Modélisation Objet

Le langage UML

Plan du cours

I - UML c'est quoi exactement ?

II – L'approche Objet

III – Tour d'horizon des diagrammes

IV – Les diagrammes un par un

V - Compléments :
- OCL,
- Correspondance UML – Java,
- UML vs Merise.

Références

- Le guide de l'utilisateur UML, I. Jacobson, G. Booch & J. Rumbaugh, ed. Eyrolles, 2000
- UML en action, P. Roques & F. Vallée ,ed. Eyrolles
- Introduction à UML, Russ Miles & Kim Hamilton, ed. O'Reilly
- Modélisation objet avec UML, PA. Muller, N. Gaertner, ed. Eyrolles
- http://www.OMG.org : les normes UML, OCL, MDA, …
- http://uml.free.fr : UML en français
- D'autres cours sur le web :
  - http:// www-igm .univ-mlv .fr/ ~dr/DESS/Elaboration/siframes.htm
  - http://www.iutc3.unicaen.fr/~moranb/cours/acsi/ menucoo.htm
  - http://www.cnam-versailles.fr/ress_uv_pres/prog_oo/Ressources%pedagogiques/NotationUML-0399.ppt

UML c'est quoi exactement ?

I

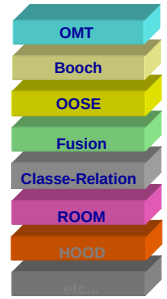
UML c'est quoi exactement ?

UML C'est quoi exactement ?

UML = Unified Modeling Language
Langage unifié pour la modélisation

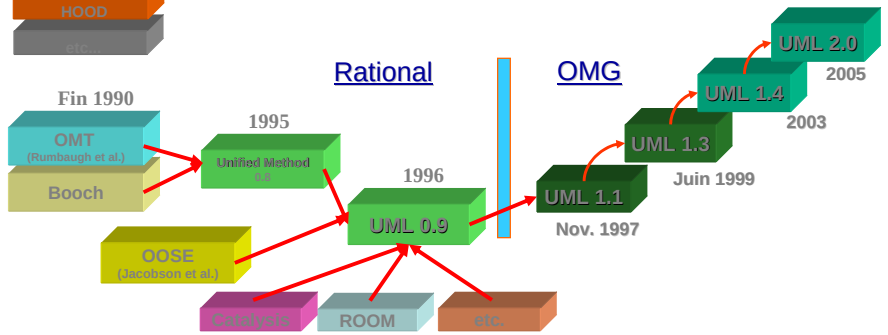
- Langage de modélisation objet, indépendant de la méthode utilisée.
- UML est un langage pour :
  - Comprendre et décrire un problème,
  - Spécifier un système,
  - Concevoir et construire des solutions,
  - Documenter un système,
  - Communiquer.

UML C'est quoi exactement ? – La genèse



- Au départ, plus de 150 méthodes et langages !
- Unification progressive de plusieurs méthodes, de remarques des utilisateurs, des partenaires
- 1989 : création de l'**OMG** (Object Management Group)

Groupe créé à l'initiative de grandes sociétés informatiques américaines afin de normaliser les systèmes à objets. Première réalisation de l'OMG : CORBA.



UML c'est quoi exactement ? – La genèse

- UML a été « boosté » par l'essor de la programmation orientée objet.
- Historique de l'approche orientée objet :
  - Simula (1967),
  - Smalltalk (1976),
  - C++ (1985),
  - Java (1995),
  - …
- UML permet de modéliser une application selon une vision objet, indépendamment du langage de programmation.

UML c'est quoi exactement ? – La portée

- UML reste au niveau d'un langage et ne propose pas de processus de développement
  - ni ordonnancement des tâches,
  - ni répartition des responsabilités,
  - ni règles de mise en œuvre.
- Il existe de (très) nombreux outils pour faire de la modélisation UML.
- Certains ouvrages et AGL basés sur UML proposent un processus en plus de UML.
  - Exemple : le processus unifié UP.

UML c'est quoi exactement ? – Le méta-modèle

- UML est bien plus qu'un outil pour dessiner des représentations mentales !
- La notation graphique n'est que le support du langage.
- UML repose sur un **méta-modèle**, qui normalise la sémantique de l'ensemble des concepts.

UML c'est quoi exactement ? – En résumé

- UML est un langage de modélisation objet
- UML n'est pas une méthode
- UML convient pour tous les types de systèmes, tous les domaines métiers, et tous les processus de développement
- UML est dans le domaine public
- Les concepts véhiculés dans UML sont définis et non équivoques

Tour d'horizon des diagrammes

II

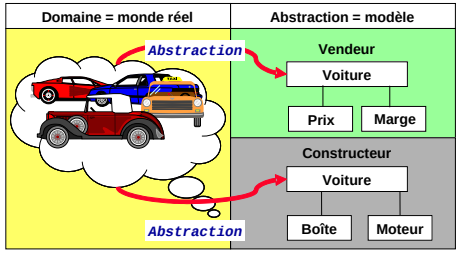
L'approche Objet

Les objets

- Les objets du monde réel nous entourent ; ils naissent, vivent et meurent.
- Les objets informatiques définissent une représentation simplifiée des entités du monde réel.
- Les objets représentent des entités
  - concrètes : avec une masse
  - abstraites : concept

Abstraction

- Une **abstraction** est un résumé, un condensé, une mise à l'écart des détails non pertinents dans un contexte donné.
- Mise en avant des caractéristiques essentielles et utiles.
- Dissimulation des détails (complexité).



Caractéristiques fondamentales des objets

- Objet = État + Comportement + Identité
- Classe (type)
- Communication entre objets par envoi de messages
- Héritage
- Polymorphisme
- Encapsulation

L'état

- L'état d'un objet :
  - regroupe les valeurs instantanées de tous les attributs d'un objet
  - évolue au cours du temps
  - est la conséquence des comportements passés
- Exemples :
  - un signal électrique : l'amplitude, la pulsation, la phase, …
  - une voiture : la marque, la puissance, la couleur, le nombre de places assises, …
  - un étudiant : le nom, le prénom, la date de naissance, l'adresse, …

Le comportement

- Le comportement
  - décrit les actions et les réactions d'un objet
  - regroupe toutes les compétences d'un objet
  - se représente sous la forme d'opérations (méthodes)
- Un objet peut faire appel aux compétences d'un autre objet
- L'état et le comportement sont liés
  - Le comportement dépend souvent de l'état
  - L'état est souvent modifié par le comportement

L'identité

- Tout objet possède une identité qui lui est propre et qui le caractérise.
- L'identité permet de distinguer tout objet de façon non ambiguë, indépendamment de son état.
- Les langages objets utilisent généralement les adresses (*références, pointeurs)* pour réaliser les identifiants

  Un attribut ***identifiant*** *n'est pas nécessaire*

Communication entre objets

- Application = société d'objets collaborant
- Les objets travaillent en synergie afin de réaliser les fonctions de l'application
- Le comportement global d'une application repose sur la communication entre les objets qui la composent
- Les objets
  - ne vivent pas en ermites
  - Les objets interagissent les uns avec les autres
  - Les objets communiquent en échangeant des messages

Communication (suite)

- Exemples de catégories de messages
  - **Constructeurs** : créent des objets
  - **Destructeurs** : détruisent des objets
  - **Accesseurs** : renvoient tout ou partie de l'état
  - **Modifieurs** : changent tout ou partie de l'état
  - **Itérateurs** : parcourent une collection d'objets

Les classes

- La classe
  - est une description abstraite d'un ensemble d'objets
  - peut être vue comme la factorisation des éléments communs à un ensemble d'objets
  - c'est un modèle d'objets ayant les mêmes types de propriétés et de comportements. Chaque instance d'une classe possède ses propres valeurs pour chaque attribut.
- Description des classes
  - Séparée en deux parties
    - La **spécification** d'une classe qui décrit le domaine de définition et les propriétés des instances de cette classe (type de donnée)
    - La **réalisation** qui décrit comment la spécification est réalisée

Conclusion

- Les objets naissent, vivent et meurent
- Les objets interagissent entre eux
- Les objets sont *regroupés* dans des classes qui les décrivent de manière abstraite
- La classe est un modèle d'objets ayant les mêmes types de propriétés et de comportements.

Tour d'horizon des diagrammes

Université d'Évry-Val d'Essonne

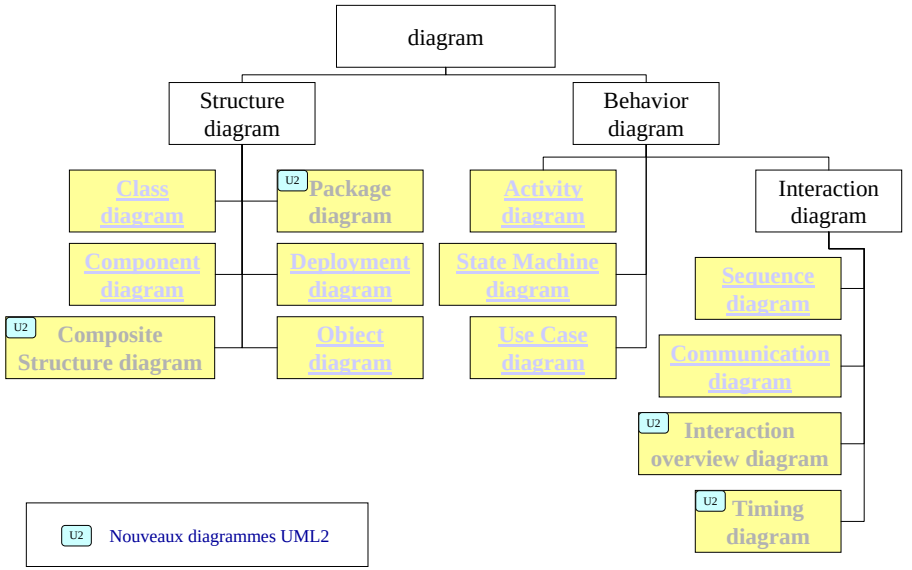
III

Tour d'horizon des diagrammes

Tour d'horizon des diagrammes – 13 diagrammes !

- Historiquement UML proposait 9 diagrammes (UML 1.x)
- UML 2 a enrichi les concepts des diagrammes existants et a ajouté 4 nouveaux diagrammes
- Les diagrammes manipulent des concepts parfois communs (ex. classe, objet, …), parfois spécifiques (ex. cas d'utilisation)

Tour d'horizon des diagrammes – 13 diagrammes !



- **Restrictions** : au sein d'une organisation ou d'une équipe, on utilise un « sous-ensemble » de UML :
  - Sous-ensemble de diagrammes,
  - Sous-ensemble des possibilités offertes par chaque diagramme.
- **Extensions** : UML possède des mécanismes d'extension, qui permettent d'adapter le langage à une organisation, une équipe ou un domaine particulier.
- Un **profil** UML est un ensemble cohérent de concepts UML, d'extensions/restrictions, de contraintes, de règles et notations.
  - Exemples : profil EJB, profil SIG, profil RT, …

IV

Les diagrammes un par un

Le diagramme de classes

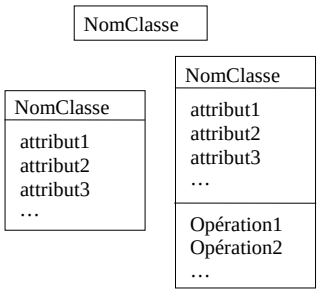
- Structure statique d'un système
  - Classes (ensembles d'objets)
  - Relations entre classes (ensemble de liens entre objets)
- Opérations / méthodes
  - Opération : service qui peut être demandé à n'importe quel objet de la classe
  - Méthode : implémentation d'une opération
  - Chaque opération (non abstraite) d'une classe doit avoir une méthode qui fournit un *algorithme* exécutable comme corps (cet algorithme est donné dans un langage de programmation ou dans du texte structuré)

Niveau de détail d'analyse

Spécification : plusieurs niveaux de détails :

- Niveau de détail d'**analyse**
  - Pas de précision sur la mise en œuvre (langage, plate-forme, …)
  - Indépendant du logiciel
  - Plusieurs niveaux de précision au fil des itérations d'analyse

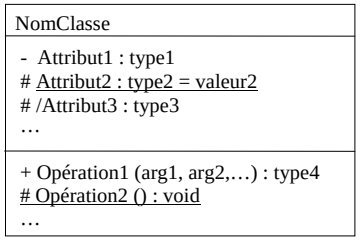
  - simplifié
    - Uniquement le nom de la classe
  - intermédiaire
    - Nom de la classe
    - Nom des attributs, ou des opérations
  - complet
    - Nom de la classe
    - Noms des attributs
    - Noms des méthodes



