
# Cours « Méthodes de conception de SI »

Tarak Chaari

**Maître-assistant à l'ISECS**
**Membre de l'unité de recherche ReDCAD**

**tarak.chaari@redcad.org**

# Présentation générale du cours

## Le nom du cours

- Méthodes de conception des Systèmes d'information (MCSI)

## Volume horaire

- 21 heures
- Cours + TD

## Objectifs

- Avoir une idée sur le processus d'ingénierie des logiciels d'une façon générale
- Maitriser le langage de modélisation (UML)

# Plan

1. Introduction au génie Logiciel
2. Importance de la modélisation
3. Modélisation orientée Objet avec UML
4. Les diagrammes UML
5. Exercices et exemples

# Introduction au Génie Logiciel

# Logiciel & Système d'information

- Un logiciel est l'ensemble des éléments informatiques qui permettent d'assurer une tâche ou une fonction (exemple : logiciel de comptabilité, logiciel de gestion des prêts)

- Ensemble de programmes liés entre eux destiné à remplir une certaine mission, généralement assez complexe

- Système d'information: Système constitué de l'équipement, des procédures, des ressources humaines, ainsi que des données qui y sont traitées, et dont le but est de fournir de l'information.

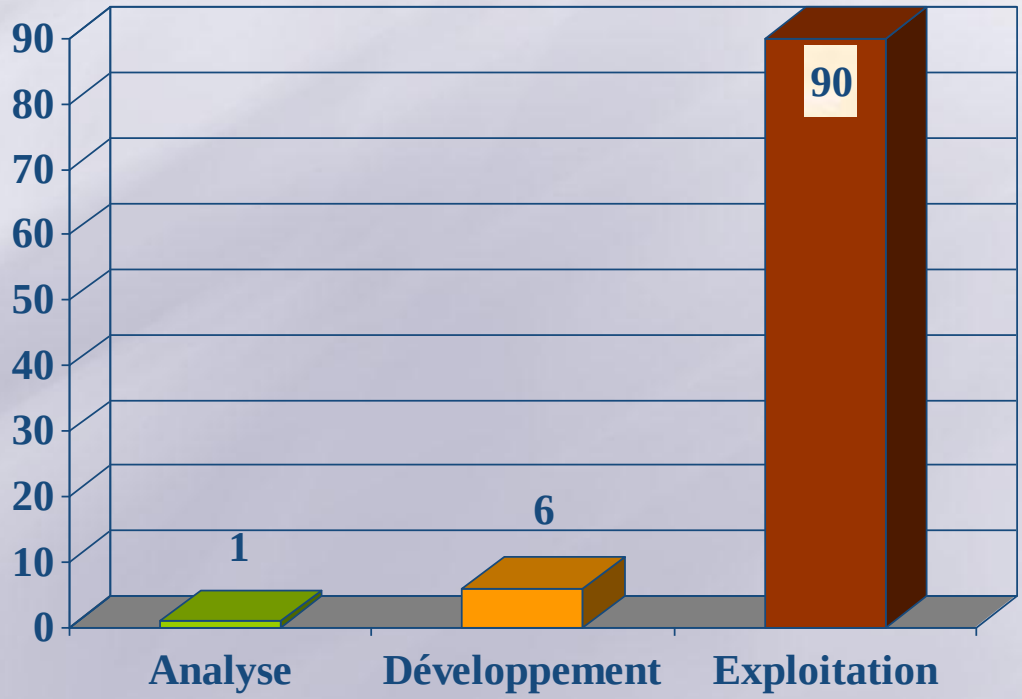
# Le logiciel, un produit pas comme les autres

- Pas de fabrication en série
- Evaluation et contrôle de qualité
- Analyse, conception développement et validation plutôt que fabrication
- Le produit « logiciel » semble flexible
  - modifier un logiciel <> modifier un produit mécanique
- Mais le software, ce n'est pas aussi « soft » que ça !!!

# Coté acheteur

- Une définition insuffisante des besoins est une cause majeure de production d'un logiciel de mauvaise qualité.

- Les coûts pour un changement du logiciel augmentent de façon exponentielle dans les dernières phases du développement.



# Coté développeur

## Les erreurs:

- Le développement est terminé quand le logiciel fonctionne.
- Tant qu'un logiciel ne fonctionne pas, il n'y a pas moyen d'en mesurer la qualité
- Ce qui compte c'est le livrable: Le logiciel

## La réalité

- 50% à 70% de l'effort consacré a lieu après la livraison.
- Les revues et les contrôles de logiciel au cours de développement sont très importantes
- La documentation pour l'utilisateur et pour les développeurs

# Coté gestionnaire

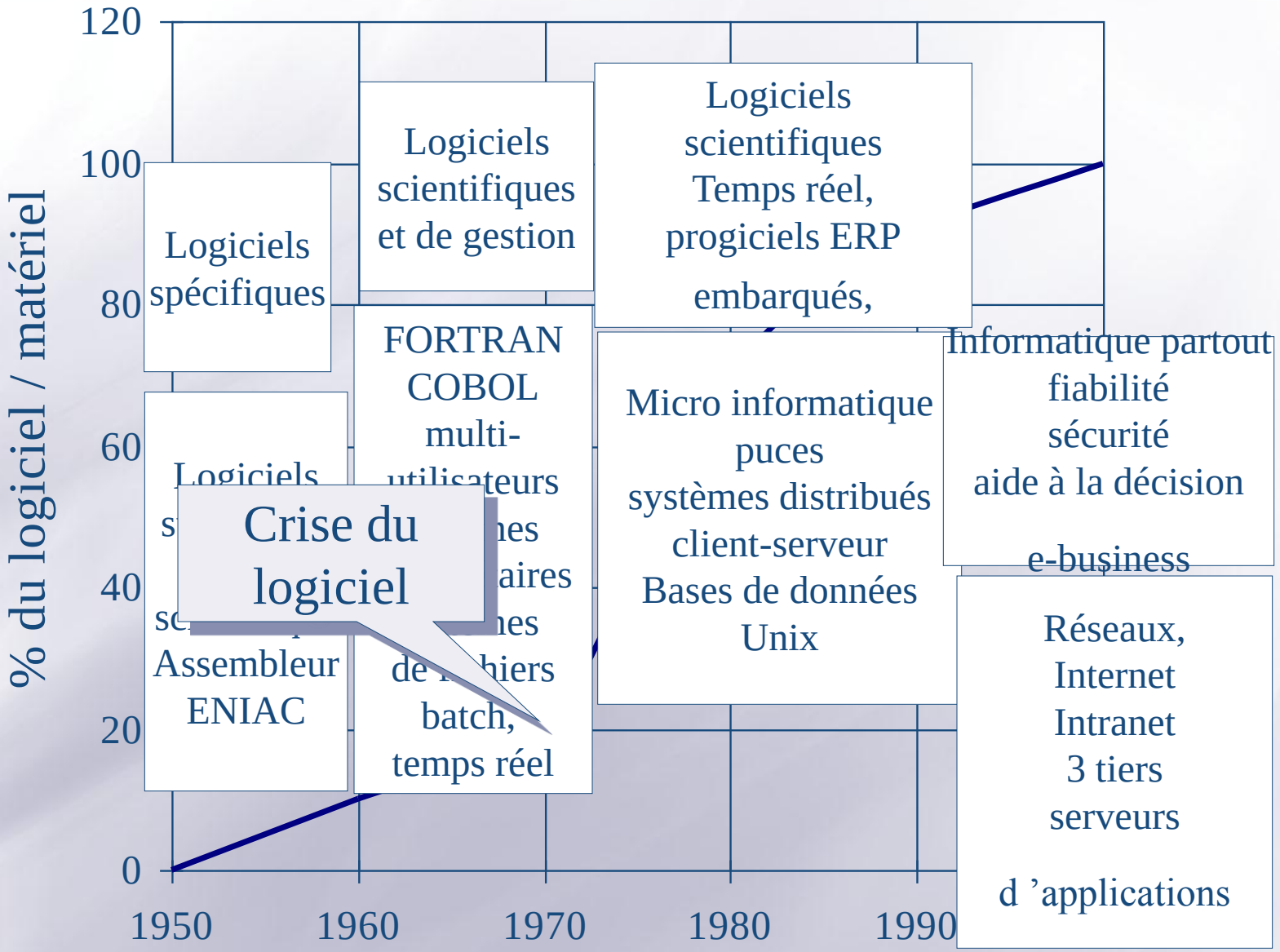
## L’erreur

- Il suffira d’ajouter des programmeurs pour rattraper le retard d’un projet.

## Conséquences

- beaucoup de temps perdu pour intégrer les nouveaux dans le projet
- incompréhensions
- incohérences
- temps perdu dans la répartition de tâches (temps de communication)

# L’explosion du logiciel



# Catastrophes et autres tracas liés aux erreurs dans les logiciels

- la sonde Mariner vers Vénus s'est perdue dans l'espace à cause d'une erreur de programme FORTRAN
- en 1972, lors d'une expérience météorologique en France 72 ballons contenant des instruments de mesure furent détruits à cause d'un défaut dans le logiciel
- en 1981, le premier lancement de la navette spatiale a été retardé de deux jours à cause d'un problème logiciel. La navette a d'ailleurs été lancé sans que l'on ait localisé exactement le problème (mais les symptôme étaient bien délimités)
- la SNCF a rencontré des difficultés importantes pour la mise en service du système Socrate
- L'explosion d'Ariane 5, le 4 juin 1996, qui a coûté un demi milliard de dollars (non assuré !), est due à une faute logicielle d'une composante dont le fonctionnement n'était pas indispensable durant le vol

# Un autre exemple…

- General Motors décide de moderniser ses usines
- Objectif : devancer les japonais
- Moyen : 250 robots + 60 véhicules de transport de pièces
- De gros problèmes surviennent
- Arrêt des chaînes
- Une étude (ultérieure) montre que les problèmes sont ailleurs

# Bill Gates lors d’un salon informatique

- **comparant l'industrie informatique avec l'industrie automobile a dit :**

  **« Si Général Motors (GM) avait suivi la même progression technologique que l'industrie informatique, nous conduirions aujourd'hui des autos coûtant 1 Dollar. Elles feraient 1 000 kilomètres avec un litre d'essence. »**

# Réponse de Général Motors

« Si Général Motors avait développé sa technologie de façon similaire à ce qu'a fait Microsoft :

- Les témoins d'huile, de température et de batterie seraient remplacés par un unique témoin « Défaillance Générale »
- Parfois une auto quitterait l'autoroute sans raison connue. Il faudrait l'accepter, et simplement redémarrer l'auto pour reprendre la route (si vous êtes encore vivant)
- L'airbag demanderait « Etes-vous sur ? » avant de s'ouvrir
- Macintosh développerait des voitures fonctionnant à l'énergie solaire, fiables, cinq fois plus rapides et deux fois plus légères. Mais elles ne pourraient emprunter que 5% des routes
- A chaque fois que GM sortirait un nouveau modèle, les conducteurs devraient réapprendre à conduire car aucune des commandes ne fonctionnerait exactement comme dans les modèles précédents.
- Enfin, il faudrait appuyer sur le bouton « Démarrer » pour stopper le moteur

# Symptômes de la crise

- Les logiciels ne correspondent souvent pas aux besoins des utilisateurs
- Ils contiennent trop d'erreurs
- Leur coût est rarement prévisible et souvent prohibitifs
- Leurs délais de réalisation sont souvent dépassés
- Les logiciels sont rarement portables
- Leur maintenance est complexe et coûteuse

# Le logiciel de nos jours

- **Taille et complexité des logiciels :**
  - **Complexité fonctionnelle**
    - **mémoriser et stocker l'information : mais en plus traiter de façon sophistiquée pour l'aide à la décision (*Entrepôt de données*).**
    - **Logiciels développés séparément et avec des démarches différentes et appelés à être interfacés pour les besoins de l'Entreprise.**
  - **Evolutions technologiques permanentes**
  - **Complexité architecturale : Client/serveur, Intranet, Corba (Common Object Request Broker Architecture), Systèmes distribués**
- **Généricité et réutilisabilité**

# SOLUTION: LE GENIE LOGICIEL

- Maîtriser les coûts
- Maîtriser les délais
- Maîtriser la qualité
- Maîtriser le processus de production de logiciels

# Cloisonnement des problèmes

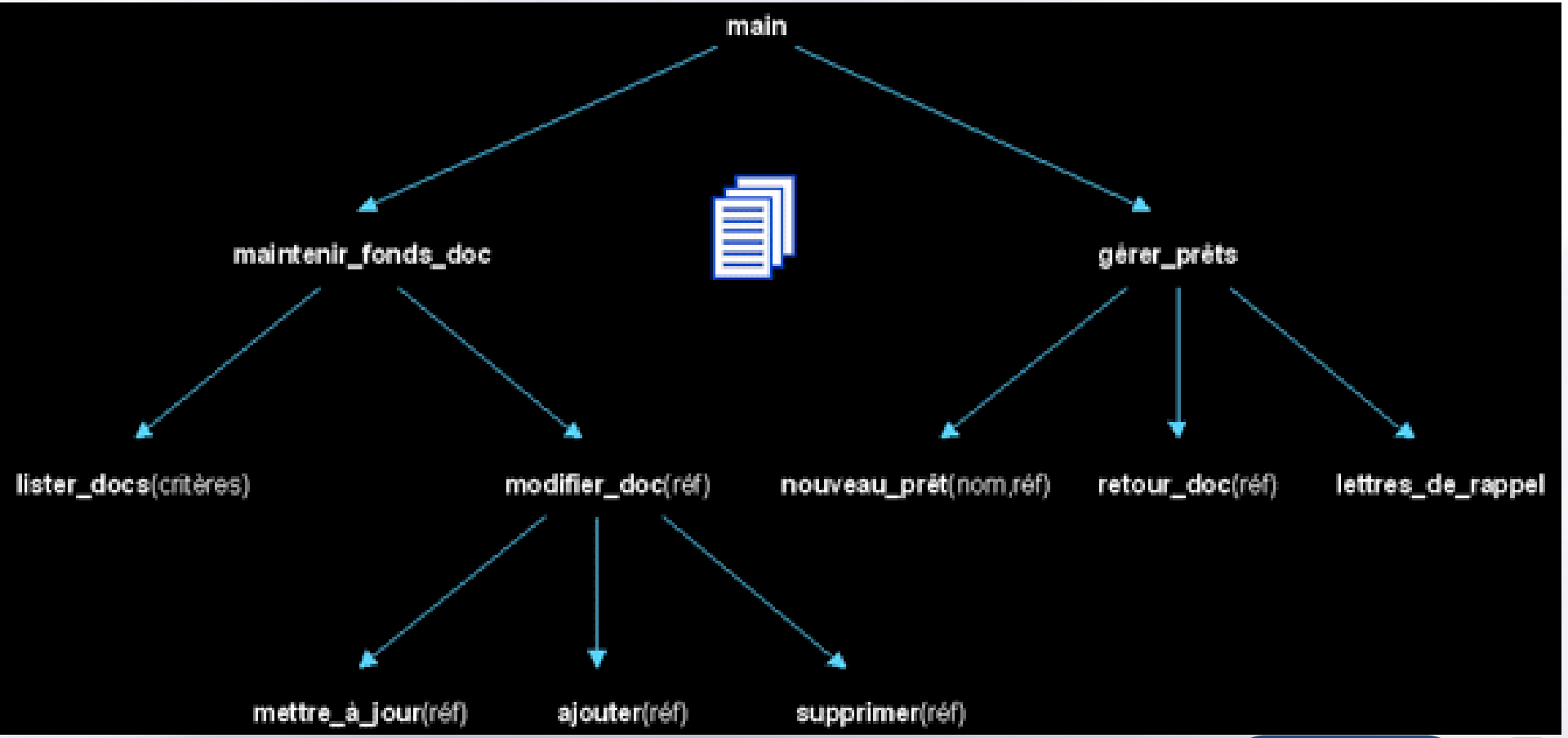
- Est-ce possible ? — Faisabilité
- Quand et qui ? — Planification
- Quoi ? — Analyse
- Comment ? — Conception
- Est-ce le produit correct ? — Validation
- Est-il fait correctement ? — Vérification

