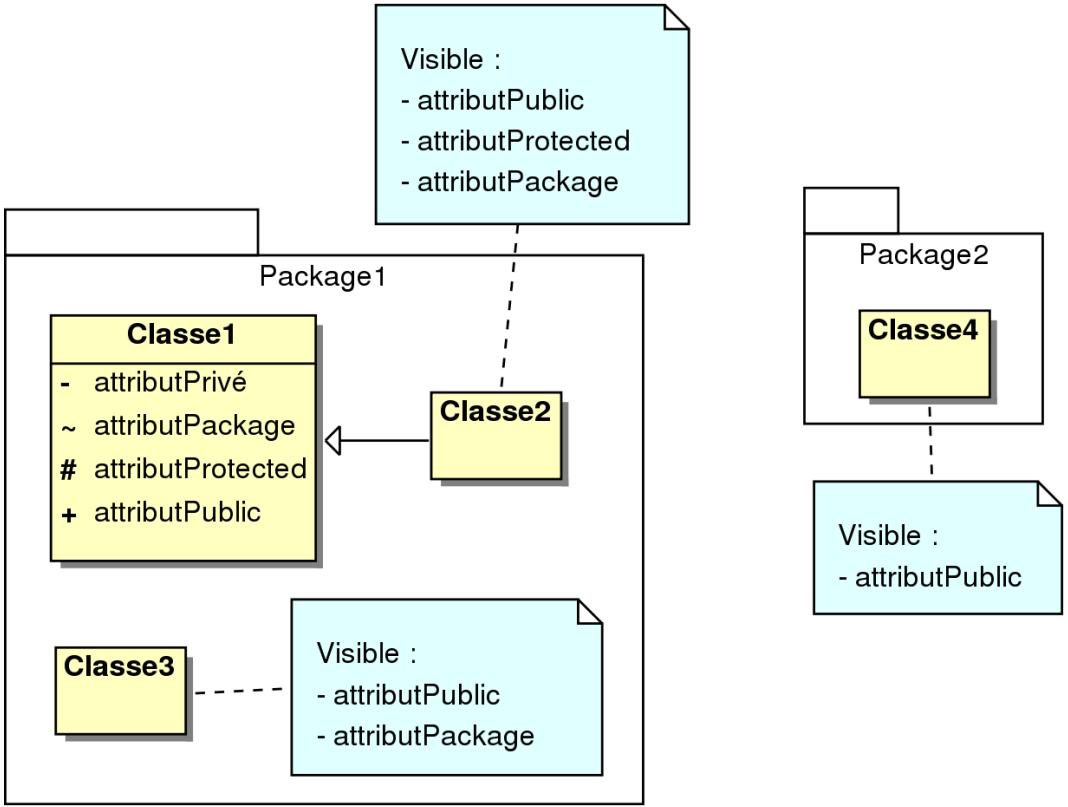
- Dès lors que l'on a une relation du tout à sa partie, on a une relation d'agrégation ou de composition.
- Pour décider de mettre une composition plutôt qu'une agrégation, on doit se poser les questions suivantes :
  - Est-ce que la destruction de l'objet composite (du tout) implique nécessairement 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.

Niveau d’accessibilité (encapsulation)

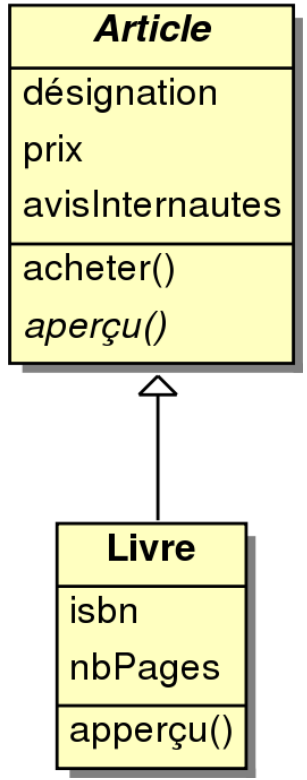
- **L'encapsulation** est un principe de conception consistant à protéger le coeur d'un système des accès intempestifs venant de l'extérieur.
- En UML, on utilise des **modificateurs d'accès** sur les attributs ou les classes :
  - **Public** ou <<+>> : propriété ou classe visible partout
  - **Protected** ou <<#>> :. propriété ou classe visible dans la classe et par tous ses descendants.
  - **Private** ou <<->> : propriété ou classe visible uniquement dans la classe
  - **Package**, ou <<~>>: propriété ou classe visible uniquement dans le paquetage
- Il n'y a pas de visibilité par défaut .

Exemple d'encapsulation



Relation d'héritage et propriétés

- La classe enfant possède toutes les propriétés de ses classes parents (attributs et opérations)
  - La classe enfant est la classe spécialisée (ici Livre)
  - La classe parent est la classe générale (ici Article)
- Toutefois, elle n'a pas accès aux propriétés privées.



Construction d'un diagramme de classes

1. Trouver les **classes du domaine étudié** ;
    - Souvent, concepts et substantifs du domaine.
2. Trouver les **associations entre classes** ;
    - Souvent, verbes mettant en relation plusieurs classes.
3. Trouver les **attributs des classes** ;
    - Souvent, substantifs correspondant à un niveau de granularité plus n que les classes. Les adjectifs et les valeurs correspondent souvent à des valeurs d'attributs.
4. Organiser et simplifier le modèle en utilisant l'héritage ;
5. Tester les chemins d'accès aux classées ;
6. **Itérer** et **raffiner** le modèle.

EXO1 — Diagrammes de classes UML : Propriétés d'une classe

Une personne est caractérisée par son nom, son prénom, son sexe et son âge. Les objets de classe Personne doivent pouvoir calculer leurs revenus et leurs charges. Les attributs de la classe sont privés ; le nom, le prénom ainsi que l'âge de la personne doivent être accessibles par des opérations publiques.

Question : Donnez une représentation UML de la classe Personne, en remplissant tous les compartiments adéquats.

Deux types de revenus sont envisagés : d'une part le salaire et d'autre part toutes les autres sources de revenus. Les deux revenus sont représentés par des nombres réels (float). Pour calculer les charges globales, on applique un coefficient xe de 20% sur les salaires et un coefficient de 15% sur les autres revenus.

Question : Enrichissez la représentation précédente pour prendre en compte ces nouveaux éléments. Un objet de la classe Personne peut être créé à partir du nom et de la date de naissance. Il est possible de changer le prénom d'une personne. Par ailleurs, le calcul des charges ne se fait pas de la même manière lorsque la personne décède.