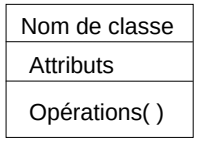
Niveau de détail de conception

- Niveau de détail de **conception**
  - Identification de l'interface des classes :
    - type des objets, comportement externe, façon interne de les mettre en œuvre
  - Indépendant du logiciel

  - Attributs :
    - Type
    - Valeurs par défaut
    - Degré de visibilité
    - Caractéristique
  - Opérations :
    - Signature
    - Degré de visibilité
    - Caractéristique



Notation des attributs et opérations



- Attribut
  <visibilité> <nomAttribut> : <type> = <valeur par défaut>
- Opération
  <visibilité> <nomOpération> (listeParamètres) : <typeRetour>
- Paramètres
  <nom> : <type> = <valeur par défaut>

Visibilité

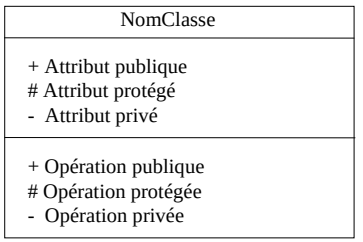
Visibilité = degré de protection

+ : publique (accessible à toutes les classes)

: protégé (accessibles uniquement aux sous-classes)

~ : paquetage (accessible uniquement aux classes du paquetage)

- : privé (inaccessible à tout objet hors de la classe)



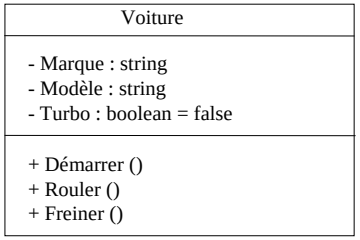
Type

- Attribut
  - type de base (entier, booléen, caractère, tableau…)
    - integer, double, char, string, boolean, currency, date, time, void
  - Au niveau analyse, pas d'attribut instance d'un classe (on utilise une association)
  - Au niveau conception, on décide quelles associations peuvent être représentées par des attributs
- Opération

  Type quelconque (de base ou classe)

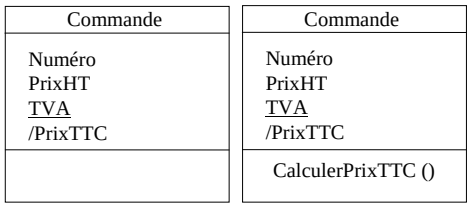
Valeur par défaut des attributs

- Affectée à l'attribut à la création des instances de la classe
- En analyse, on se contente souvent d'indiquer le nom des attributs



Attributs et opérations dérivés

- Notation : /nomAttribut
- Propriété redondante, dérivée d'autres propriétés déjà allouées
- En conception, un attribut dérivé peut donner lieu à une opération qui encapsulera le calcul effectué

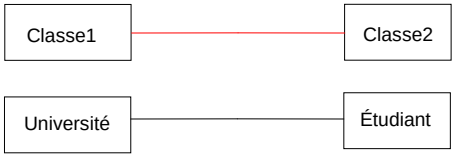


Les relations entre classes

- L'association
- L'agrégation
- La composition
- La généralisation
- *La dépendance*

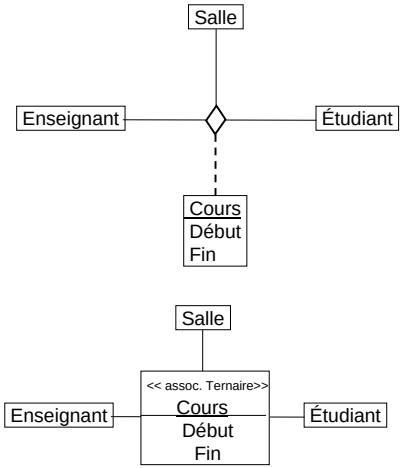
L'association

- L'association exprime une connexion structurelle entre classes
- La plupart des associations sont binaires connectent 2 classes
- Notation :



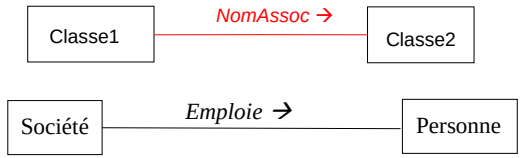
Association n-aire

- Représentation au niveau de l'association
- Représentation au moyen d'une classe
  + contrainte qui exprime que les multiples branches de l'association s'instancient simultanément en un même lien



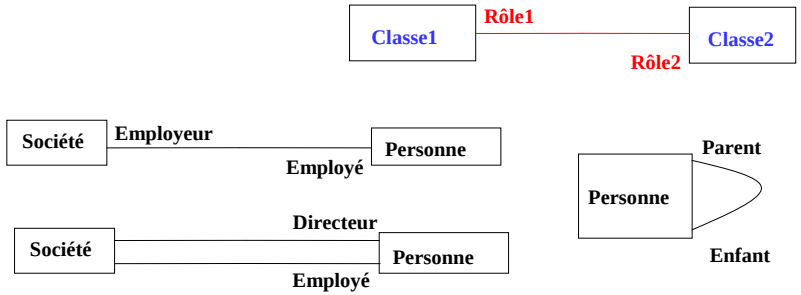
Nommage des associations

- Indication du sens de **lecture** de l'association
- Une association peut se lire dans les 2 sens, en fonction des besoins
- Usage : forme verbale, active ou passive
- Surtout utilisé dans les modèles d'analyse
  (en conception, on utilise plutôt le nommage des rôles)



Nommage des rôles

- Le rôle décrit comment une classe voit une autre classe à travers une association
- Une association a par essence 2 rôles, selon le sens dans lequel on la regarde
- Usage : Forme nominale



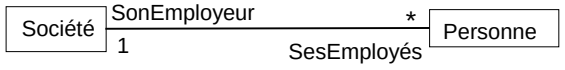
Multiplicité

- Chaque rôle porte une indication de multiplicité : nombre d'objets de la classe considérée pouvant être liés à un objet de l'autre classe
- Information portée par le rôle

Valeur :	signification :
1	Un et un seul
0..1	Zéro ou un
M .. N	De M à N (entiers naturels)
*	De zéro à plusieurs
0 .. *	De zéro à plusieurs
1 .. *	D'un à plusieurs

Lecture d'un association

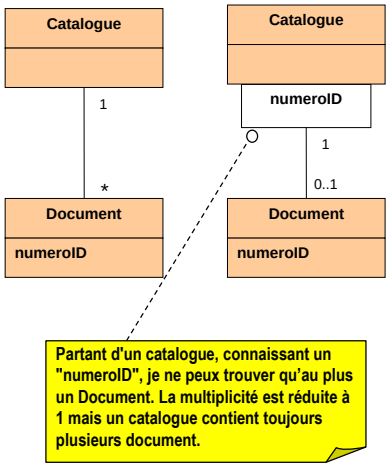
« Un employeur emploie plusieurs personnes »



« Un employé est employé par un seul employeur »

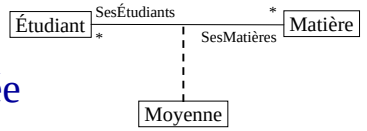
Les restrictions

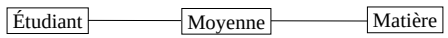
Une restriction (ou qualification) consiste à sélectionner un sous-ensemble d'objets parmi l'ensemble des objets qui participent à une association (association qualifiée)



Classe associative

- Ensemble d'attributs qualifiant la relation
- Représentation simplifiée





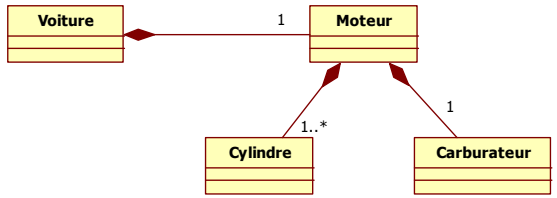
L'agrégation

- Connexion bidirectionnelle dissymétrique
  - une des extrémités est prédominante par rapport à l'autre
  - Ne concerne qu'un seul rôle
- Représentation des relations de type
  - tout et parties
  - composé et composants
  - maître et esclaves
- Deux types d'agrégation
  - **Agrégation** partagée – notion de co-propriété
    - La création (resp. la destruction) des composants est indépendante de la création (resp. la destruction) du composite
    - Un objet peut faire partie de plusieurs composites à la fois
  - **Composition**
    - Cas particulier de l'agrégation : attributs contenus physiquement par l'agrégat
    - La création (resp. la destruction) du composite entraîne la création (resp. la destruction) des composants.
    - Un objet ne fait partie que d'un seul composite à la fois.

Exemples

- Agrégation

- Composition

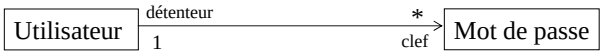


Navigabilité d'une association

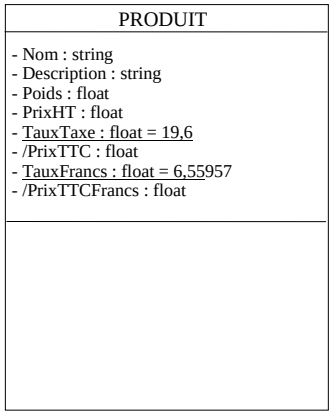
- Qualité d'une association qui permet le passage d'une classe vers une autre
- Par défaut, on peut naviguer dans les 2 sens
- On peut cependant **limiter** la navigabilité

- Exemple :

Une instance d'utilisateur peut accéder à des instances de Mot de passe, mais pas l'inverse.

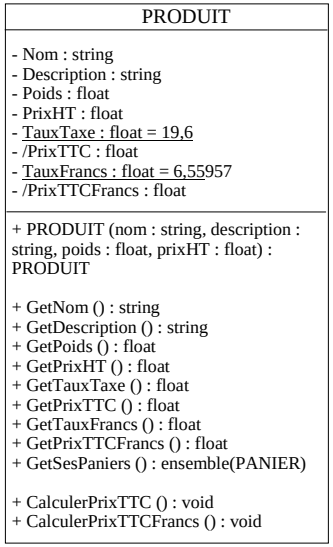


Méthode - *1ers attributs*



- Attributs décrivant l'objet
- Attributs de classes
- Attributs dérivés

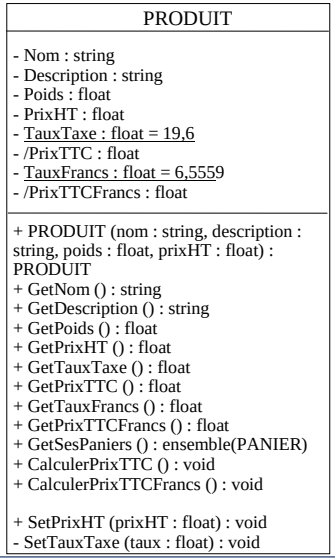
Méthode - *1ères méthodes*



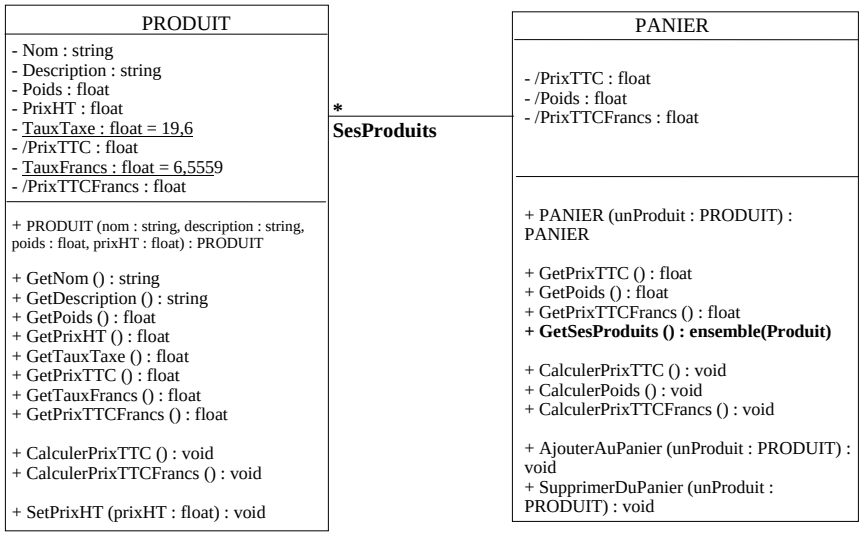
- Constructeur(s)
- Accesseurs (1 pour chaque attribut)
- Calcul des attributs dérivés

Méthode - *Autres méthodes*

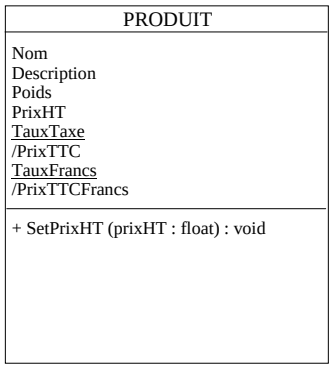
- Modifieurs
  - Publics
  - Privés
- Autres méthodes



Méthode - *Associations et rôles*



Méthode - *Version "light"*



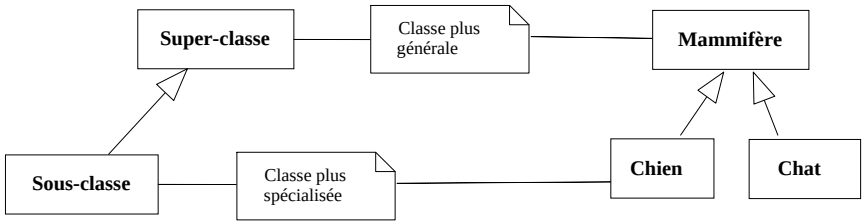
- Attributs
  - Explicite : nom
  - Implicite
    - Visibilité (privée)
    - Type, valeur par défaut
- Méthodes
  - Explicite : méthodes spécifiques
  - Implicite :
    - Constructeur
    - Accesseurs "GetToto" (1 par attribut et rôle)
    - Modifieurs privés "SetToto"
    - Calcul des attributs dérivés

Un peu de pratique

- Diagramme de classe
  - Une voiture :
    - Quatre roues, des pneus
    - Un moteur,
    - Un coffre,
    - Des passagers….
  - Un chien : c'est un mammifère qui appartient à un propriétaire, possède quatre membres, donne naissance éventuellement à des chiots, ….