# En rodage !!!

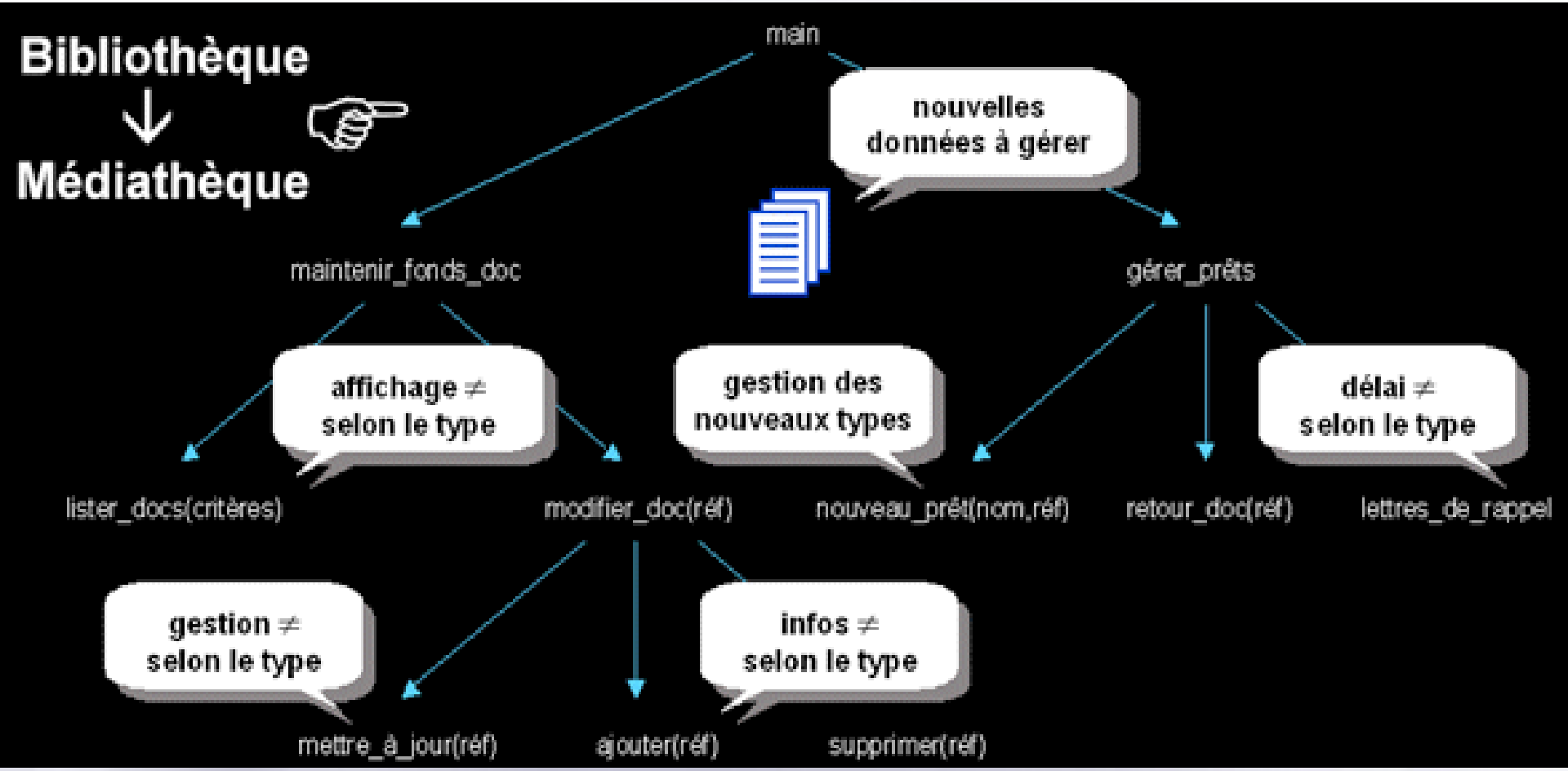
## Introduction générale et rappels

# C’est quoi un système d’information?

- **Un système d’information:**
  - un ensemble de fonctions bien organisées
  - des données (entités manipulées)



# STOP! il y a un problème…

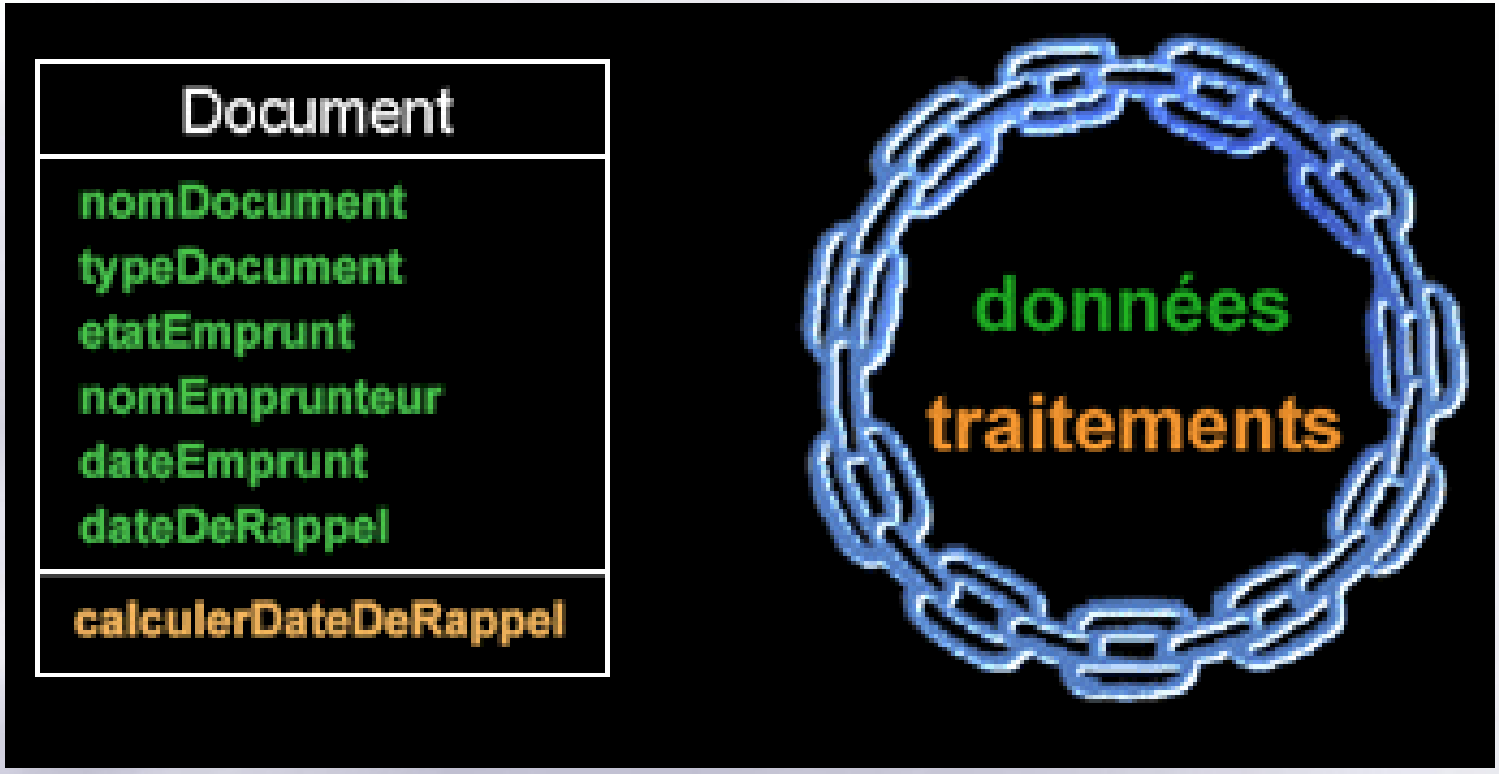



- **Problème:** .....
- **Cause:** .....

# Approches fonctionnelles

- Séparer les données des traitements
- Se focalise sur les fonctions (gérer des prêt, ajouter des documents…)
- Découper les fonctions en sous – fonctions
- Avantages: ……
- Inconvénients: ……...

# La solution: l'approche objet



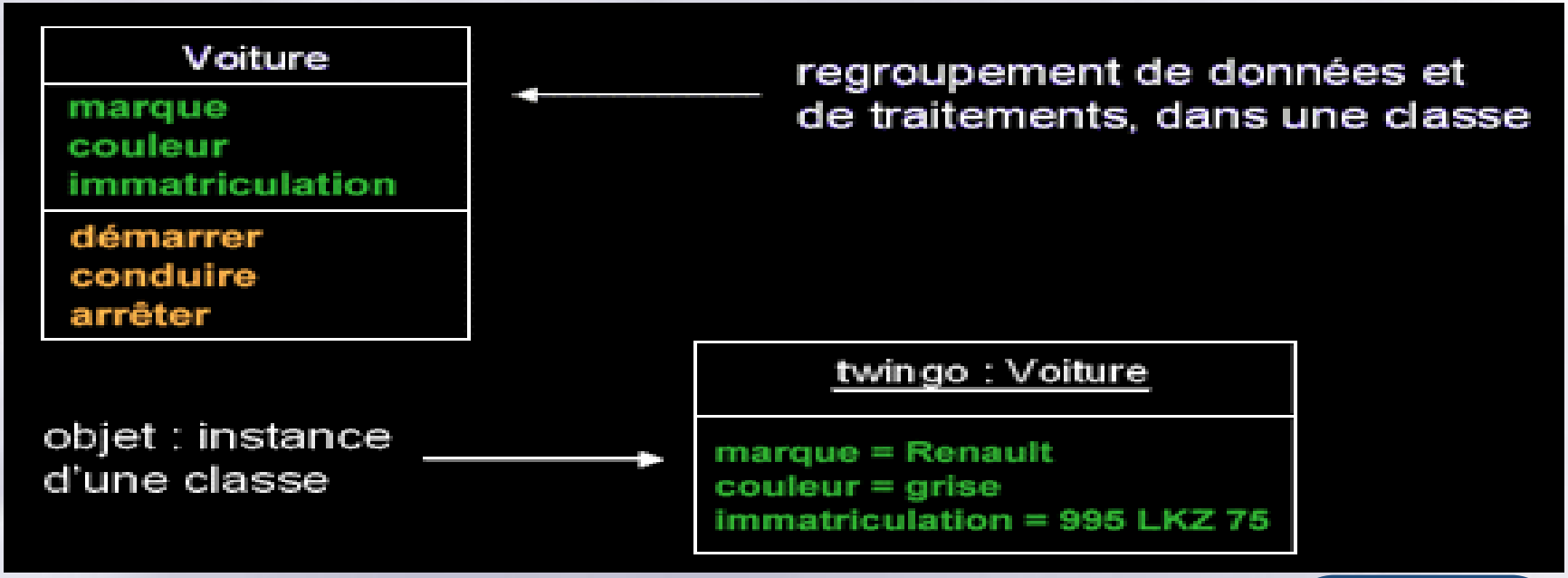
# Vision objet d'un système d'information (1)

- Un SI = un ensemble d'objets qui collaborent entre eux

- Un objet représente une entité du système qui est caractérisée par:
  - Des frontières précises
  - Une identité (ou référence)
  - Un ensemble d'attributs (propriétés) décrivant son état
  - Un ensemble de méthodes (opérations) définissant son comportement

# Vision objet d'un système d'information (2)

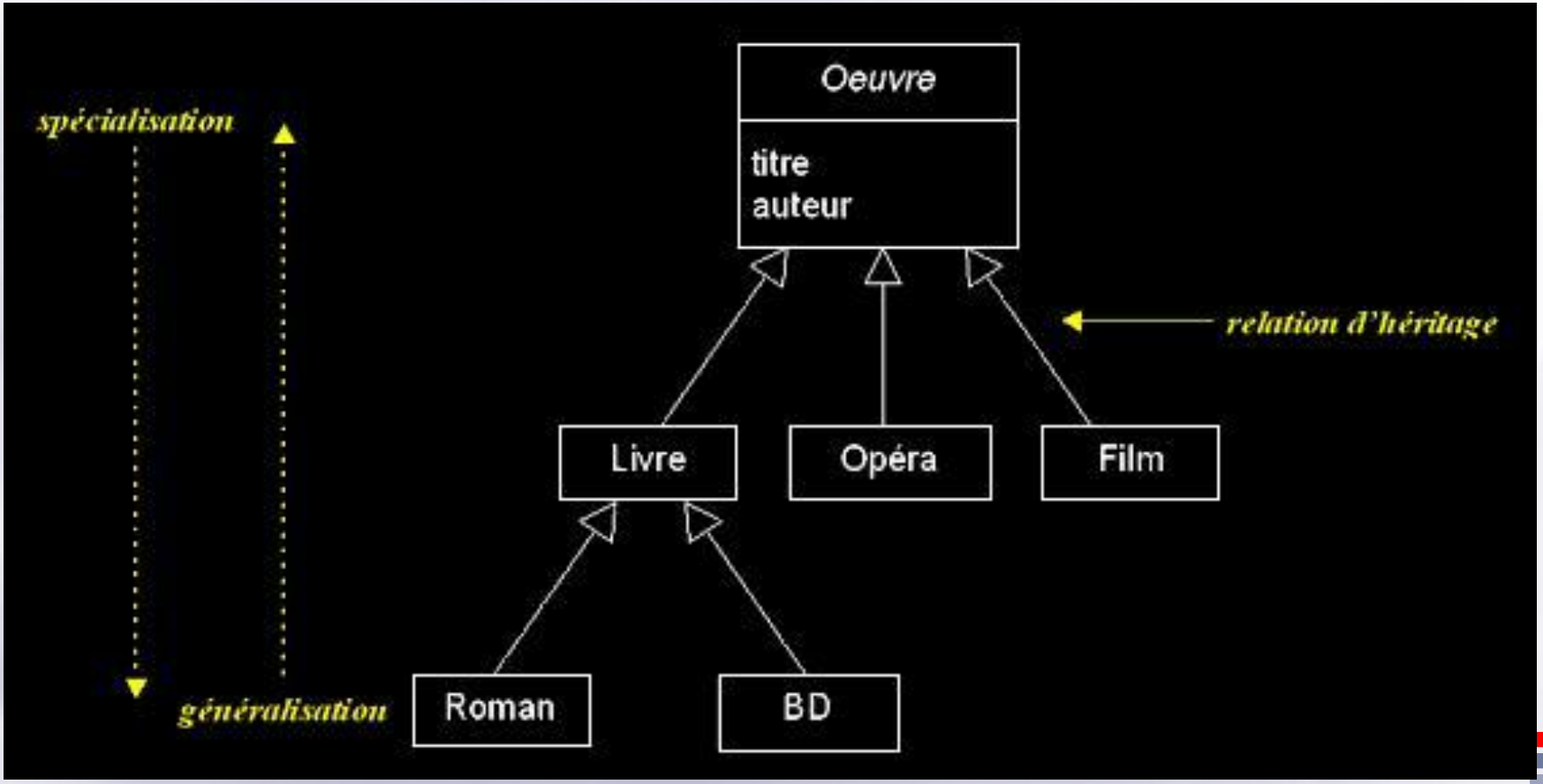
- Un objet est une instance de classe (une occurrence d'un type abstrait)
- Une **classe** est un type de données abstrait(modèle) , caractérisé par des propriétés (attributs et méthodes) communes à des objets et permettant de créer des objets possédant ces propriétés.