Question : Enrichissez encore la représentation précédente pour prendre en compte ces nouveaux éléments

EXO2 — Diagramme de classe : Gestion de bibliothèque

On désire automatiser la gestion d'une petite bibliothèque municipale. Pour cela, on a analysé son fonctionnement pour obtenir la liste suivante de règles et d'affirmations :

- Les adhérents ont un prénom (chaîne de caractères) et un nom (chaîne de caractères).
- La bibliothèque comprend un ensemble de documents et un ensemble d'adhérents.
- Les adhérents sont inscrits ou désinscrits sur une simple demande.
- De nouveaux documents sont ajoutés régulièrement à la bibliothèque.
- Ces documents sont soit des journaux, soit des volumes.
- Les volumes sont soit des dictionnaires, soit des livres, soit des BD.
- Les documents sont caractérisés par un titre (chaîne de caractères).
- Les volumes ont en plus un auteur (chaîne de caractères). Les Bd ont en plus un nom de destinataire (chaîne de caractères).
- Les journaux ont, outre les caractéristiques des documents, une date de parution (une date).
- Seuls les livres sont empruntables.
- Un adhérent peut emprunter ou restituer un livre.
- Les adhérents peuvent emprunter des livres (et uniquement des livres) et on doit pouvoir savoir à tout moment quels sont les livres empruntés par un adhérent.
- Un adhérent peut emprunter au plus 3 livres.
- La date de restitution d'un livre emprunté est fixée au moment du prêt. Cette date peut être prolongée sur demande.

Réalisez le diagramme de classes permettant d'automatiser la bibliothèque municipale et définissez les attributs et les méthodes de chaque classe de ce diagramme, ainsi que le type et les cardinalités des associations entre les classes.

Plan

- Concepts de l’approche objet
- Modélisation UML
- **Modélisation objet élémentaire avec UML**
  - Diagrammes de cas d’utilisation
  - Diagrammes de classes
  - **Diagrammes d’objets**
  - Diagrammes de séquences

Objectif

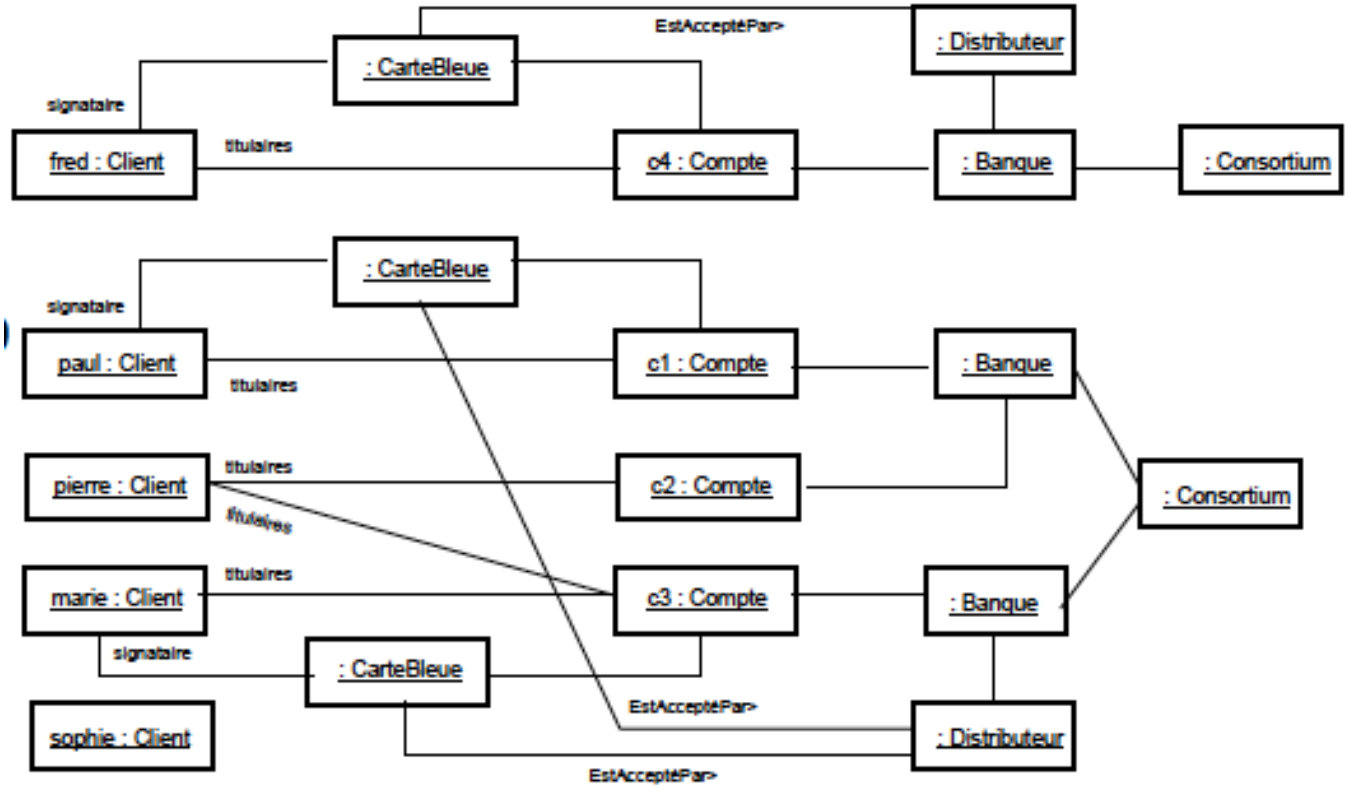
- Le **diagramme d’objets** représente les objets d’un système à un instant donné. Il permet de :
  - Illustrer le modèle de classes (en montrant un exemple qui explique le modèle)
  - Préciser certains aspects du système (en mettant en évidence des détails imperceptibles dans le diagramme de classes).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

Le diagramme de classe modélise des *règles* et le diagramme d’objets modélise des *faits.*

Définitions

- Objet : instance d’une classe
- **Diagrammes d’objets**: ensemble d’objets respectant les contraintes d’un diagramme de classe.
  - respect des cardinalités
  - Chaque attribut d’une classe a une valeur affectée dans chaque instance de cette classe.
- Diagramme de classe = définition d’un cas générale
- Un diagramme d’objets est une instance de diagramme de classes représentant des objets et leur liens à un instant donnée.

Exemple de diagramme d'objets



Représentation des objets

- Comme les classes, on utilise des **cadres compartimentés**.
- En revanche, **les noms des objets sont soulignés** et on peut rajouter son identifiant devant le nom de sa classe.
- Les valeurs (a) ou l'état (f) d'un objet peuvent être spécifiées.
- Les instances peuvent être **anonymes** (a,c,d), **nommées** (b,f), **orphelines** (e), **multiples** (d) ou **stéréotypées** (g).

