

UML (Unified Modeling Language)

Modélisation Objet

La modélisation

Elle est essentielle pour :

- Comprendre le fonctionnement d'un système
- Maîtriser la complexité
- Faciliter la communication au sein de l'équipe

Et particulièrement en génie logiciel :

- Être un facteur de réduction des coûts et des délais,
- Être un facteur d'accroissement de la qualité du produit,
- Permettre d'assurer une maintenance facile et efficace,
- Permettre de contrôler l'avancement d'un projet.

Modèles et techniques utilisés par les méthodes objets

Les méthodes de modélisation 'classiques' sont basées sur :

- Un modèle de données et un modèle des traitements séparés
- Une modélisation de flots de données

Insatisfaisantes pour modéliser des systèmes objet !!

Aucune méthode ne couvre toutes les étapes du cycle de développement

Triple perception du système d'information

- Dimension statique: objets
- Dimension dynamique: événements/états
- Dimension fonctionnelle : flux/processus

Grandes étapes

- Identifier les entités du domaine.
- Structurer le domaine en analysant les propriétés de ces entités et leurs relations.
- Identifier les opérations que savent effectuer ces entités.
- Décrire précisément ces opérations en les reliant à des messages.
- Décrire le lancement du programme.

Avantages de la conception objet

Avantages de l'utilisation de l'approche objet au niveau conceptuel

- Réduction de la « distance » entre langage utilisateur et langage conceptuel
- Regroupement de l'analyse des données et des traitements
- Simplification des transformations entre niveau conceptuel et niveau physique
- Abstraction forte
- Orienté vers la réutilisation : notion de composants, modularité, extensibilité, adaptabilité, souplesse.

Historique

Début des années 1990

- Les premiers processus de développement OO apparaissent
- Prolifération des méthodes et notations étaient la cause de grande confusion Méthode OOD de Grady Booch (1991)
- Méthode OMT de James Rumbaugh (1991)
- Méthode OOSE de Ivar Jacobson (1991)
- Méthode OOA/OOD de Coad and Yourdon (1992)
- Méthode de Schlaer and Mellor (1992)

Fin 1994(J. Rumbaugh rejoint G. Booch chez Rational Software).

OMT + OOD → Unified Method (oct 1995)

Fin 1995(I. Jacobson les rejoint chez Rational Software)

Unified Method + OOSE → UML 0.9 (juin 1996)

Début 1997(Partenaires divers : Microsoft, Oracle, IBM, HP et autres leaders collaborent)

→ UML 1.0 (jan 1997)

Fin 1997(l'OMG (Object Management Group) retient UML 1.1 comme norme de modélisation)

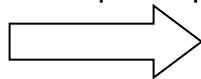
Les versions se succèdent :

- Début 1998(UML 1.2)
- En 1998(UML 1.3)
- En 2001(UML1.4)
- En 2003(UML 1.5)
- En 2005(UML 2.0)

Qu'est ce que UML ?

Langage = syntaxe + sémantique :

- Syntaxe : notations graphiques consistant essentiellement en des représentations conceptuelles d'un système
- Sémantique : sens précis pour chaque notation



UML : langage de modélisation

graphique et textuel

- UML unifie
- Les concepts, quels que soient le domaine d'application
- Les notations et concepts orientés objet
- UML est indépendant
- Du type du système-logiciel, matériel, organisation..
- Du domaine métier : gestion, ingénierie, finance...
- UML permet de :
 - Comprendre et de décrire les besoins,
 - Concevoir et construire des solutions,
 - Documenter un système tout au long du cycle de développement,
 - Communiquer entre les membres de l'équipe de projet.

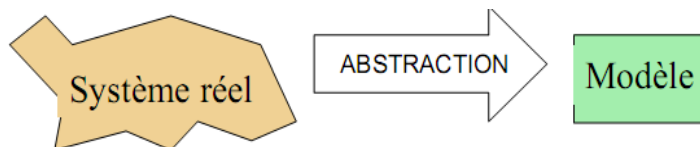
- Supporté par plusieurs outils (AGL) : Objecteering, Open tools, Rational Rose, PowerAMC, WinDesign, WINDEV...

Attention

UML est un langage (et non pas une méthode) qui :

- Permet de représenter les modèles
- Ne définit pas le processus d'élaboration des modèles.

Modélisation en diagrammes



Modèle = ensemble d'éléments de modélisation vus dans un ensemble de diagrammes

Diagramme

- Support graphique de modélisation, chaque diagramme propose un point de vue différent.
- Mise en œuvre d'un ensemble d'éléments de visualisation représentant des éléments de modélisation (graphe)

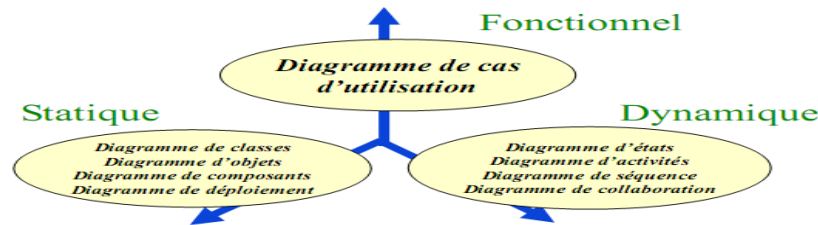
Diagrammes d'UML

UML propose de décrire le système à l'aide de 9 (13) (4 autres nouveaux diagrammes en 2004) diagrammes. Chacun de ces diagrammes correspond :

- Soit à la description d'une partie du système
- Soit à la description du système selon un point de vue particulier.
- **Diagramme de cas d'utilisation** : destiné à représenter les besoins des utilisateurs par rapport au système. Point de vue de l'utilisateur.
- **Diagramme de classes** : Structure statique en termes de classes et relations qui les lient.
- **Diagramme objets** : Représentation des objets et de leurs relations (liens).
- **Diagramme d'état/transitions** : montre les différents états par lesquels passe un objet en réactions aux événements.
- **Diagramme d'activités** : Représentation du comportement d'une opération, d'un cas d'utilisation, ou d'un processus métier en terme d'actions
- **Diagramme de séquences** : : Représentation temporelle des objets et de leurs interactions
- **Diagramme de collaboration** : une autre représentation des scénarios des cas d'utilisation qui met l'accent sur les objets et les messages échangés.
- **Diagramme de composants** : représente les différents constituants logiciel d'un système en termes de modules : fichiers source, bibliothèques, exécutables, etc.
- **Diagramme de déploiement** : montre la disposition physique des matériels qui composent le système et la répartition des composants sur ces matériels, modes de connexion...

Classification des diagrammes

- L'ensemble des 9 diagrammes peut être réparti sur les trois axes de modélisation



Diagrammes de cas d'utilisation

Introduction Cas d'utilisation :

- Le concept de cas d'utilisation introduit par Ivar Jacobson dans la méthode Object-Oriented Software Engineering (OOSE).
- Les fonctionnalités du système sont décrites comme un ensemble de cas d'utilisation.
- Chaque cas représente un flot spécifique d'événements vers le système.
- La description du cas d'utilisation définit ce qui arrive dans le système lors de sa réalisation.

Diagramme des cas d'utilisation

- Décrit, sous forme d'actions et de réactions, le comportement d'un système du point de vue d'un utilisateur.
- Permet de définir les limites du système et ses relations avec l'environnement.
- Sert à modéliser les aspects dynamiques d'un système (Contrairement aux diagrammes de classes).
- Fait ressortir les acteurs et les fonctions offertes par le système.
- Utilisé pour modéliser les exigences (besoins) du client Comportent plusieurs éléments :
 - Acteurs
 - Cas d'utilisation
 - Relations de dépendances, de généralisations et d'associations

Acteurs



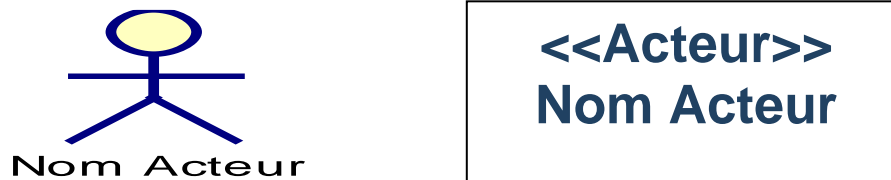
- Entité qui agit sur le système; représente un ensemble cohérent de rôles qu'un utilisateur peut effectuer.
- Le terme acteur ne désigne pas seulement des utilisateurs humains mais également les autres systèmes (machines, programmes, ...)

- Un acteur est un rôle joué par une entité externe qui agit sur le système (Comptabilité, service commercial, ...), en échangeant de l'information (en entrée et en sortie)

Remarques

- La même personne physique peut jouer le rôle de plusieurs acteurs (Chef d'agence est un client de la banque).
- D'autres parts, plusieurs personnes peuvent jouer le même rôle, et donc agir comme un même acteur (plusieurs personnes peuvent jouer le rôle d'administrateur).

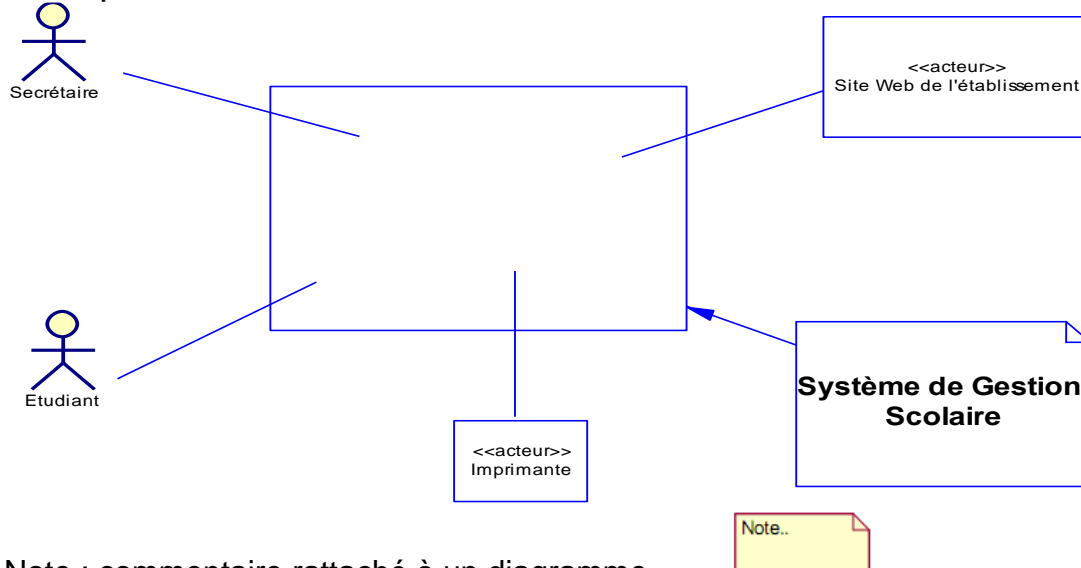
Peut être représenté de deux manières différentes :



Les acteurs peuvent être de trois types :

- Humains : utilisateurs du logiciel à travers son interface graphique, par exemple.
- Logiciels : disponibles qui communiquent avec le système grâce à une interface logicielle (API, ODBC, ...)
- Matériels : exploitant les données du système ou qui sont pilotés par le système (Imprimante, robots, automates, ...)

Exemple



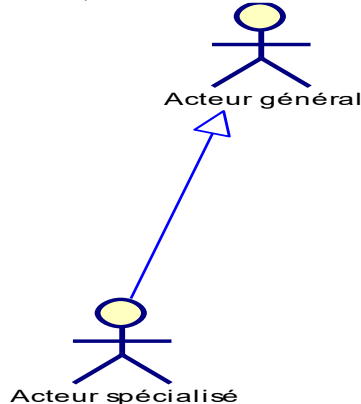
Note : commentaire rattaché à un diagramme

Différentes catégories d'acteurs

Catégories, du point de vue système on distingue deux types :

- **Acteurs principaux** : utilisent les fonctions principales du système. Par exemple, le client pour un distributeur de billets.
- **Acteurs secondaires** : effectuent des tâches administratives ou de maintenance. Par exemple, la personne qui recharge la caisse contenue dans le distributeur

- Un acteur peut être une spécialisation d'un autre acteur déjà défini. Dans ce cas, on utilise la relation de généralisation/spécialisation.



Cas d'utilisation (Use Case):

Introduit par Ivar Jacobson en 1992 dans sa méthode Object-Oriented Software Engineering (OOSE).

- Technique de description du système étudié privilégiant le point de vue de l'utilisateur.
- Repris par UML dans la but de :
 - Effectuer une bonne délimitation du système
 - Améliorer la compréhension de son fonctionnement interne

Questions à se poser :

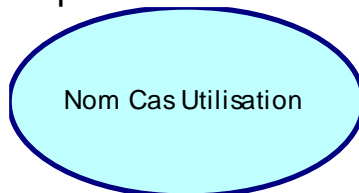
- Quelles sont les tâches de l'acteur ?
- Quelles informations l'acteur doit-il créer, sauvegarder, modifier, détruire, ou simplement lire ?
- L'acteur doit-il informer le système de changements externes ?
- On s'intéresse au domaine du 'quoi faire', pas du 'comment' (sinon on rentre dans la phase de conception).
- On doit rester au niveau de l'interaction acteur/système.

Cas d'utilisation

Les cas d'utilisations

- Permettent de modéliser les attentes (besoins) des utilisateurs.
- Représentent les fonctionnalités du système.
- Suite d'événements, initiée par des acteurs, qui correspond à une utilisation particulière du système.
- L'image d'une fonctionnalité du système, déclenchée en réponse à la stimulation d'un acteur externe.

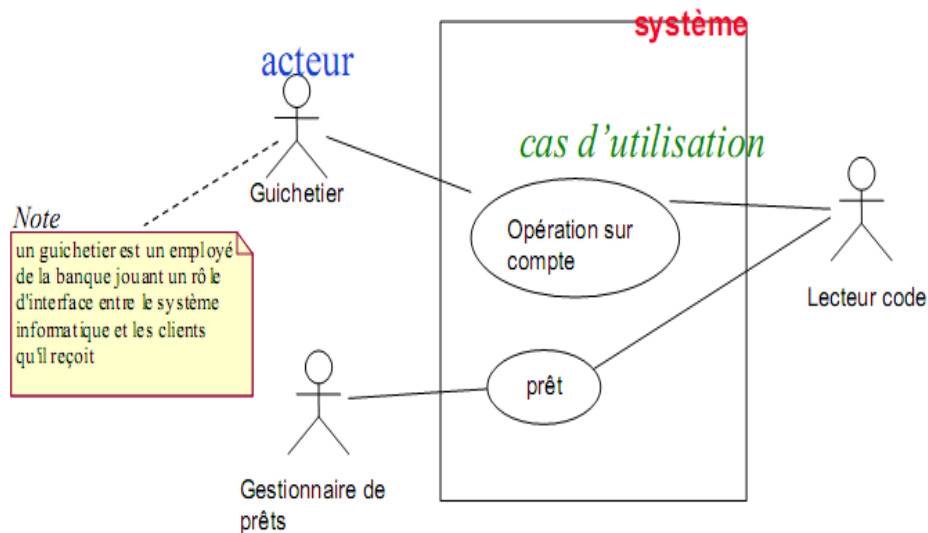
Représentation Cas d'utilisation



Démarche

- Recherche des acteurs externes
- Pour chaque acteur les cas d'utilisation
- Pour chaque cas d'utilisation :
 - rechercher les interactions
 - rechercher les objets manipulés
- Faire la maquette de chaque cas d'utilisation

Remarque : Les diagrammes des cas d'utilisation se retrouveront à tous les stades du projet.

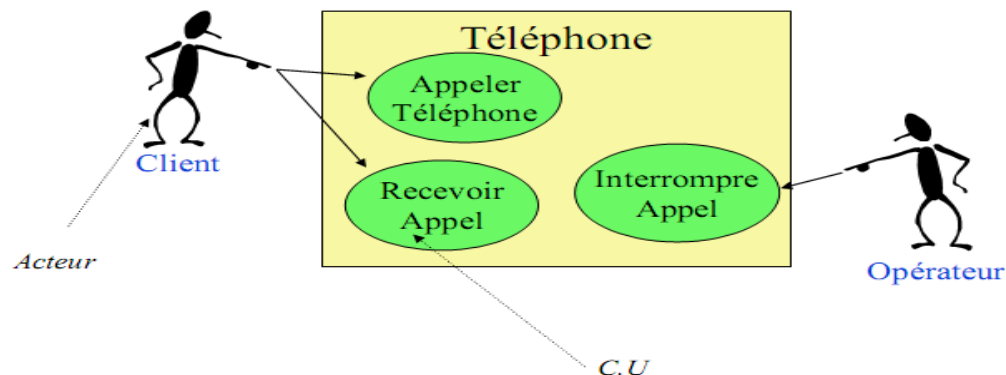
Acteurs et cas d'utilisation

Cas d'utilisation : description générique d'une transaction complète entre l'acteur et le système (claire et précise).

Remarque : pas d'interactions entre acteurs

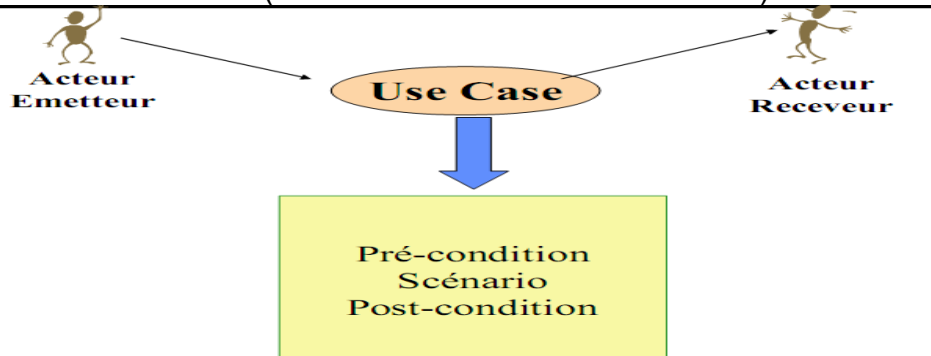
C.U : exemple

Diag. C.U Système téléphonique

**Caractéristiques**

- Identification d'une finalité de l'utilisateur
- Un stimulus de départ
- Une pré-condition du système au déclenchement

- Un enchaînement d'interactions
- Une post-condition du système à la fin du cas d'utilisation
- Enchaînement d'actions à effectuer
- Une fin normale (conditions d'exécution éventuelles)



Exemple (1)

- Nom du cas d'utilisation: **Attribution d'une place sur un vol.**
- Un stimulus de départ **Client présente sa réservation.**
- Une pré-condition: **Guichet ouvert & réservation sur un vol.**
- Un enchaînement d'interactions: **Scénarios.**
- Une post-condition: **Place affectée au passager & Fin-réservation**

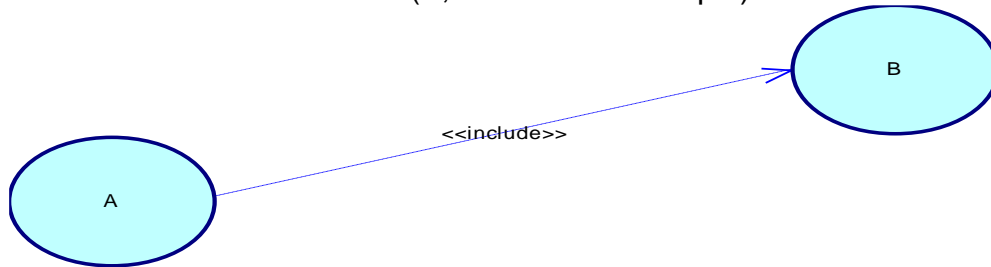
Structuration des cas d'utilisation

UML définit trois types de relations standardisées entre cas d'utilisation :

- Une relation d'inclusion, formalisée par la dépendance «**include**»
- Une relation d'extension, formalisée par la dépendance «**extend**»
- Une relation de **généralisation/spécialisation**

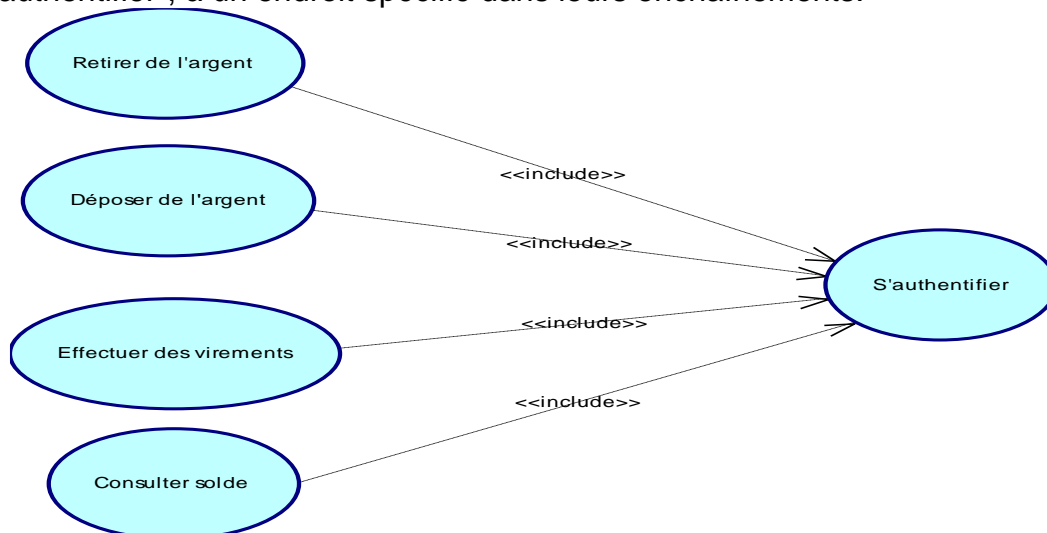
Relation d'inclusion

- A inclut B : le cas A inclut obligatoirement le comportement défini par le cas B; permet de factoriser des fonctionnalités partagées
- Le cas d'utilisation pointé par la flèche (dans notre cas B) est une sous partie de l'autre cas d'utilisation (A, dans notre exemple).