Hiérarchie de classes

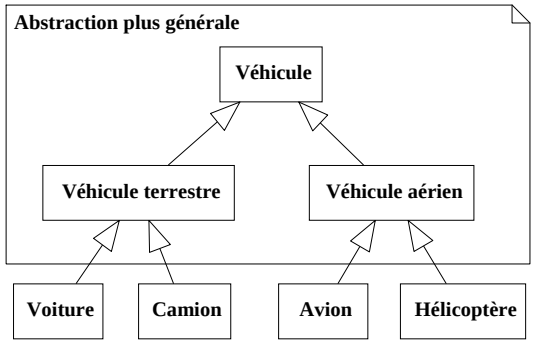
- Relation « est du type » ou « est une sorte de »
- Généralisation : Super-classe
- Spécialisation : Sous-classe



Généralisation

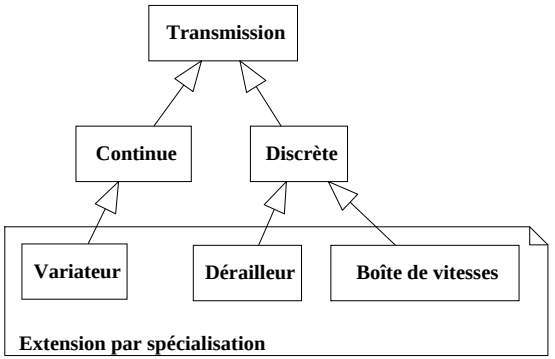
- Factorisation des éléments communs

  attributs, opérations et contraintes



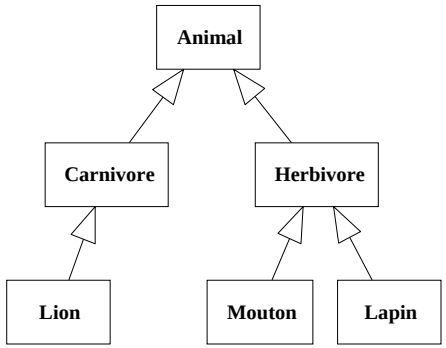
Spécialisation

- Extension cohérente d'un ensemble de classes



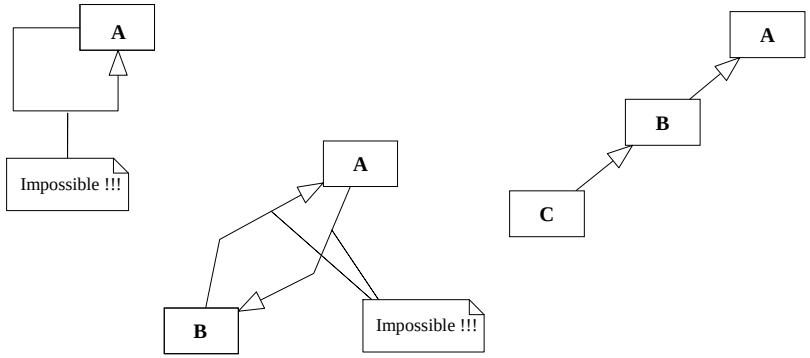
Propriétés de la généralisation

- Signifie toujours « ***est un*** » ou « ***est une sorte de*** »

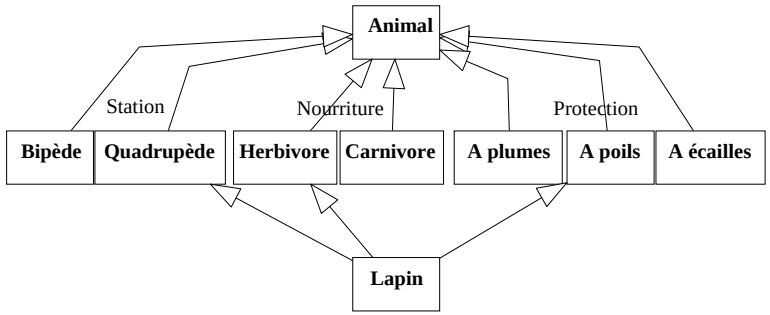


Propriétés de la généralisation

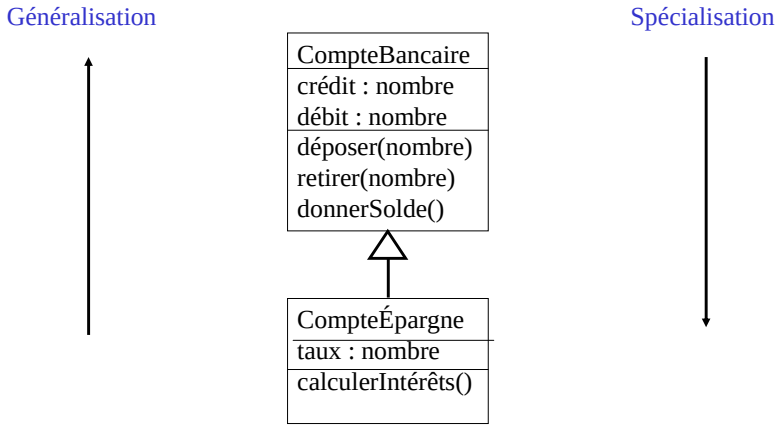
- Non-réflexive, non-symétrique, transitive



Généralisation multiple

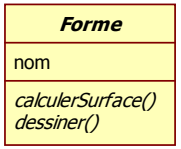


Exemple d'héritage



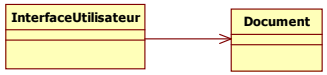
Classes et Opérations abstraites

- Classe abstraite : classe qui ne peut avoir aucune instance directe ; on écrit son nom en *italique*
- Opération abstraite : opération incomplète qui a besoin de la classe fille pour fournir une implémentation ; on écrit son nom en italique.



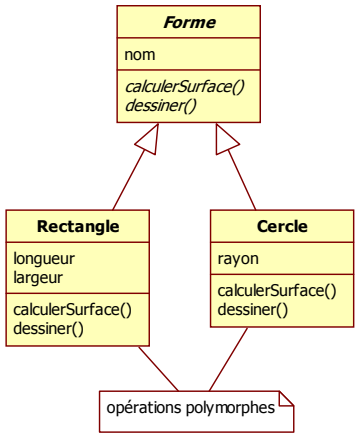
Dépendance

- Relation sémantique entre 2 éléments selon laquelle un changement apporté à l'un peut affecter l'autre.
- Implique uniquement que des objets d'une classe peuvent fonctionner ensemble.
- Notation :



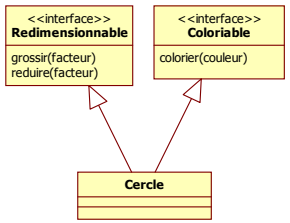
Polymorphisme

- Le même message peut être interprété de différentes façons, selon le récepteur.
- Une méthode peut être définie par le même nom dans différentes classes.



Interface

- Définition d'un contrat pour toutes les classes qui l'implémentent. Évite l'utilisation de la relation d'héritage.
- Ensemble d'opérations sans implémentation.
- Notation :



Conclusion

- Les classes sont connectées par des **relations :**

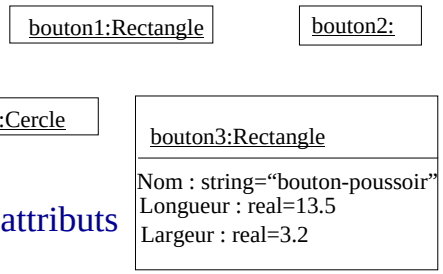
- L'**association** exprime une connexion structurelle
- L'**agrégation** est une forme d'association plus forte
- La **généralisation** permet d'ordonner les objets au sein de hiérarchies de classes
- La **dépendance** exprime que des objets travaillent brièvement avec d'autres.

Les diagrammes un par un

Le diagramme d'objets

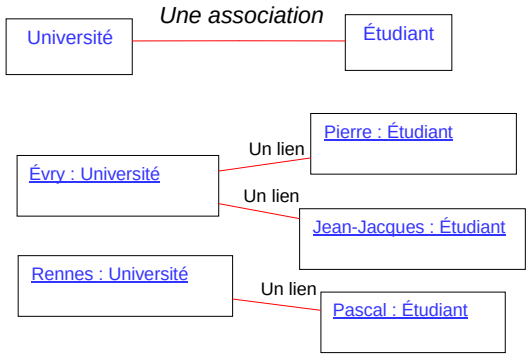
Définition

- Modélisent les instances d'éléments qui apparaissent sur les diagrammes de classe.
- Montrent un ensemble d'objets et leurs relations à un moment donné
  - Instances nommées
  - Instances anonymes
  - Instances avec valeurs d'attributs

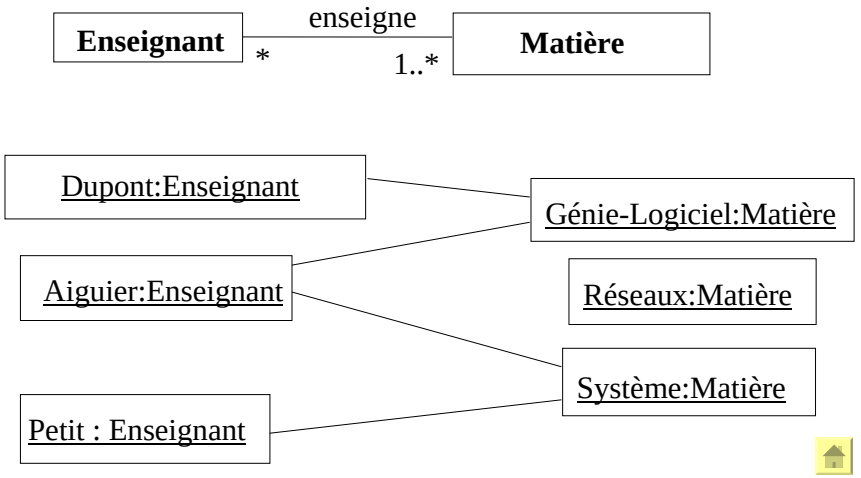


Association/Lien

- Une association est une abstraction des liens qui existent entre les objets instances des classes associées
- Les liens se représentent de la même manière que les associations



Exemple



Les diagrammes un par un

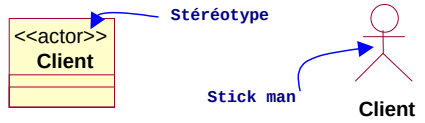
Le diagramme de cas d'utilisation

Le diagramme de cas d'utilisation

- Ce diagramme permet une description, en prenant le **point de vue de l'utilisateur**, du système à construire.
- Pas d'aspects techniques.
- Les deux concepts de base du diagramme :
  - l'acteur,
  - Le cas d'utilisation.