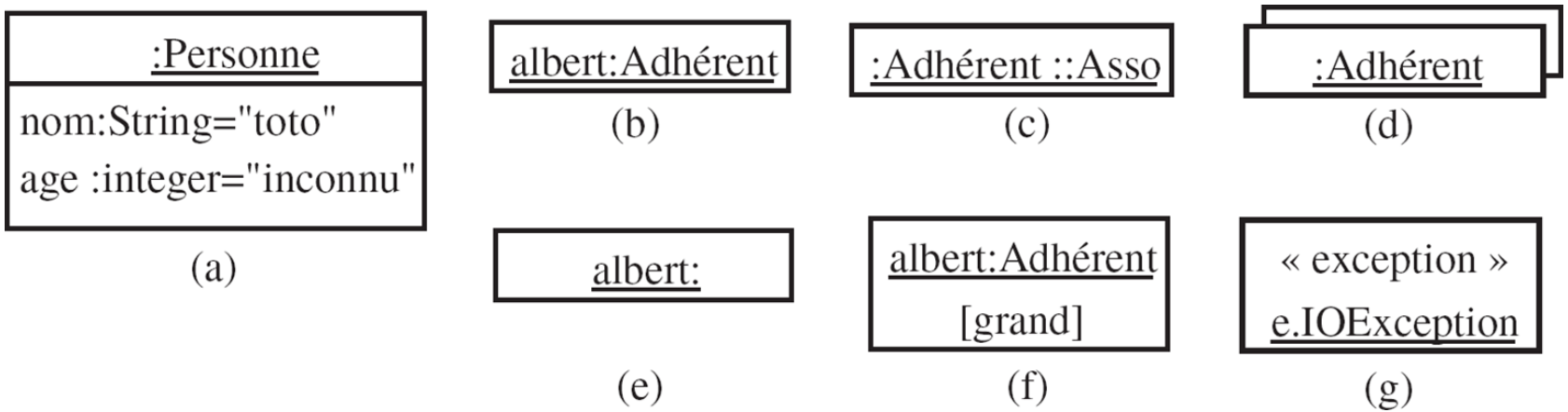


Diagramme de classe et diagrammes d'objets

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

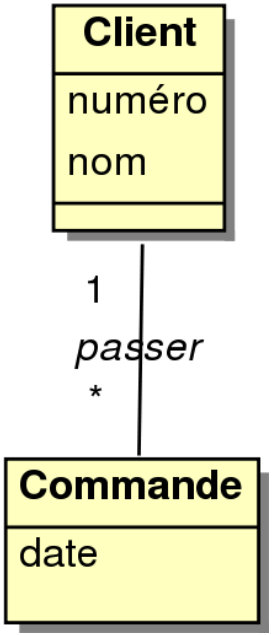


Diagramme **cohérent** avec le diagramme de classe.

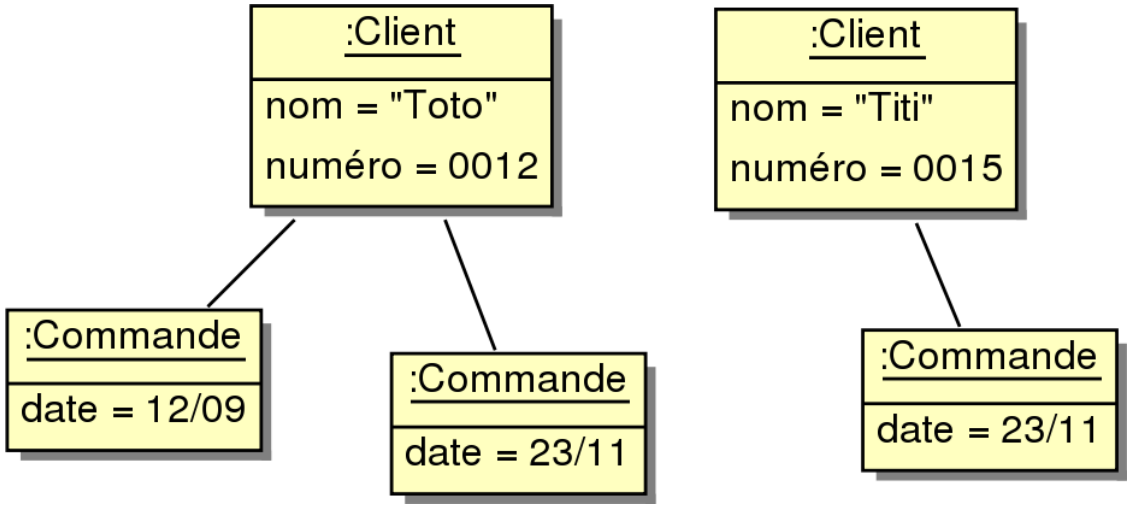


Diagramme de classe et diagrammes d'objets

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

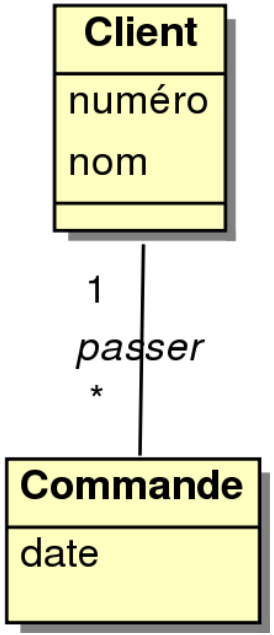
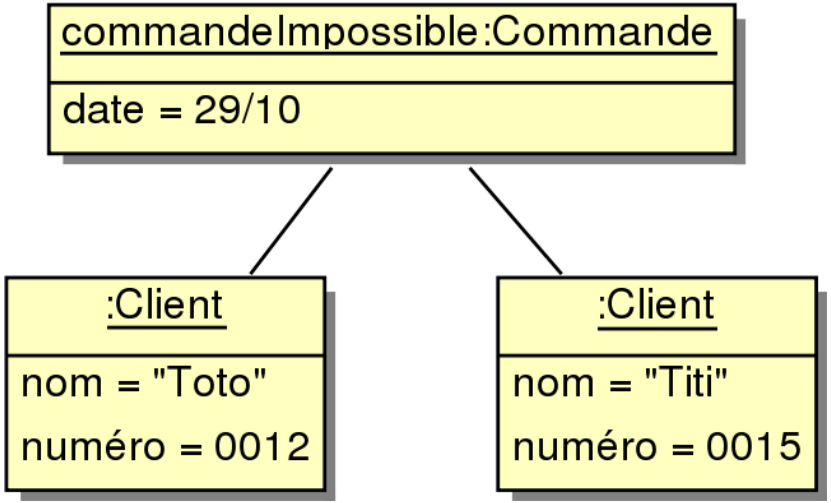


Diagramme **incohérent** avec le diagramme de classe.



Liens entre objets

- Un **lien** est une instance d'une association.
- Un lien se représente comme une association mais s'il a un nom, il est souligné.

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

Liens entre objets : exemple

Une association est une abstraction des liens qui existent entre les objets instances des classes associées.