Exemple :

Les cas d'utilisation "Déposer de l'argent", "Retirer de l'argent", "Effectuer des virements" et "Consulter solde" incorporent de façon explicite le cas d'utilisation "S'authentifier", à un endroit spécifié dans leurs enchaînements.



Relation d'inclusion : On utilise cette relation pour éviter de décrire plusieurs fois un même enchaînement d'actions. Ainsi, on est amené à factoriser un comportement commun à plusieurs cas d'utilisation dans un cas d'utilisation à part.

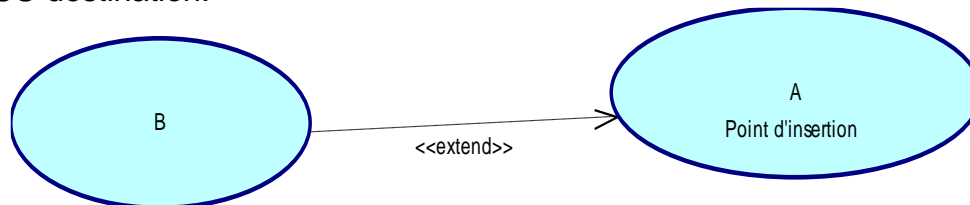
Remarques

- La relation include n'a pour seul objectif que de factoriser une partie de la description d'un cas d'utilisation qui serait commune à d'autres cas d'utilisation.
- Le cas d'utilisation inclus dans les autres cas d'utilisation n'est pas à proprement parlé un vrai cas d'utilisation car il n'a pas d'acteur déclencheur ou receveur d'évènement. Il est juste un artifice pour faire de la réutilisation d'une portion de texte.

Relation d'extension

La relation stéréotypée «extend» permet d'étendre les interactions et donc les fonctions décrites dans les cas d'utilisation, mais sous certaines contraintes.

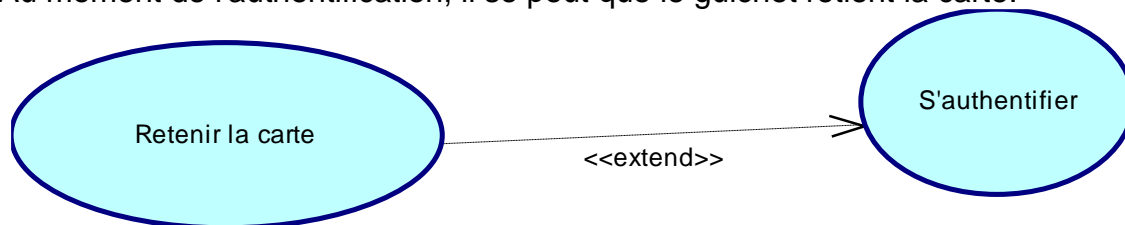
- Le CU source (B) ajoute, sous certaines conditions, son comportement au CU destination (A)
- En d'autres termes, le CU B peut être appelé au cours de l'exécution du CU A
- Le comportement ajouté s'insère au niveau d'un point d'extension défini dans le CU destination.



- Le cas d'utilisation de destination peut fonctionner tout seul, mais il peut également être complété par un autre cas d'utilisation, sous certaines conditions.
- On utilise principalement cette relation pour séparer le comportement optionnel (les variantes) du comportement obligatoire.

Exemple :

Au moment de l'authentification, il se peut que le guichet retient la carte.



Relations d'inclusion d'extension

- La relation « extend » montre une possibilité d'exécution d'interactions qui augmenteront les fonctionnalités du cas étendu, mais de façon optionnelle, non obligatoire,
- La relation "include" suppose une obligation d'exécution des interactions dans le cas de base.

Relation d'héritage

- Il peut également exister une relation d'héritage entre cas d'utilisation.
- Cette relation exprime une relation de spécialisation/généralisation au sens classique.

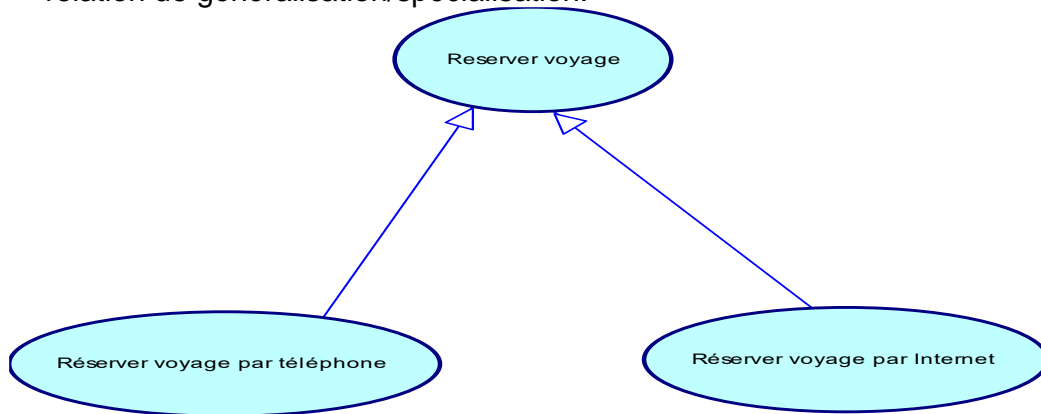
Relation d'héritage : Exemple

Dans un système d'agence de voyage, un acteur "Touriste" peut participer à un cas d'utilisation de base qui est "Réserver voyage", qui suppose par exemple, des interactions basiques au comptoir de l'agence. Une réservation peut être réalisée par téléphone ou par Internet.

Analyse du problème :

- On voit qu'il ne s'agit pas d'une relation "extend", car la réservation par Internet n'étend pas les interactions ni les fonctionnalités du cas d'utilisation "Réserver voyage".

- Les deux cas d'utilisation "Réservation voyage" et "Réserver voyage par Internet" sont liés : la réservation par Internet est un cas particulier de réservation.
- De façon générale en objet, une situation de cas particulier se traduit par une relation de généralisation/spécialisation.



Structuration entre cas d'utilisation

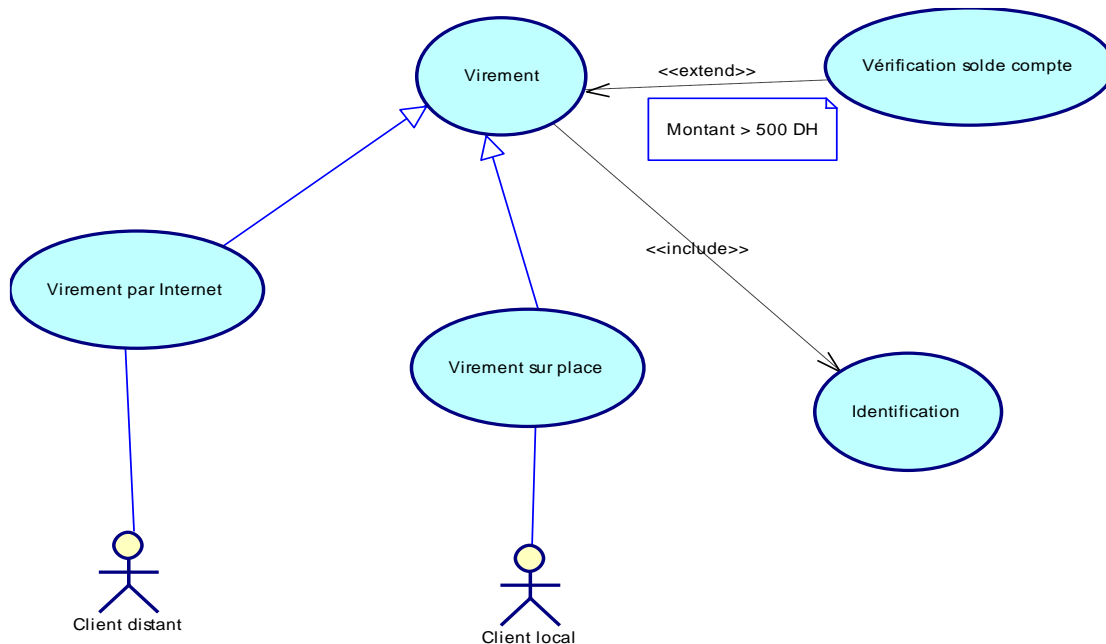
Résumé : Les cas peuvent être structurés par des relations :

- A inclut B : le cas A inclut obligatoirement le comportement défini par le cas B; permet de factoriser des fonctionnalités partagées
- A étend B : le cas A est une extension optionnelle du cas B à un certain point de son exécution.
- A généralise B : le cas B est un cas particulier du cas A.


Relations entre cas d'utilisation : Exemple

Un client peut effectuer un **retrait bancaire**. Le retrait peut être **effectué sur place** ou par **Internet**. Le client **doit être identifié** (en fournissant son code d'accès) pour effectuer un retrait, mais si le **montant dépasse 500DH**, la **vérification du solde de son compte** est réalisée.

Relations entre cas d'utilisation



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 il n'expose pas de façon détaillée le dialogue entre les acteurs et les cas d'utilisation.  Nécessité de décrire ce dialogue


Deux façons sont couramment utilisées pour décrire les cas d'utilisation :

- Description textuelle
- Description à l'aide d'un diagramme de séquence.

Description des cas d'utilisation (description textuelle)

- Identification
 - Nom du cas : retrait d'argent.
 - Objectif : détaille les étapes permettant à un guichetier d'effectuer des opérations de retrait par un client.
 - Acteurs : Guichetier (Principal), Système central (Secondaire).

Description des cas d'utilisation (description textuelle)

- Scénarios  (Scénario = chemin dans le CU)
 - Scénario nominal
 - Le Guichetier saisit le numéro de compte client
 - L'application valide le compte auprès du SC(système central).
 - L'application demande le type d'opération au Guichetier
 - Le Guichetier sélectionne un retrait de 200 DH
 - Le système interroge le SC pour s'assurer que le compte est suffisamment approvisionné.
 - Le SC effectue le débit du compte
 - Le système notifie au guichetier qu'il peut délivrer le montant demandé.
- Scénarios
 - Scénario alternatif (exception)
 - En (5) : si le compte n'est pas suffisamment approvisionné

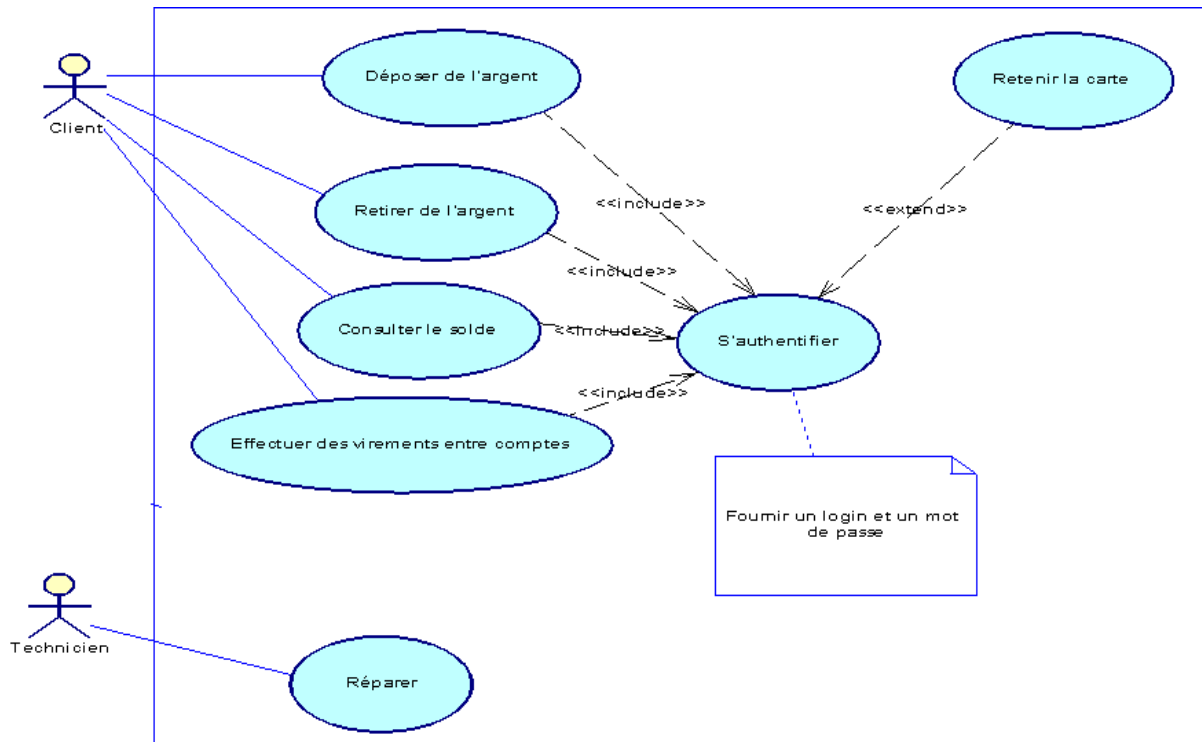
Intérêts des cas d'utilisation

Les CU obligent les utilisateurs à :

- Définir la manière dont ils voudraient interagir avec le système
- Préciser quelles informations ils entendent échanger avec le système
- Décrire ce qui doit être fait pour obtenir le résultat escompté.

Diagramme des cas d'utilisation :

- Le diagramme des cas d'utilisation regroupe dans un même schéma les acteurs et les cas d'utilisation en les reliant par des relations. Le système étant délimité par un cadre rectangulaire.
- La représentation de base d'un cas d'utilisation est une ellipse contenant le nom du cas. L'interaction entre un acteur et un cas d'utilisation se représente comme une association. Elle peut comporter des multiplicités comme toute association entre classes.



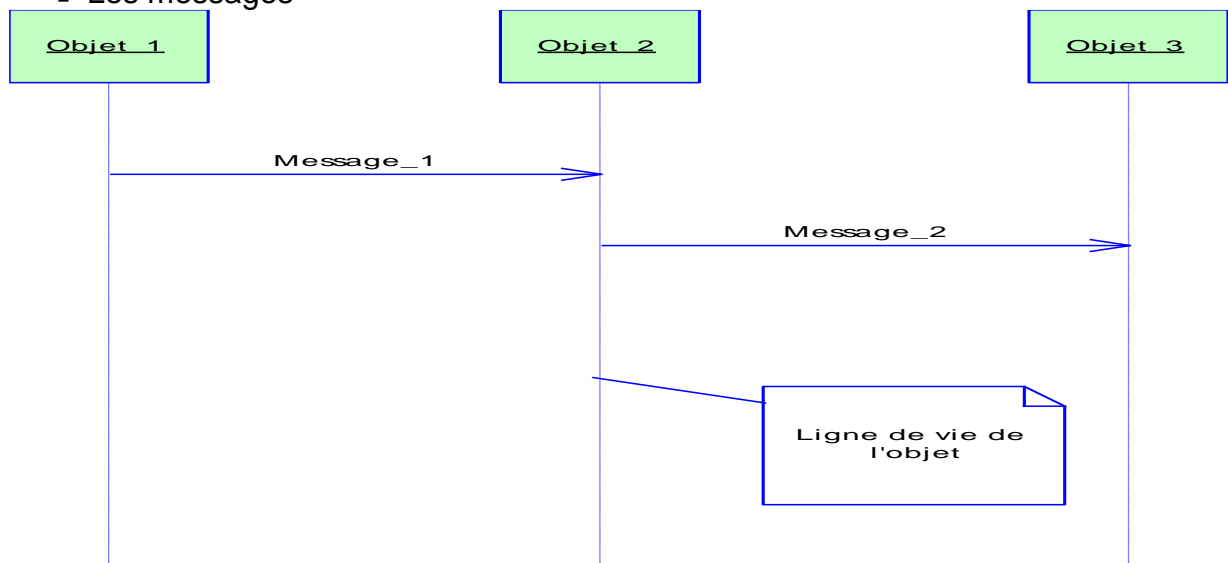
Étapes de construction du diagramme des cas d'utilisation :

Pour modéliser le diagramme des cas d'utilisation, il faut :

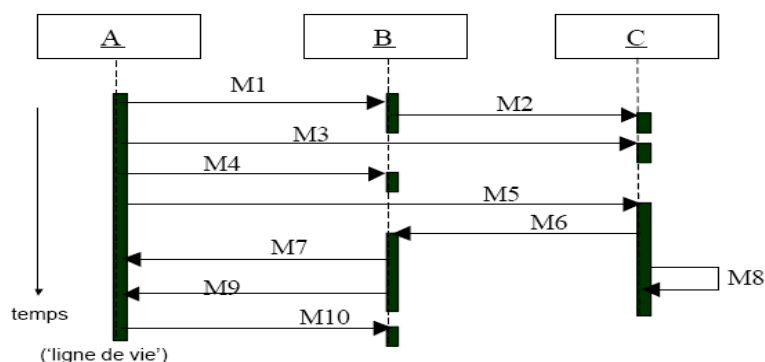
- Identifier les acteurs qui entourent le système. Certains acteurs utilisent le système pour accomplir des tâches (acteurs principaux), d'autres effectuent des tâches de maintenance ou d'administration (acteurs secondaires).
- Organiser les acteurs selon une hiérarchisation de généralisation/spécialisation
- Intégrer les acteurs au diagramme en spécifiant les cas d'utilisation auxquels ils se rapportent
- Structurer les cas d'utilisation pour faire apparaître les comportements partagés (relation d'inclusion), les cas particuliers (généralisation/spécialisation) ou options (relation d'extension)

Diagramme de séquences

- Le diagramme de séquence montre quels sont les objets qui participent à l'exécution du use-case et quels sont les messages qu'ils échangent : description des interactions entre les objets d'un point de vue temporel.
- Chaque bloc ou objet participant dans le processus est représenté par une barre verticale.
- Remarque : l'ordre dans lequel apparaissent les barres n'a pas d'importance (lisibilité).
- Le diagramme de séquence fait ressortir :
 - Les acteurs
 - Les objets
 - Les messages

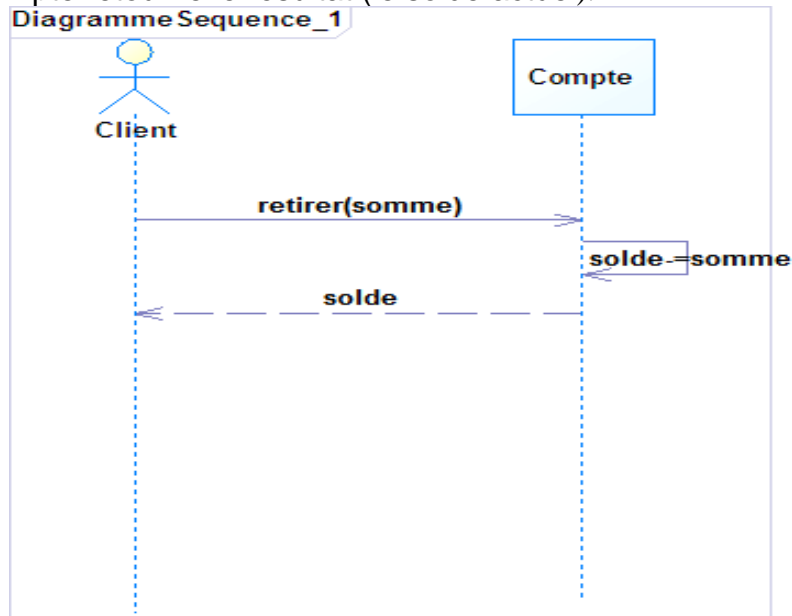


- Un objet est représenté par un rectangle et une ligne verticale (ligne de vie de l'objet)
- Les objets communiquent en échangeant des messages représentés par des flèches orientées de l'émetteur au récepteur
- L'ordonnancement vertical des messages indique la chronologie
- Un message reçu par un objet déclenche l'exécution d'une opération
- Un message envoyé par objet correspond :
 - Demander un service d'un autre objet
 - Renvoyer le résultat d'une opération



Exemple

- ✱ Le client demande un service (déposer de l'argent) à l'objet Compte.
- ✱ Le compte reçoit le message et déclenche l'opération de même nom.
- ✱ Le compte retourne le résultat (le solde actuel).