Acteur - définition

- Un acteur est une **entité externe** au système :
  - qui attend un ou plusieurs services du système,
  - à qui le système fournit une interface d'accès,
  - qui interagit avec le système par échange de messages.
- C'est une personne ou un autre système
- Les acteurs sont décrits par une abstraction ne retenant que le rôle qu'ils jouent vis à vis du système
- Notation :



Acteurs vs Utilisateurs

- On ne doit pas raisonner en terme d'entité physique, mais en terme de **rôle** que l'entité physique joue
- Un acteur représente un rôle joué par un utilisateur qui interagit avec le système
- La même personne physique peut jouer le rôle de plusieurs acteurs (joueur, banquier)
- Plusieurs personnes peuvent également jouer le même rôle, et donc agir comme le même acteur (tous les joueurs)

Cas d'utilisation - définition

- Un cas d'utilisation modélise un **service rendu** par le système
- Il exprime les interactions entre les acteurs et le système
- Il apporte une valeur ajoutée "notable" aux acteurs concernés.
- Règle de nommage : verbe (+ complément)
- Notation :

Cas d'utilisation vs scénario

- **Cas d'utilisation** : représente un cas en général, une représentation générale et synthétique d'un ensemble de scénarios similaires décrits sous la forme d'enchaînements

  exemple : un joueur joue un coup

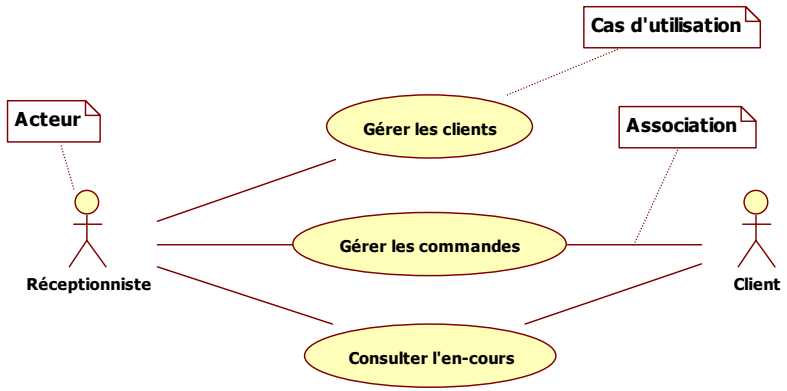
- **Scénario** : cas d'utilisation *spécifique*

  exemple : le joueur Pierre joue : il obtient 7 avec les dés, se déplace rue de Belleville et achète la propriété

Utilité

- Un moyen de déterminer le **but** et le **périmètre** d'un système
- Utilisé par les utilisateurs finaux pour exprimer leur attentes et leur besoins ➔ permet d'impliquer les utilisateurs dès les premiers stades du développement
- Support de communication entre les équipes et les clients
- Découpage du système global en grandes tâches qui pourront être réparties entre les équipes de développement
- Permet de concevoir les Interfaces Homme-Machine
- Constitue une base pour les tests fonctionnels

Exemple de diagramme de cas d'utilisation



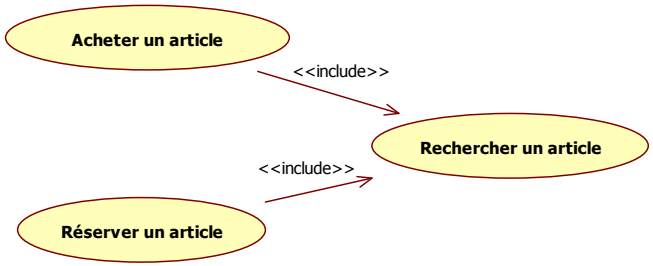
Relations entre cas d'utilisation

- UML définit trois types de relations standardisées entre cas d'utilisation :
  - une relation d'inclusion,
  - une relation d'extension,
  - une relation de généralisation/spécialisation.
- On peut également généraliser les acteurs

Inclusion <<include>>

<<include>> permet d'incorporer le comportement d'un autre cas d'utilisation.

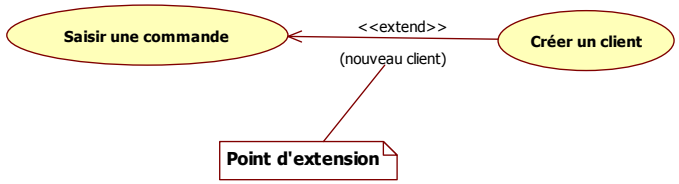
Le cas de base en incorpore explicitement un autre, à un endroit spécifié dans sa description.



Extension <<extend>>

<<extend>> permet de modéliser la partie d'un cas d'utilisation considérée comme facultative dans le comportement du système.

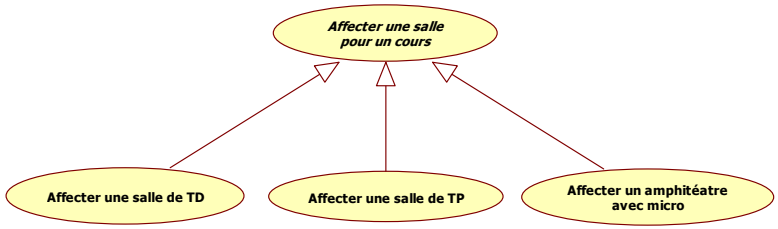
Le cas de base peut fonctionner seul, mais il peut aussi être complété par un autre, sous certaines conditions, et uniquement à certains points particuliers de son flot d'événements appelés points d'extension.



Généralisation

Les cas d'utilisation peuvent être hiérarchisés par généralisation.

Les cas d'utilisation descendants héritent de la sémantique de leur parent. Ils peuvent comprendre des interactions spécifiques supplémentaires, ou modifier les interactions héritées.



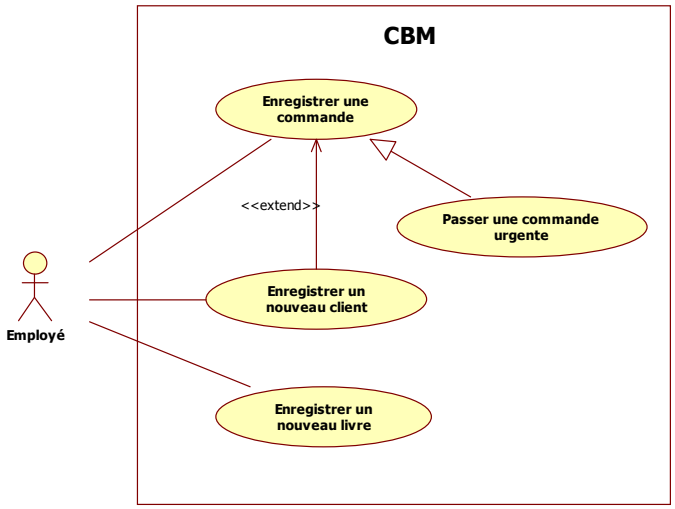
Spécification

- La description textuelle n'est pas normalisée
- Convergence vers un modèle standard :
  - Sommaire d'identification
    inclut titre, but, résumé, dates, version, responsable, acteurs…
  - Description des enchaînements
    décrit les enchaînements nominaux, les enchaînements alternatifs, les exceptions, mais aussi les préconditions, et les postconditions.
  - Exigences fonctionnelles
  - Besoins d'IHM
    ajoute éventuellement les contraintes d'interface homme-machine
  - Contraintes non-fonctionnelles
    ajoute éventuellement les informations suivantes : fréquence, volumétrie, disponibilité, fiabilité, intégrité, confidentialité, performances, concurrence, etc.
- On peut aussi réaliser des diagrammes

Exemple : La CBM

- La CBM (Computer Books by Mail) est une société de distribution d'ouvrages d'informatique qui agit comme intermédiaire entre les librairies et les éditeurs.
- Elle prend des commandes en provenance des libraires, s'approvisionne (à prix réduit) auprès des éditeurs concernés et livre ses clients à réception des ouvrages
- Il n'y a donc pas de stockage de livres.
- Seules les commandes des clients solvables sont prises en compte.
- Les commandes « urgentes » font l'objet d'un traitement particulier.
- La CBM désire mettre en place un Système Informatique lui permettant de gérer les libraires et les livres, et d'enregistrer les commandes.

Diagramme de cas d'utilisation

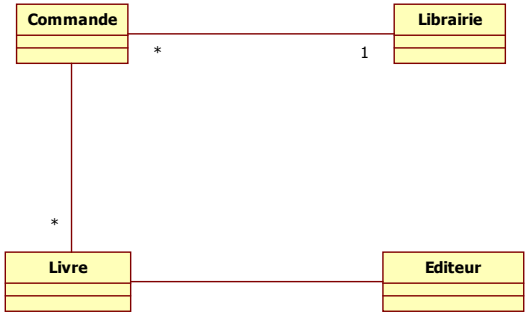


Spécification textuelle du cas « *Enregistrer une commande* »

Acteur : l'employé de la coopérative
Objectif : enregistrer une commande de livres
Précondition : le libraire existe
Scénario nominal :
1 - l'employé sélectionne le libraire et vérifie sa solvabilité
2 - l'employé vérifie l'existence des livres
3 - l'employé précise la quantité pour chaque livre
4 – L'employé confirme la commande
Postcondition : une nouvelle commande est créée.
Scénario d'exception 1 :
1a - le libraire n'est pas solvable
1b - le système alerte l'employé et lui propose d'arrêter l'enregistrement
Scénario d'exception 2 :
2a - un des livres n'existe pas
2b – Le système édite une lettre qui pourra être envoyée au libraire. La commande est placée en attente.

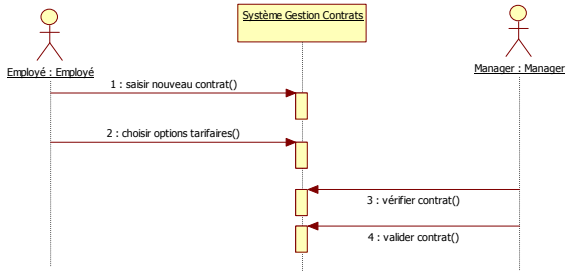
Diagrammes complémentaires

On peut ajouter à chaque cas d'utilisation un diagramme de classes simplifié, appelé Diagramme de Classes Participantes (DCP)



Diagrammes complémentaires

- Autres diagrammes possibles pour compléter la description d'un cas d'utilisation :
  - Diagramme d'activités
  - Diagramme de séquence (vision boîte noire)



Pour finir : quelques pièges à éviter

- Des cas d'utilisation trop petits et trop nombreux
  - Jacobson : pas plus de 20 UC !
  - Larman : test du patron
  - Test de la taille
- Trop d'importance au diagramme
  - Pas trop de relations entre UC !
- Décomposition fonctionnelle
  - Garder le point de vue de l'utilisateur et pas le point de vue interne !
- Confusion entre processus métier et UC
  - Pas d'interactions entre acteurs directement !
  - Se concentrer sur les actions à automatiser !

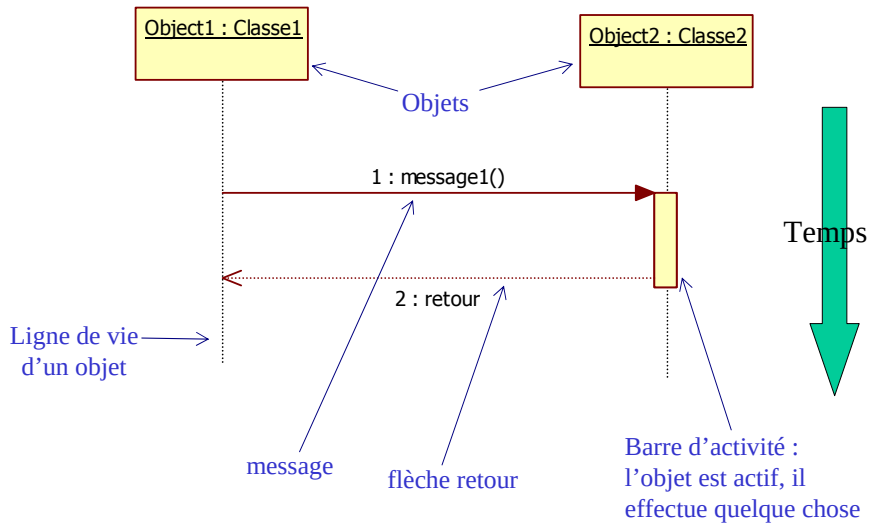
Le diagramme de séquence
Le diagramme de communication

- Décrire des scénarios particuliers
- Capturer l'ordre des interactions entre les parties du système (objets, composants, acteurs, …)
- Décrire les interactions déclenchées lorsqu'un scénario d'un cas d'utilisation est exécuté

- Ces diagrammes comportent :
  - des objets dans une situation donnée (instances)
  - les messages échangés entre les objets
- Les objets sont les instances des classes identifiées et décrites dans le diagramme de classes.