Diagramme de classes

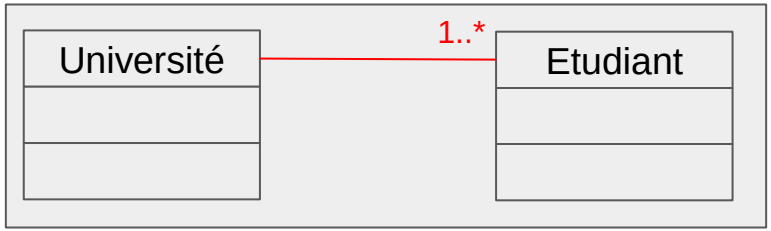
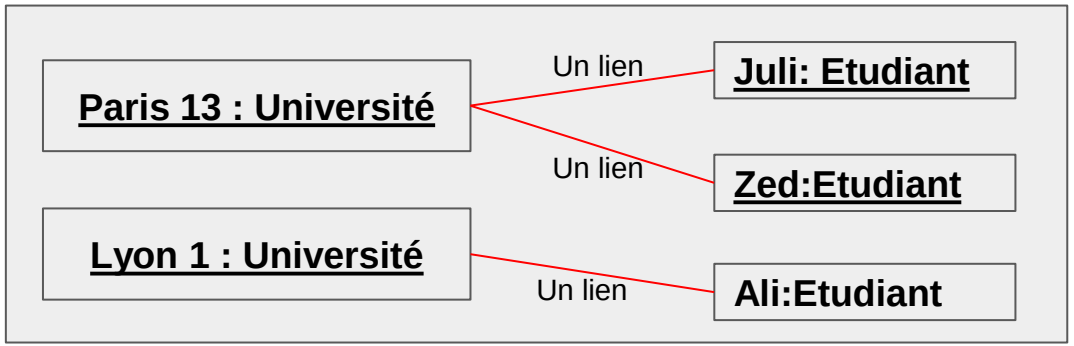
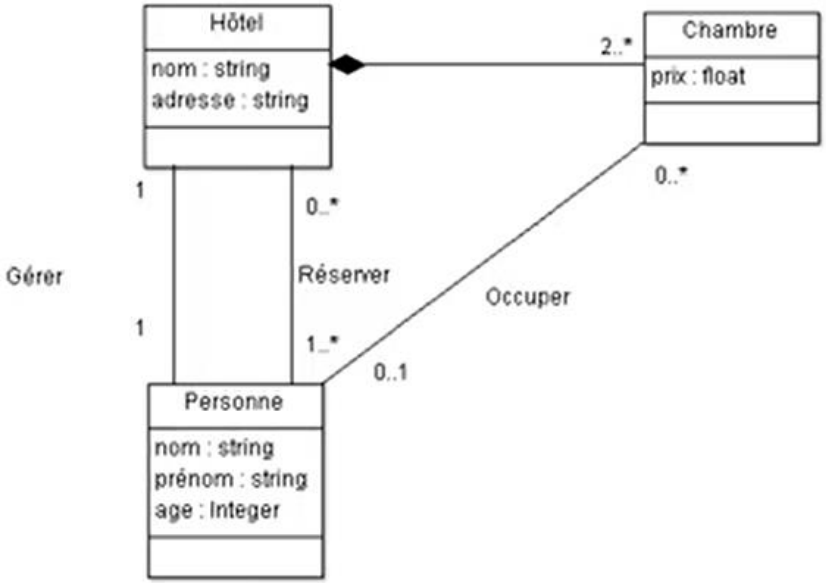


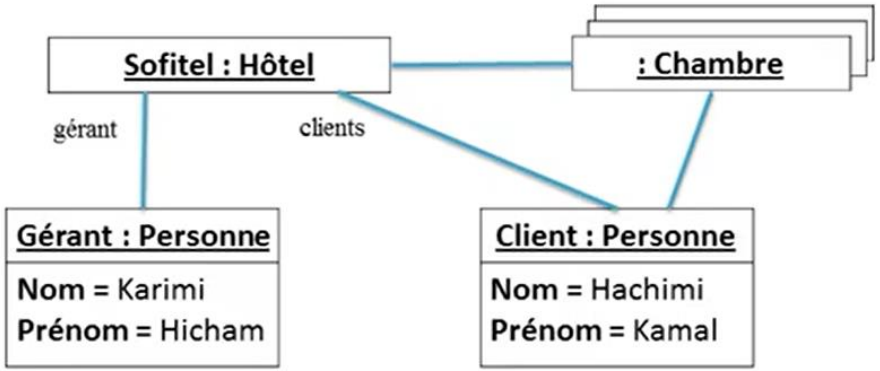
Diagramme d'objets



Exemple de diagramme d’objets



En se basant sur ce diagramme de classe, fait un diagramme d’objet qui représente la situation suivante : **L’hôtel Sofitel dont le gérant Mr. Julien Auberger s’occupe, possède 50 chambres. L’une des chambres est loué à Mr. Brahimi Jazz.**



Plan

- Concepts de l’approche objet
- Modélisation UML
- **Modélisation d’objet élémentaire avec UML**
  - Diagrammes de cas d’utilisation
  - Diagrammes de classes
  - Diagrammes d’objets
  - **Diagrammes de séquences**

Objectif des diagrammes de séquence (interactivité)

- Les **diagrammes de cas d'utilisation** modélisent à **QUOI** sert le système, en organisant les interactions possibles avec les acteurs.
- Les **diagrammes de classes** permettent de spécifier la structure et les liens entre les objets dont le système est composé : ils spécifie QUI sera à l'oeuvre dans le système pour réaliser les fonctionnalités décrites par les diagrammes de cas d'utilisation.
- Les **diagrammes de séquences COMMENT** permettent de décrire les éléments du système interagissent entre eux et avec les acteurs.
    - Les objets au coeur d'un système interagissent en s'échangent des messages.
    - Les acteurs interagissent avec le système au moyen d'IHM (Interfaces Homme-Machine).

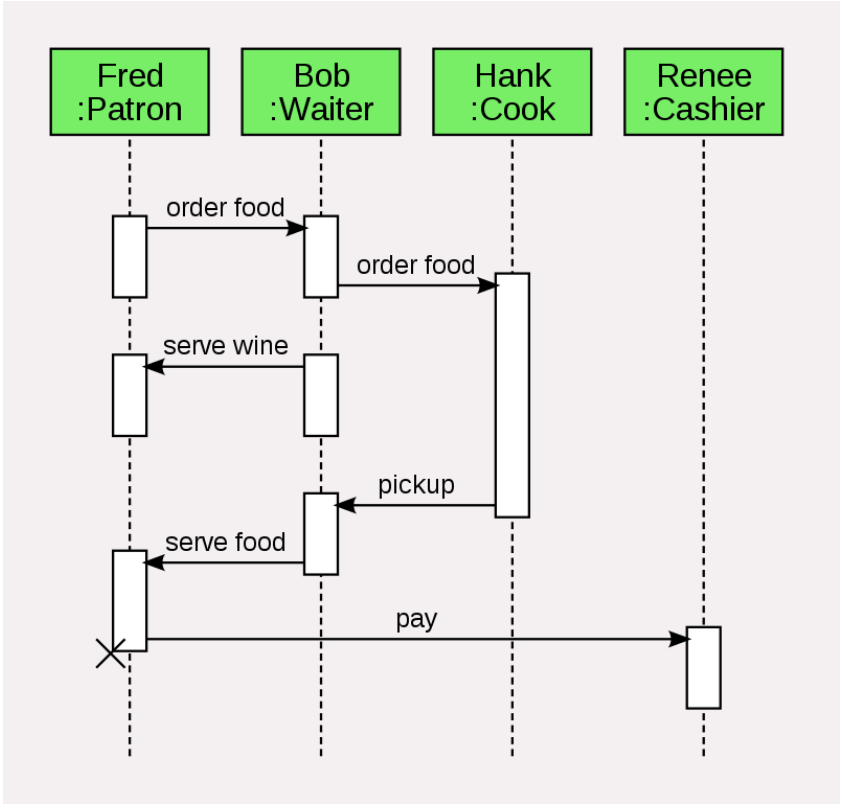
Avantages des diagrammes de séquence

On peut dessiner un diagramme de séquence pour :