# Vision objet d'un système d'information (3)

## Héritage

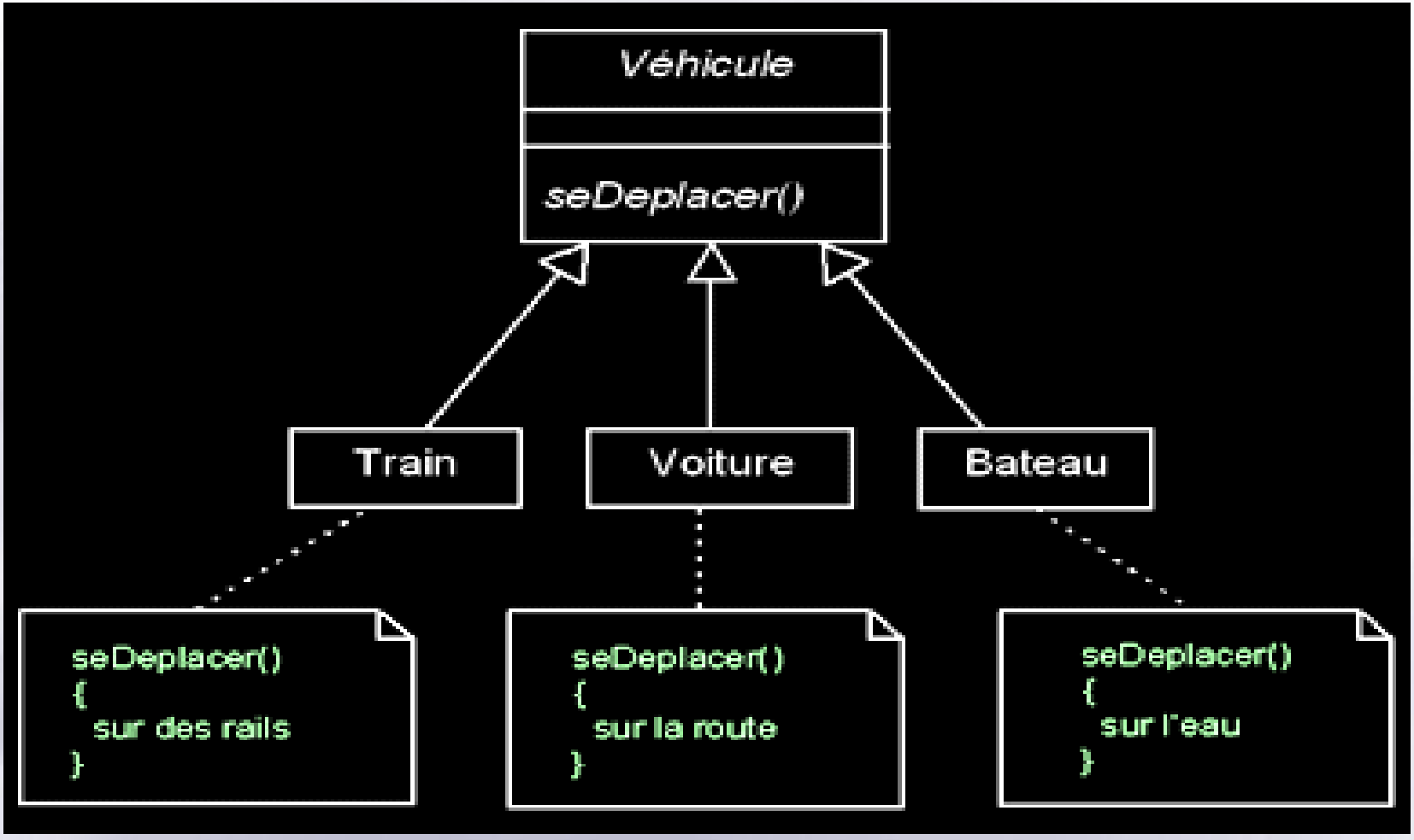
- Transmission de propriétés (attributs et méthodes) d'une classe à une sous classe d'objets



# Vision objet d'un système d'information (4)

## Polymorphisme

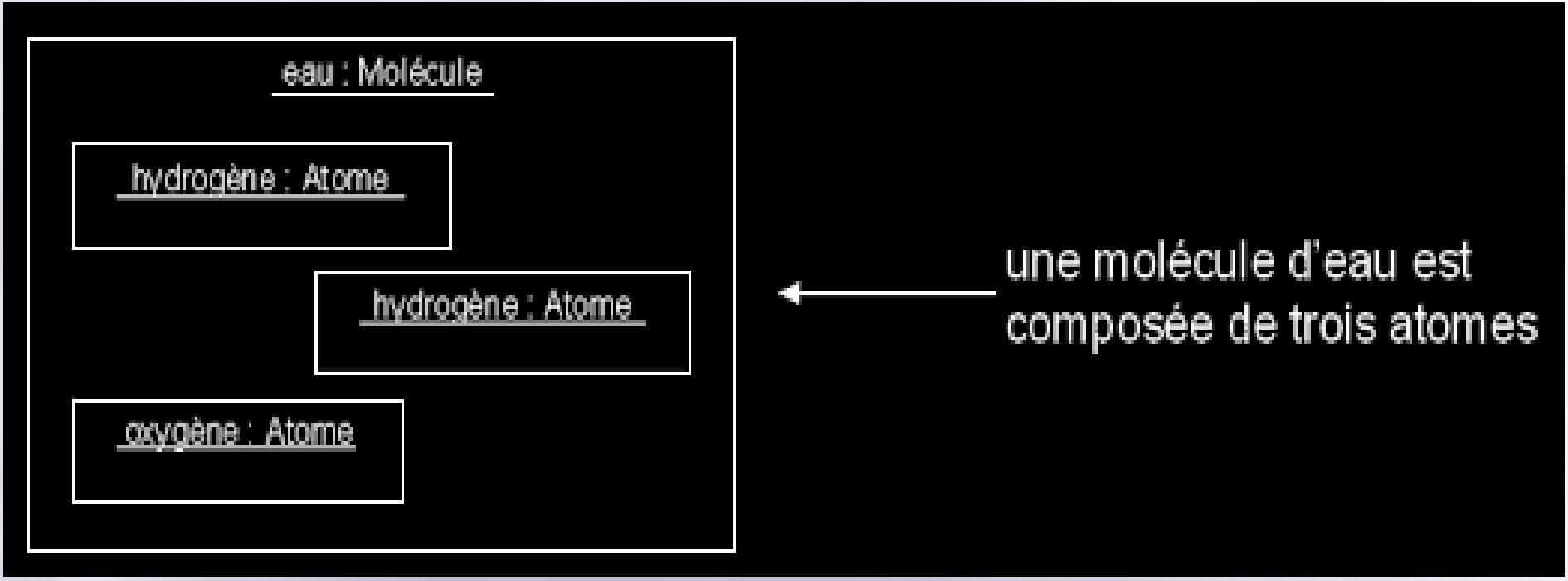
- Factorisation de comportement (méthodes) commun d'objets



# Vision objet d'un système d'information (5)

## L'agrégation

- Une relation d'agrégation permet de définir des objets composés d'autres objets.
- L'agrégation permet d'assembler des objets de base, afin de construire des objets plus complexes.



# Résumé des concepts fondateurs de l'approche objet (1)

- L'héritage est un mécanisme de transmission des propriétés d'une classe (ses attributs et méthodes) vers une sous-classe.

- Une classe peut être spécialisée en d'autres classes, afin d'y ajouter des caractéristiques spécifiques ou d'en adapter certaines.

- Plusieurs classes peuvent être généralisées en une classe qui les factorise afin de regrouper les caractéristiques communes d'un ensemble de classes.

# Résumé des concepts fondateurs de l'approche objet (2)

- La spécialisation et la généralisation permettent de construire des hiérarchies de classes. L'héritage peut être simple ou multiple.
- L'héritage évite la duplication et encourage la réutilisation.
- Le polymorphisme représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes.
- Le polymorphisme augmente la généricité du code.

# L’approche objet : une solution parfaites?

- L'approche objet est moins intuitive que l'approche fonctionnelle
  - Quel moyen utiliser pour faciliter l’analyse objet?
  - Quels critères identifient une conception objet pertinente ?
- L'application des concepts objets nécessite une grande rigueur
  - Le vocabulaire présente des d'ambiguïtés et des d’incompréhension
  - Comment décrire la structure objet d'un système de manière pertinente?
  - Comment décrire l’interaction entre ces objets de manière précise?

# Remèdes aux inconvénients de l’approche objet

- Un langage (ou modele) pour exprimer les concepts objet qu'on utilise, afin de pouvoir :
  - Représenter des concepts abstraits (graphiquement par exemple)
  - Limiter les ambiguïtés (parler un langage commun)
  - Faciliter l'analyse (simplifier la comparaison et l'évaluation de solutions)
- Une démarche d'analyse et de conception objet pour :
  - Ne pas effectuer une analyse fonctionnelle et se contenter d'une implémentation objet, mais penser objet dès le départ
  - Définir les vues qui permettent de couvrir tous les aspects d'un système, avec des concepts objets

# Dans quel cas modéliser et analyser?

- **Construire**
  - **une niche: c'est facile**
  - **une maison: c'est un peu plus compliqué**
  - **un immeuble: bon là, comment s'y prendre?**

- **Nombreux fabricants de logiciels veulent construire « des immeubles » mais abordent le problème comme d'ils devaient bricoler une niche**

- ***Nous construisons des modèles pour les systèmes complexes parce que nous ne sommes pas en mesure d'appréhender de tels systèmes dans leur intégralité.***

  Le guide de l'utilisateur UML, Grady Booch, James Rumbaugh et Ivar Jacobson Eyrolles, Octobre 20002.

# Mais pourquoi analyser et modéliser?

- **Pour éviter les résultats négatifs suivant:**
  - Les modules ne fonctionnent pas ensemble
  - Le produit est non conforme avec les besoins
  - Le produit est difficile à maintenir et à faire évoluer
  - Le produit présente beaucoup d'anomalies et de bugs

- **Mais aussi pour**
  - Simplifier la problématique posée par le client
  - « Visualiser » le système
  - Spécifier sa *structure* et son *comportement*
  - Aider à sa construction
  - Travailler en équipe

# Et c’est quoi un modèle?

- Un modèle est une abstraction de la réalité
- Il s'agit d'un processus qui consiste à identifier les caractéristiques intéressantes d'une entité, en vue d'une utilisation précise.
- L'abstraction désigne aussi le résultat de ce processus, c'est-à-dire l'ensemble des caractéristiques essentielles d'une entité, retenues par un observateur.
- Un modèle est une vue subjective mais pertinente de la réalité
- Un modèle définit une frontière entre la réalité et la perspective de l'observateur. Ce n'est pas "la réalité", mais une vue très subjective de la réalité.
- Bien qu'un modèle ne représente pas une réalité absolue, un modèle reflète des aspects importants de la réalité, il en donne donc une vue juste et pertinente.

# Exemples de modèles

- **Modèle météorologique :**
  - **partir de données d'observation (satellite ...), permet de prévoir les conditions climatiques pour les jours à venir.**
- **Modèle économique :**
  - **peut par exemple permettre de simuler l'évolution de cours boursiers en fonction d'hypothèses macro-économiques (évolution du chômage, taux de croissance...).**
- **Modèle démographique :**
  - **définit la composition d'un panel d'une population et son comportement, dans le but de fiabiliser des études statistiques, d'augmenter l'impact de démarches commerciales, etc...**

# Caractéristiques fondamentales d'un modèle

- Le caractère abstrait d'un modèle doit notamment permettre :
  - de faciliter la compréhension du système étudié
    - un modèle réduit la complexité du système étudié.
  - de simuler le système étudié
    - un modèle représente le système étudié et reproduit ses comportements.

- Un modèle réduit (décompose) la réalité, dans le but de disposer d'éléments de travail exploitables par des moyens mathématiques ou informatiques

# Et UML dans tout ça?

- UML: *langage standard* de *modélisation* des systèmes d'information

- UML: langage visuel pour
  - comprendre le système
  - communiquer et travailler à plusieurs
  - aider à spécifier, concevoir et développer un système d'information

  → avec différents modèles et différentes vues

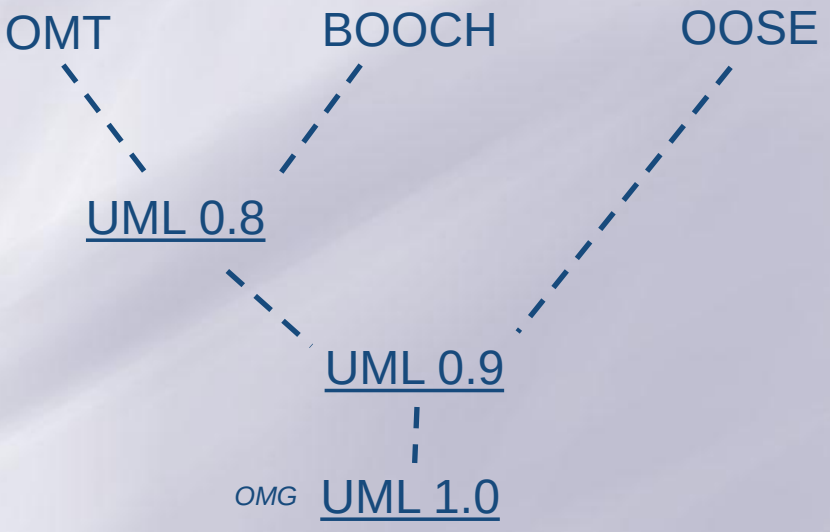
# Et dans quel cas l'utiliser?

- **Systèmes à forte composante logicielle**
  - systèmes d'information pour les entreprises
  - les services bancaires et financiers
  - les télécommunications
  - les transports
  - la défense / l'aérospatiale
  - le commerce de détail
  - l'électronique médicale
  - les services distribués et les applications WEB
- **Pas limité à un domaine précis**

# UML: un peu d'histoire

## 3 fondateurs principaux

- Grady BOOCH (BOOCH)
- James Rumbaugh (OMT)
- Ivar Jacobson (OOSE)




## Historique

- En 1995: Méthode unifiée 0.8 (intègrant les méthodes Booch'93 et OMT)
- En 1995: UML 0.9 (intègrant la méthode OOSE)
- En 1996: UML 1.0 (proposée à l'OMG)
- En 1997: UML 1.1 (standardisée par l'OMG)
- En 1998: UML 1.2
- En 1999: UML 1.3
- En 2000: UML 1.4
- En 2003: UML 1.5
- En 2004: UML 2.0
- …
- En 2007: UML 2.1.2

# Un peu de publicité ☺

- UML est le résultat d'un large consensus (industriels, chercheurs...)
- UML est le fruit d'un travail d'experts reconnus
- UML est issu du terrain
- UML est riche (il couvre toutes les phases d'un cycle de développement)
- UML est ouvert (il est indépendant du domaine d'application et des langages d'implémentation)
- Après l'unification et la standardisation, bientôt l'industrialisation d'UML :
  les outils qui supportent UML se multiplient (GDPro, ObjectTeam, Objecteering, OpenTool, Rational Rose, Rhapsody, STP, Visio, Visual Modeler, WithClass...).