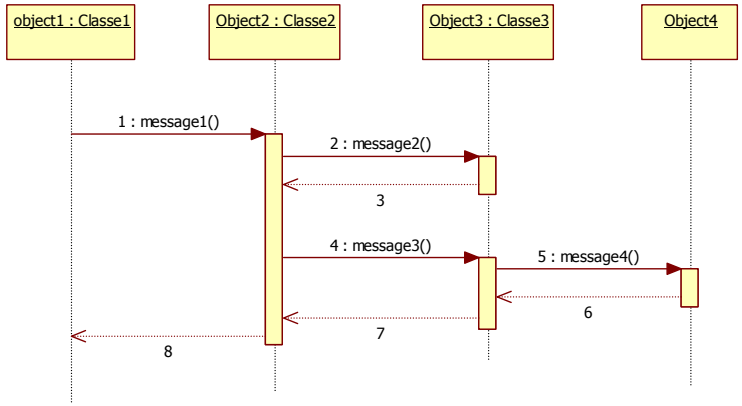
- UML propose deux types de diagrammes pour modéliser la collaboration entre les objets du système :
  - Le diagramme de séquences,
  - Le diagramme de communication
- Ces deux diagrammes sont regroupés sous le terme de diagrammes d'interactions.
- Nous nous focaliserons dans un premier temps sur le diagramme de séquences.

Diagramme de séquence en détail



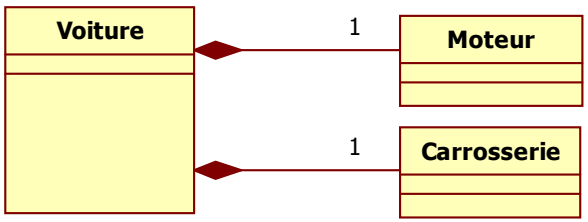
Messages imbriqués

- Un message peut conduire à l'envoi d'un ou plusieurs messages par le récepteur



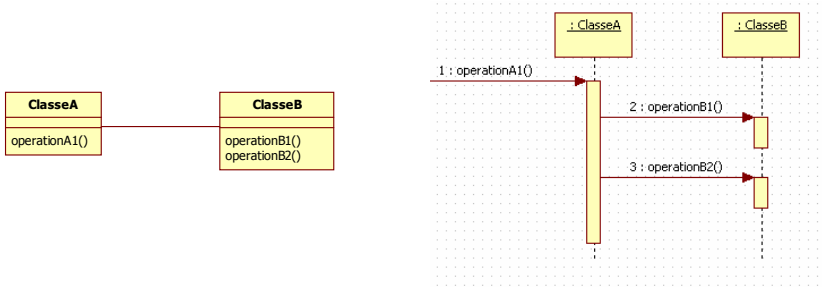
Exemple

- Comment calculer le poids d'une voiture, égal au poids du moteur plus le poids de la carrosserie ?



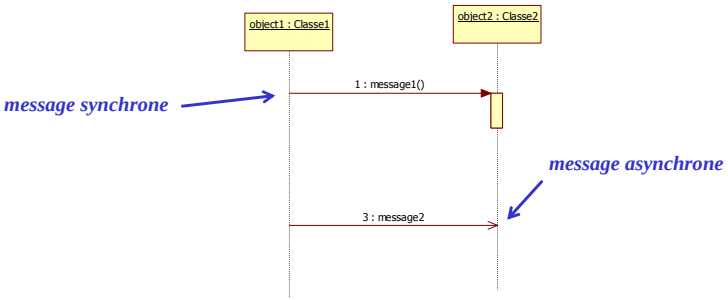
Diagrammes de séquences et diagramme de classes

- Le diagramme de classes présente une vue statique du système.
- Le diagramme de séquences présente une vue dynamique du système.



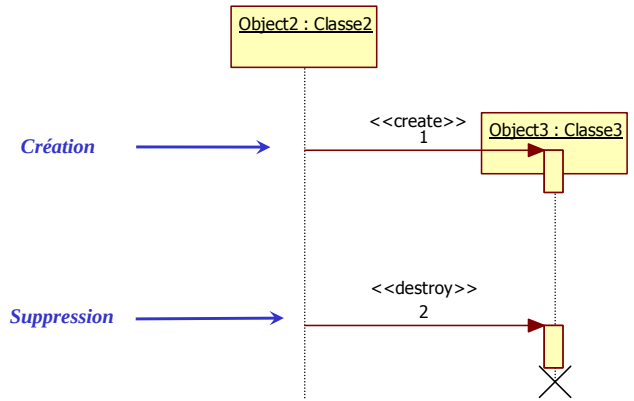
Messages synchrones/asynchrones

- Message synchrone : l'émetteur attend le retour du récepteur du message
- Message asynchrone : l'émetteur n'est pas bloqué, il continue ses traitements



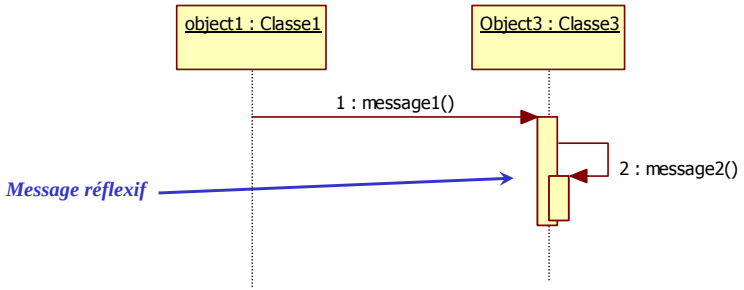
Messages de création/destruction

- Rappel : les objets naissent, vivent et meurent



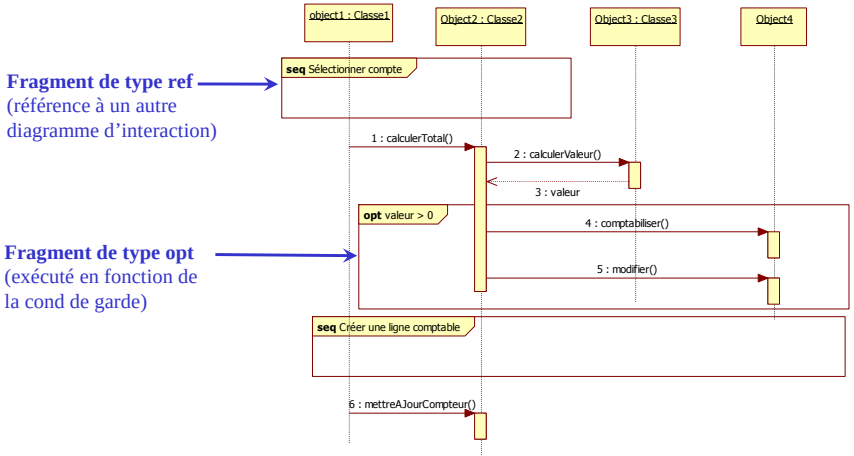
Messages réflexifs

- Envoi d'un message d'un objet à lui-même



Fragments (UML2)

- Structurer les interactions complexes



Utilisation du diagramme de séquence

- Dans la vue fonctionnelle du système
  - Diagramme complémentaire de description d'un cas d'utilisation
  - Décrire un scénario d'un cas d'utilisation : décrire les interactions entre le système et son environnement (vision boîte noire)
  - Appelé diagramme de séquence « système »
- Dans l'analyse détaillée et la conception OBJET du système
  - Décrire les interactions internes du système : les interactions entre les objets

Diagramme de communication

- Diagramme de séquences : l'accent est mis sur l'ordre temporel des interactions
- Diagramme de communication : l'accent est mis sur l'examen des interactions vis-à-vis des liens entre objets.
- Équivalents en UML1, quelques nouveautés sur le diagramme de séquences en UML2, non disponibles sur le diagramme de communication

Diagramme de communication

- Focalisation sur les liens entre objets d'une interaction

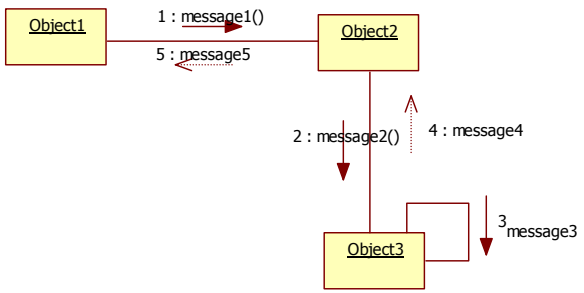
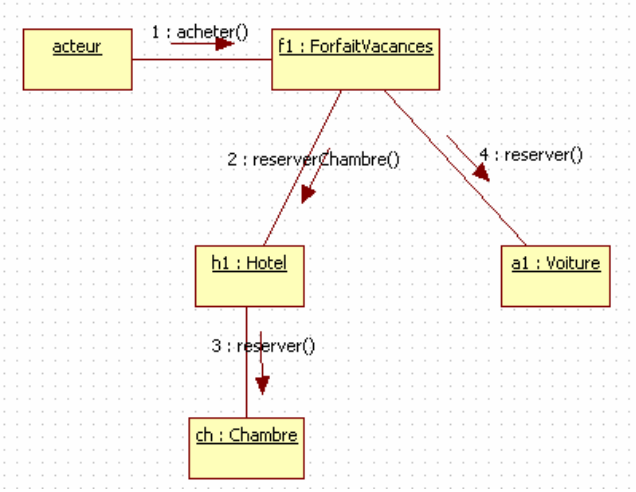


Diagramme de communication

- Exemple :

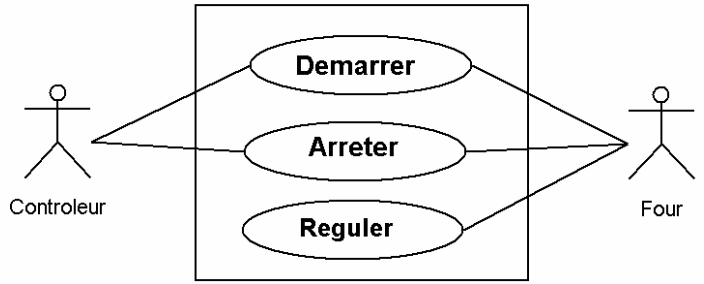


Exemple : Système de régulation d'un four

- Description du système de régulation :
  - Grâce au système de régulation, le four peut être mise en marche et arrêté.
  - Le système régule la température du four
- Objectif :
  - Identifier les acteurs et cas d'utilisation
  - Réaliser un diagramme de séquence « système » pour un des cas d'utilisation

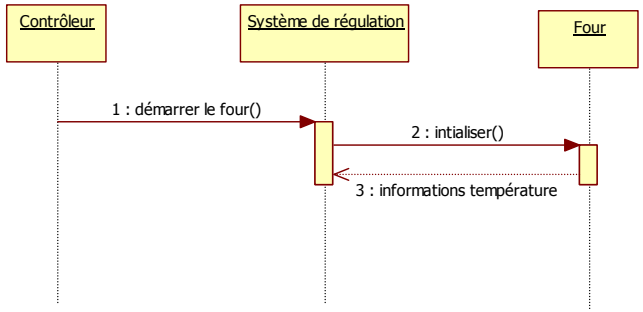
Système de régulation d'un four

- Diagramme de cas d'utilisation :



Système de régulation d'un four

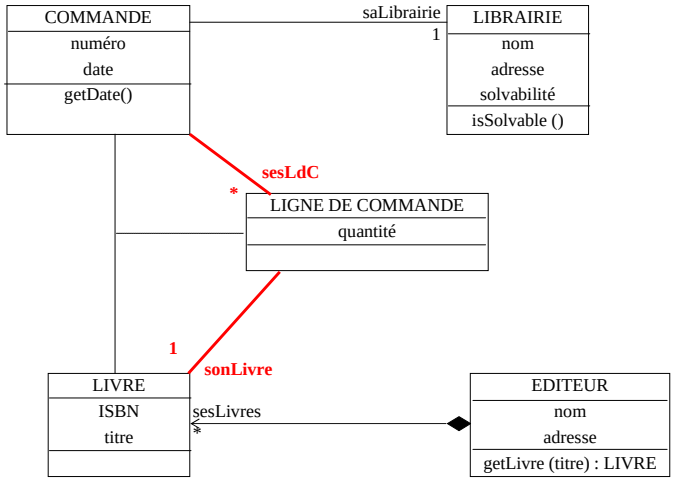
- Diagramme de séquence « système » du cas d'utilisation « Démarrer » :



Exemple : La CBM

- Description du système :
  - Reprendre l'énoncé de l'exercice sur les cas d'utilisation (planche 82).
- Objectif :
  - Réaliser un diagramme de classes pour le système CBM,
  - Réaliser un diagramme de séquence objet lorsque qu'un utilisateur demande les détails de toutes les commandes d'une librairie donnée.