- Représenter les détails d’un cas d’utilisation
- Modéliser le déroulement logique d’une procédure, fonction ou une opération complexe
- Voir comment les objets et les composants interagissent entre eux pour effectuer un processus
- Schématiser et comprendre le fonctionnement détaillé d’un scénario existant ou à venir.

Diagramme de séquences : exemple de compréhension

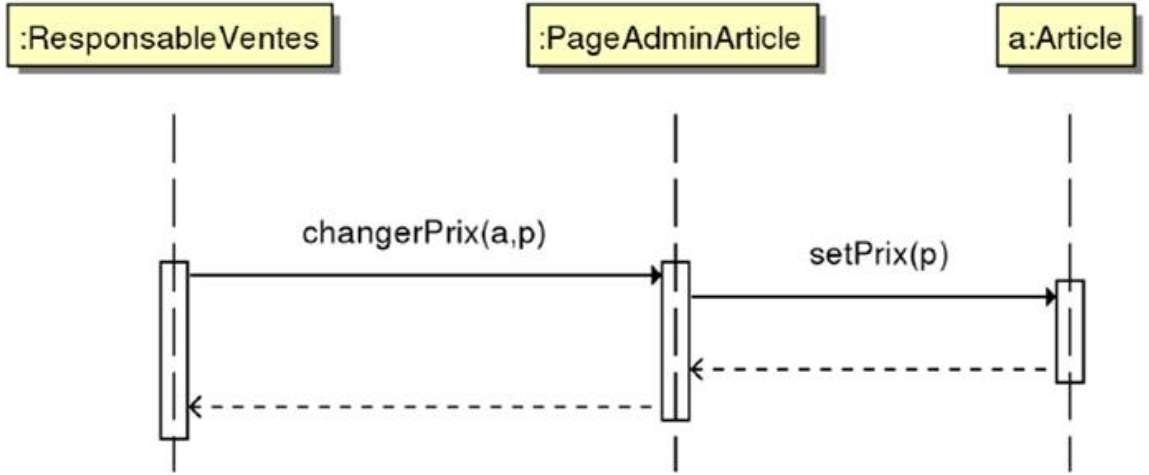
- **Diagramme de séquences** d'un restaurant :
  1. Le client passe la commande au serveur
  2. Le serveur passe la commande au cuisinier
  3. Le serveur sert le vin
  4. Le serveur récupère la commande au cuisinier
  5. Le serveur sert le repas
  6. Le client paie



Exemple d'interaction

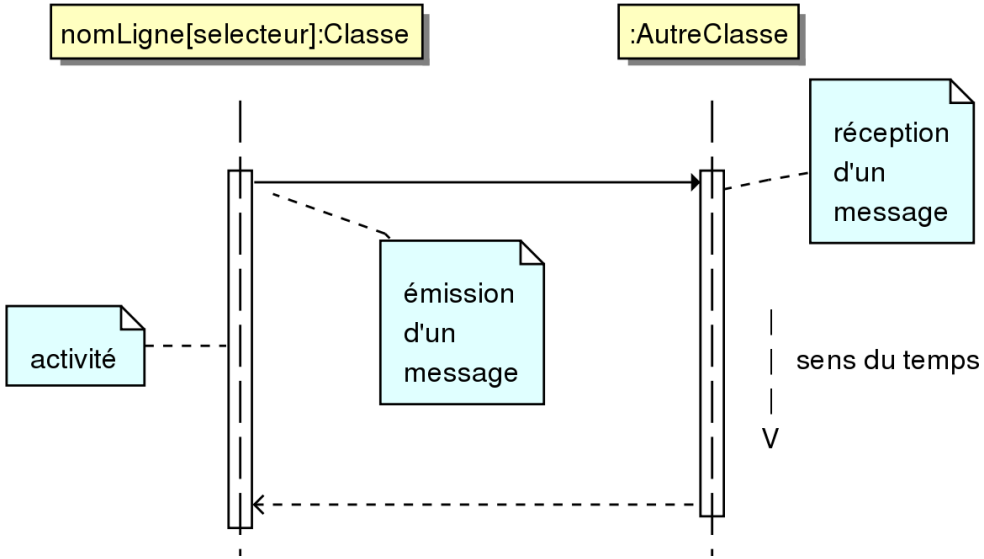
- Cas d'utilisation :

- Diagramme de séquences correspondant :



Ligne de vie

- Une ligne de vie représente un participant à une interaction (objet ou acteur).
- Dans le cas d'une collection de participants, un sélecteur permet de choisir un objet parmi n (par exemple objets[2]).