Plusieurs concepts additionnels :

- ☐ Période d'activité
- ☐ Types de messages
- ☐ Création et destruction d'objets
- ☐ Structures de contrôles

Période d'activité

- ☐ Correspond au temps pendant lequel un objet fait une action
- ☐ Représentée par une bande rectangulaire superposée à la ligne de vie de l'objet.



Messages

- ☐ Traduisent les interactions (échange d'informations) entre objets
- ☐ Représentés par des flèches orientées de l'émetteur au récepteur
- ☐ Plusieurs types :
 - Message simple
 - Message minuté (Timeout)
 - Message synchrone
 - Message asynchrone
 - Message récursif

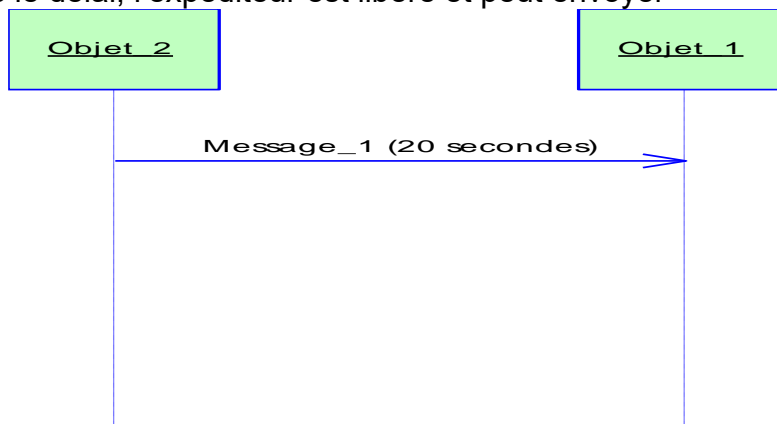
Message simple

- Message pour lequel on ne spécifie aucune information d'envoi ou de réception.



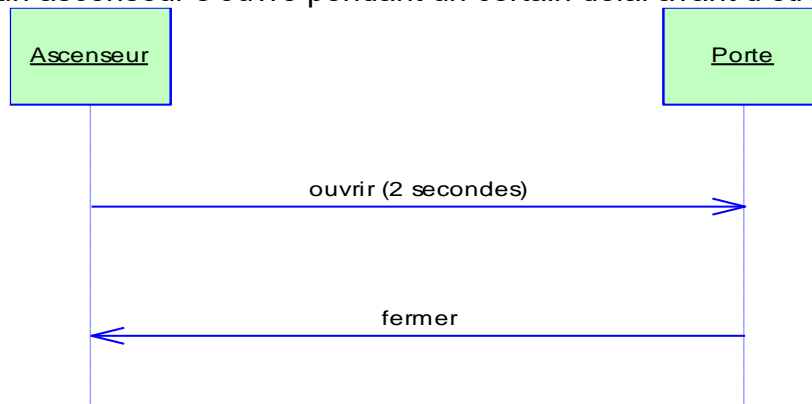
Message minuté (Timeout) :

- Bloque l'expéditeur pendant un temps donné, en attendant la prise en compte du message par le récepteur
- Après le délai, l'expéditeur est libéré et peut envoyer



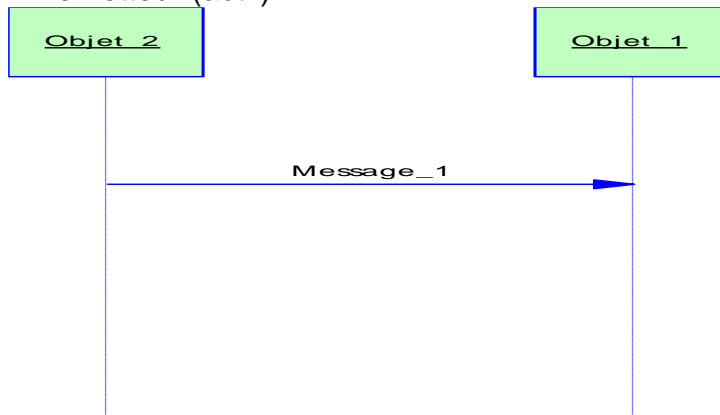
Exemple :

La porte d'un ascenseur s'ouvre pendant un certain délai avant d'être refermée.



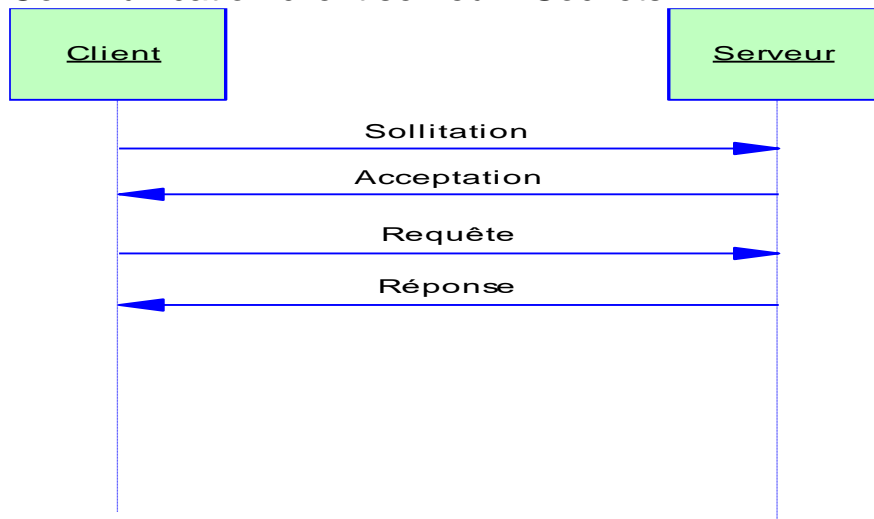
Message synchrone (appel de procédure)

- Bloque l'expéditeur jusqu'à la prise en compte du message par le récepteur.
- Le contrôle est passé de l'émetteur au récepteur qui devient à son tour émetteur (actif).



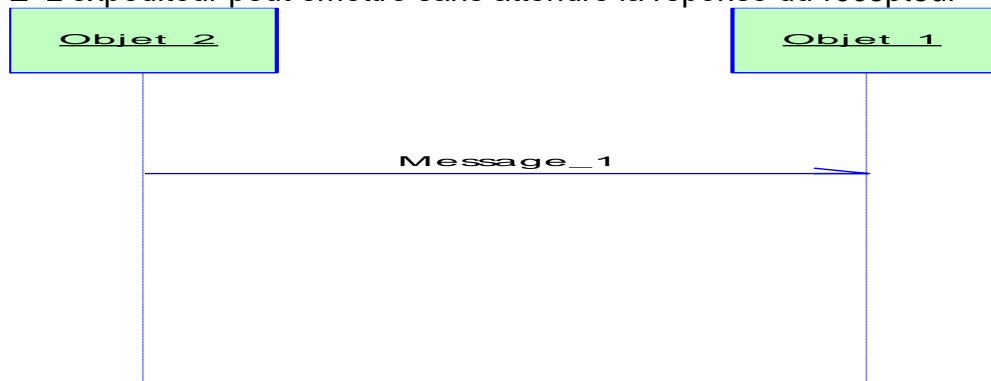
Message synchrone (appel de procédure) : Exemple

Communication client serveur : Sockets



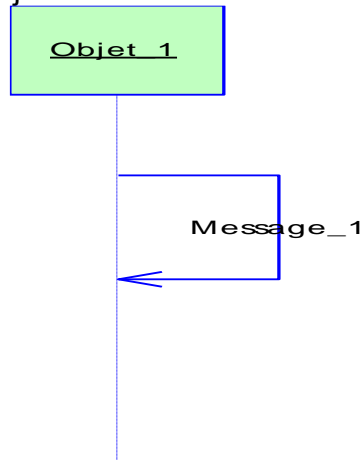
Message asynchrone

- N'interrompt pas l'exécution de l'expéditeur
- L'expéditeur peut émettre sans attendre la réponse du récepteur



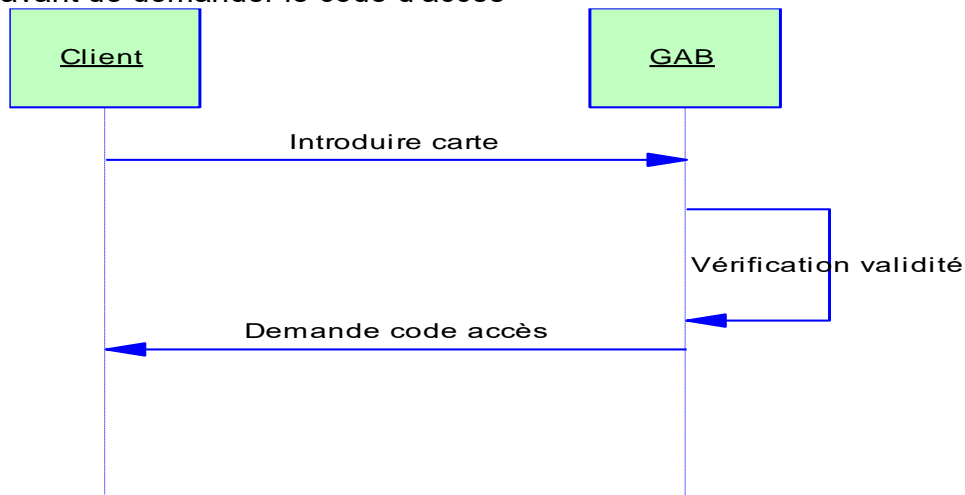
Message récursif

- Appelé aussi message réflexive
- Message envoyé d'un objet vers lui-même.



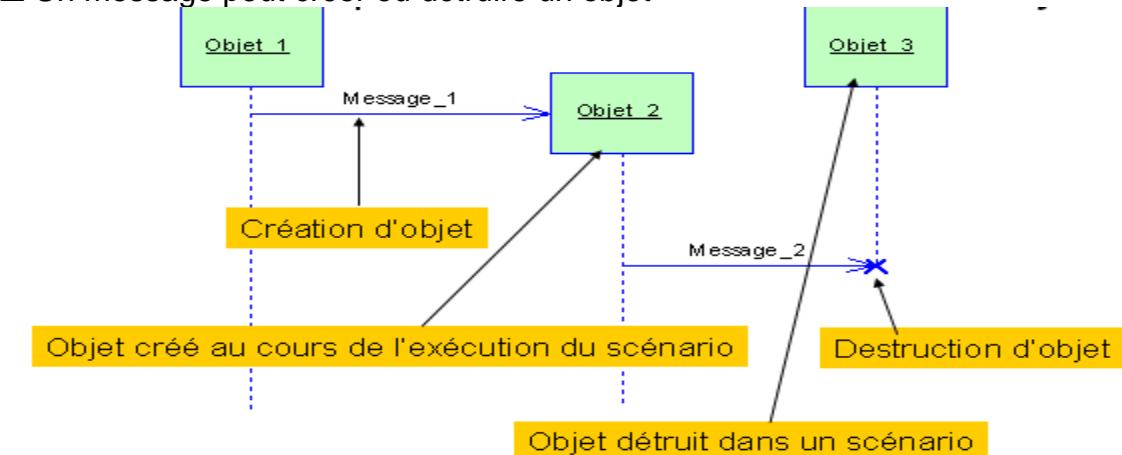
Exemple

Lorsque le client introduit sa carte de guichet, ce dernier vérifie la validité de la carte avant de demander le code d'accès



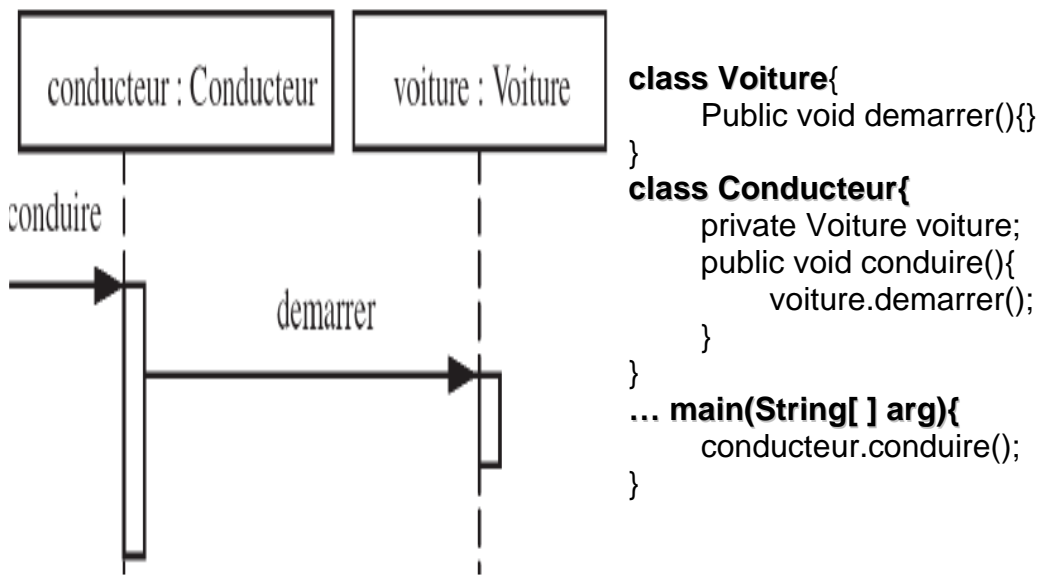
Création et destruction d'objets

- Un message peut créer ou détruire un objet



Traduction des messages

- Envoyer un message c'est demander un service d'un autre objet (sauf le cas d'un message de retour).
- Les messages sont traduits par des opérations dans la classe de l'objet ayant reçu le message



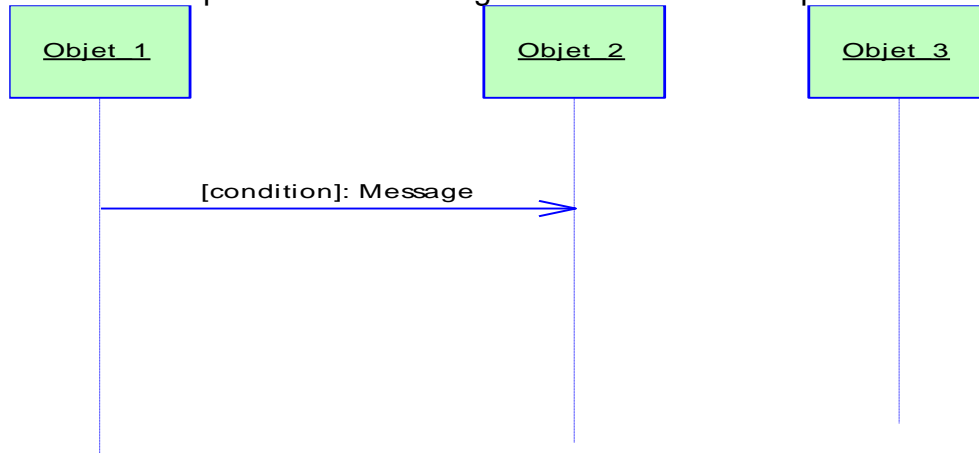
Structures de contrôle

Le diagramme de séquences peut inclure un certain nombre de structures

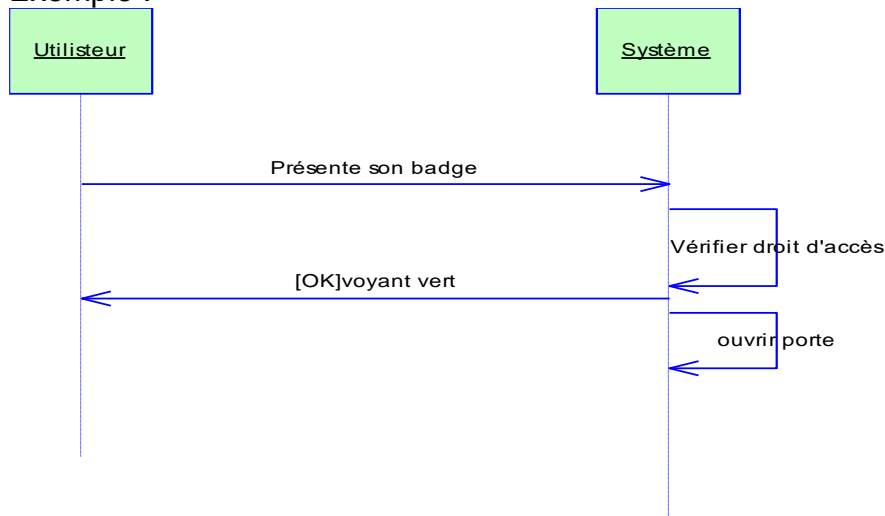
- Branchements (tests)
- Répétitions (itérations, boucles)

Les tests (branchements)

- La condition précède le message et elle est délimitée par des crochets

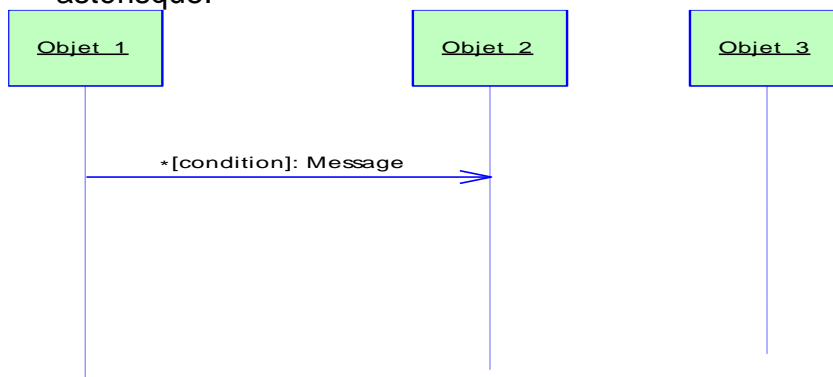


Exemple :



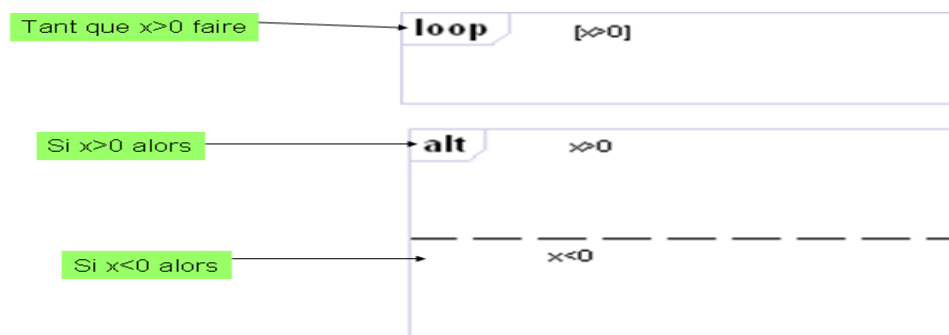
Les boucles (répétitions)

- La boucle se note comme le test, mais la condition est précédée d'un astérisque.