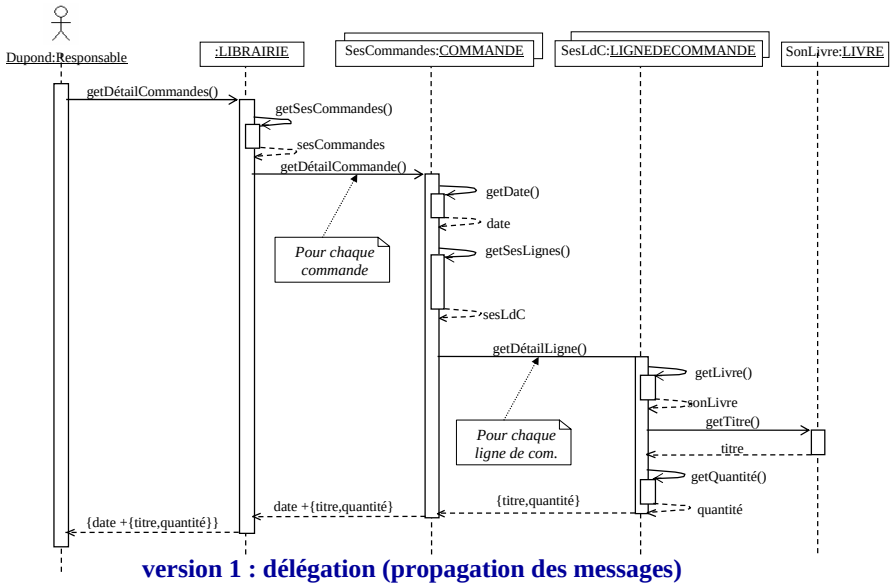
CBM : diagramme de classes V1



CBM : un diagramme de séquence

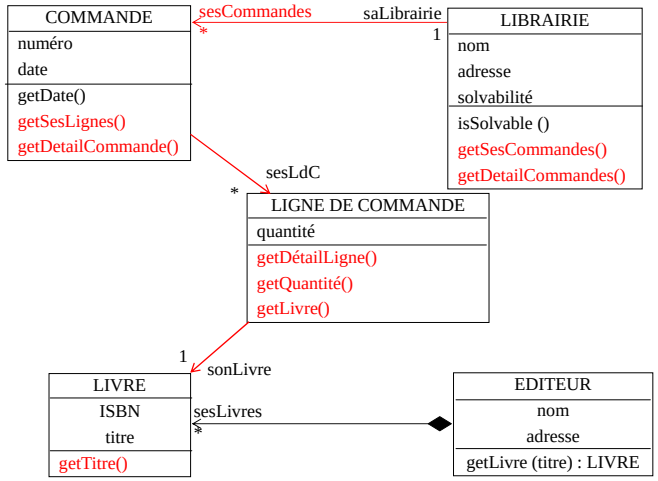
- Diagramme de séquence : « Communiquer les détails de toutes les commandes d'une librairie donnée »
- Ce diagramme correspond à la méthode ***getDétailCommandes()*** de la classe LIBRAIRIE

CBM : diagramme de séquence V1

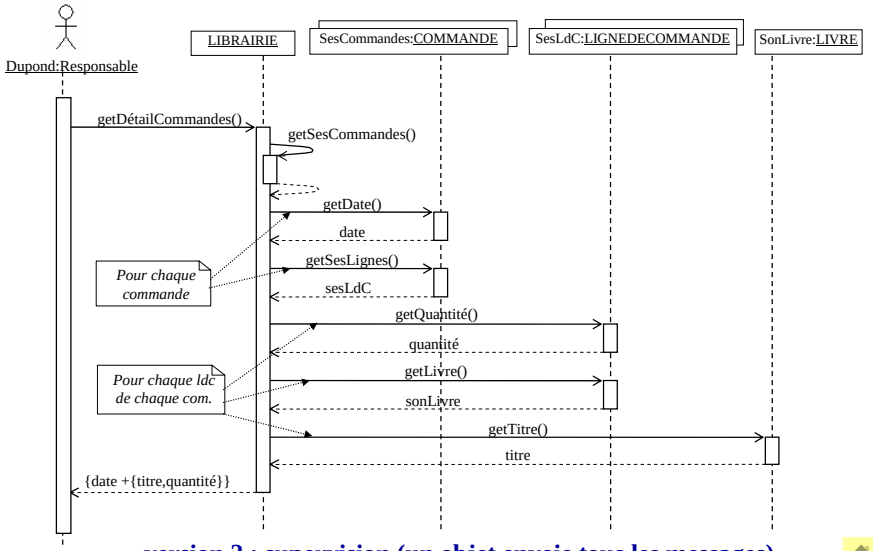


version 1 : délégation (propagation des messages)

CBM : diagramme de classes V2



CBM : diagramme de séquence V2



version 2 : supervision (un objet envoie tous les messages)

Les diagrammes un par un

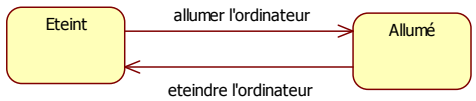
Le diagramme d'états

Diagramme d'états

- Vue synthétique du fonctionnement dynamique d'un objet
- Description du comportement d'un objet tout au long de son cycle de vie :
  - Description de tous les états possibles d'un **unique** objet à travers l'**ensemble** des cas d'utilisation dans lequel il est impliqué
  - Utile pour les objets qui ont un **comportement complexe**. Un diagramme d'états est alors réalisé pour la classe qui décrit ces objets au comportement complexe.

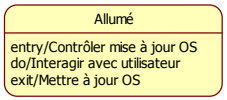
Diagramme d'états

- Éléments fondamentaux du diagramme :
  - Les états : représente une situation dans la vie d'un objet pendant laquelle il satisfait une certaine condition, exécute certaines activités, attend certains évènements.
  - Les transitions entre états : pour marquer le changement d'état d'un objet.
  - Les évènements (ou déclencheurs) qui provoquent des changements d'état.
- Exemple : diagramme d'état d'un ordinateur :



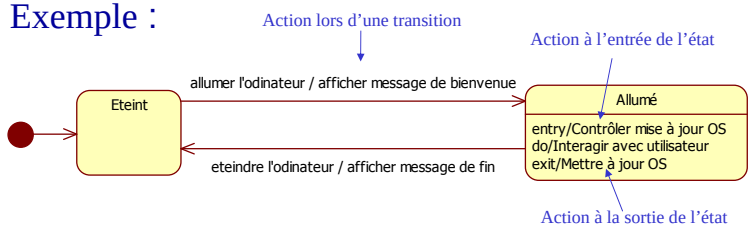
Les états en détail

- Plusieurs types d'états :
  - État initial (un seul par diagramme) :
  - État final (aucun ou plusieurs possibles) :
  - État intermédiaire (plusieurs possibles) : Nom de l'état
- Contenu d'un état intermédiaire :
  - le nom
  - l'activité attachée à cet état
  - les actions réalisées pendant cet état
- Exemple : état « Allumé » de l'ordinateur :



Actions

- Opération instantanée (durée négligeable) toujours intégralement réalisée.
- Exécutée :
  - lors d'une transition : $event_i$ / $action_i$
  - à l'entrée dans un état : **entry** / $action_i$
  - à la sortie d'un état : **exit** / $action_i$
  - interne, sans changer d'état : $event_i$ / $action_i$
- Exemple :

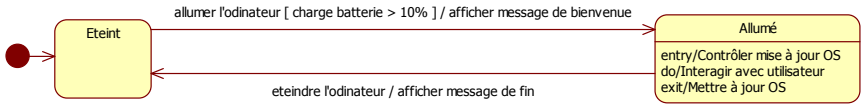


Activités

- Opération qui nécessite un certain temps d'exécution.
- Peut être interrompue à chaque instant.
- Exécutée entre l'entrée de la sortie de l'état.
- Notation : **do** / activité
- Exemple :

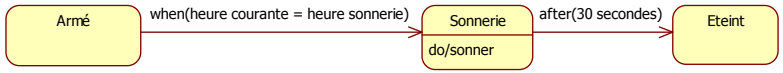
Transitions

- Passage unidirectionnel et instantané d'un état à un autre état
- Déclenchée :
  - par un **événement,**
  - automatiquement à la fin d'une **activité** (transition automatique).
- **Condition de garde** : condition booléenne qui autorise ou bloque la transition
- **Action** : réalisée lors du changement d'état
- Syntaxe complète : **<Événement> [<Garde>] / <Action>**
- Exemple :



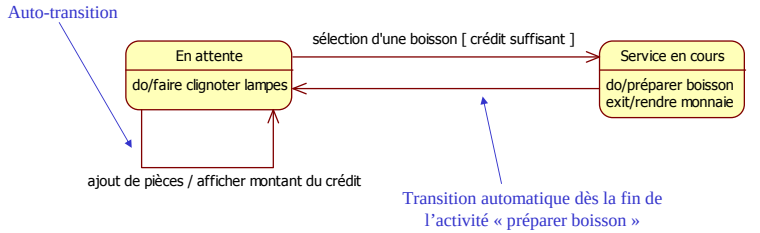
Transitions

- Deux types d'évènements particuliers :
  - Le passage du temps (time event) : représente une durée décomptée à partir de l'entrée dans l'état courant
    Notation : after(expression représentant une durée)
  - Un changement interne à l'objet (change event) : sans réception de message (souvent le temps)
    Notation : when(expression booléenne)
- Exemple : diagramme d'états d'un réveil :



Transitions

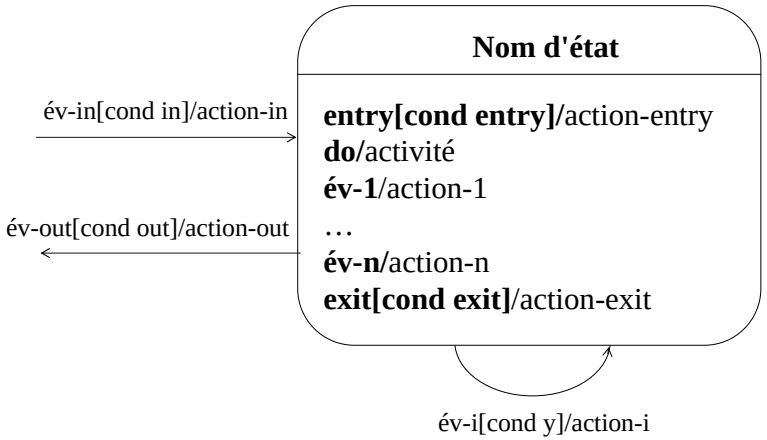
- **Transition automatique** : lorsque qu'il n'y a pas de nom d'événement sur une transition, il est sous-entendu que la transition aura lieu dès la fin de l'activité.
- **Auto-transition** : transition d'un état vers lui-même
- Exemple : diagramme d'états du distributeur de boissons :



Exercice

- Représenter sous forme de diagramme d'états le fonctionnement d'un lecteur de DVD, dont voici la description :
  - Le lecteur possède un tiroir qui peut recevoir plusieurs disques.
  - A la fin de la lecture d'un disque, le lecteur démarre la lecture du disque suivant.
  - A la fin du dernier disque, le lecteur s'arrête.
  - L'utilisateur peut lancer la lecture des disques, arrêter la lecture ou mettre en pause le lecteur.