Messages

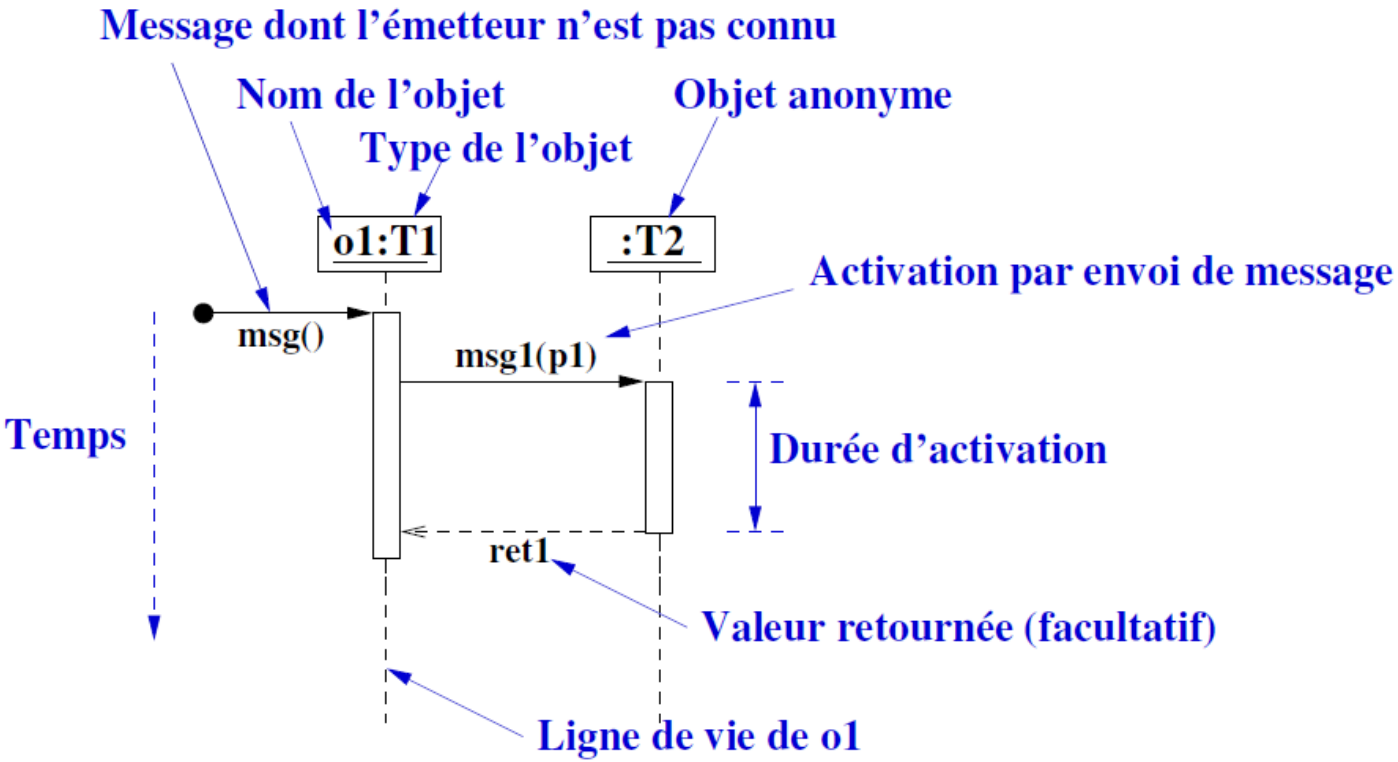
- Les principales informations contenues dans un diagramme de séquence sont les messages échangés entre les lignes de vie, présentés dans un ordre chronologique.
  - Un message définit une communication particulière entre des lignes de vie (objets ou acteurs).
  - Plusieurs types de messages existent, dont les plus courants :
    - l'envoi d'un signal ;
    - l'invocation d'une opération (appel de méthode) ;
    - la création ou la destruction d'un objet.
- La réception des messages provoque une période d'activité (rectangle vertical sur la ligne de vie) marquant le traitement du message (spécification d'exécution dans le cas d'un appel de méthode).

Principaux types de messages

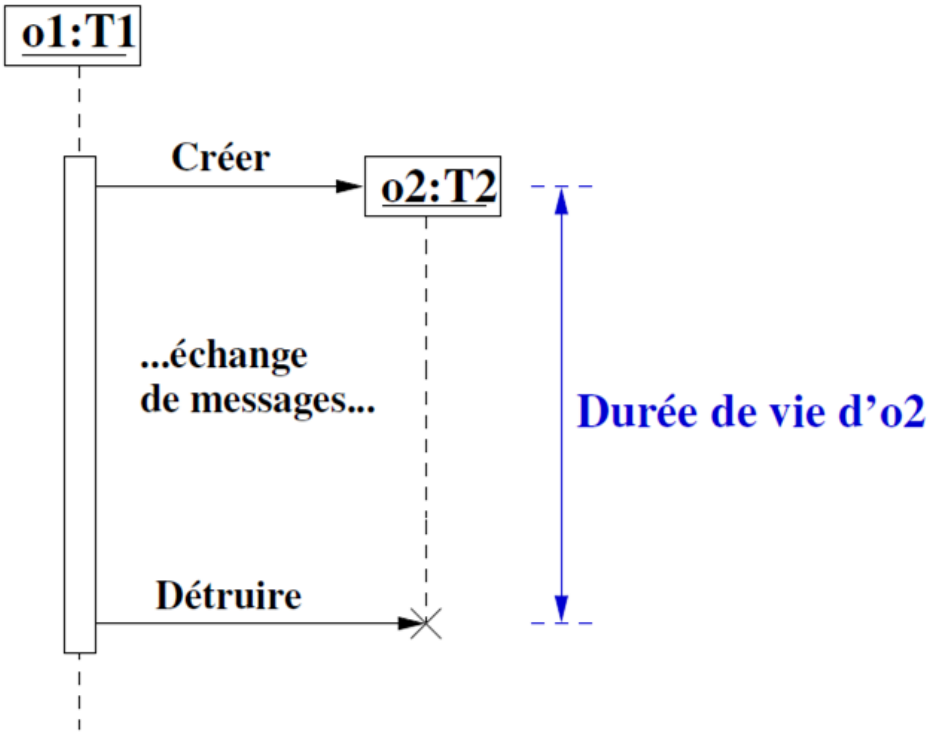
- Un message **synchrone** bloque l'expéditeur jusqu'à la réponse du destinataire. Le ot de contrôle passe de l'émetteur au récepteur.
  - Typiquement : appel de méthode
    - Si un objet A invoque une méthode d'un objet B, A reste bloqué tant que B n'a pas terminé

  - On peut associer aux messages d'appel de méthode un message de retour (en pointillés) marquant la reprise du contrôle par l'objet émetteur du message synchrone.
- 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).