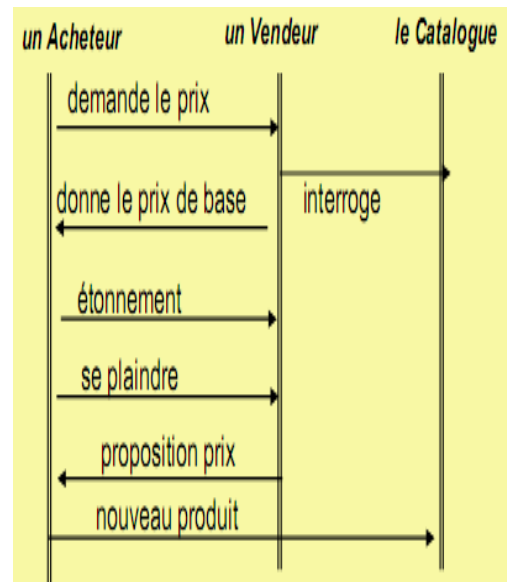
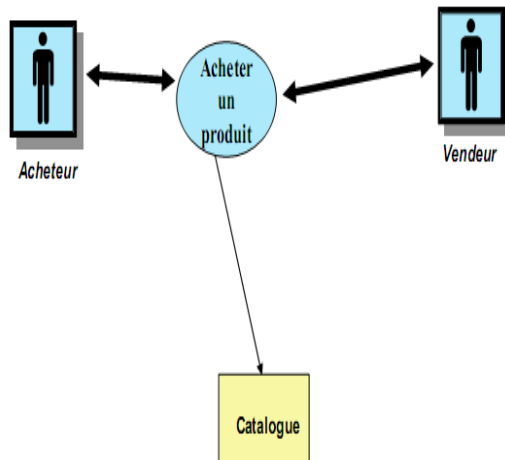


Fragments :

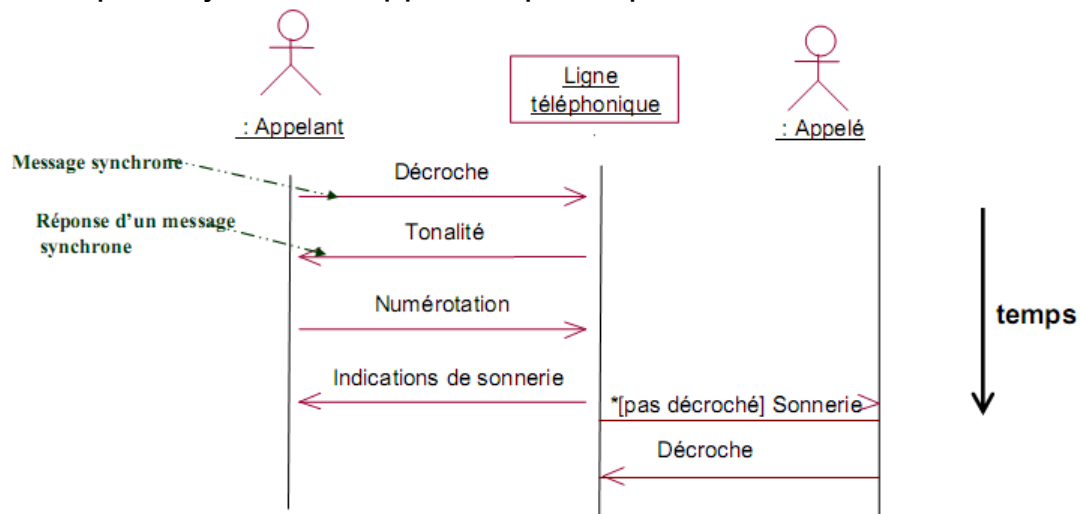
- Permet de décomposer une interaction complexe en fragments simples
- Représenté par un rectangle dont le coin supérieur gauche contient un pentagone
- Dans le pentagone figure le type du fragment
 - loop : boucle
 - alt : alternative
 - ref : référence, ...



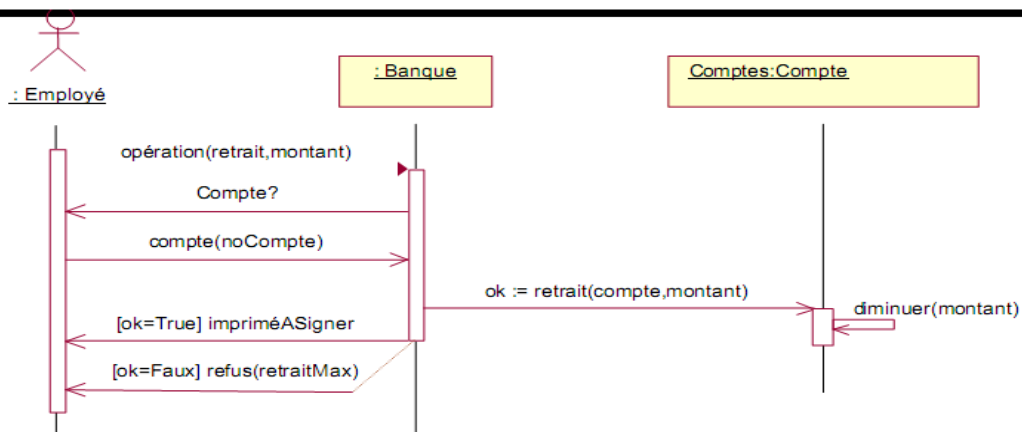
Exemples: Vente



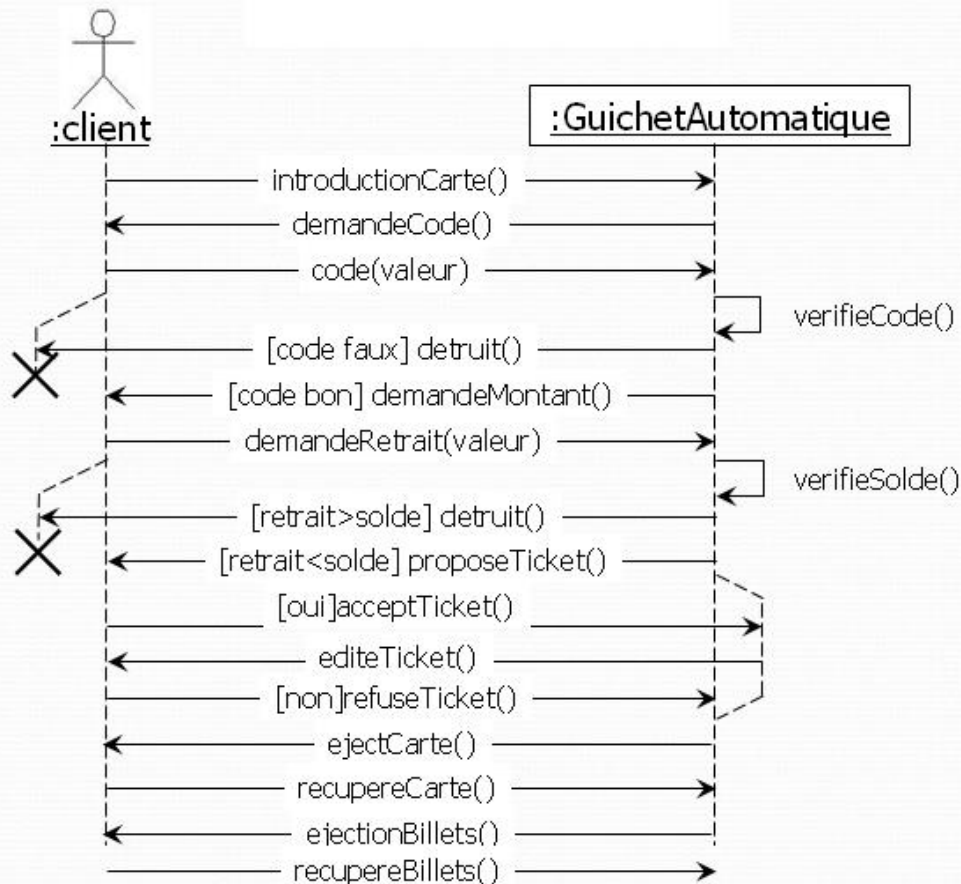
Exemple : Système d'appel téléphonique



Diagrammes de séquence exemple : système de gestion de comptes bancaires



Diagrammes de séquence exemple : Guichet Automatique

**Diagrammes de classes et d'objets**

- Permet de donner une vue statique du système en terme de :
 - Classes d'objets
 - Relations entre classes
 - Associations
 - agrégation/composition
 - héritage
- La description du diagramme de classes est centrée sur trois concepts :
 - Le concept d'objets
 - Le concept de classes d'objets comprenant des attributs et des opérations
 - Les différents types de relations entre classes.

Concept d'objet

Objet = un concept, abstraction ou une chose autonome qui a un sens dans le contexte du système à modéliser

- une **personne** : le client
- un **objet concret** : le livre intitulé « Initiation à... »
- un **objet abstrait** : le compte bancaire n° 151423
- ...

❑ Exemples

- Gestion de stock : Clients, Commandes, Articles, ...
- Gestion scolaire : Étudiants, Modules, Filières, ...

Concept d'attribut

- ❑ Un attribut est une propriété, caractéristique d'un objet. Par exemple :
 - un client a un nom, un prénom, une adresse, un code client, ...
 - un compte bancaire a un numéro, un solde, ...

- ❑ Un attribut doit (généralement) avoir une **valeur atomique**

La description d'un attribut comporte :

Visibilité attribut:type[= valeur initiale]

Où :

- ❑ Visibilité :
 - + (publique, public) : visible par tous
 - - (privée, private) : visible seulement dans la classe
 - # (protégée, protected) : visible seulement dans la classe et dans les sous-classes de la classe.
- ❑ Nom d'attribut
- ❑ Type de l'attribut
- ❑ Valeur initiale (facultative)

Le type d'un attribut peut être :

- Un type de base : entier, réel, ...
- Une expression complexe : tableaux, enregistrements, ...
- Une classe

Exemples d'attributs :

- - *couleur* : *enum{Rouge, Vert, Bleu}*
- # *b* : *boolean = vrai*
- - *Client* : *Personne*

Lorsqu'un attribut peut être dérivé ou calculé à partir d'autres attributs, il est précédé d'un */*. Par exemple, une classe « Rectangle » peut contenir les attributs suivants :

- ❑ longueur : réel,
- ❑ largeur : réel,
- ❑ /surface : réel.

Rectangles			
-	Largeur	: float	= 10
-	Longueur	: float	
-	/Surface	: float	

On distingue deux types d'attributs :

- ❑ Attribut d'instance :
 - Chaque instance de la classe possède une valeur particulière pour cet attribut
 - Notation : **Visibilité attribut:type[= valeur initiale]**
- ❑ Attribut de classe
 - Toutes les instances de la classe possède la même valeur pour cet attribut

- Notation : **Visibilité attribut:type[= valeur initiale]**
- Équivalent en C++, Java : static

NOM CLASSE
Attribut_1 : int
Attribut_2 : int
Attribut_3 : int
Operation_1 () : void
Operation_2 () : void

NOM CLASSE
+ Attribut public : int
Attribut protégé : int
- Attribut privé : int

Concept opération

Une opération est :

- ☐ un service offert par la classe
- ☐ une fonction ou une transformation qui peut être appliquée aux objets d'une classe.
- ☐ permet de décrire le comportement d'un objet. Par exemple, « Embaucher », « Licencier » et « Payer » sont des opérations de la classe « Société ».
- ☐ Une opération est une transformation qui peut être appliquée à ou par des objets d'une classe.

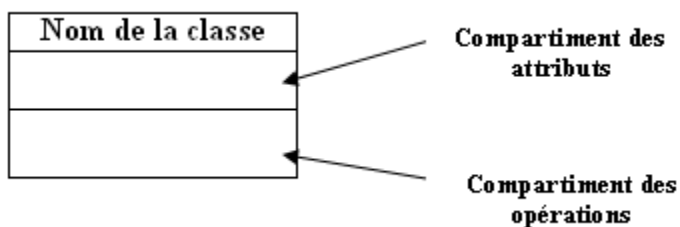
Concept de classes d'objets

- ☐ Une classe est la description abstraite d'un ensemble d'objet. Il peut être vu comme la factorisation des éléments communs à un ensemble d'objet.

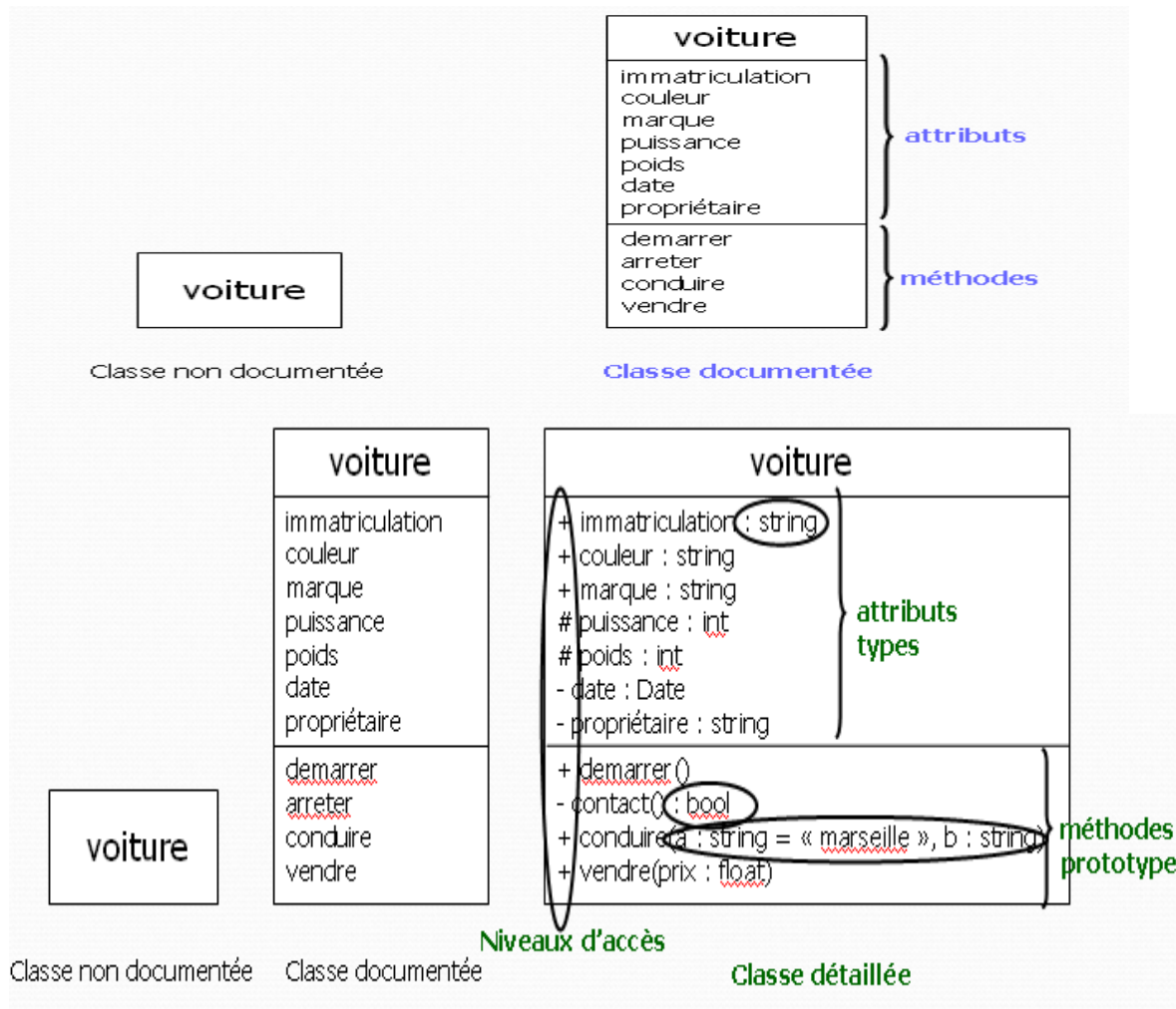
Classe représentée par un rectangle à trois parties :

- Partie 1 : Nom de la classe
- Partie 2 : Attributs (propriétés, champs)
- Partie 3 : Méthodes (fonctions, opérations)

- ☐ syntaxe

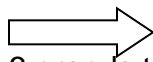


Exemple de classe



Relations entre les classes

- Possibilité de communication entre objets :
 - interactions entre objets.



Relation entre classes

- 3 grands types:

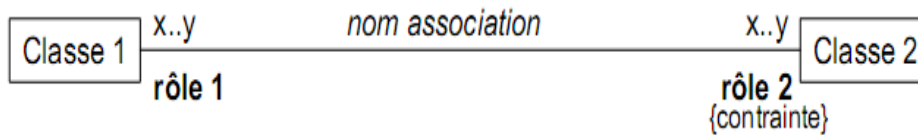
- **Associations** ("est produit par", "est affilié à", "se trouve à", "est conduit par"...): dépendance entre classes, communication entre Objets, Ex: « un lecteur lit un livre » (forme verbale)
- **Agrégation/composition** ("fait partie de"): objets composites / Composants, Ex: « un train est composé de wagons »
- **Généralisation/spécialisation** ("est une sorte de"): héritage entre objets, Exemple: « un chat est une sorte d'animal ».