**UML n'est pas une mode, c'est un investissement fiable !**

# Caractéristiques d'UML

- **UML cadre l'analyse objet, en offrant**
  - différentes vues (perspectives) complémentaires d'un système, qui guident l'utilisation des concept objets,
  - plusieurs niveaux d'abstraction, qui permettent de mieux contrôler la complexité dans l'expression des solutions objets.

- **UML est un support de communication**
  - Sa notation graphique permet d'exprimer visuellement une solution objet.
  - L'aspect formel de sa notation limite les ambiguïtés et les incompréhensions.
  - Son aspect visuel facilite la comparaison et l'évaluation de solutions.
  - Son indépendance (par rapport aux langages d'implémentation, domaine d'application, processus...) en font un langage universel.

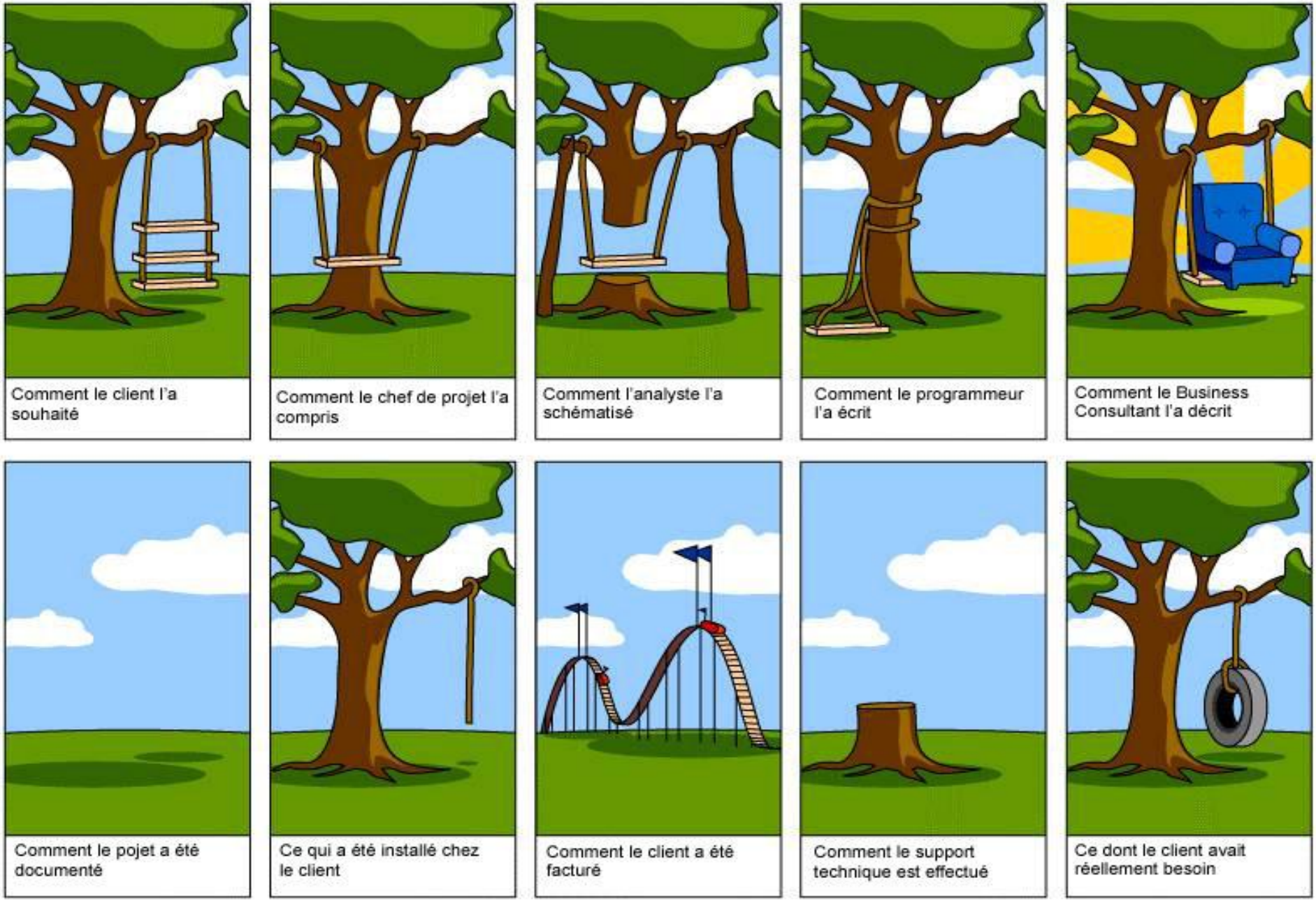
# Alors UML, c'est la solution à tout?

- UML n'est qu'un ensemble de formalismes permettant d'appréhender un problème et de le modéliser
- Un formalisme n'est qu'un outil
- Le succès = savoir utiliser les outils
- Ce n'est pas un algorithme ou une méthode automatique à appliquer
- UML laisse la liberté de « penser » ☺
- IMAGINATION IS MORE IMPORTANT THAN KNOWLEGDE (A. Einstein)

# Et comment l'utiliser alors?

- **Avant tout, une bonne démarche qui permet de :**
  - Bien comprendre les demandes et exigences des utilisateurs finaux
  - Bien communiquer avec le client
  - Tenir compte des changements du cahier des charges
  - Empêcher la découverte tardive de défauts sérieux dans le projet
  - Traiter au plus tôt tous les points critiques du projet
  - Bien maîtriser la complexité
  - Favoriser la réutilisation
  - Définir une architecture robuste
  - Faciliter le travail en équipe

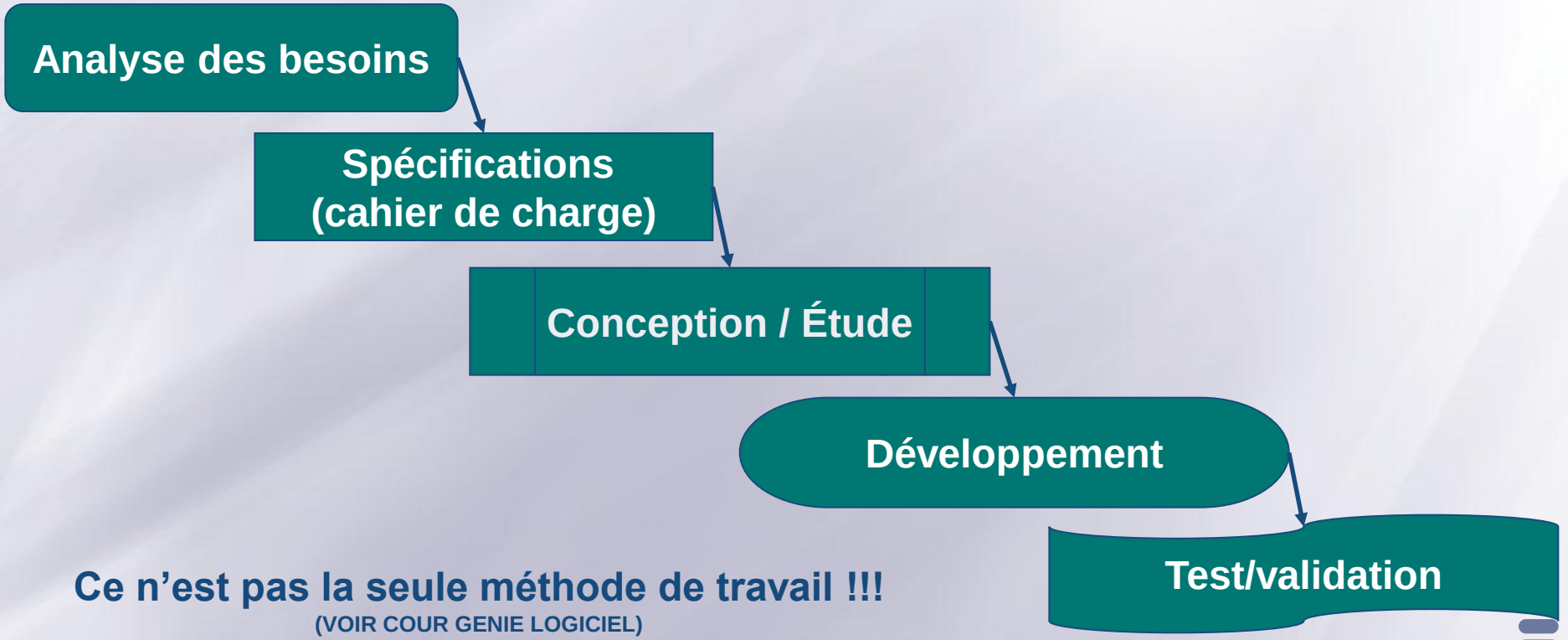
# Et pourquoi une démarche?



# Les démarches ou méthodes

## 3 types de démarches:

- Itérative et incrémentale
- guidée par l'utilisateur
- centrée sur l'architecture




**Ce n'est pas la seule méthode de travail !!!**
(VOIR COUR GENIE LOGICIEL)

# Démarche itérative et incrémentale

- L'idée est simple : pour modéliser (comprendre et représenter) un système complexe, il vaut mieux s'y prendre en plusieurs fois, en affinant son analyse par étapes

- Cette démarche devrait aussi s'appliquer au cycle de développement dans son ensemble, en favorisant le prototypage

- Le but est de mieux maîtriser la part d'inconnu et d'incertitudes qui caractérisent les systèmes complexes.

# Démarche guidée par l'utilisateur

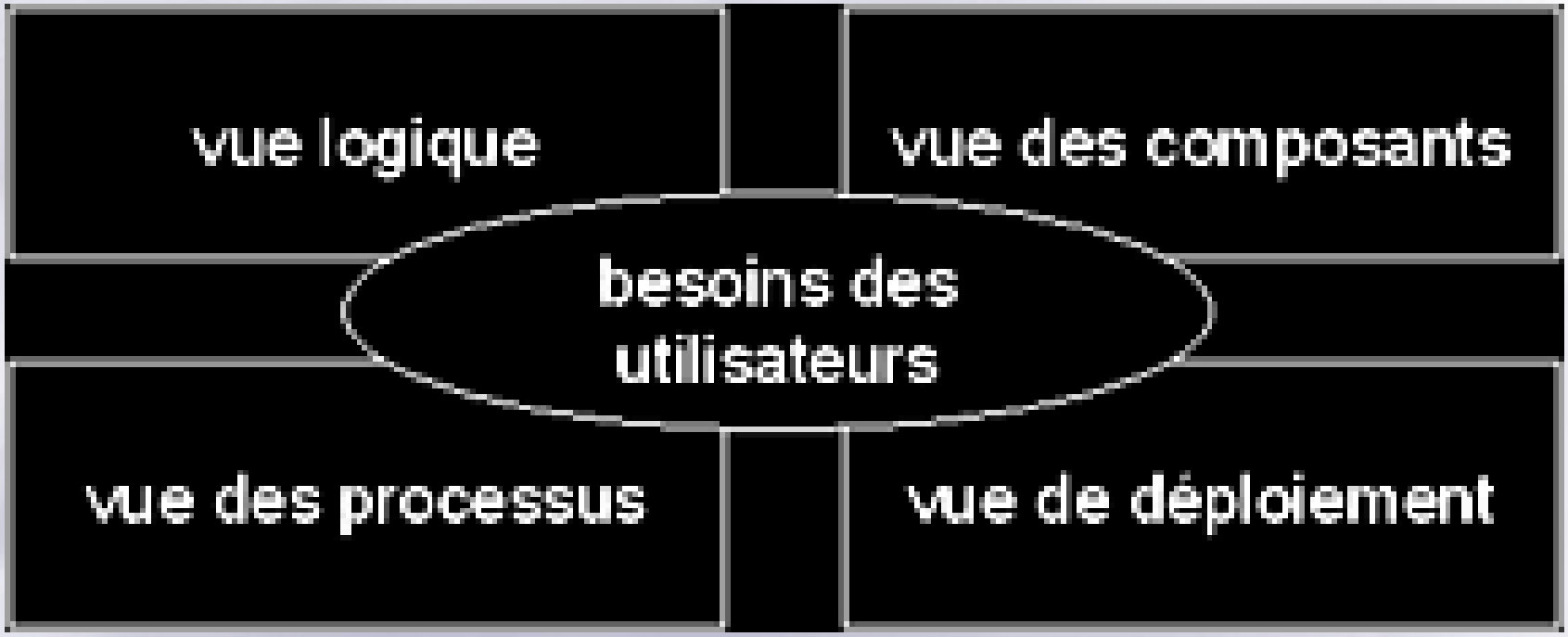
- Ce sont les utilisateurs qui guident la définition des modèles :
  - Le périmètre du système à modéliser est défini par les besoins des utilisateurs (les utilisateurs définissent ce que doit être le système).
  - Le but du système à modéliser est de répondre aux besoins de ses utilisateurs (les utilisateurs sont les clients du système).

- Les besoins des utilisateurs servent aussi de fil rouge, tout au long du cycle de développement (itératif et incrémental) :
  - A chaque itération de la phase d'analyse, on clarifie, affine et valide les besoins des utilisateurs.
  - A chaque itération de la phase de conception et de réalisation, on veille à la prise en compte des besoins des utilisateurs.
  - A chaque itération de la phase de test, on vérifie que les besoins des utilisateurs sont satisfaits.

# Démarche centrée architecture

- Une architecture adaptée est la clé de voûte du succès d'un développement.

- Elle décrit des choix stratégiques qui déterminent en grande partie les qualités du logiciel (adaptabilité, performances, fiabilité...)



# Vue logique du système

- Cette vue de haut niveau se concentre sur l'abstraction et l'encapsulation, elle modélise les éléments et mécanismes principaux du système

- Elle identifie les éléments du domaine, ainsi que les relations et interactions entre ces éléments :
  - les éléments du domaine sont liés au(x) métier(s) de l'entreprise,
  - ils sont indispensables à la mission du système,
  - ils gagnent à être réutilisés (ils représentent un savoir-faire).

- Cette vue organise aussi (selon des critères purement logiques), les éléments du domaine en *"catégories"* :
  - pour répartir les tâches dans les équipes,
  - regrouper ce qui peut être générique,
  - isoler ce qui est propre à une version donnée, etc...

# Vue de composants

- **Cette vue de bas niveau (aussi appelée "vue de réalisation"), montre :**
  - **L'allocation des éléments de modélisation dans des modules (fichiers sources, bibliothèques dynamiques, bases de données, exécutables, etc...)**
  - **En d'autres termes, cette vue identifie les modules qui réalisent (physiquement) les classes de la vue logique**
  - **L'organisation des composants, c'est-à-dire la distribution du code en gestion de configuration, les dépendances entre les composants...**
  - **Les contraintes de développement (bibliothèques externes...)**
  - **La vue des composants montre aussi l'organisation des modules en *"sous-systèmes"*, les interfaces des sous-systèmes et leurs dépendances (avec d'autres sous-systèmes ou modules).**

# Vue de processus

- **Cette vue est très importante dans les environnements multitâches. Elle montre :**
  - **La décomposition du système en terme de processus (tâches)**
  - **Les interactions entre les processus (leur communication)**
  - **La synchronisation et la communication des activités parallèles (threads).**

# Vue de déploiement

- **Cette vue très importante dans les environnements distribués, décrit les ressources matérielles et la répartition du logiciel dans ces ressources :**
  - **La disposition et nature physique des matériels, ainsi que leurs performances**
  - **L'implantation des modules principaux sur les noeuds du réseau**
  - **Les exigences en terme de performances (temps de réponse, tolérance aux fautes et pannes...)**

# Vue des besoins de l’utilisateur

- Cette vue (dont le nom exact est "vue des cas d'utilisation"), guide toutes les autres:
  - Dessiner le plan (l'architecture) d'un système informatique n'est pas suffisant, il faut le justifier !
  - Cette vue définit les besoins des clients du système et centre la définition de l'architecture du système sur la satisfaction (la réalisation) de ces besoins.
  - A l'aide de scénarios et de cas d'utilisation, cette vue conduit à la définition d'un modèle d'architecture pertinent et cohérent.
  - Elle est la "colle" qui unifie les quatre autres vues de l'architecture.
  - Elle motive les choix, permet d'identifier les interfaces critiques et force à se concentrer sur les problèmes importants.