Ligne de vie et activation

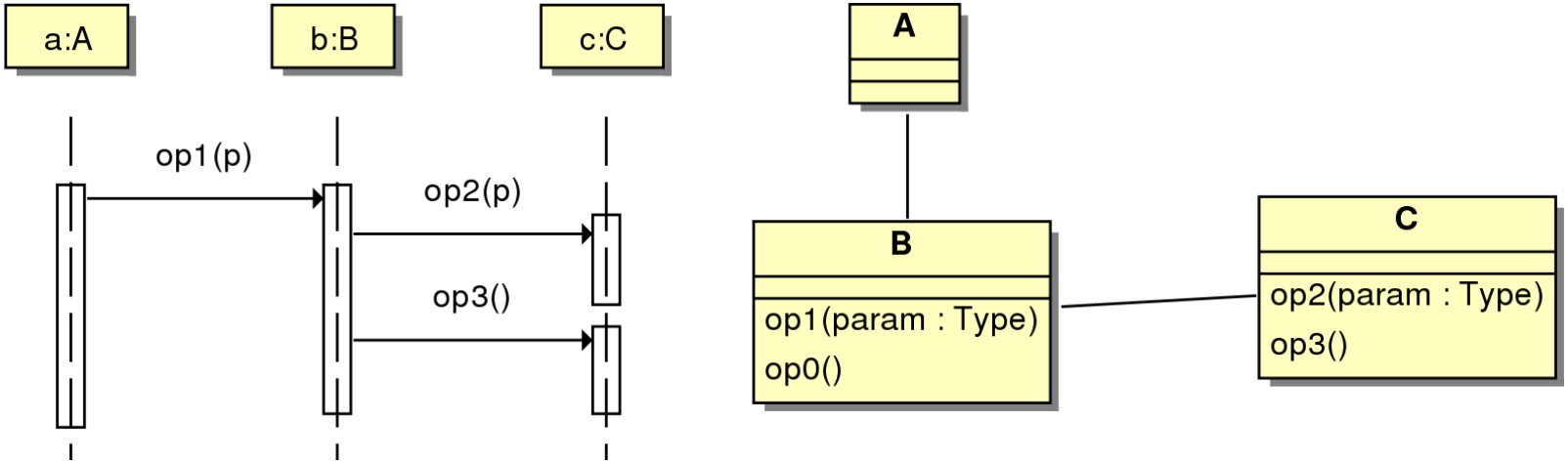


Création et destruction d'objets



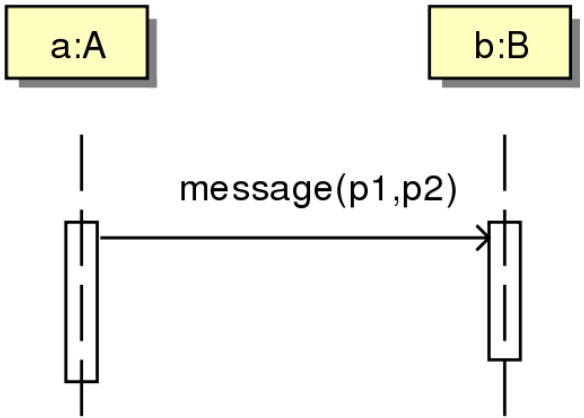
Correspondance messages / opérations

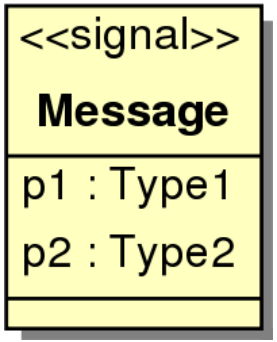
- Les **messages synchrones** correspondent à des **opérations** dans le diagramme de classes.
- Envoyer un message et attendre la réponse pour poursuivre son activité revient à invoquer une méthode et attendre le retour pour poursuivre ses traitements.



Correspondance messages / signaux

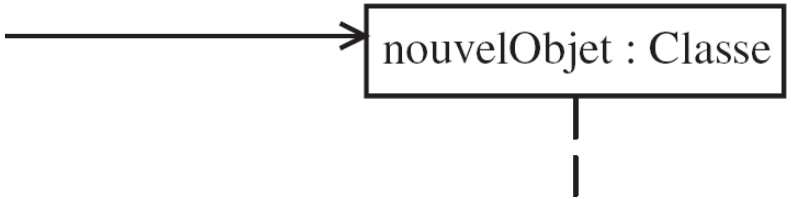
- Les messages asynchrones correspondent à des signaux dans le diagramme de classes.
- Les signaux sont des objets dont la classe est stéréotypée signal et dont les attributs (porteurs d'information) correspondent aux paramètres du message.



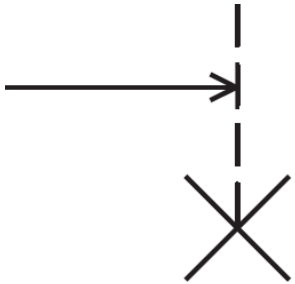


Création et destruction de lignes de vie

- La **création** d'un objet est matérialisée par une flèche qui pointe sur le sommet d'une ligne de vie.
  - On peut aussi utiliser un message asynchrone ordinaire portant le nom <<create>>.



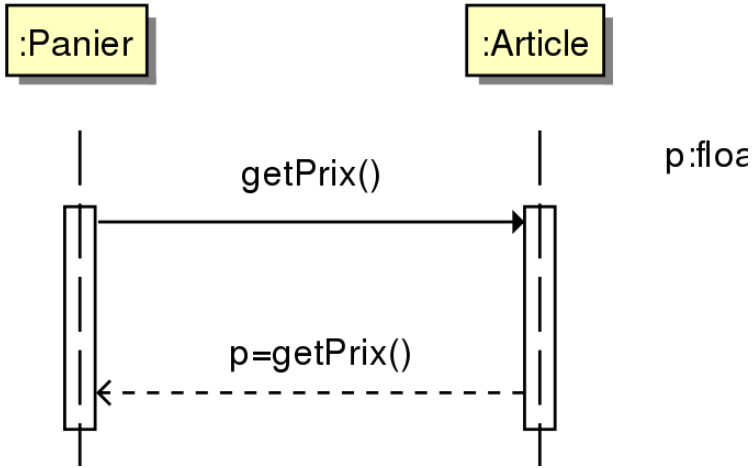
- La **destruction** d'un objet est matérialisée par une croix qui marque la fin de la ligne de vie de l'objet.



Messages de retour

- Le récepteur d'un message **synchrone** rend la main à l'émetteur du message en lui envoyant un **message de retour**

- Les messages de retour sont optionnels : la fin de la période d'activité marque également la fin de l'exécution d'une méthode.
- Ils sont utilisés pour spécifier le résultat de la méthode invoquée.
- Le retour des messages asynchrones s'effectue par l'envoi de nouveaux messages asynchrones.