Associations

- Relation existant entre une, deux ou plusieurs classes.
- Une association porte un nom (signification) Représentée par une ligne rectiligne

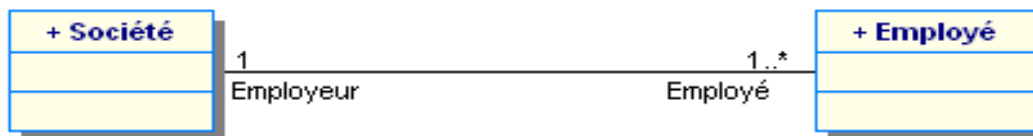


- ☐ Nom : forme verbale, sens de lecture avec flèche
- ☐ Rôles : forme nominale, identification extrémité association
- ☐ Multiplicité : 1, 0..1, M..N, *, 0..*, 1..*



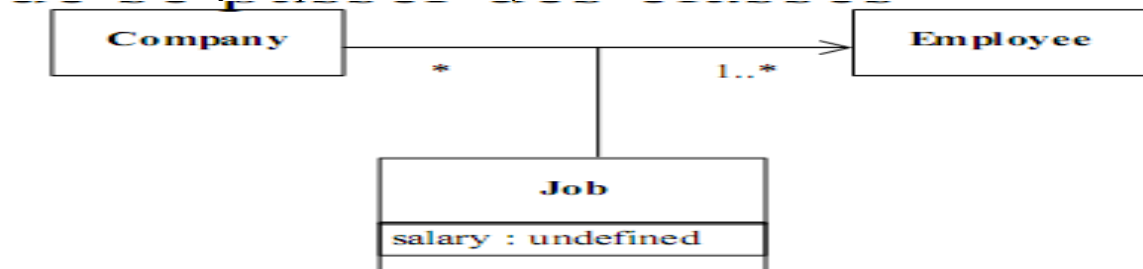
Rôle d'une association

- ☐ Décrit le rôle d'une classe dans une association

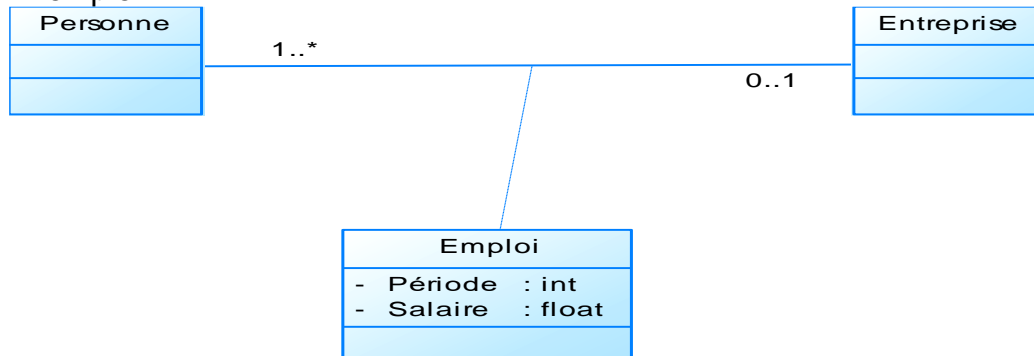


Classes-Associations

- ☐ Une classe-association est une association qui est aussi une classe.
- ☐ Les classes-associations sont utilisées lorsque les associations doivent porter des informations
- ☐ Il est toujours possible de se passer des classes-associations.



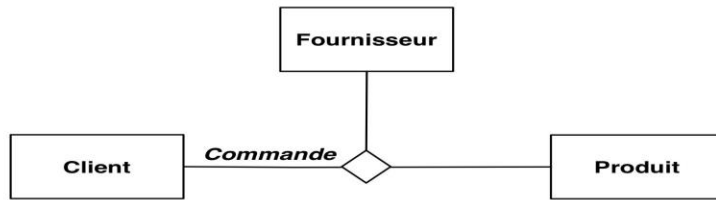
Exemple



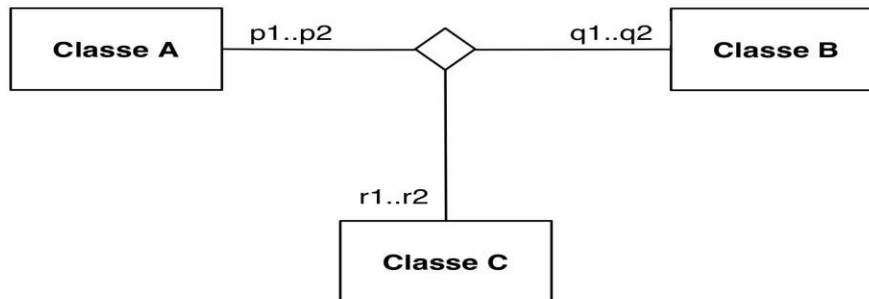
Degré d'une association :

- **degré d'une association** = nombre de classes participantes

- Association **unaire** : relie 2 instances d'une classe
- association **binaire** : relie 2 classes
- association **ternaire** : relie 3 classes
- association **n-aire** : relie n classes



Exemple ternaire



- ☐ Pour un couple d'instances de la classe A et de la classe B,
- ☐ il y a au min. **r1** instances de la classe C et au max. **r2** instances,
- ☐ ...

Multiplicité

Multiplicité	Description
1	Indique un et un seul
0..1	Indique 0 ou 1
M..N	De M à N (M<N)
*	Plusieurs
0..*	De 0 à plusieurs (0, n)
1..*	De 1 à plusieurs (1, n)

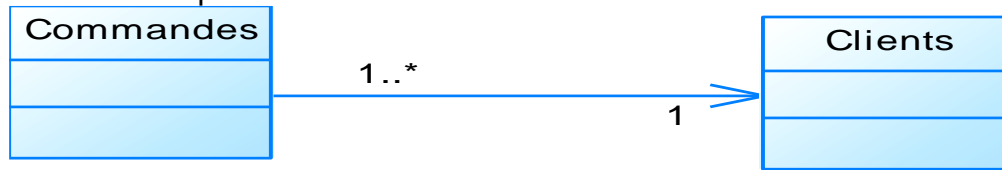
Association Navigabilité

- ☐ Une association est par défaut bidirectionnelle.
- ☐ Cependant, il peut être utile de se limiter à une seule direction → association navigable.

Exemple

- ☐ Spécification : on doit être en mesure de savoir le client qui a fait la commande et non toutes les commandes d'un client

□ Conception :



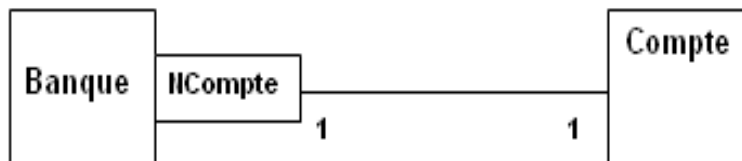
□ Implémentation : la classe commande doit avoir un champ faisant référence à la classe client.

Associations qualifiées

Exemple : une banque contient plusieurs comptes, d'où le diagramme :



Par contre, si on connaît le N°Compte, il y a un et un seul compte, on obtient alors :



Qualification d'une association

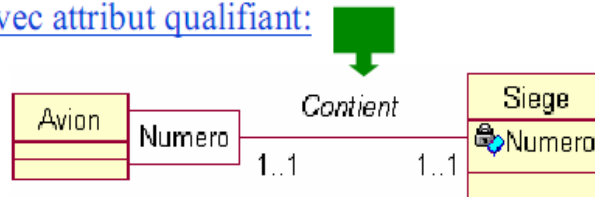
Exercice

□ Un avion est composé de plusieurs sièges, mais dans une rangée il y a seulement quatre sièges.

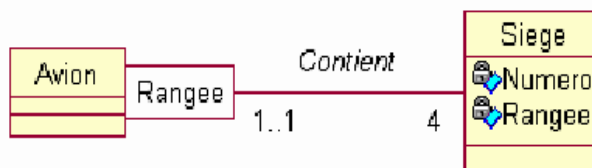


ce schéma ne permet pas de dire que chaque siège a un numéro qui est unique pour chaque avion.

Avec attribut qualifiant:



« un avion contient *un* siège *pour un numéro donné* ».



dans un avion, pour une rangée donnée, il y a 4 sièges.

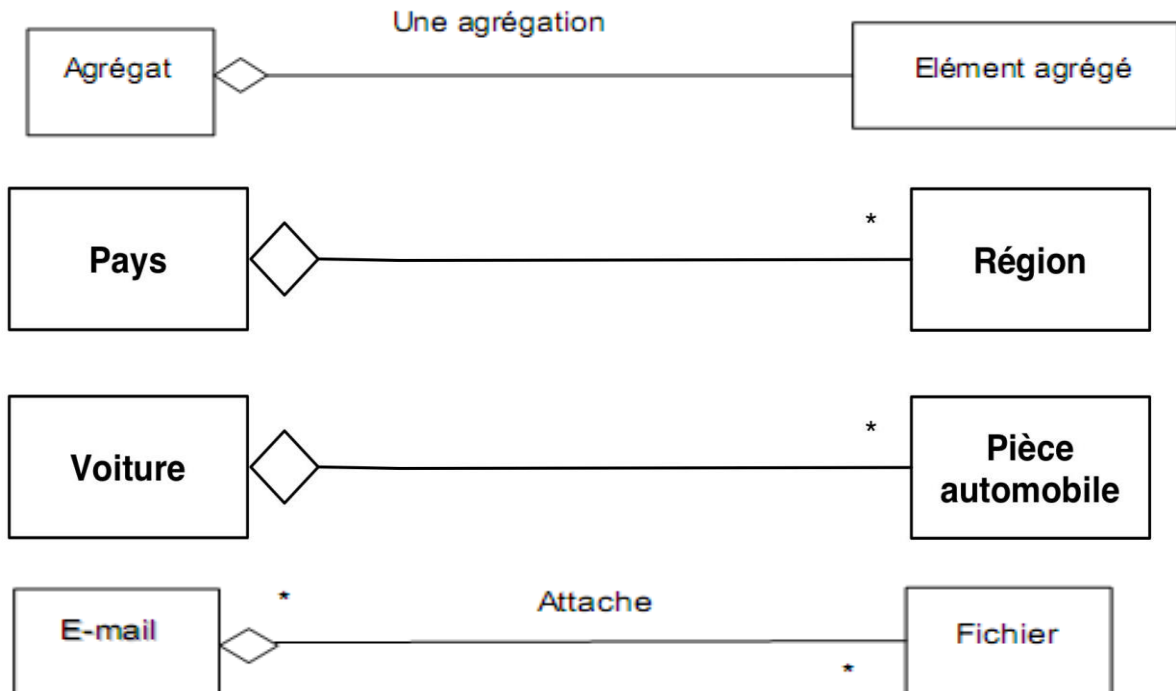
Agrégation

- L'agrégation est une association **non symétrique**, qui exprime un couplage fort et une relation de subordination
- Représente une relation de type "**ensemble / élément**"
- A un même moment, une instance d'élément agrégé peut être liée à plusieurs instances d'autres classes : **l'élément agrégé peut être partagé**
- Une instance « ensemble » peut exister sans élément (et inversement) :

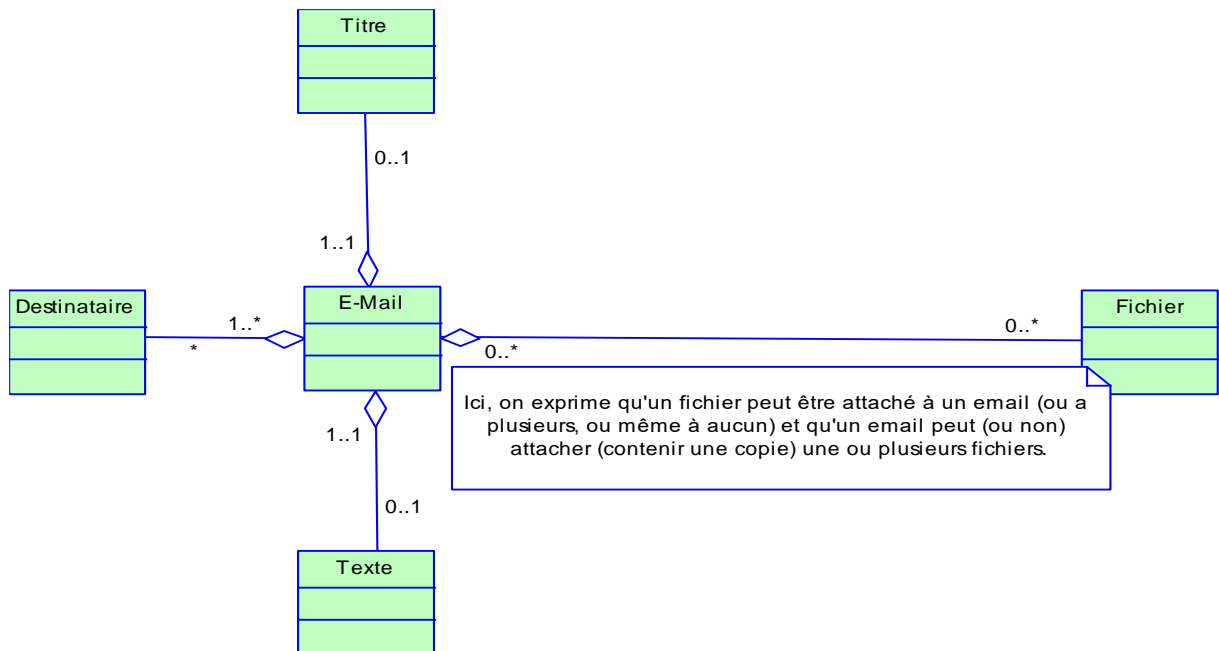
Cycles de vies indépendants

- Une agrégation peut notamment (mais pas nécessairement) exprimer :
 - qu'une classe (un "élément") fait partie d'une autre ("l'ensemble"),
 - 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

Exemples



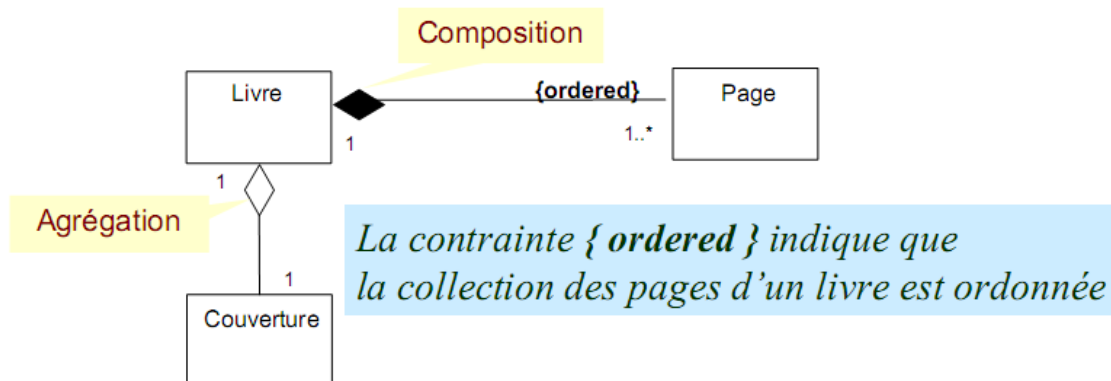
Exemple agrégation



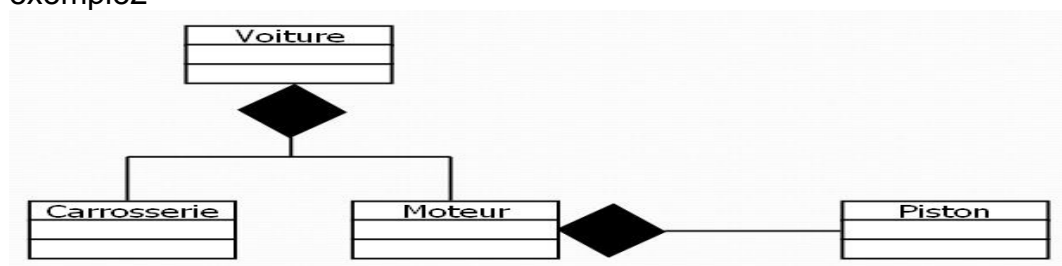
Composition

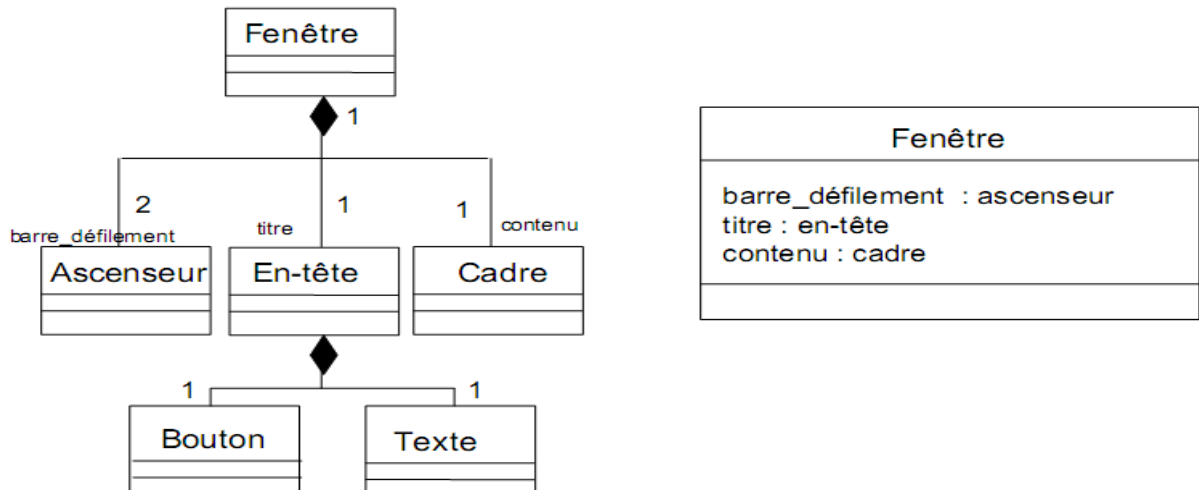
- ☐ Cas particulier d'agrégation \Rightarrow contenance physique
- ☐ « Si destruction objet composé \Rightarrow destruction des composants »
- ☐ Agrégation forte
- ☐ Destruction du composite (x est une partie de y?) \Rightarrow Destruction automatique de tous ses composants

exemple1



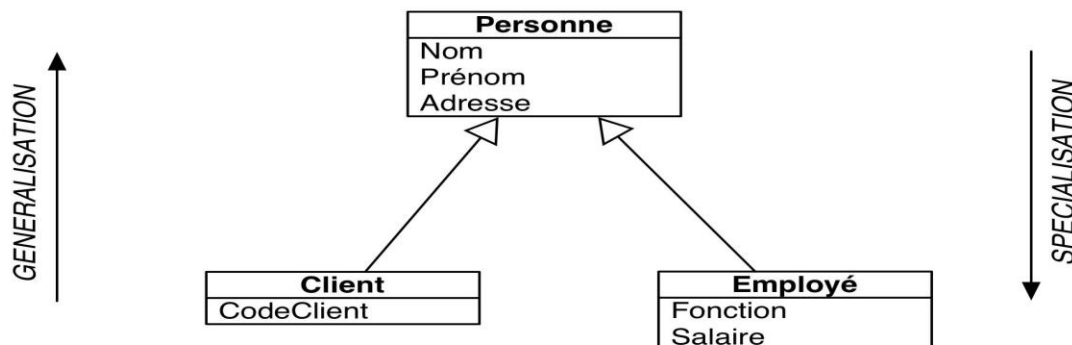
exemple2





Généralisation / Spécialisation et héritage

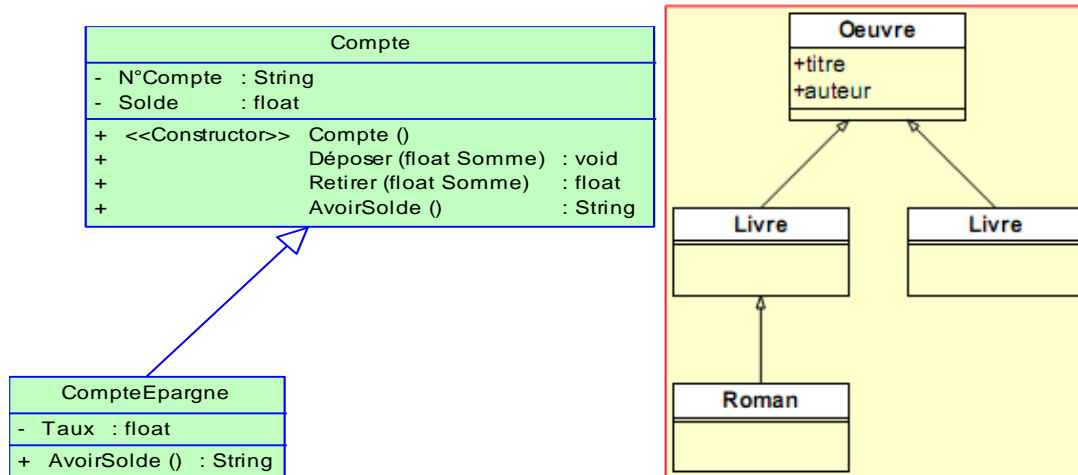
- ☐ La généralisation est la relation entre une classe et une ou plusieurs de ses versions raffinées.
- ☐ On appelle la classe dont on tire les précisions la super-classe et les autres classes les sous-classes.
- ☐ C'est une relation de type « est un (is a) » ou « est une sorte de ».
- ☐ La notation utilisée pour la généralisation est le triangle



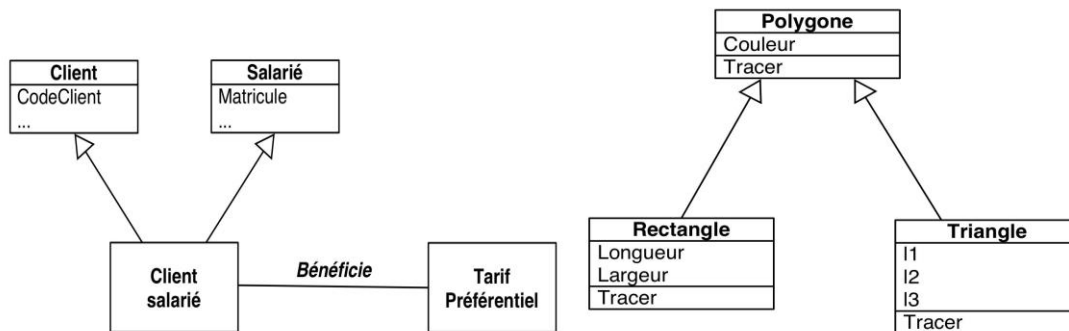
- ☐ **Généralisation :**
 - regroupement, « est-un », « sorte-de »
- ☐ **Spécialisation**
 - (symétrique généralisation):
 - raffinement de classe
 - par extension de propriétés (ajouter un attribut)
 - par restriction de domaines de valeurs d'attributs.