# Un petit exercice de Génie logiciel

- Quelle méthode utiliser dans les cas suivants?
  - Gestion d'une bibliothèque
  - Gestion des expériences de puces à ADN
  - Systèmes d'information communicants pour la médecine

# Passons aux choses sérieuses !!!

## MODELISATION OBJET AVEC UML

# Les éléments de modélisation (1)

## Les objets

- la description d'une entité du monde réel ou virtuel


## Les classes

- la description d'un ensemble d'objets


## Les états

- une étape de la vie d'un objet


## Les acteurs

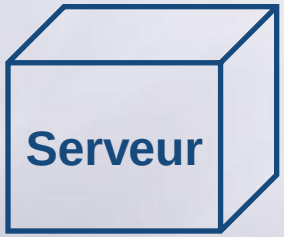
- utilisateurs finaux du système

# Les éléments de modélisation (2)

## Les cas d'utilisation


- Une manière dont un acteur utilise le système

## Les noeuds




- Un dispositif matériel

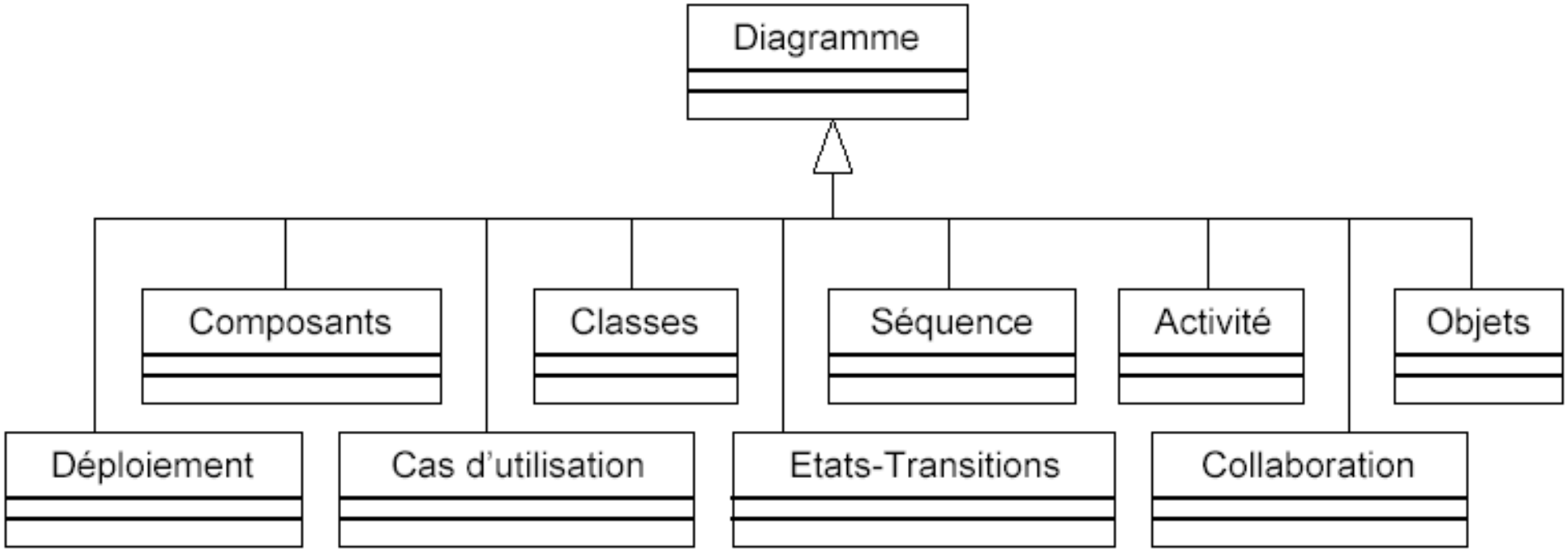
## Les paquetages


- Une partition du modèle

## Les notes

- Un commentaire, une explication ou une annotation

# Les diagrammes UML



# Diagramme des cas d'utilisation

*(Ivar Jacobson)*

# Diagramme de cas d'utilisation (0)

- Une réflexion sur les fonctionnalités attendues du futur système avant la conception
- Avoir une idée (même assez vague) sur les grands modules du système
- Les fonctionnalités que doit fournir chaque composant
- Ces fonctionnalités vont aider les utilisateurs à effectuer leur « mission »

# Diagramme de cas d'utilisation (1)

- La détermination et la compréhension des besoins sont souvent difficiles

➔ il faut clarifier et organiser les besoins des clients (les modéliser).

- Les use cases permettent de structurer les besoins des utilisateurs et les objectifs correspondants d'un système
- Ils centrent l'expression des exigences du système sur ses utilisateurs
- Ils se limitent aux préoccupations "réelles" des utilisateurs ; ils ne présentent pas de solutions d'implémentation et ne forment pas un inventaire fonctionnel du système
- Ils identifient les utilisateurs du système (acteurs) et leur interaction avec le système
- Ils permettent de classer les acteurs et structurer les objectifs du système

# Diagramme de cas d'utilisation (2)

- Le DCU décrit sous la forme d'*actions* et de *réactions* le comportement *externe* du système
- Le comportement externe = du point de vue des utilisateurs
- Il permet de définir les frontières du système et les relations entre le système et l'environnement
- Un DCU comprend les acteurs, le système et les ses cas d'utilisation.

# Diagramme de cas d'utilisation (3)

## Qu'est ce qu'un cas d'utilisation?

- Un cas d'utilisation (use case) représente un ensemble de séquences d'actions réalisées par le système et produisant un résultat observable intéressant pour un acteur particulier

- Un cas d'utilisation modélise un service rendu par le système. Il exprime les interactions acteurs/système et apporte une valeur ajoutée « notable » à l'acteur concerné

- Exemples: Achat billet, Ouverture compte, Livraison Client, Traiter commande, établir facture…

# Diagramme de cas d'utilisation (4)

## Qu'est ce qu'un acteur?

- Un acteur représente le rôle joué par quelque chose ou quelqu'un se trouvant dans l'environnement du système étudié
- Un acteur est en relation avec le métier de l'entreprise et interagit avec le système dans différents cas d'utilisation
- Il peut être un élément de la structure de l'entreprise tel qu'une direction, un service ou un poste de travail

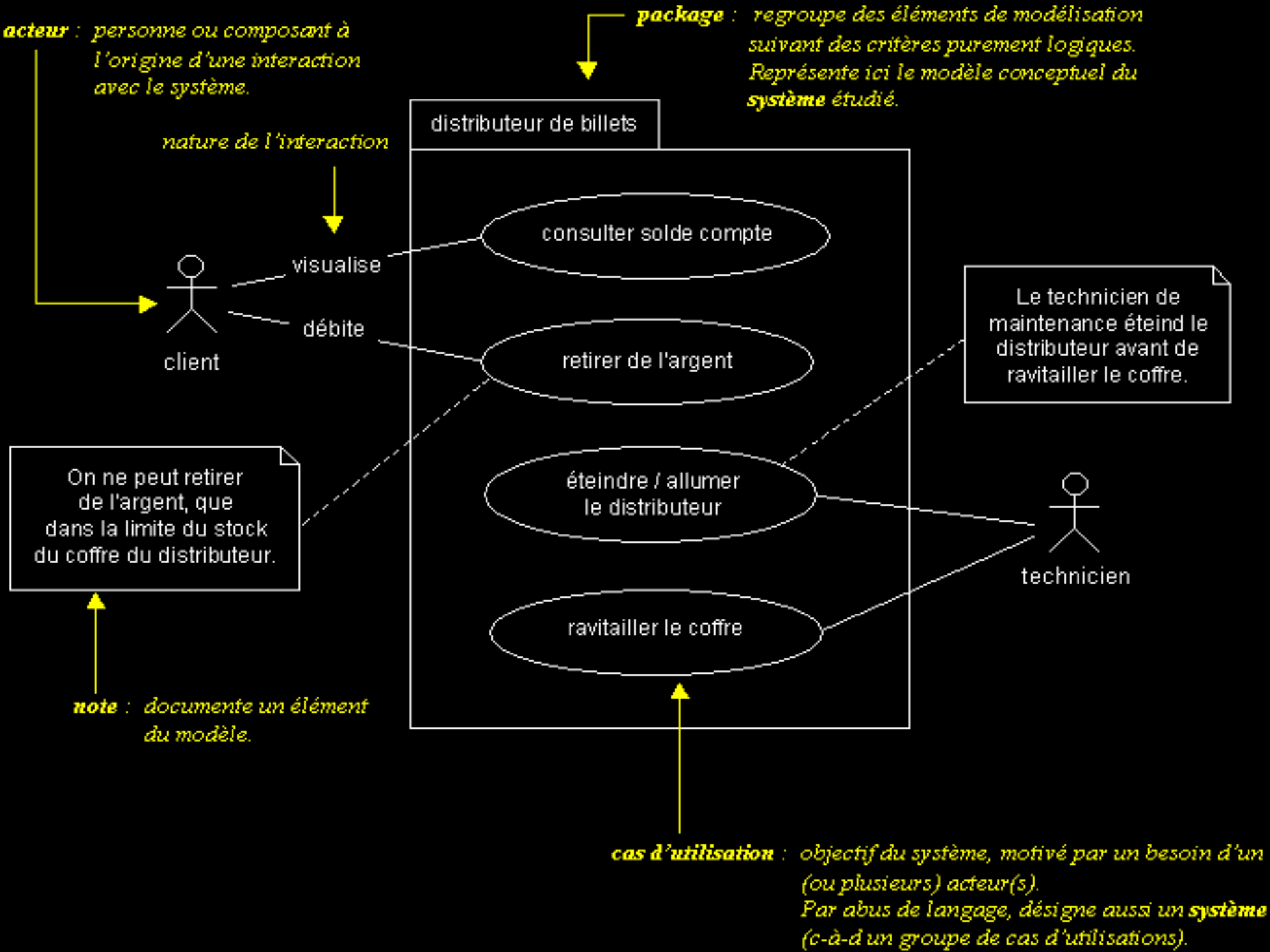
# Diagramme de cas d'utilisation (5)

## Types d'acteurs

- Par défaut, le rôle d'un acteur est « principal ». Si ce n'est pas le cas on peut indiquer le rôle « secondaire » sur l'acteur
- Si un acteur a pour rôle unique de consommer des informations du système sans modifier l'état de celui-ci au niveau métier alors son rôle est secondaire
- Un acteur peut aussi être un périphérique externe ou un système externe

## Les cas d'utilisation: portée

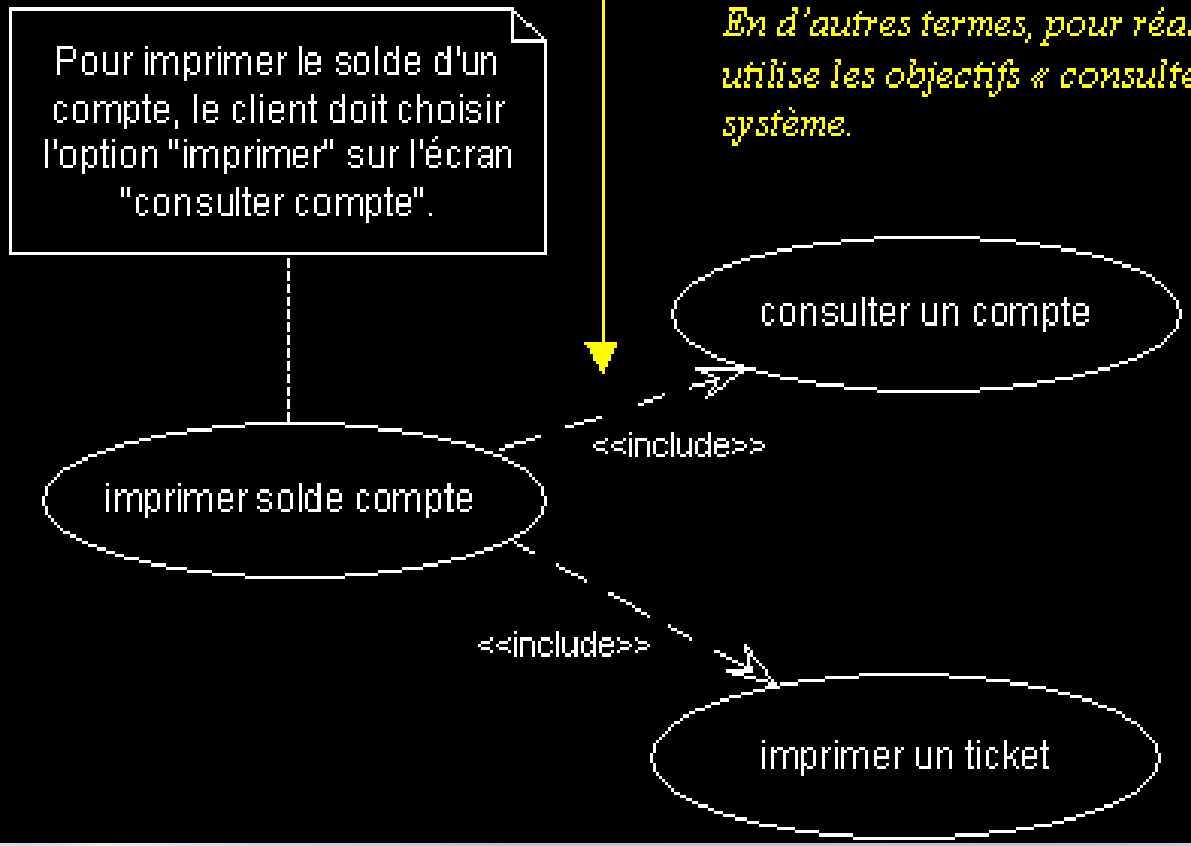
- Les Cas d'utilisation interviennent tout au long du cycle de vie d'un projet


acteur : personne ou composant à l'origine d'une interaction avec le système.
package : regroupe des éléments de modélisation suivant des critères purement logiques. Représente ici le modèle conceptuel du système étudié.
distributeur de billets
nature de l'interaction
visualise
débite
client
consulter solde compte
retirer de l'argent
éteindre / allumer le distributeur
ravitailler le coffre
technicien
Le technicien de maintenance éteind le distributeur avant de ravitailler le coffre.
On ne peut retirer de l'argent, que dans la limite du stock du coffre du distributeur.
note : documente un élément du modèle.
cas d'utilisation : objectif du système, motivé par un besoin d'un (ou plusieurs) acteur(s).
Par abus de langage, désigne aussi un système (c-à-d un groupe de cas d'utilisations).