Forme générale d'un état

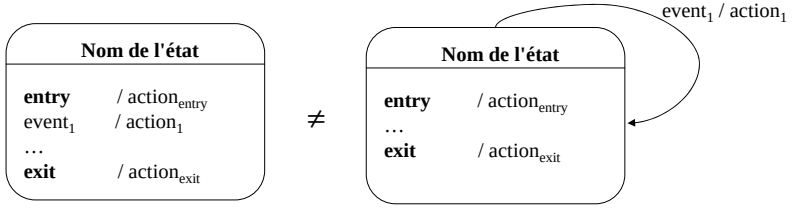


Ordonnancement des actions

- En entrée
  - Action sur la transition d'entrée
  - Action d'entrée
  - Activité associée à l'état
- En interne
  - Interruption de l'activité en cours (contexte sauvé)
  - Action interne
  - Reprise de l'activité
- En sortie
  - Interruption de l'activité en cours (contexte perdu)
  - Action de sortie
  - Action sur la transition de sortie
- Auto-transition
  - Interruption de l'activité en cours (contexte perdu)
  - Action de sortie
  - Action sur l'auto-transition
  - Action d'entrée
  - Activité associée à l'état

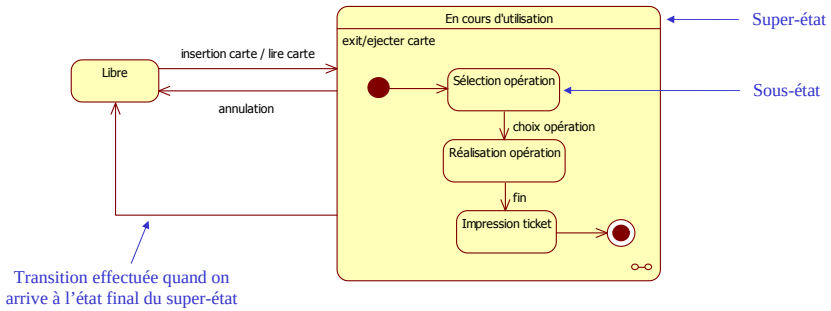
Auto-transition / Action interne

- Action interne, le contexte de l'activité est préservé (on ne sort pas de l'état)
- Auto-transition : le contexte est réinitialisé (on sort et on re-rentre dans l'état)



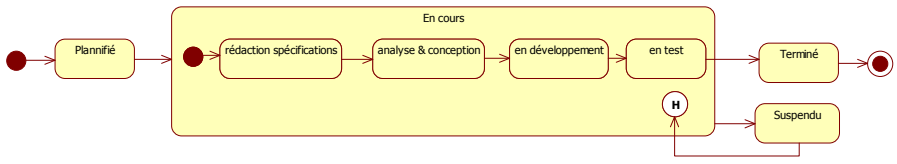
Hiérarchie d'états

- Permet de structurer les diagrammes complexes.
- Factorisation des actions, activités et transitions dans un super-état ou état composé.
- Exemple : diagramme d'états du distributeur d'argent :



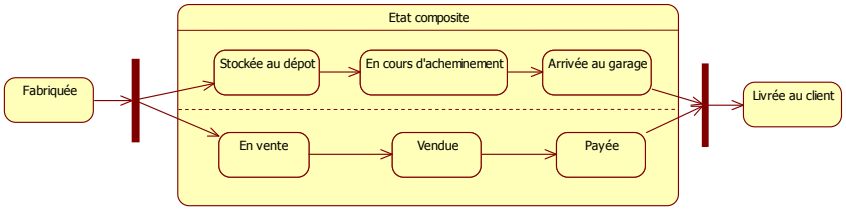
Pseudo-état historique

- Quand une transition sort d'un état à sous-états :
  - le dernier sous-état atteint n'est plus connu,
  - une nouvelle entrée dans l'état redémarre au premier sous-état.
- Un pseudo-état historique permet de mémoriser le dernier sous-état.
- Notation : Ⓗ
- Exemple : diagramme d'états d'un développement informatique :



Comportements concurrents

- Un objet peut se trouver dans plusieurs états à la fois. On crée alors un état composé de plusieurs **régions** concurrentes.
- Chaque région possède son propre diagramme d'état.
- Lorsqu'un objet est dans l'état composite, il se trouve dans un état de chaque diagramme d'état.
- Exemple : diagramme d'état d'une automobile :



Exercice

- Représenter par un diagramme d'états les états que peut prendre un individu du point de vue de l'INSEE :
  - vivant,
  - décédé,
  - mineur,
  - majeur,
  - célibataire,
  - marié,
  - veuf,
  - divorcé.

Les diagrammes un par un

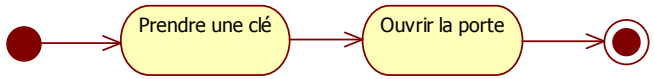
Le diagramme d'activités

Diagramme d'activités

- Permet la modélisation dynamique d'un système.
- Mettre en l'accent sur les enchaînements et les conditions pour exécuter et coordonner des actions.
- Utilisation :
  - Décrire un processus métier,
  - Décrire un cas d'utilisation,
  - Décrire un algorithme ou une méthode.

Éléments de base du diagramme

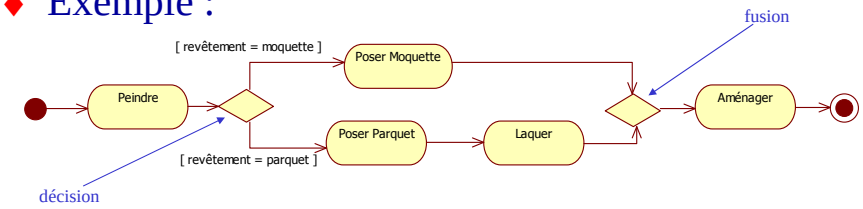
- Une **action** modélise une étape dans l'exécution d'un flot (algorithme ou processus).
- Les actions sont reliées par des **transitions**, en général automatiques.
- Exemple :



- Noter la similitude avec le diagramme d'états !

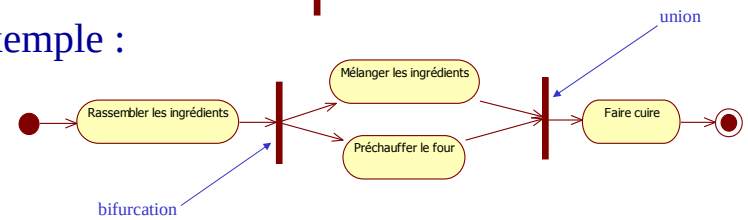
Décisions et fusions

- Une **décision** modélise un choix entre plusieurs flots.
- Une **fusion** rassemble plusieurs flots alternatifs.
- Équivalent d'un OU dans un texte.
- Notation : ◇
- Exemple :



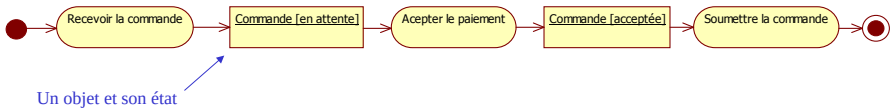
Bifurcation et union

- Une **bifurcation** (ou débranchement) modélise une séparation d'un flot en plusieurs flots concurrents.
- Une **union** (ou jointure) synchronise des flots multiples.
- Équivalent d'un ET dans un texte.
- Notation : ——— ou |
- Exemple :



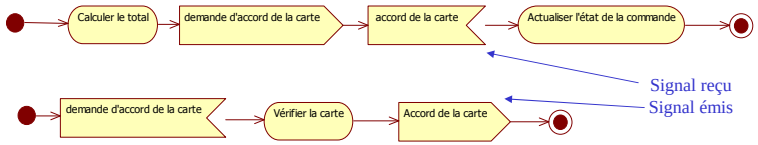
Objets

- Il est possible de faire apparaître des **objets** dans le diagramme.
- L'**état** de l'objet peut être indiqué si l'objet change d'état durant l'exécution du diagramme.
- Permet de montrer la circulation et l'utilisation d'objets au sein de l'activité.
- Exemple :



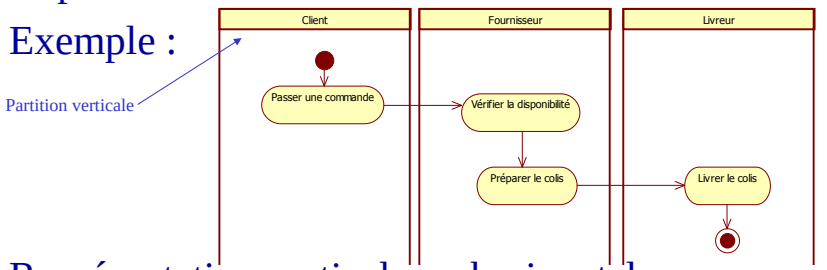
Signaux

- Les **signaux** permettent de représenter des interactions avec des participants externes (acteurs, systèmes, autres activités,…).
- Un **signal reçu** déclenche une action du diagramme. Un **signal émis** est un signal envoyé à un participant externe.
- Exemple : interactions entre 2 diagrammes d'activités:



Partitions

- Les **partitions** (ou couloirs) sont utilisées pour structurer le diagramme en opérant des regroupements.
- Souvent utilisées pour représenter les acteurs ou les responsabilités des actions.
- Exemple :



- Représentation verticale ou horizontale.

Utilisation du diagramme d'activité

- Le diagramme d'activité est plus facile à lire et à comprendre que les autres diagrammes. Il permet donc de communiquer avec des acteurs « non techniques ».
- Il s'utilise à différents niveaux :
  - Modélisation d'un processus métier (BPM),
  - Modélisation d'un cas d'utilisation,
  - Modélisation du comportement interne d'une méthode (algorithme).

Exercice

- Réaliser un diagramme d'activités pour décrire le passage au guichet d'enregistrement d'un vol dans un aéroport.

1. Lorsqu'un passager se présente au guichet d'enregistrement, l'hôtesse lui demande son billet et une pièce d'identité.
2. Elle vérifie alors que le billet correspond bien au vol en cours d'enregistrement, et la correspondance entre le nom écrit sur le billet et le nom écrit sur la pièce d'identité.
3. En cas de problème, le passager peut être réorienté vers l'agence de voyage qui se trouve dans l'aéroport.
4. Si tout est correct, l'hôtesse demande les préférences au passager (placement dans l'avion, …).
5. Puis, si le passager est muni de bagages, elle prend les bagages et édite un reçu à destination du passager.
6. En parallèle, la carte d'enregistrement est imprimée.
7. Enfin la carte d'enregistrement est remise au passager, ainsi qu'une documentation sur la compagnie aérienne.

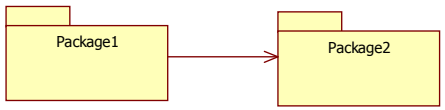
Le diagramme de paquetages

Paquetage

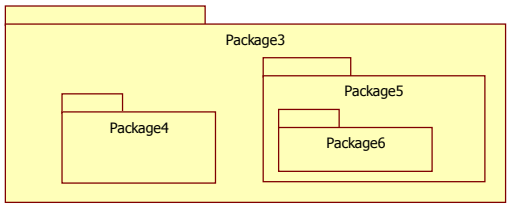
- Problème : un système peut contenir plusieurs centaines de classes. Comment organiser ces classes ?
- Solution : rassembler les classes dans des groupes logiques.
- Le **paquetage** est un regroupement d'éléments UML, par exemple un regroupement de classes.
- Un paquetage peut aussi être un regroupement d'autres éléments comme les cas d'utilisation.

Diagramme de paquetages

- Le diagramme de paquetages permet de représenter les paquetages et leurs dépendances.
- Notation :



- Il est possible d'imbriquer des paquetages dans d'autres paquetages :



Espaces de noms

- Pour qu'un élément utilise un élément d'un autre paquetage, il doit spécifier où il se trouve, en précisant son nom complet, sous la forme :

  nomPaquetage::nomClasse

- Le nom complet permet de distinguer 2 classes de même nom mais de paquetages différents.
- Le nom complet permet de rendre unique un élément.

Visibilité d'un élément

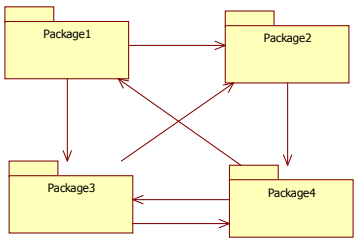
- Les éléments peuvent avoir une visibilité publique ou privée:
- **Visibilité publique** : visible et accessible de l'extérieur du paquetage.
- Notation : +
- **Visibilité privée** : non visible et non disponible de l'extérieur du paquetage.
- Notation : -

Architecture logique d'un système

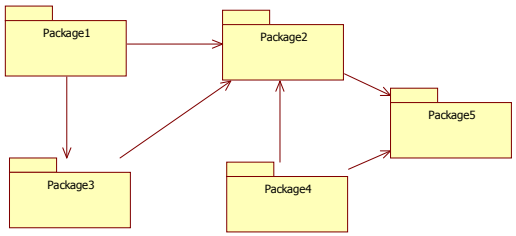
- Les **dépendances** entre paquetages sont issues des dépendances entre éléments des paquetages.
- De trop nombreuses dépendances entre paquetages conduisent à un système fragile et peu évolutif.
- Objectif de l'**architecture** : limiter les dépendances entre paquetages.

Architecture logique d'un système

- Exemples :
  - Mauvais !



  - Beaucoup mieux !



Les conseils de l'architecte

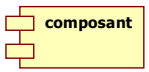
- Faire des regroupements en assurant une cohésion forte à l'intérieur des paquetages et un couplage faible entre les paquetages.
- Éviter les dépendances circulaires.
- Aller dans le sens de la stabilité.

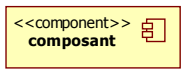
Le diagramme de composants

Diagramme de composants

- Le diagramme de composants présente l'**architecture applicative** statique du système, alors que le diagramme de classes présente la structure logique du système.
- Il permet de montrer les **composants** du système et leurs **dépendances** dans leur environnement d'implémentation.
- Les composants permettent d'organiser un système en « morceaux » logiciels.