La classe spécialisée (sous-classe)

- ☐ hérite les méthodes et les attributs de la classe générale (super-classe).
- ☐ Principe de substitution : toutes les propriétés de la classe 'parent' doivent être valables pour les classes 'enfant'.
- ☐ peut ajouter ses propres attributs et méthodes.
- ☐ peut redéfinir le comportement d'une méthode.



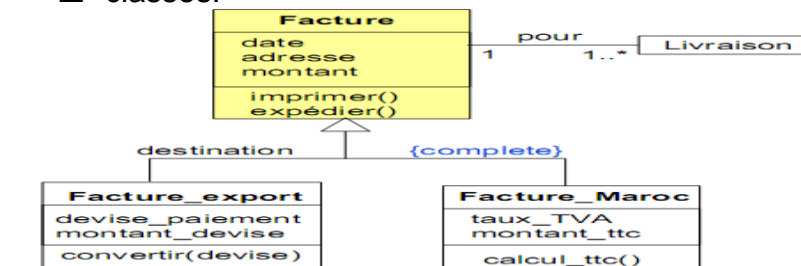
- une classe peut hériter de plusieurs super-classes → **héritage multiple**
- **polymorphisme** = opérations de même nom, comportement spécifique.



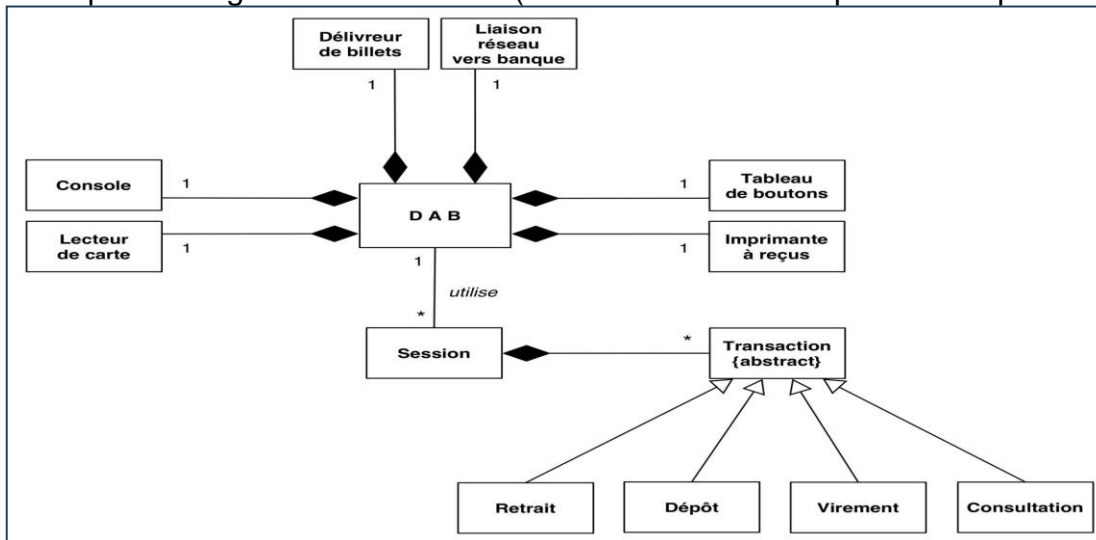
Hierarchies de classes (Contraintes sur les associations) :

Contraintes :

- {complète},
- {incomplète},
- {disjoint}
- Permettent d'imposer des règles à respecter lors du passage à l'implémentation
- Il est possible d'attribuer toutes sortes de contraintes à une association
- Les contraintes sont représentées entre accolades et peuvent être exprimées dans n'importe quel langage.
- Attributs / opérations: communs sont montrées au niveau le plus haut.
- La contrainte {complète} indique que la généralisation est terminée et qu'il n'est plus possible de rajouter des sous-classes.

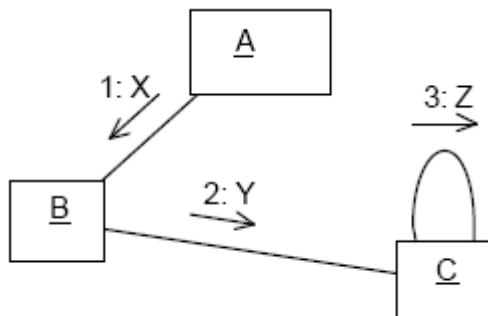


Exemple de diagramme de classes (Distributeur Automatique de Banque : DAB)



Diagrammes de collaboration

- Représente les interactions entre objets et relations structurelles permettant celles-ci.
 - Description:
 - Du comportement collectif d'un ensemble d'objets
 - Des connexions entre ces objets
 - Des messages échangés par les objets
 - Interaction réalisée par un groupe d'objets qui collaborent en échangeant des messages
 - Temps non représenté de manière implicite (numérotation des messages).

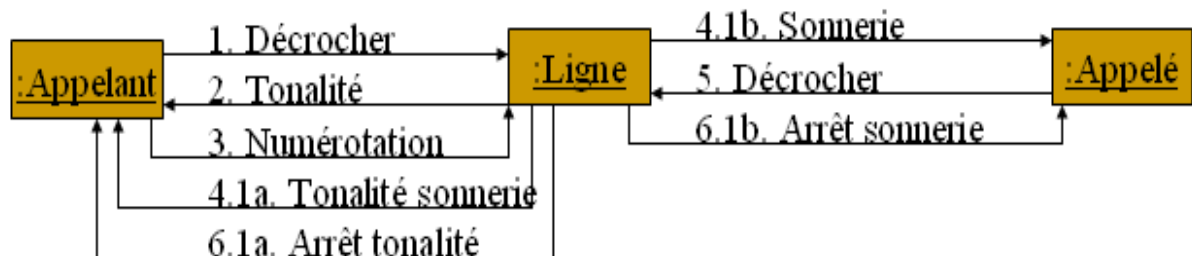


Exemple:

- A envoie un message X à B.
- B envoie un message Y à C.
- C s'envoie un message à B.

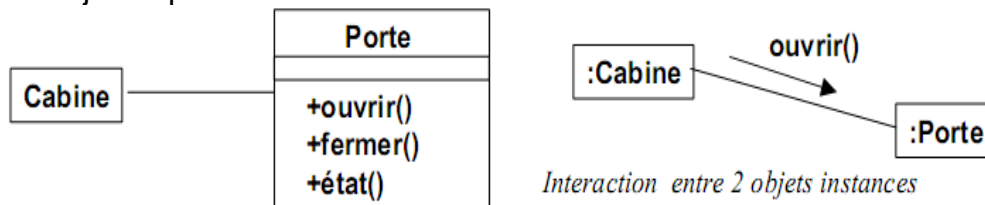
Temps

Exemple : Appel téléphonique



Les interactions

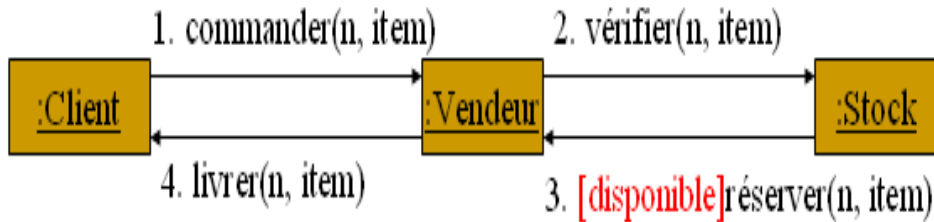
- Eléments d'une interaction:
 - instances.
 - liens (support messages).
 - messages déclenchant les opérations.
 - rôles joués par les extrémités de liens.



Interaction entre 2 objets instances
des classes Cabine et Porte

Modélisation d'une cabine d'ascenseur qui 'demande' à une porte de s'ouvrir.

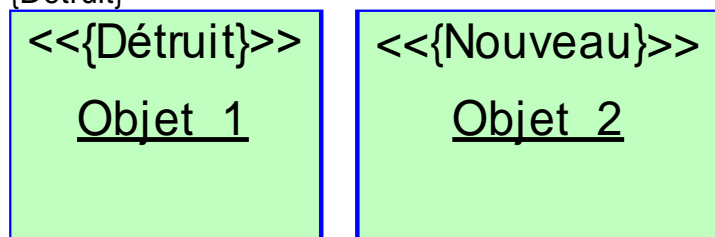
- Mêmes types contraintes que pour les diagrammes de séquence
 - Itération : *[condition]
 - Conditions : [condition]
- Exemple : réservation d'articles



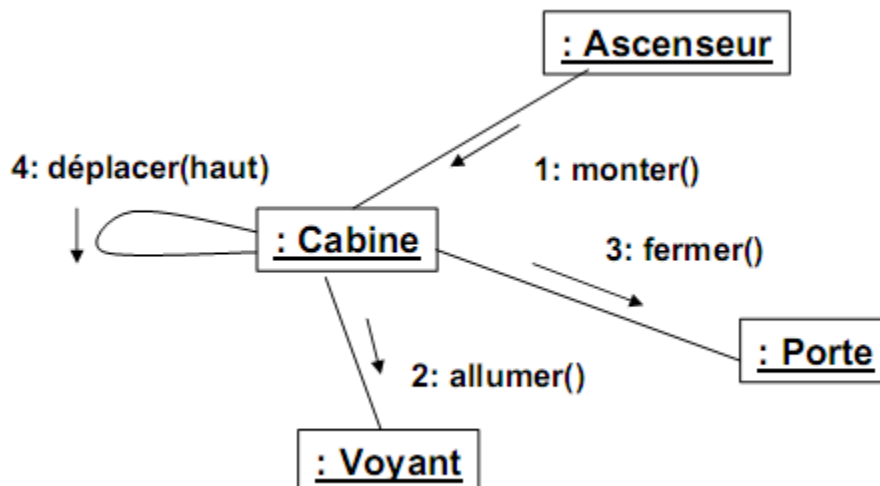
Les **objets créés** ou **détruits** au cours d'une interaction peuvent **respectivement** porter les contraintes :

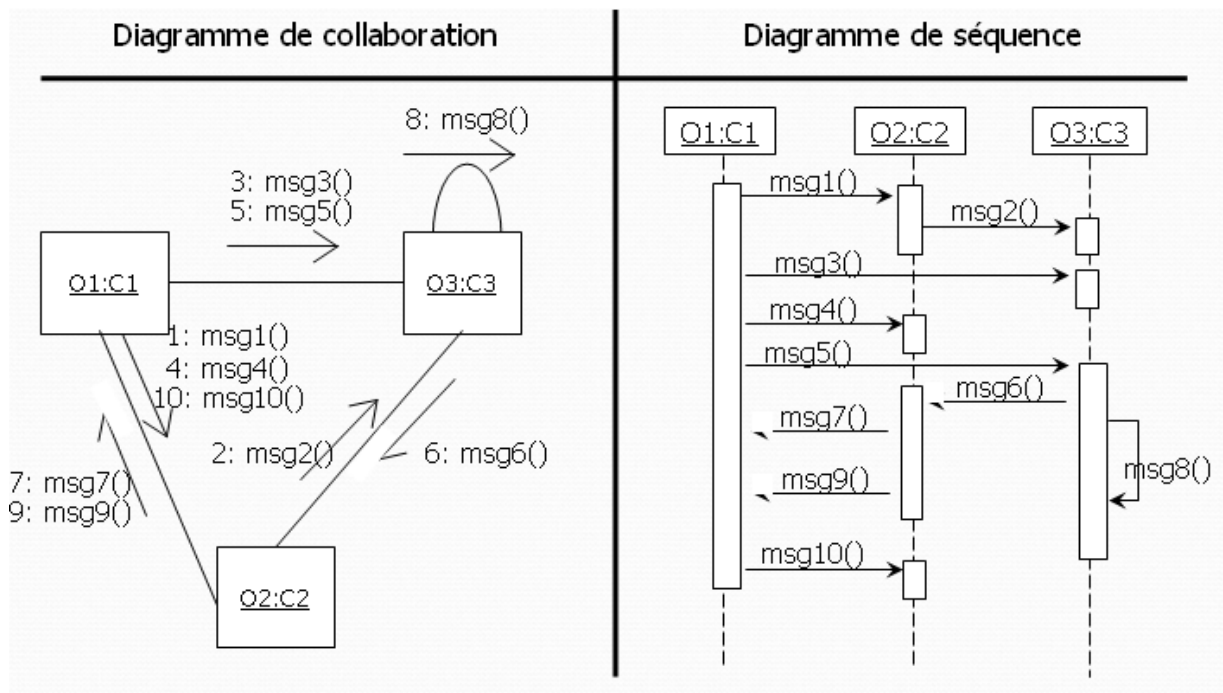
{Nouveau}

{Détruit}



Représentation de l'ordre des envois de messages pour l'exemple de l'ascenseur :





Équivalence diagrammes de séquence / collaboration

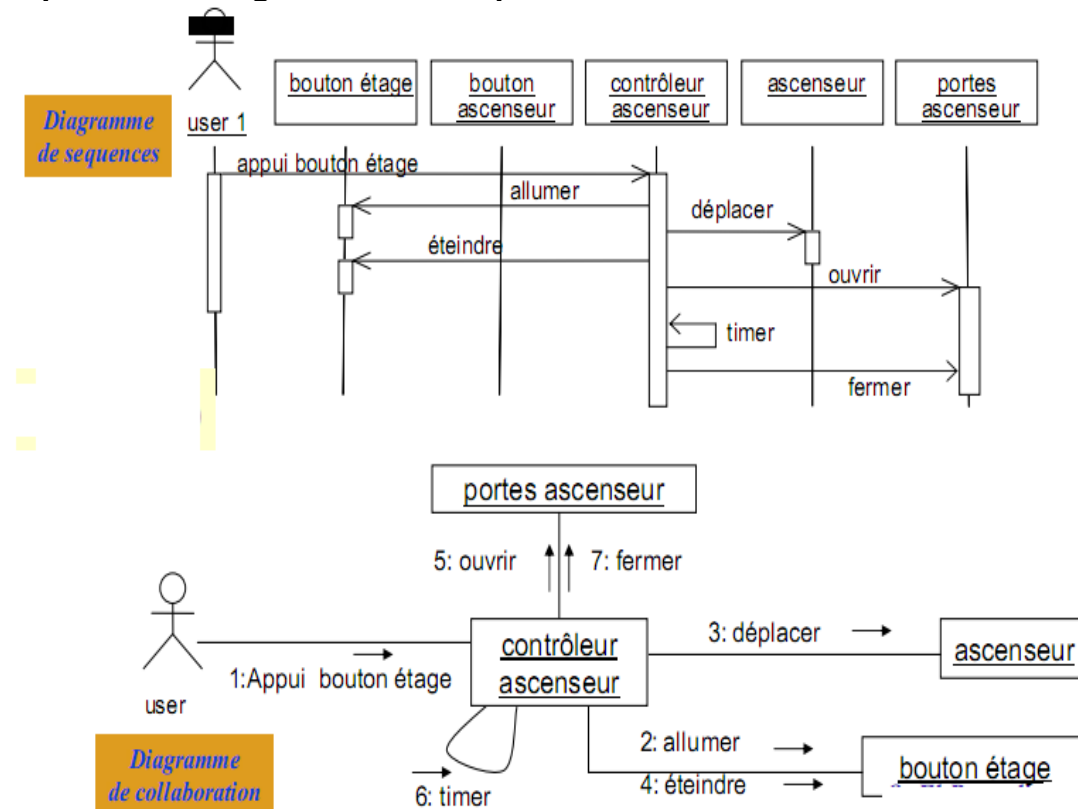


Diagramme état-transition

□ Attaché à une classe ou à un cas d'utilisation, il doit présenter une classe par rapport à ses états possibles et aux transitions qui le font évoluer. Lorsque [évènement] => changement d'état.

- L'action peut être conditionnelle : [condition] action.

Un état = image de la conjoncture instantanée des valeurs des attributs d'un objet & Présence ou non de ses liens à d'autres objets.

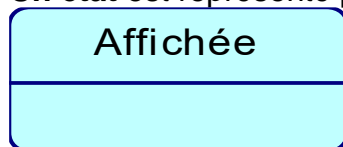
- Une transition représente un changement d'état ; elle est déclenchée par un événement.

- Transitions: peuvent être automatiques

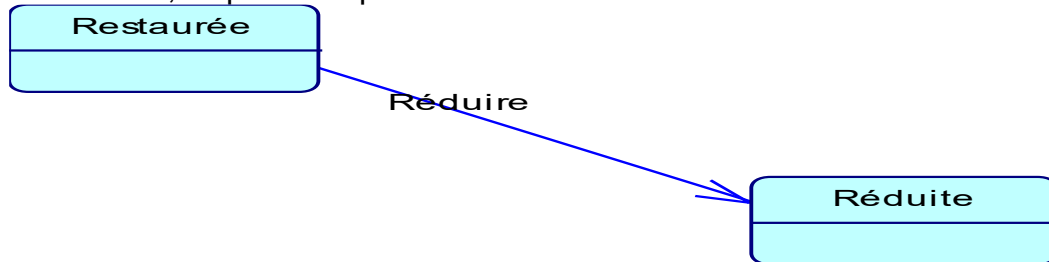
- peuvent être conditionnées

- Evènements: déclenchent les transitions

Un état est représenté par un rectangle aux coins arrondis

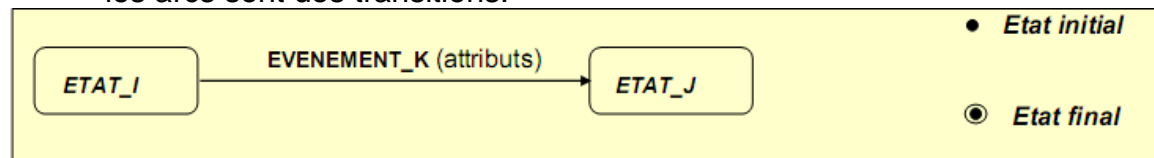


Transition: C'est le passage d'un état à un autre, Peut être nommé par un événement, Représenté par une flèche orientée de l'état source vers l'état cible.