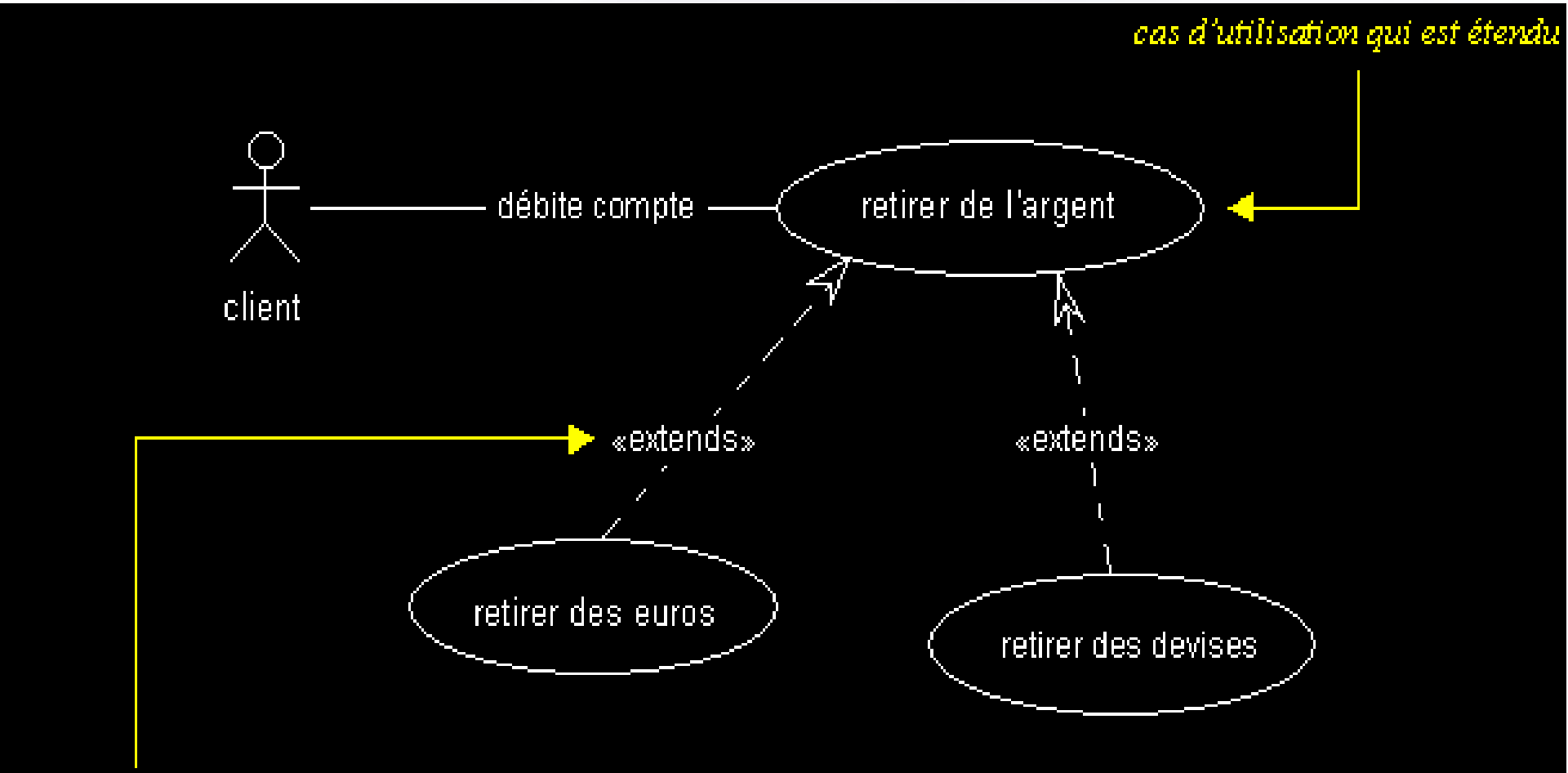
# Diagramme de cas d'utilisation (8)

***relation d'utilisation*** : *indique que le cas d'utilisation source contient aussi le comportement décrit dans le cas d'utilisation destination.*

*En d'autres termes, pour réaliser l'objectif « imprimer solde compte », on utilise les objectifs « consulter compte » et « imprimer un ticket » du système.*



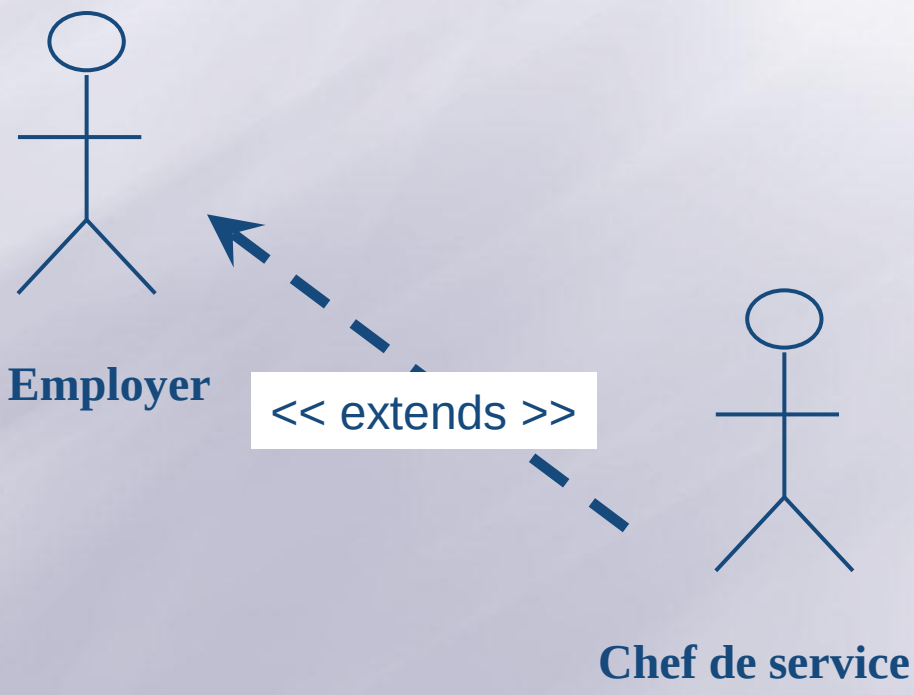
# Diagramme de cas d'utilisation (9)



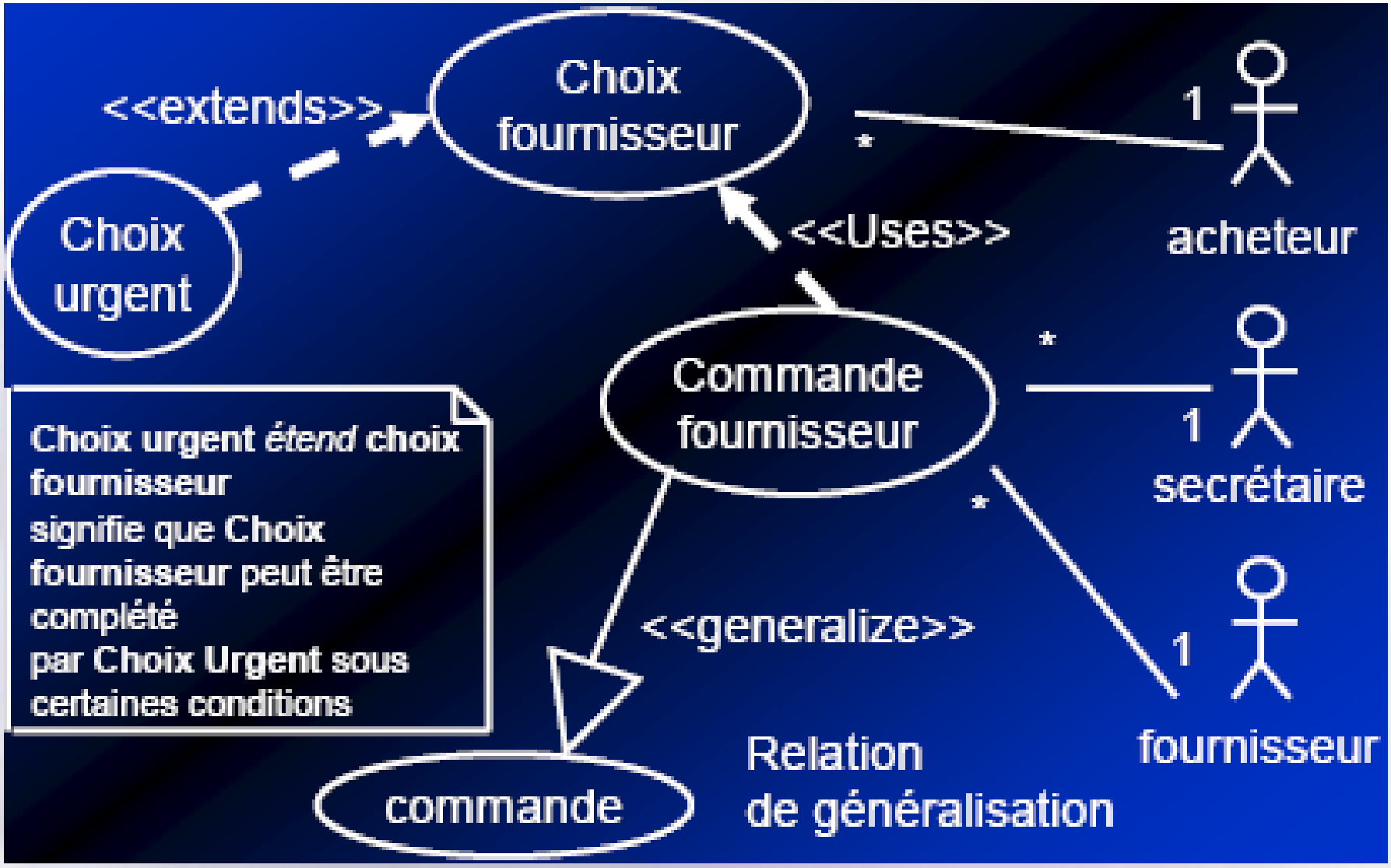

***relation d'extension*** : *indique que le cas d'utilisation source étend (précise) les objectifs (le comportement) du cas d'utilisation destination.*

# Diagramme de cas d'utilisation (10)

## Extension des acteurs



# Diagramme de cas d'utilisation (11)



# Diagramme de cas d'utilisation (12)

- **Première étape à faire (capture des besoins)**
- **Cas d'utilisation: chaque comportement du système attendu par l'utilisateur**
  - **Définir le périmètre du système : Paquetage**
  - **Inventorier les acteurs (ceux qui utiliseront le futur logiciel)**
  - **Évaluer les besoins de chaque acteur en CU**
  - **Regrouper ces CU dans un diagramme présentant une vue synthétique du paquetage**
  - **Possibilité de documentation des CU complexes (cahier de charge)**
  - **Na pas descendre trop bas et réinventer l'analyse fonctionnelle**

# A vous de jouer (1)

- **Élaborez le diagramme des cas d'utilisation d'une agence de voyage. Pour organiser un voyage, l'agent doit réserver au client une chambre d'hôtel, lui réserver un billet d'avion ou de train, lui réserver un taxi pour venir de l'aéroport ou de la gare et enfin, lui établir une facture. Certains clients peuvent demander une facture plus détaillée**

# A vous de jouer (2)

- Élaborez le diagramme des cas d'utilisation correspondant à la gestion des inscriptions à l'ISECS
  - étudiant, scolarité, département
  - dépôt de dossier, enregistrement de dossier, sélection des dossiers, notification d'acceptation ou de refus

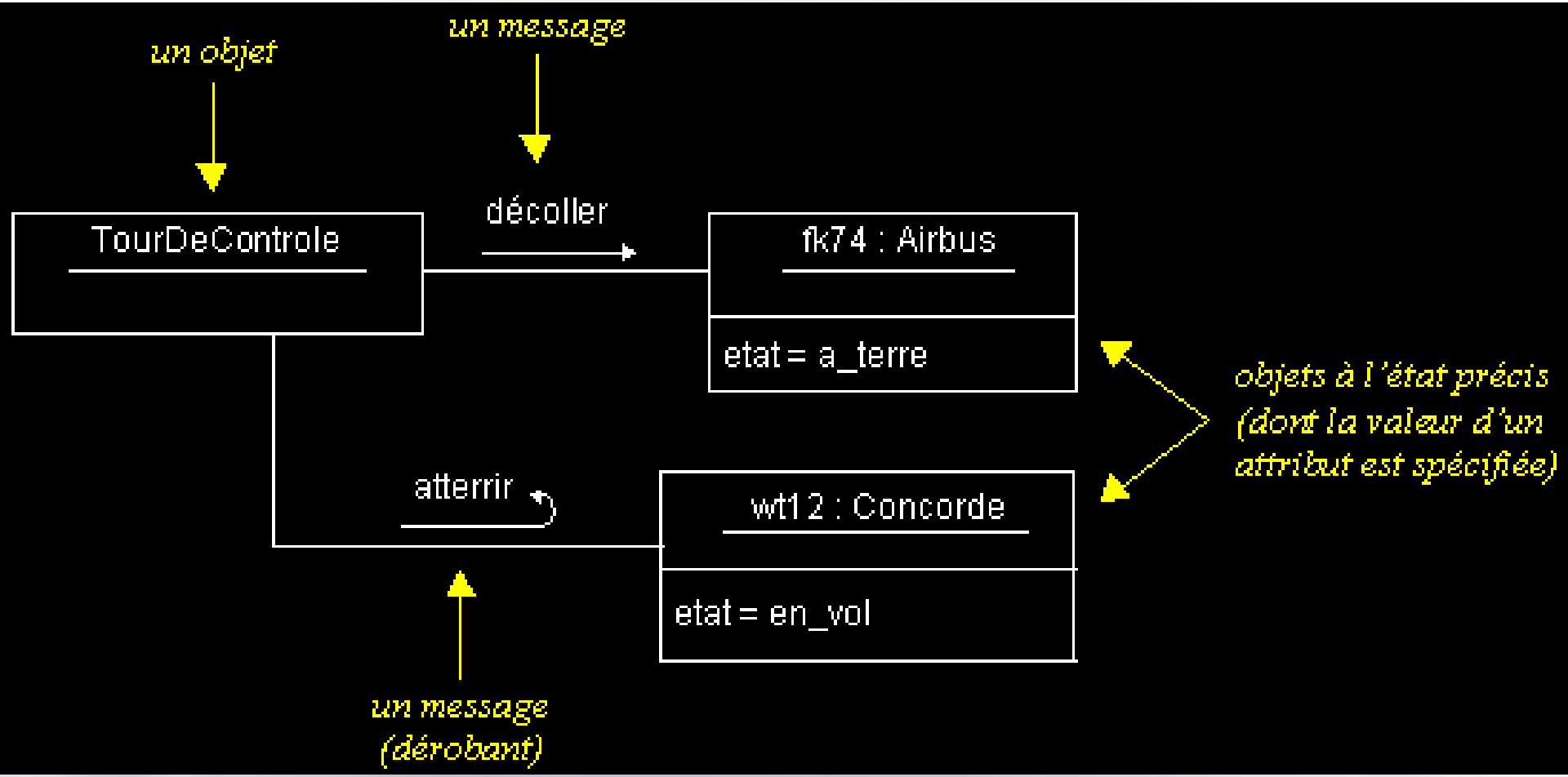
- Élaborez le diagramme des cas d'utilisation correspondant à la gestion des paiements en caisse dans un magasin