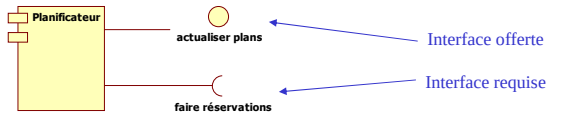
Composant

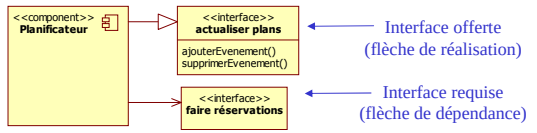
- Un composant est un élément encapsulé, réutilisable et remplaçable du logiciel.
- Ce sont des « briques de construction », qui permettent en les combinant de réaliser un système.
- Notation : 

ou 

- Exemples : enregistreur d'évènements, éditeur PDF, convertisseur de format, …

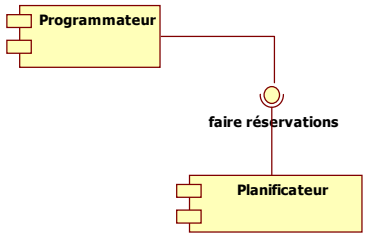
Interfaces de composants

- Les interfaces d'un composant définissent les comportements offerts à d'autres composants (interfaces offertes) ou attendus d'autres composants (interfaces requises).
- Notation : 

- Autre notation possible, avec affichage des opérations offertes par les interfaces : 

Dépendances entre composants

- Les dépendances entre composants peuvent être matérialisées par les interfaces :



- Ou bien par le lien de dépendance standard UML :

Conception d'un composant

- Un composant doit fournir des **services** bien précis, qui doivent être cohérents et génériques pour être réutilisables dans différents contextes.
- Le comportement interne, réalisé par un ensemble de classes, est masqué aux autres composants ; seules les interfaces sont visibles.
- On peut substituer un composant à un autre si les interfaces sont identiques.
- Le « découpage » d'un système en composants se fait en recherchant les éléments qui sont employés fréquemment dans le système.

Vues d'un composant

- **Vue boîte noire** : montre l'aspect extérieur d'un composant : interfaces fournies et requises, liens avec d'autres composants.
  → pour se concentrer sur les problèmes d'architecture applicative générale du système.
- **Vue boîte blanche** : montre les classes, les interfaces et les autres composants inclus.
  → Pour montrer comment un composant rend ses services au travers des classes qu'il utilise.

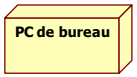
Les diagrammes un par un

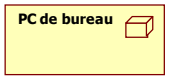
Le diagramme de déploiement

Diagramme de déploiement

- Le diagramme de déploiement montre la **configuration physique** des différents matériels qui participent à l'exécution du système, et montre la répartition des composants sur ces matériels.
- Utile :
  - dans les premières phases du projet pour montrer une esquisse grossière du système,
  - à la fin du développement pour servir de base au guide d'installation par exemple.

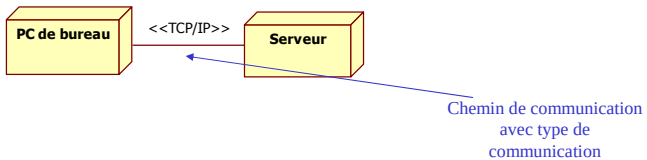
Noeud

- Un **nœud** est une ressource matérielle ou logicielle qui peut héberger des logiciels ou des fichiers associés.
- Notation : 

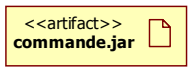
ou 

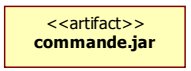
- Exemples :
  - nœuds matériels : serveur, PC de bureau, disque dur
  - nœuds logiciels : système d'exploitation, serveur web

Chemin de communication

- Un nœud peut avoir besoin de communiquer avec d'autres nœuds.
- Les **chemins de communication** sont utilisés pour spécifier que des nœuds communiquent les uns avec les autres à l'exécution.
- Notation : 

Artefact

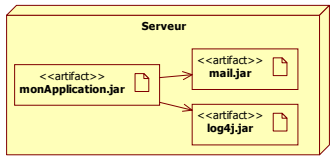
- Un **artéfact** est fichier physique qui s'exécute ou est utilisé par le système.
- Notation : 

ou 

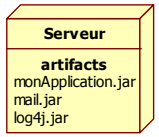
- Exemples :
  - fichiers exécutables (.exe, .jar),
  - fichiers de bibliothèque (.dll),
  - fichiers source (.java),
  - fichiers de configuration (.xml, .properties).

Artefacts, nœuds et composants

- Un **artéfact** est déployé sur un **nœud** : il réside sur (ou est installé sur) le nœud.
- Un artefact peut dépendre d'autres artefacts pour s'exécuter.
- Exemple :



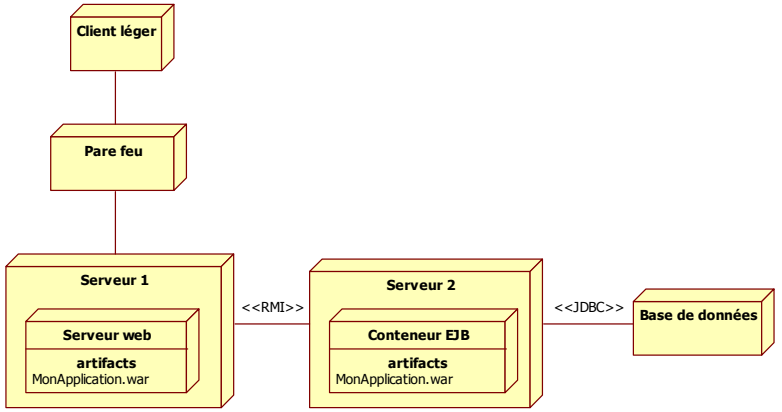
ou



- Un artefact peut être la représentation physique d'un **composant**.
- Exemple :

Exemple

- Diagramme de déploiement d'une architecture J2EE :



Compléments

Université d'Évry-Val d'Essonne

V

Compléments

Compléments

OCL

Object Constraint Language

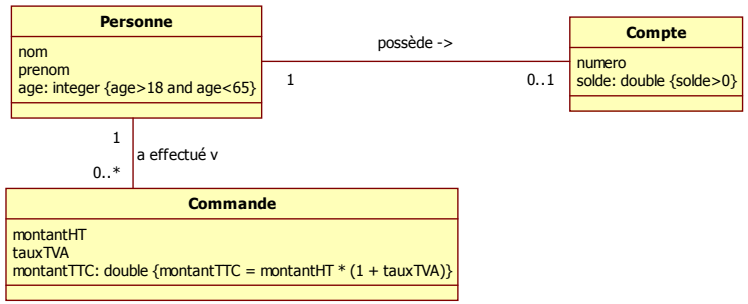
- Le **constat** : on ne peut pas tout exprimer avec UML. Il convient d'ajouter des spécifications en langage naturel. Or le langage naturel peut être ambigu.
- La **solution** : pour remédier à cela, l'OMG a décidé de définir un langage formel : OCL, pour l'expression des contraintes sur les éléments d'un modèle UML.

Object Constraint Language

- OCL comme UML est indépendant du langage d'implémentation.
- Une contrainte décrite en OCL peut s'appliquer à n'importe quel élément du modèle.
- Notation : {contrainte} après l'élément concerné, ou dans une note, ou en dehors du modèle.

Exemple

- Contraintes sur les attributs d'un diagramme de classes :



Types et opérateurs OCL

- OCL définit 4 types :
  - Booléen : true, false,
  - Entier : 1, 2, 322, -10, …
  - Réel : 2.34, 4.5, …
  - Chaîne de caractère : « Hello world ».
- OCL possède des opérateurs :
  - Arithmétiques : +, -, *, /,
  - Arithmétiques avancés : abs(), max(), min(),
  - Comparaisons : <, ≤, >, ≥, =, <>,
  - Booléens : and, or, xor, not,
  - Chaînes de caractères : concat(), size(), substring().

Contexte

- Il est possible d'écrire des contraintes qui ne sont pas physiquement rattachées à des éléments du modèle. Il faut alors indiquer le nom d'une classe.
- Notation : **context** nomClasse
- On ajoute le mot-clé **inv:** pour indiquer que la contrainte qui suit doit être respectée en permanence.
- Exemple :

```
context Compte
inv: solde>0
context Personne
inv: age>18 and age<65
```


Contraintes sur les méthodes

- Deux types de contraintes sur les méthodes :
  - **Pré condition** : vérifiée avant l'exécution de la méthode. Souvent utilisée pour vérifier les paramètres d'entrée de la méthode.

    Notation : pre:
  - **Post condition** : vérifiée après l'exécution de la méthode. Souvent utilisée pour décrire comment des valeurs ont été modifiée par la méthode.

    Notation : post:

Contraintes sur les méthodes

- On précise les contraintes sur une méthode en indiquant le contexte de la méthode :

  Context Classe::méthode(paramètres) : typeRetour
- Exemple :
  - L'opération qui consiste à créditer un compte n'accepte que les montants positifs,
  - Après exécution de l'opération, le solde doit avoir pour valeur la somme du solde avant l'appel de la méthode et du montant passé en paramètre.

  ⇨Expressions OCL correspondantes :

```
Context Compte::crediter(montant)
Pre: montant>0
Post: solde = solde@pre + montant
```


Compléments

Correspondance UML - Java

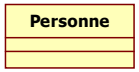
UML - Java

- UML est un langage de **modélisation**
- Java est un langage de **programmation**

- Il n'y a pas toujours de correspondance directe entre un concept UML et une implémentation dans un langage de programmation.
- Cependant il existe généralement une façon naturelle de traduire un concept UML en Java.

UML – Java : Classes

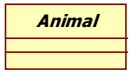
- UML :



- Java :

```
public class Personne {
        …
}
```

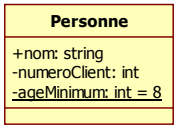
Classe abstraite




```
abstract public class Animal {
        …
}
```


UML – Java : Attributs

- UML :

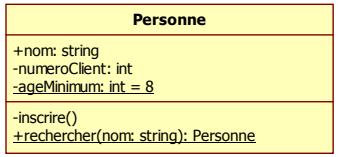


- Java :

```
public class Personne {
        public string nom;
        private int numeroClient;
        private static int ageMinimum=8;
}
```


UML – Java : Opérations

- UML :



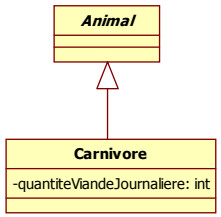
- Java :

```
public class Personne {
        …

        private void inscrire() {
                …
        }
        public static Personne rechercher(string nom) {
                …
        }
}
```

UML – Java : Généralisation

- UML :



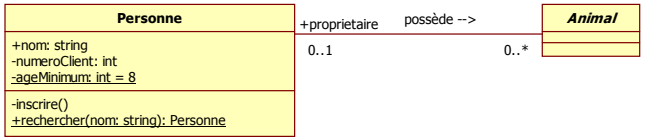
- Java :

```
public class Carnivore extends Animal {
        private int quantiteViandeJournaliere;

}
```


UML – Java : Relations

- UML :



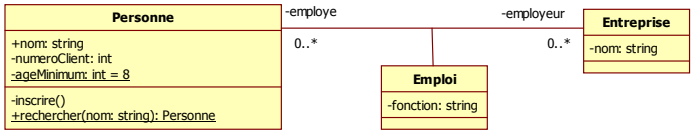
- Java :

```
public class Personne {
        …
        private Animal lesAnimaux[];

}
abstract public class Animal {
        …
        public Personne proprietaire;
}
```


UML – Java : Relations

- UML :

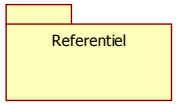


- Java :

```
public class Emploi {
        private string fonction;
        private Personne employe;
        private Entreprise employeur;
}
```


UML – Java : Paquetages

- UML :



- Java :

```
package Referentiel;
        …
```

UML vs MERISE

- Peut on comparer UML et MERISE ?
- UML est un langage de modélisation
- MERISE propose un langage ET une méthode
- Le passage de MERISE à UML (et l'inverse) n'est donc pas uniquement d'ordre syntaxique.

- La comparaison la plus connue entre les deux langages : modéliser les données
- MERISE : approche disjointe et parallèle des données et des traitements (modèles spécifiques).
- UML : approche objet qui intègre données et traitements (approche composants).

- La démarche sous jacente à UML est :
  - centrée sur les besoins utilisateurs (cas d'utilisation),
  - itérative et incrémentale,
  - le même concept d'objet peut parcourir tout le cycle d'abstraction.
- La démarche proposée par MERISE est :
  - proche du cycle de développement en cascade (on procède une phase après l'autre),
  - conduite par les différents modèles proposés.