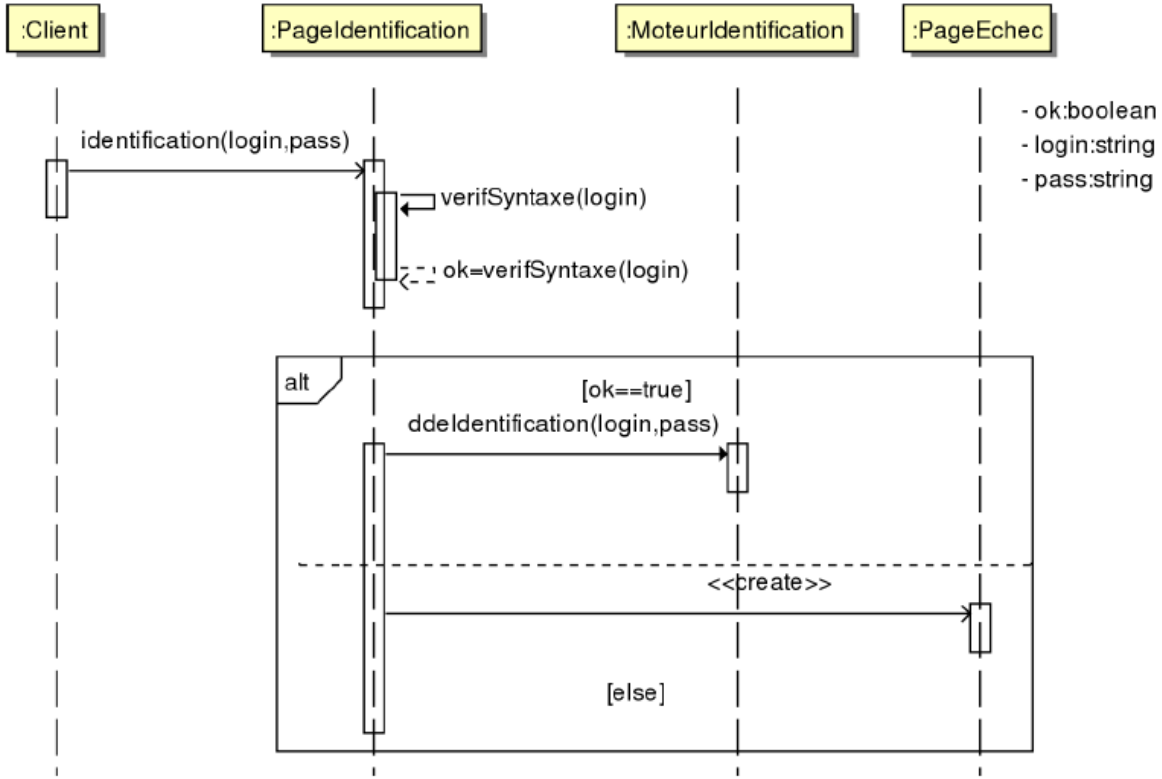
Fragment combiné

- Un **fragment combiné** permet de décomposer une interaction complexe en fragments suffisamment simples pour être compris.
  - Recombiner les fragments restitue la complexité.
  - Syntaxe complète avec UML 2 : représentation complète de processus avec un langage simple (ex : processus parallèles).
- Un **fragment combiné** se représente de la même façon qu'une interaction. Il est représenté un rectangle dont le coin supérieur gauche contient un pentagone.
  - Dans le pentagone figure le type de la combinaison (appelé opérateur d'interaction).

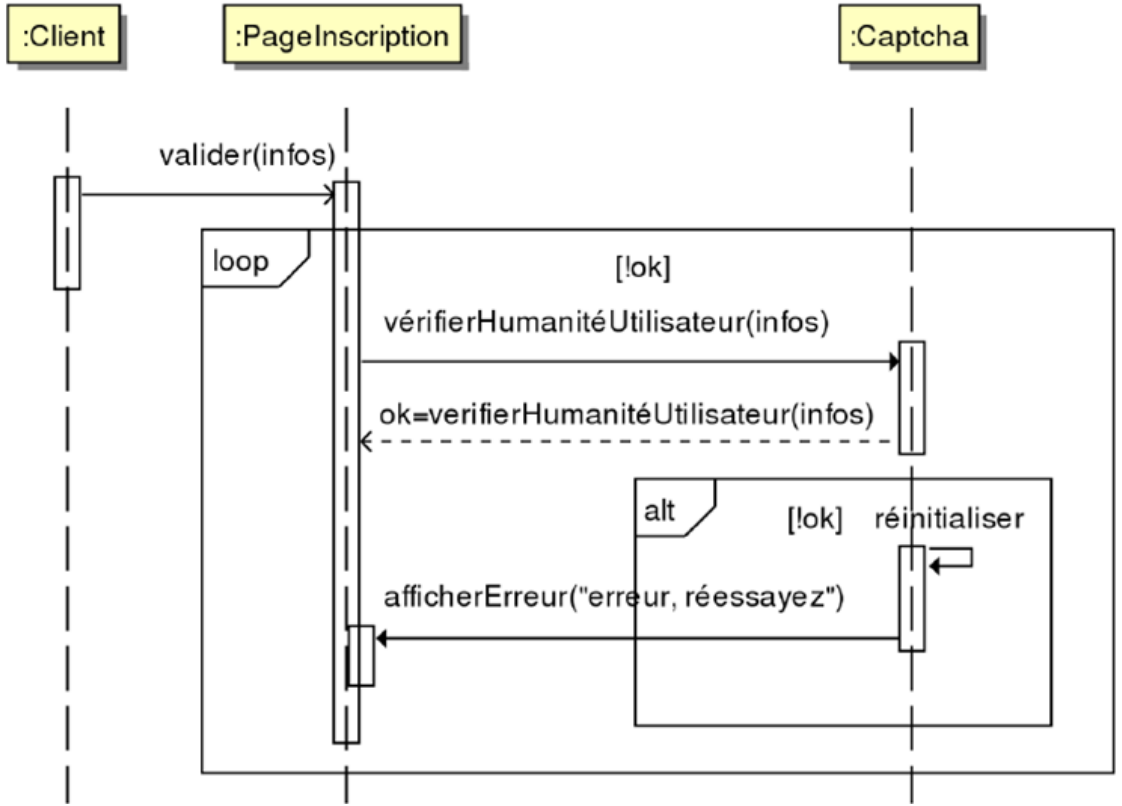
Différentes type de fragments combiné

- **alt** : fragments multiple alternatifs (si alors sinon)
- **opt** : fragment optionnel
- **par** : fragment parallèle (traitements concurrents)
- **loop** : le fragment s'exécute plusieurs fois
- **region** : région critique (un seul thread à la fois)
- **neg** : une interaction non valable
- **break** : représente des scénario d'exception
- **ref** : référence à une interaction dans un autre diagramme
- **sd** : fragment du diagramme de séquence en entier

Exemple de fragment avec l'opérateur conditionnel

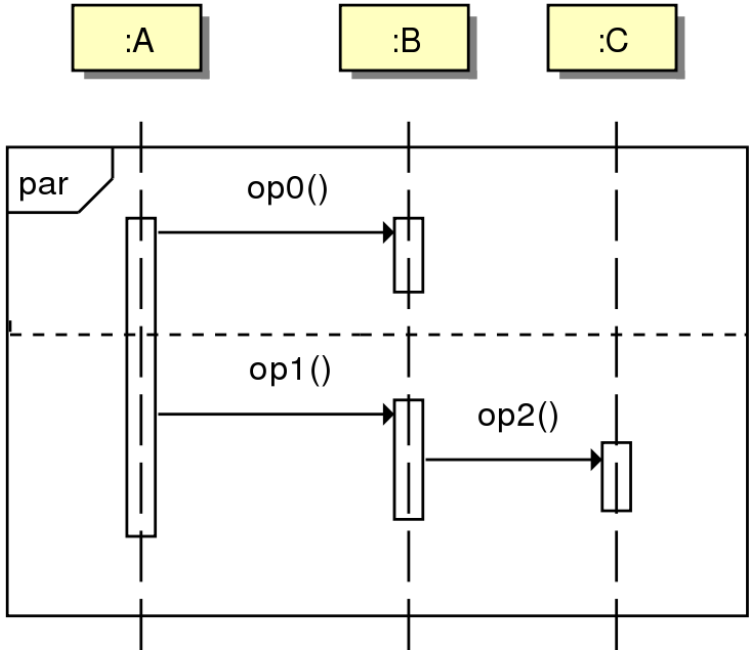


Opérateur de boucle



Opérateur parallèle

- 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.



Réutilisation d'une interaction

- **Réutiliser une interaction** consiste à placer un fragment portant la référence <<**ref**>> là où l'interaction est utile.
- On spécie le nom de l'interaction dans le fragment.