# Diagramme de collaboration

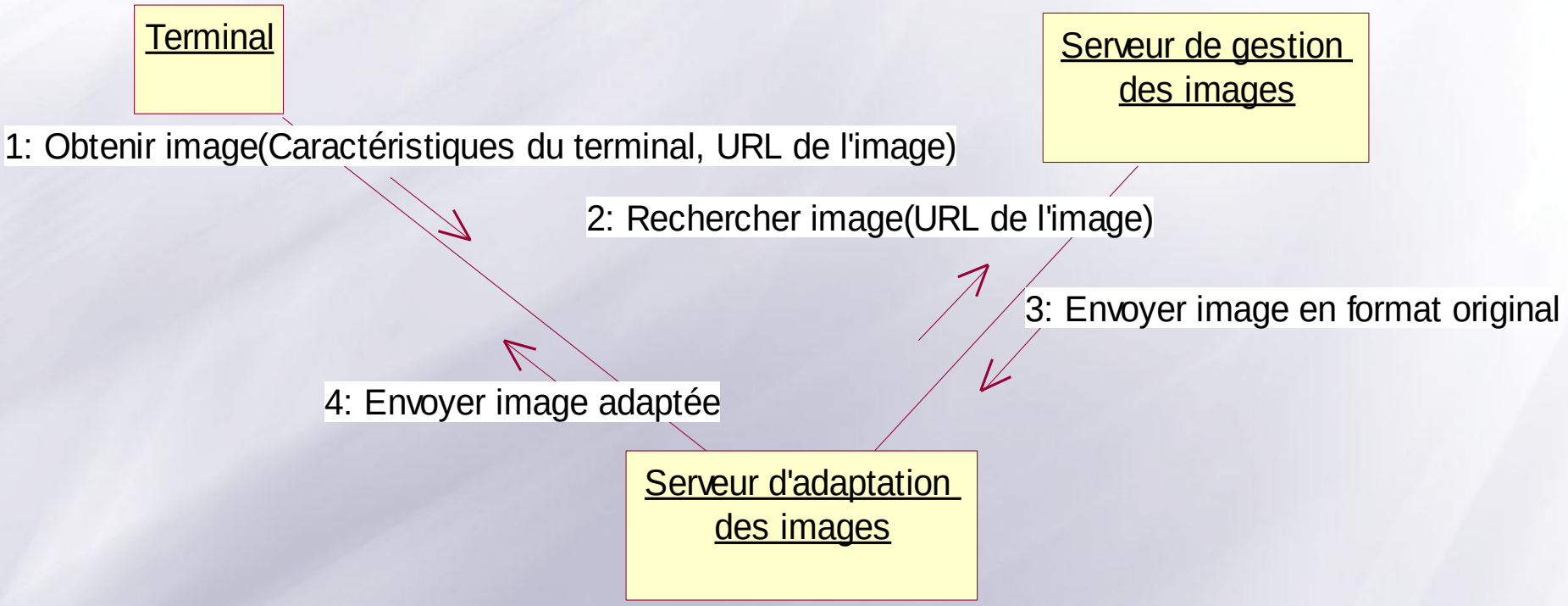
# Diagramme de collaboration (1)

- Les diagrammes de collaboration montrent des interactions entre objets (instances de classes et acteurs)
- Ils offrent une représentation spatiale d'une interaction
- Ils permettent de représenter le contexte d'une interaction, car on peut y préciser les états des objets qui interagissent

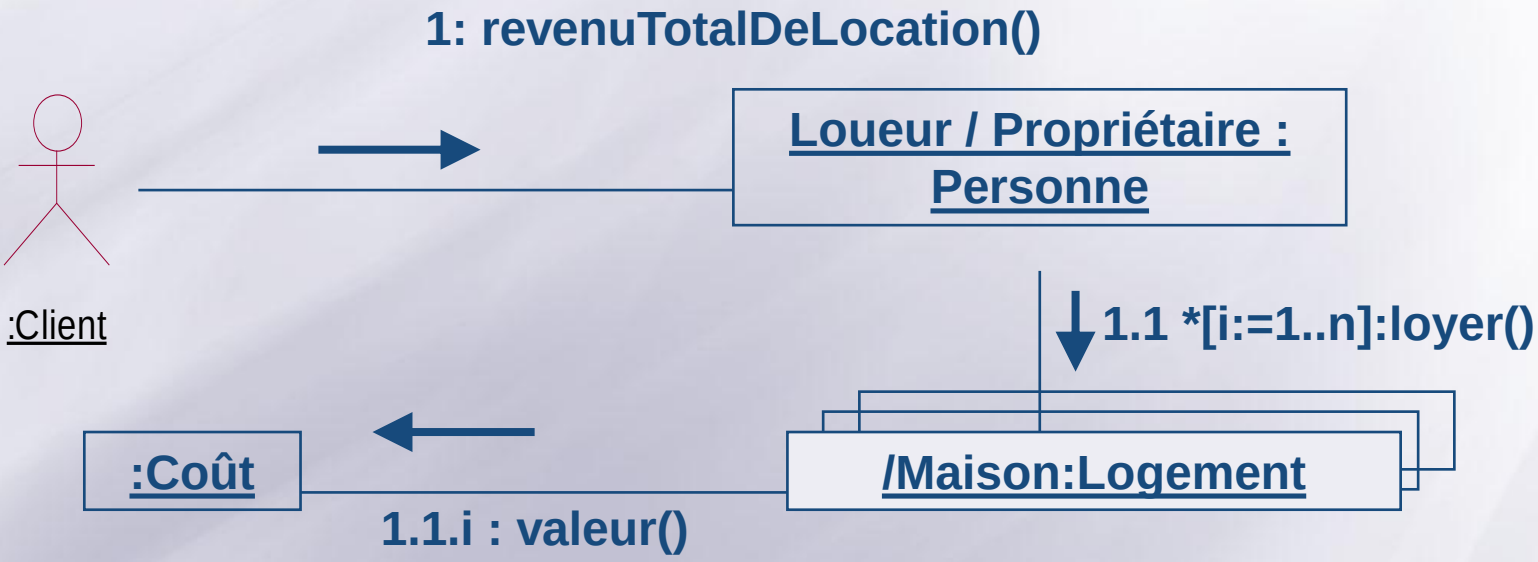
# Diagramme de collaboration (2)



# Diagramme de collaboration (3)



# Diagramme de collaboration : séquence de messages



# Diagramme de séquence

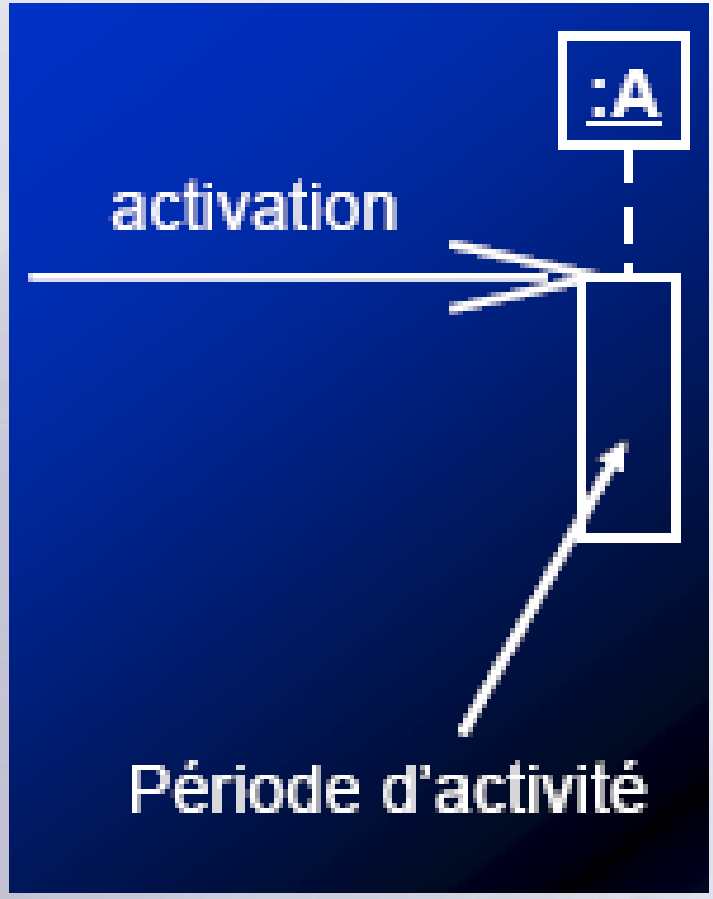
# Diagramme de séquence (1)

- Les diagrammes de séquences permettent de représenter des collaborations entre objets selon un point de vue temporel, on y met l'accent sur la chronologie des envois de messages
- Contrairement au diagramme de collaboration, on n'y décrit pas le contexte ou l'état des objets, la représentation se concentre sur l'expression des interactions
- Les diagrammes de séquences peuvent servir à illustrer un cas d'utilisation
- L'ordre d'envoi d'un message est déterminé par sa position sur l'axe vertical du diagramme ; le temps s'écoule "de haut en bas" de cet axe
- La disposition des objets sur l'axe du temps n'a pas de conséquences pour la sémantique du diagramme
- Les diagrammes de séquences et les diagrammes d'état-transitions sont les vues dynamiques les plus importantes d'UML

# Diagramme de séquence (2)

- Présente les interactions chronologiques entre les objets

- Focalise sur les
  - objets (les classes)
  - les acteurs (en partie)
  - échange de messages
  - scénarii d'exécution
  - ligne de vie des objets

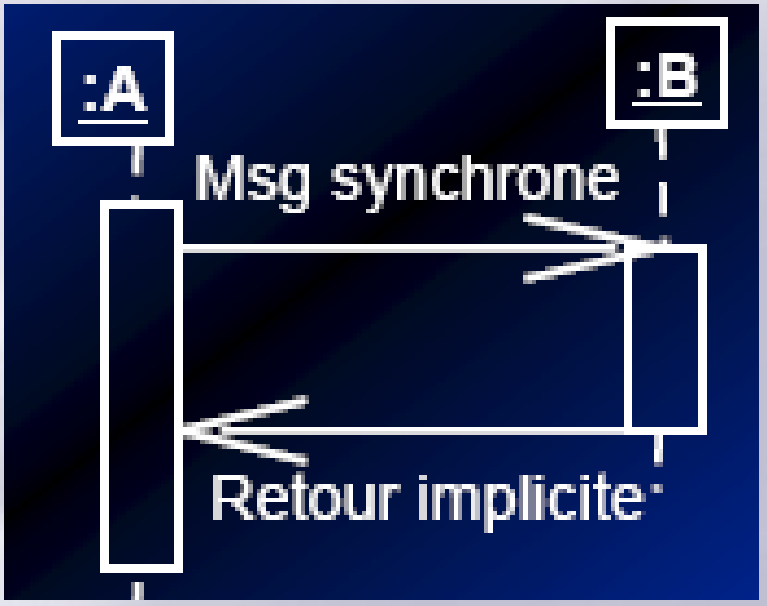


# Diagramme de séquence (3)

## Messages synchrones et asynchrones

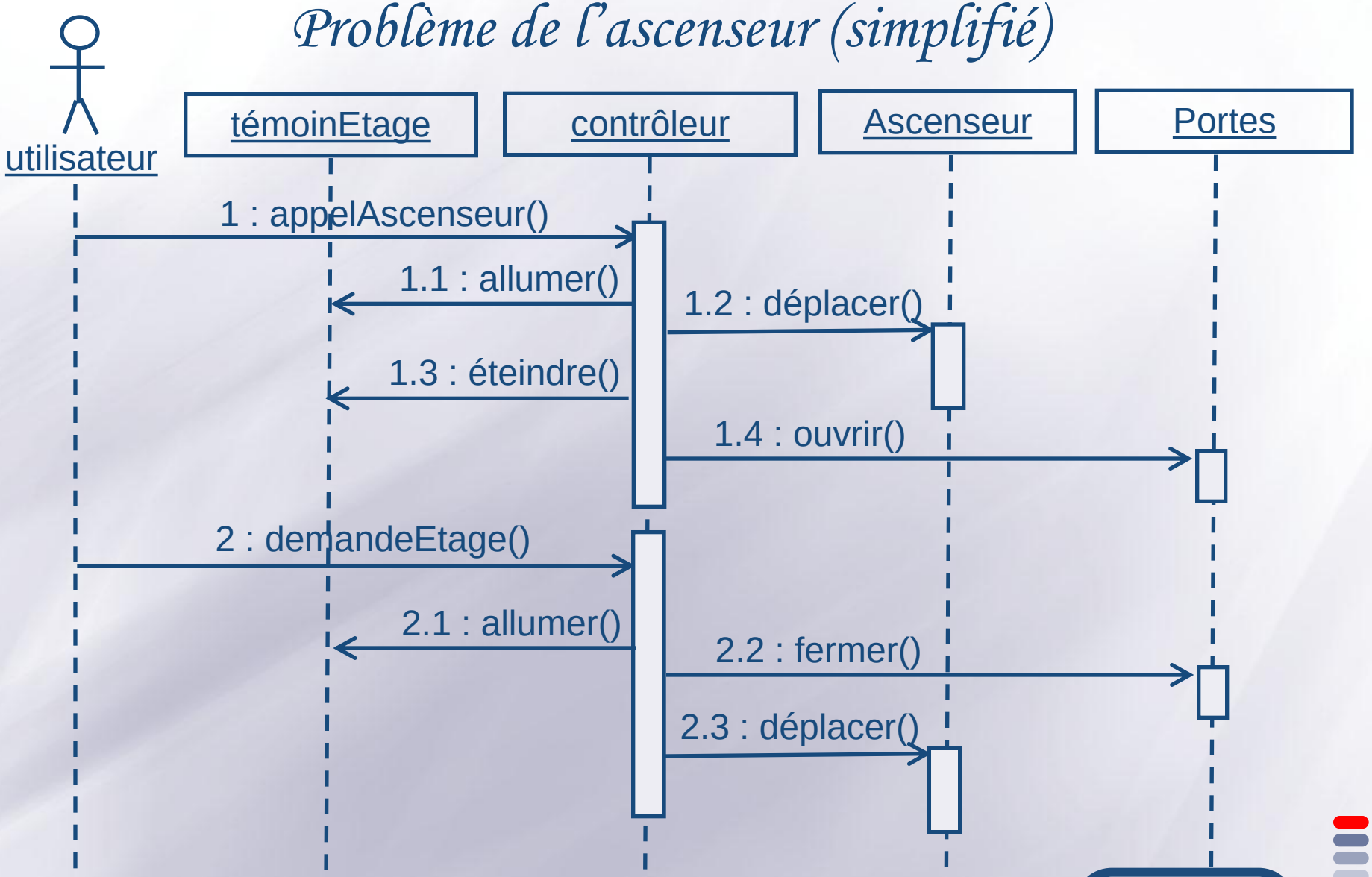
- messages synchrones
  - l'émetteur est bloqué jusqu'au traitement effectif du message

- Messages asynchrones
  - l'émetteur n'est pas bloqué et il peut poursuivre son exécution