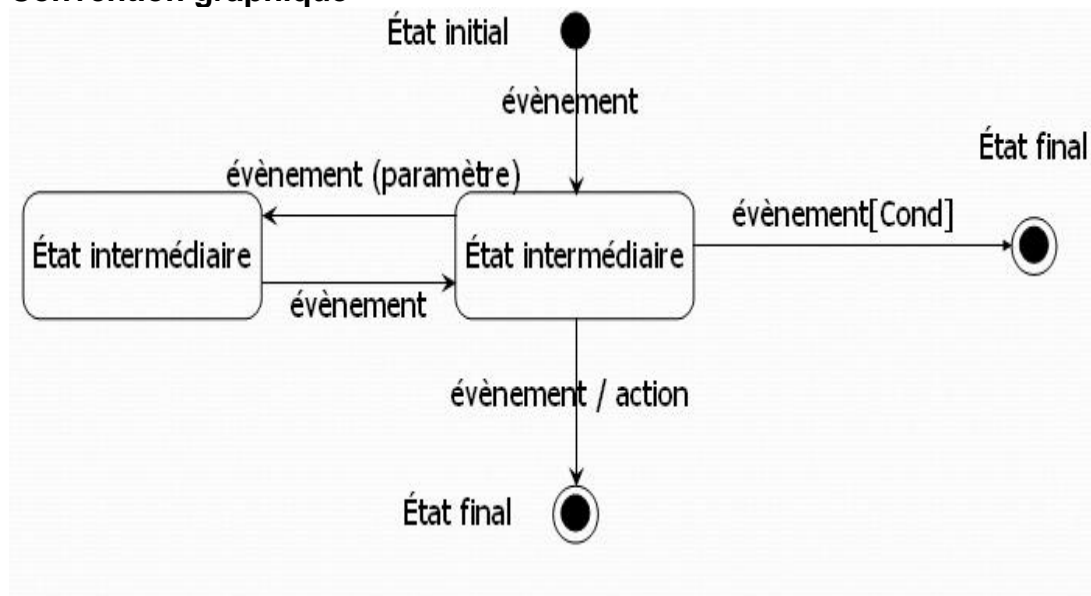


Construction du diagramme états – transition Statecharts

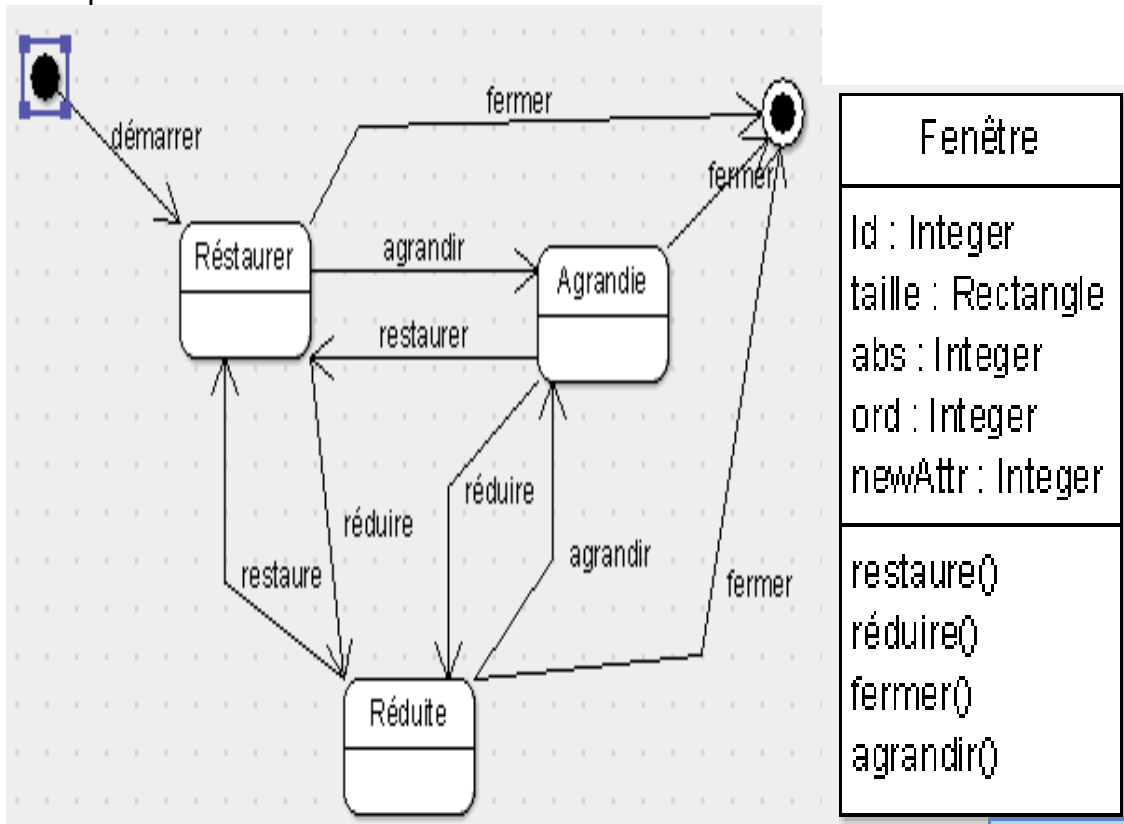
- Un Statechart est un graphe orienté dont les nœuds sont des états et les arcs sont des transitions.



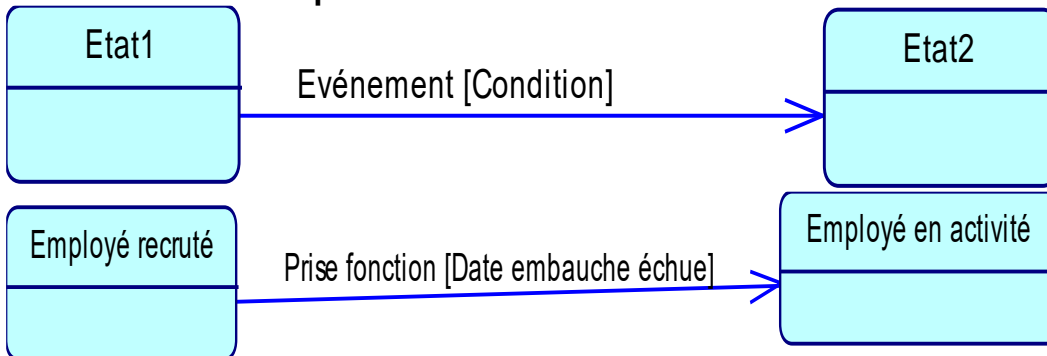
Convention graphique



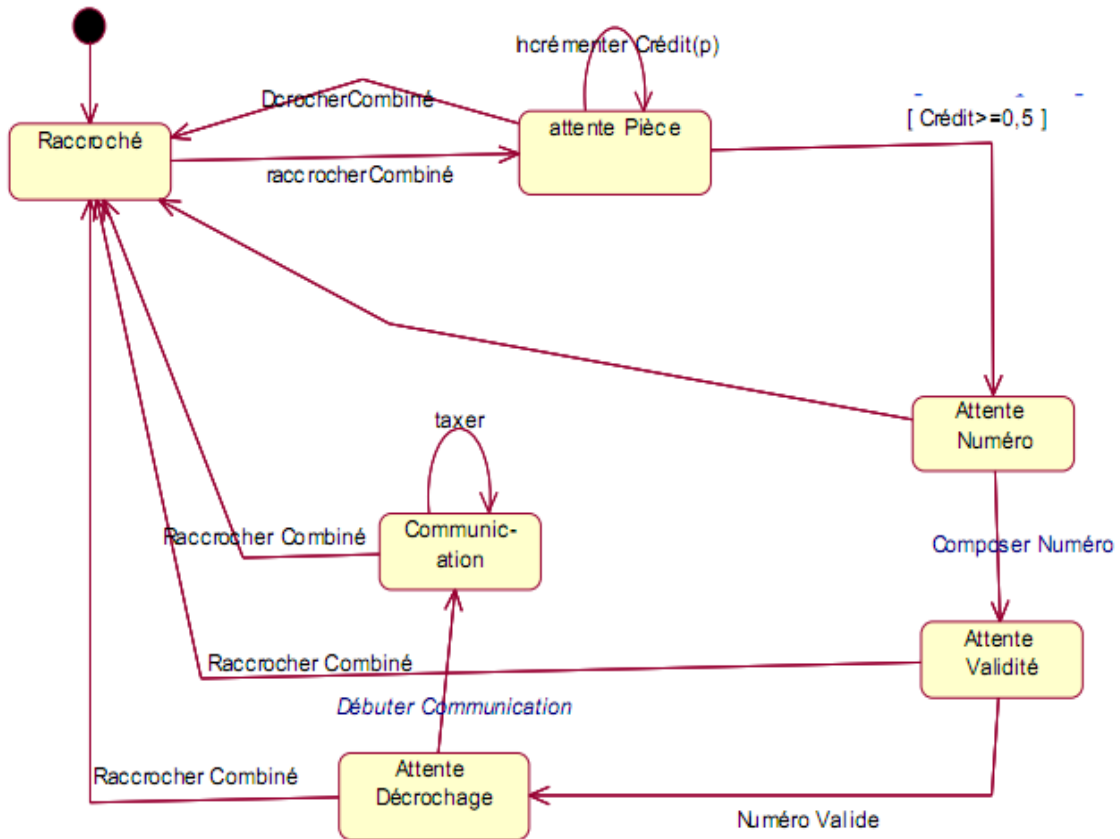
Exemple



Formalisme et exemple



Exemple



Exemple (Feu de signalisation)

Feu	
- ID	: int
- Couleur	: {Vert, Orange, Rouge}

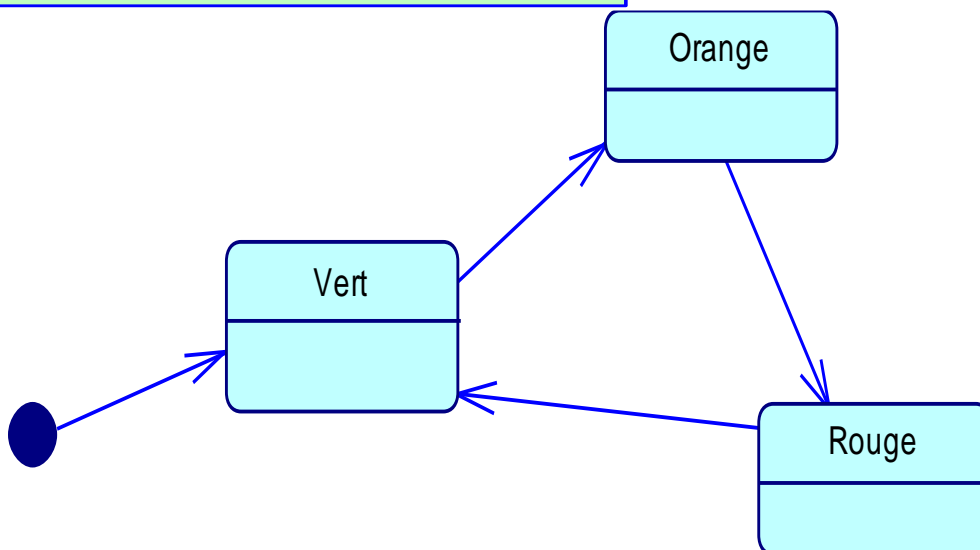


Diagramme d'activités

Introduction

- Le diagramme d'activité est attaché à une catégorie de classe et décrit le déroulement des activités de cette catégorie. Le déroulement s'appelle "flot de contrôle". Il indique la part prise par chaque objet dans l'exécution d'un travail. Il sera enrichi par les conditions de séquençement.
- Il pourra comporter des synchronisations pour représenter les déroulements parallèles.
- La notion de couloir d'activité va décrire les responsabilités en répartissant les activités entre les différents acteurs opérationnels.

Le déroulement séquentiel des activités

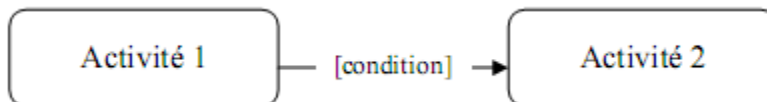
- Le diagramme d'états-transitions vu précédemment présente déjà un séquençement des activités d'une classe.



- Le diagramme d'activités va modifier cette représentation pour n'en conserver que le séquençement. La notion d'état disparaît. On obtient ce graphe :

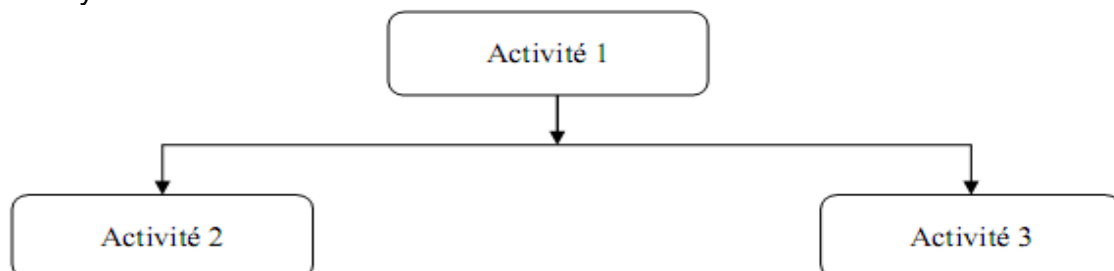


- Comme dans le diagramme d'états-transitions, la transaction peut être complétée par une condition de garde.



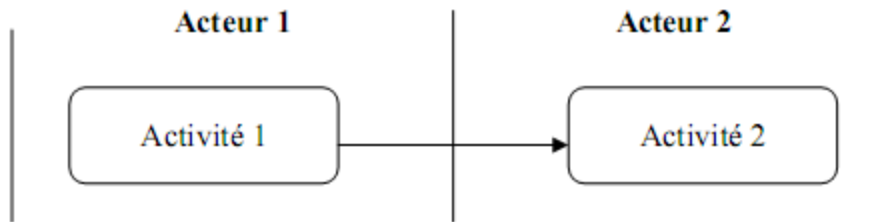
La synchronisation

- Les flots de contrôle parallèles sont séparés ou réunis par des barres de synchronisation. Les activités 2 et 3 seront simultanées.



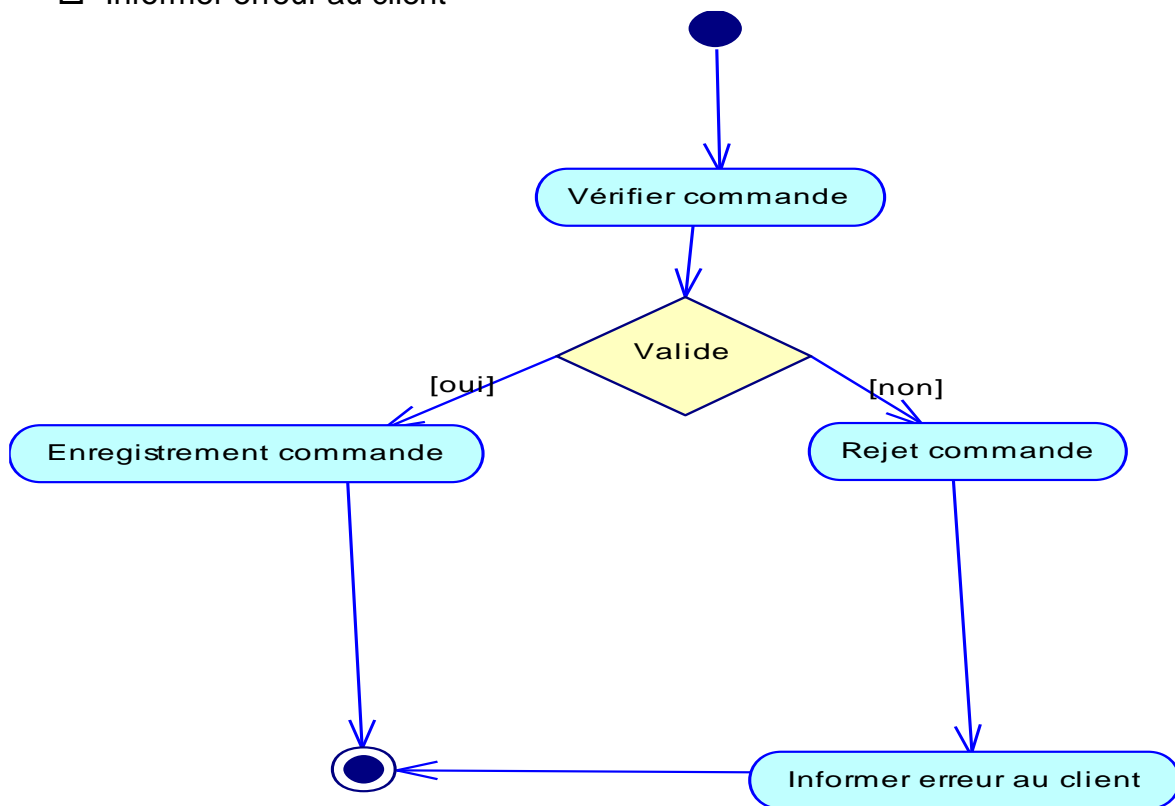
Les couloirs d'activités

- Le diagramme d'activités fait intervenir les acteurs de chaque activité. Chaque activité sera placée dans une colonne (couloir) qui correspond à l'acteur.

**Exercice 1**

Représenter les états suivants sous forme de diagramme d'activité :

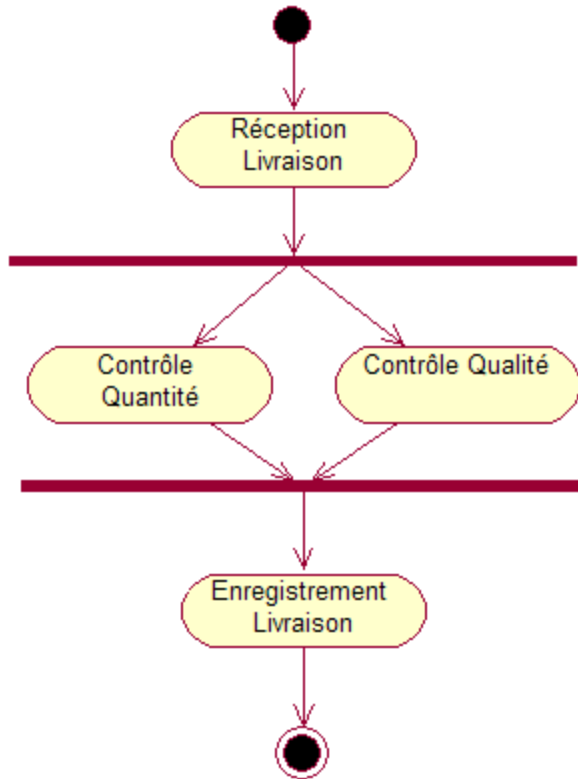
- ☐ Vérification commande
- ☐ Enregistrement commande
- ☐ Rejet commande
- ☐ Informer erreur au client

**Exercice 2**

Dans le domaine de gestion de stock, on considère les états suivants indiquant le flot de contrôle de réception d'une livraison :

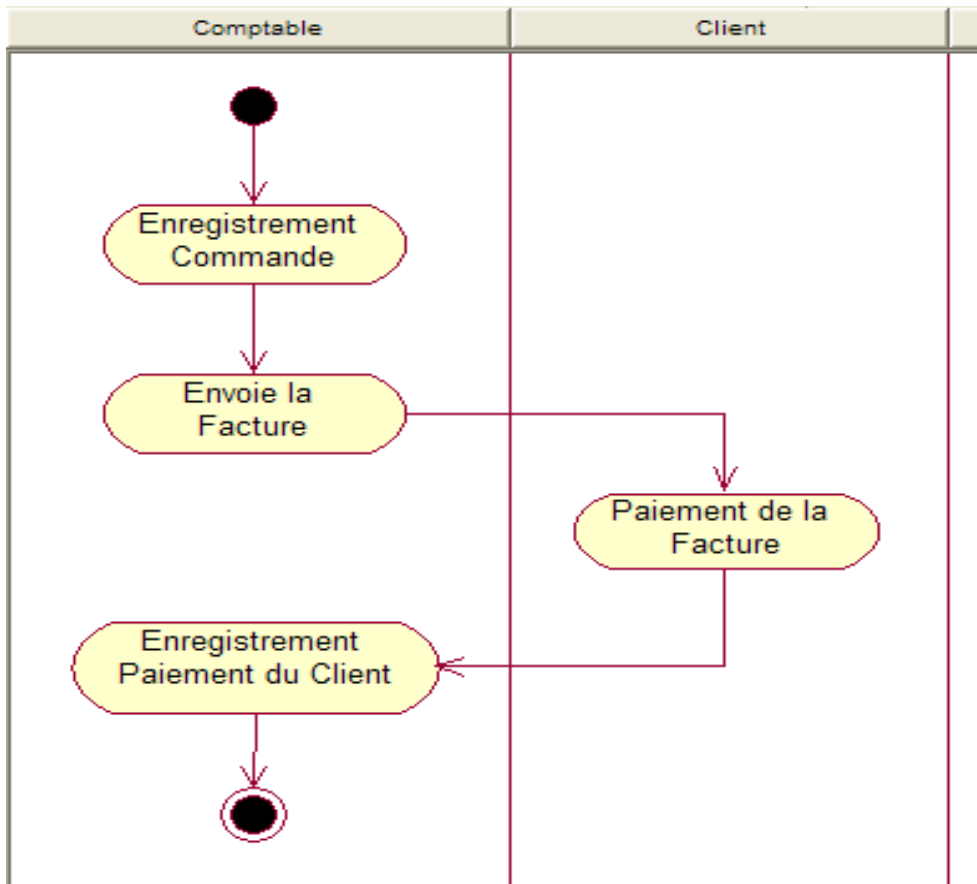
Réception livraison, contrôle qualité, contrôle quantité et enregistrement livraison.

Proposez un diagramme d'activité représentant ce flot d'information

**Exercice 3**

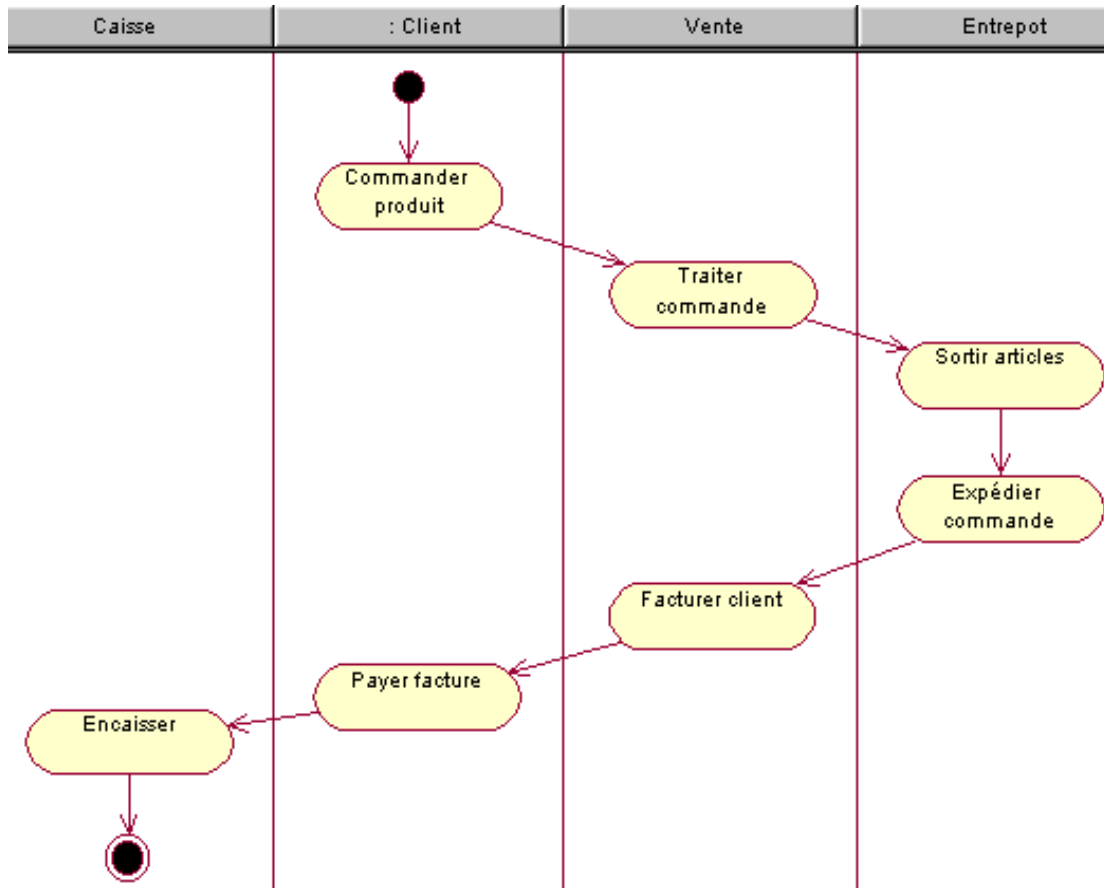
Construire un diagramme d'activité pour modéliser le processus de commander d'un produit. Le processus concerne les acteurs suivants:

- ☐ **Comptable** : enregistrement commande, envoie la facture et enregistrement paiement du client
- ☐ **Client** : paiement de la facture

**Exercice 4**

Construire un diagramme d'activité pour modéliser le processus de commander d'un produit. Le processus concerne les acteurs suivants:

- ☐ **Client**: qui commande un produit et qui paie la facture
- ☐ **Caisse**: qui encaisse l'argent du client
- ☐ **Vente**: qui s'occupe de traiter et de facturer la commande du client
- ☐ **Entrepôt**: qui est responsable de sortir les articles et d'expédier la commande.



Exemple : activité de vente par correspondance (VPC)

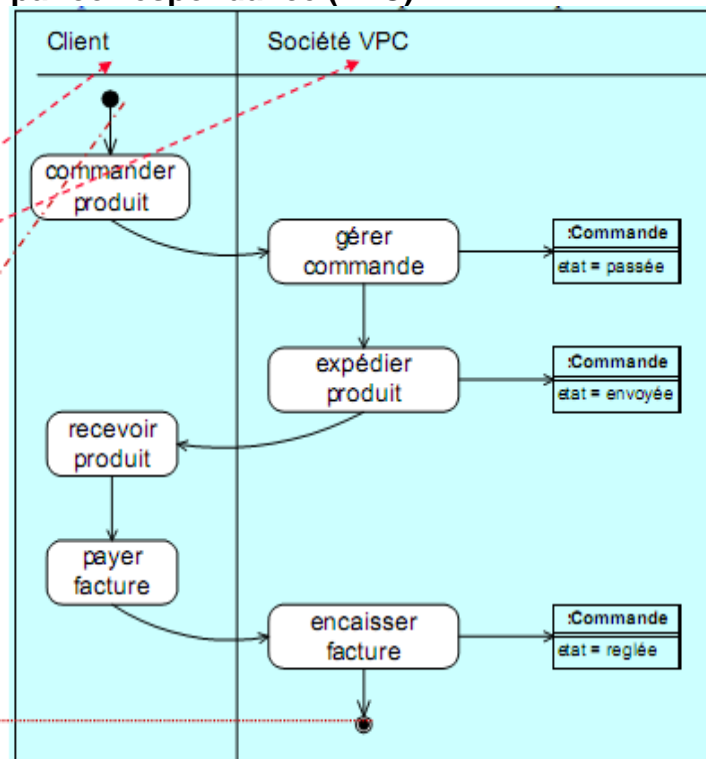
**Montre le cycle de vie d'une classe donnée.*

**En général divisé en couloirs d'activité*

(Swimlanes)

Début

Fin



Diagrammes de Composants et de Déploiement

- Un diagramme de composant représente les composants logiciels d'un système Diagramme utilisé dans la phase de conception / réalisation : éléments physiques constituant le système et leurs relations
- Décrit les éléments physiques et leurs relations dans l'environnement de réalisation sous forme de modules.
- Modules : toutes sortes d'éléments physiques (fichiers sources/données, exécutables, bibliothèques)
- **Objectif**
 - représenter l'organisation et les dépendances entre composants logiciels
 - description des composants et de leurs relations dans le système en construction
- **Composant**
 - partie physique et remplaçable d'un système qui se conforme à et fournit la réalisation d'interfaces
 - doit être compris comme un élément qu'on peut acheter, associer à d'autres composants
 - division en composants
- **Nom du composant :**
 - Permet de distinguer un composant des autres composants
 - Il peut être un nom simple ou un nom composé qui indique le packaging auquel appartient le composant



Système Vente (diagramme de classes)

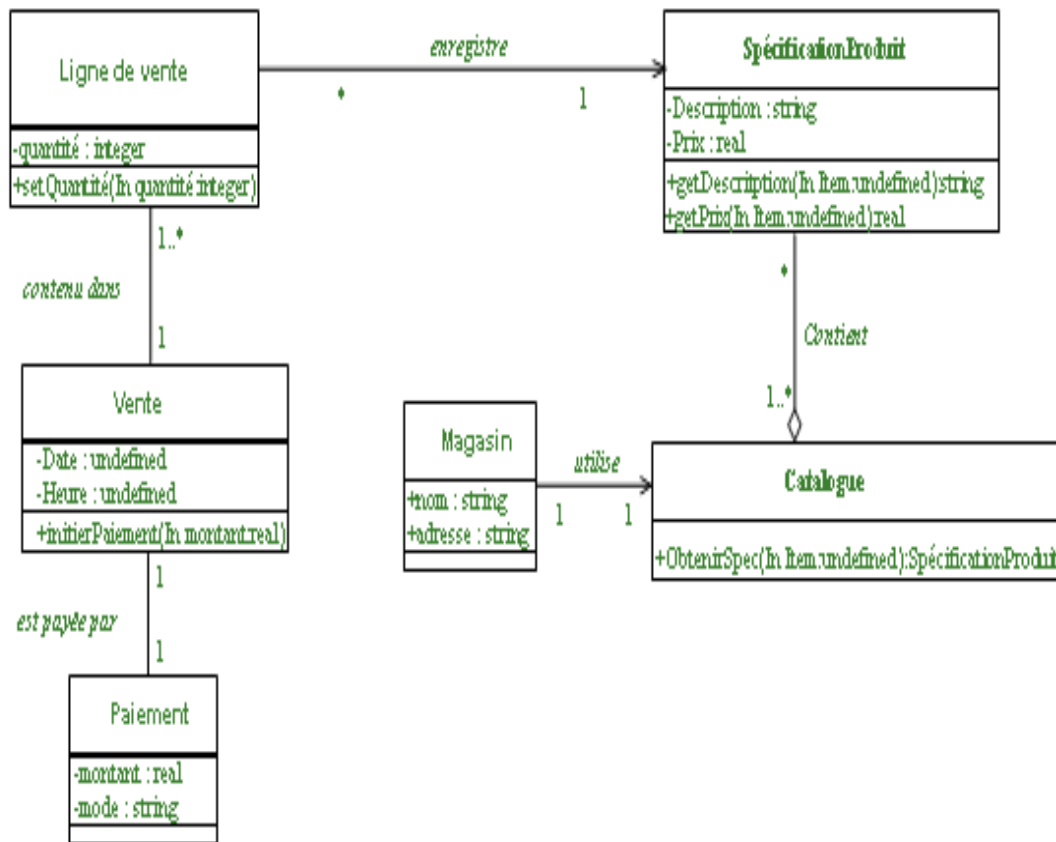


Diagramme de composants (Exemple)

■ Système Vente

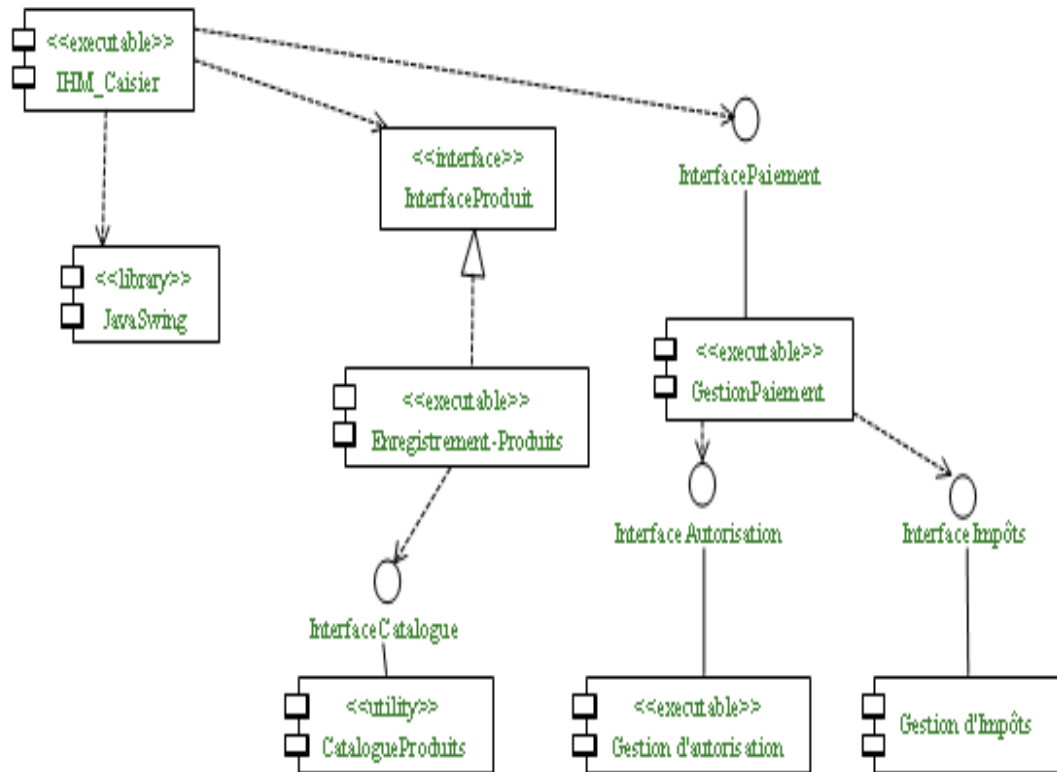


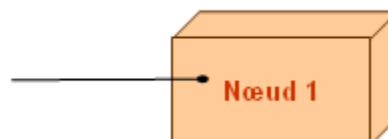
Diagramme de déploiement

- ☐ Montre la configuration des nœuds de exécution et des composants qu'y résident
- ☐ Montre les relations physiques entre les composants logiciels et matériels d'un système
- ☐ Permet de spécifier
 - Nœuds
 - Relations : (dépendance, associations)

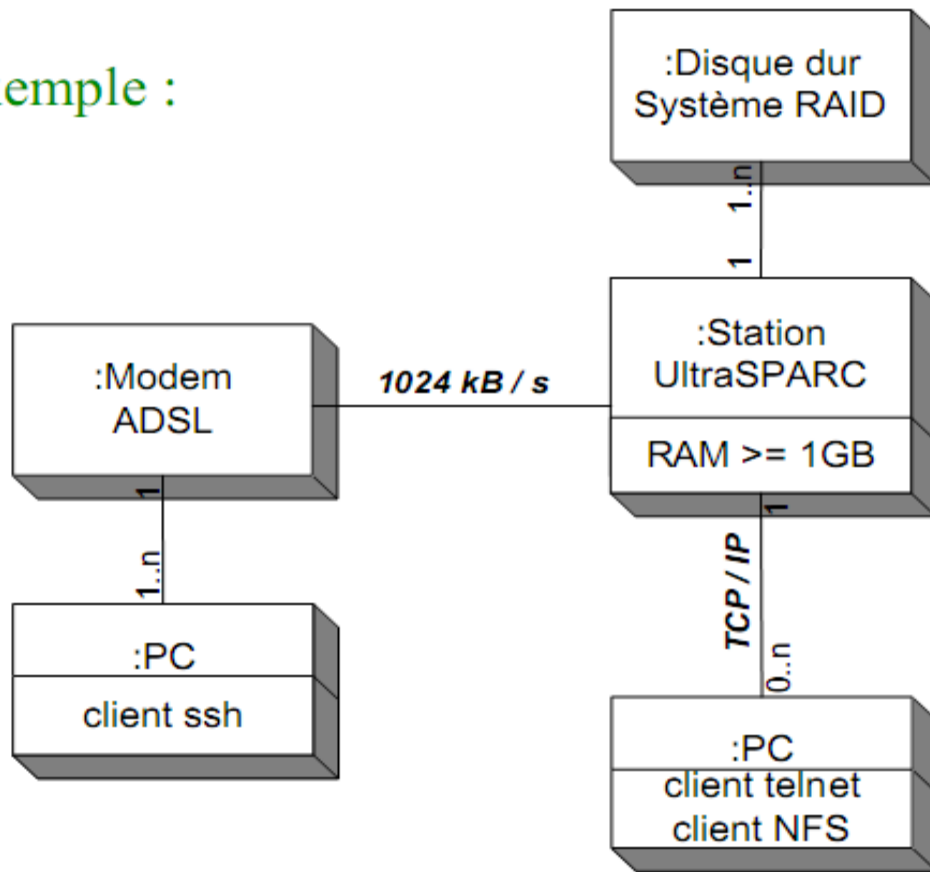
Nœud

- ☐ **Nom du nœud :**
 - Permet de distinguer un nœud des autres nœuds
 - Le nom peut être composé du nom de paquetage qui contient le nœud

Nom du nœud



Exemple :



Démarche d'application D'UML

- ☐ Étape 1 : élaboration d'un diagramme de contexte du système à étudier: Précision du champ du système à étudier
- ☐ Étape 2 : identification et représentation des cas d'utilisation : Identification des fonctions du système
- ☐ Étape 3 : description et représentation des scénarios: chaque cas d'utilisation se traduit par un certain nombre de scénarios. Chaque scénario fait l'objet d'une description textuelle. Chaque scénario est ensuite décrit sous forme graphique à l'aide du diagramme de séquence et/ou diagramme de collaboration.
- ☐ Étape 4 : identification des objets et classes
- ☐ Étape 5 : élaboration du diagramme de classes
- ☐ Étape 6 : élaboration du diagramme état-transition : pour chaque classe importante c'est à dire présentant un intérêt pour le système à modéliser, un diagramme état-transition est élaboré.
- ☐ Étape 7 : consolidation et vérification des modèles: Itération des étapes 3, 4, 5 et 6 => niveau suffisant pour la description du système.