# Diagramme de séquence (4)

*Problème de l'ascenseur (simplifié)*



# A vous de jouer

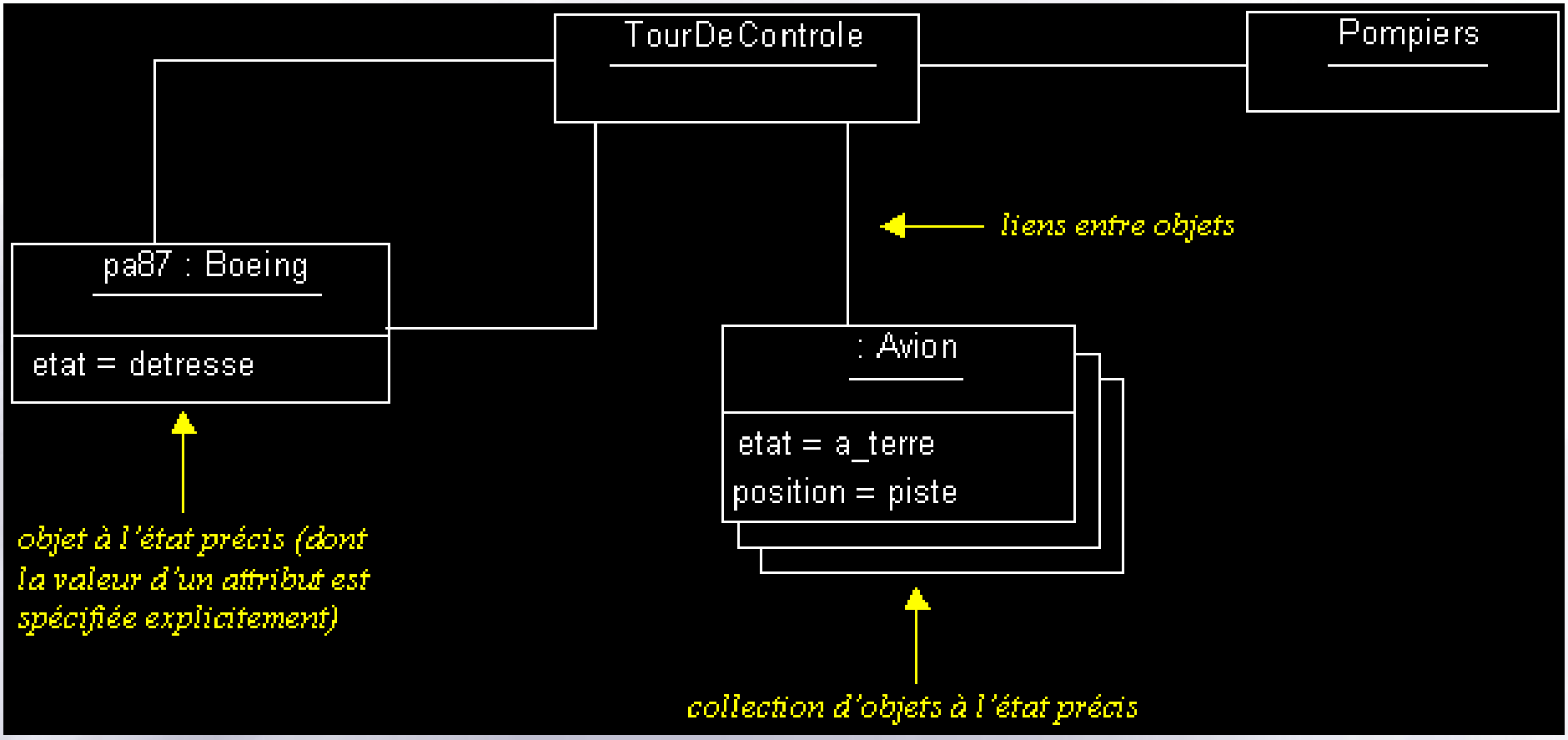
- **Une entreprise désire développer un logiciel de gestion de formation de ses employés. Un employé saisie sa demande de formation dans un formulaire après avoir consulté le catalogue des formations proposées par l'entreprise. Le module de gestion des formations, ajoute cette demande dans la base de données et envoie une notification à la direction par email (date de saisie, code de l'employé et le code de la formation). La direction consulte les détails de la formation à travers le même catalogue et répond à l'email du module de gestion par un refus ou un accord. Cet email est transmis par ce module à l'employé. En cas d'accord, le module envoie les formulaires d'inscription nécessaires avec le planning des sessions de formation à l'employé.**

# Diagramme d'objets

# Diagramme d’objets (1)

- Ce type de diagramme UML montre des objets (instances de classes dans un état particulier) et des liens (relations sémantiques) entre ces objets
- Les diagrammes d'objets s'utilisent pour montrer un contexte (avant ou après une interaction entre objets par exemple)
- Ce type de diagramme sert essentiellement en phase exploratoire, car il possède un très haut niveau d'abstraction

# Diagramme d'objets (2)



# A vous de jouer

1) Une voiture comporte un moteur et 4 roues

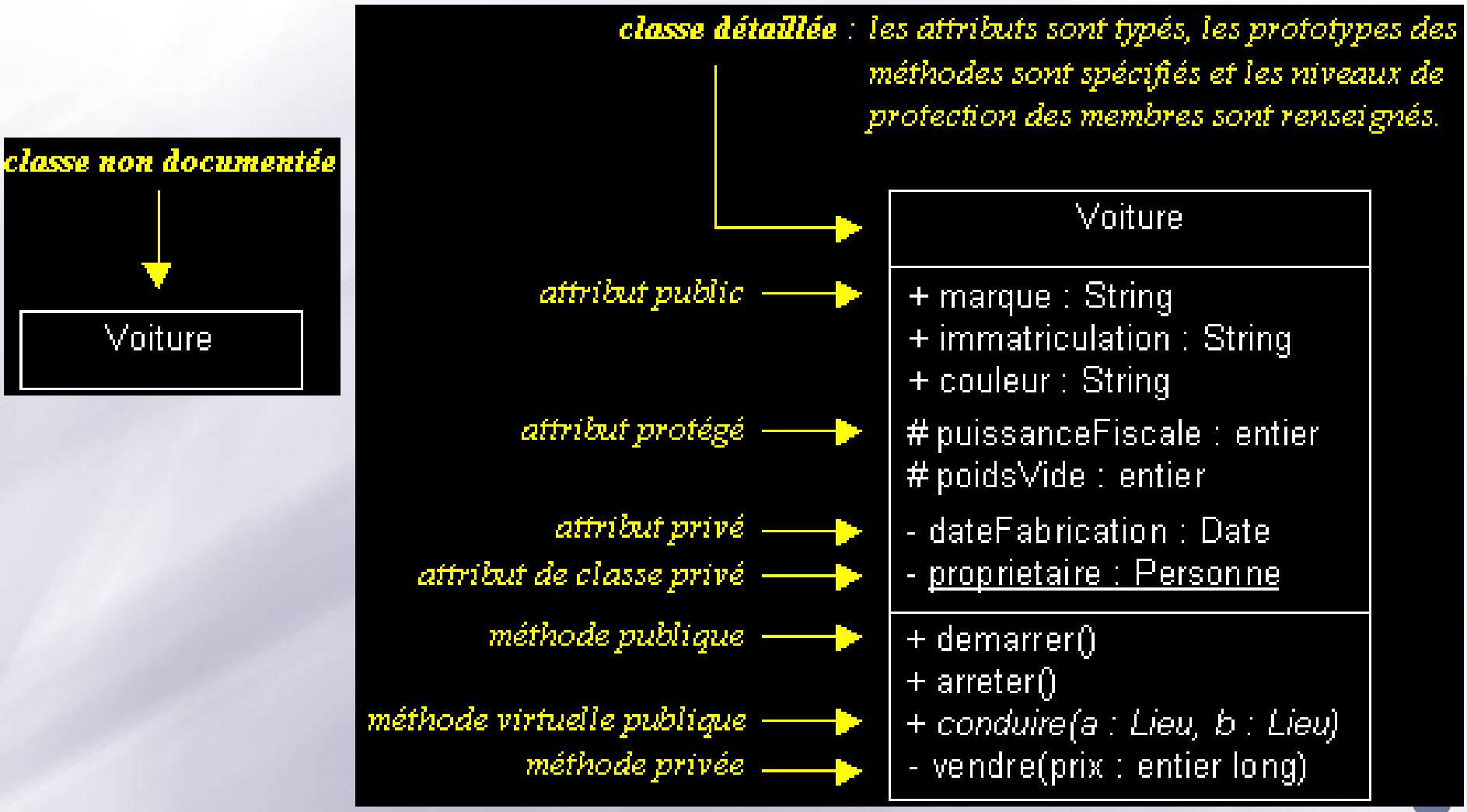
2) Le bus numéro 6723 roule en direction de Paris avec 3 passagers. Son chauffeur est Jean-Marc.

Élaborez les diagramme d'objets représentant les différentes entités citées.

# Diagramme de classes

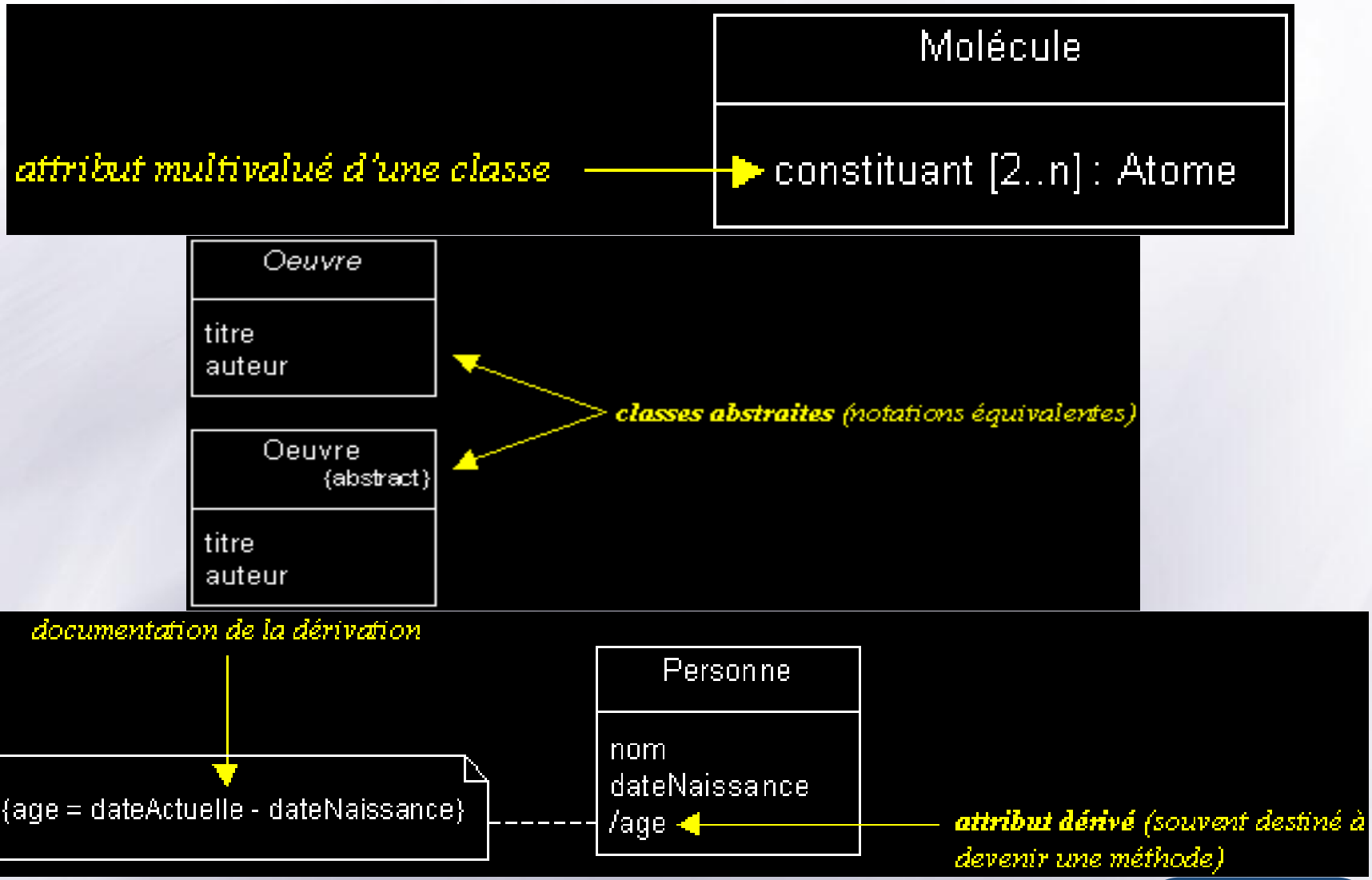
# Diagramme de classes (1)

## Les classes (1)



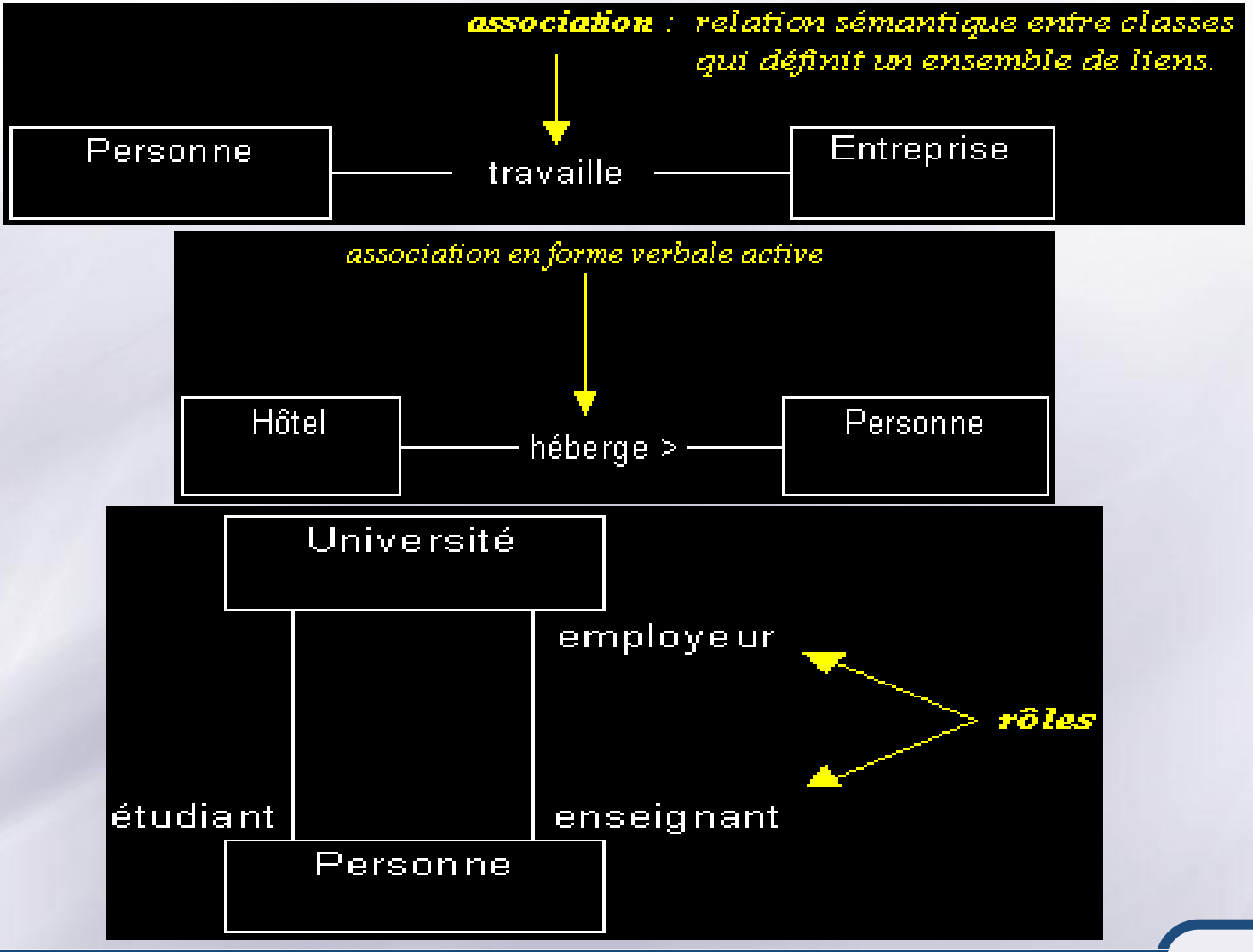
# Diagramme de classes (2)

## Les classes (2)



# Diagramme de classes (3)

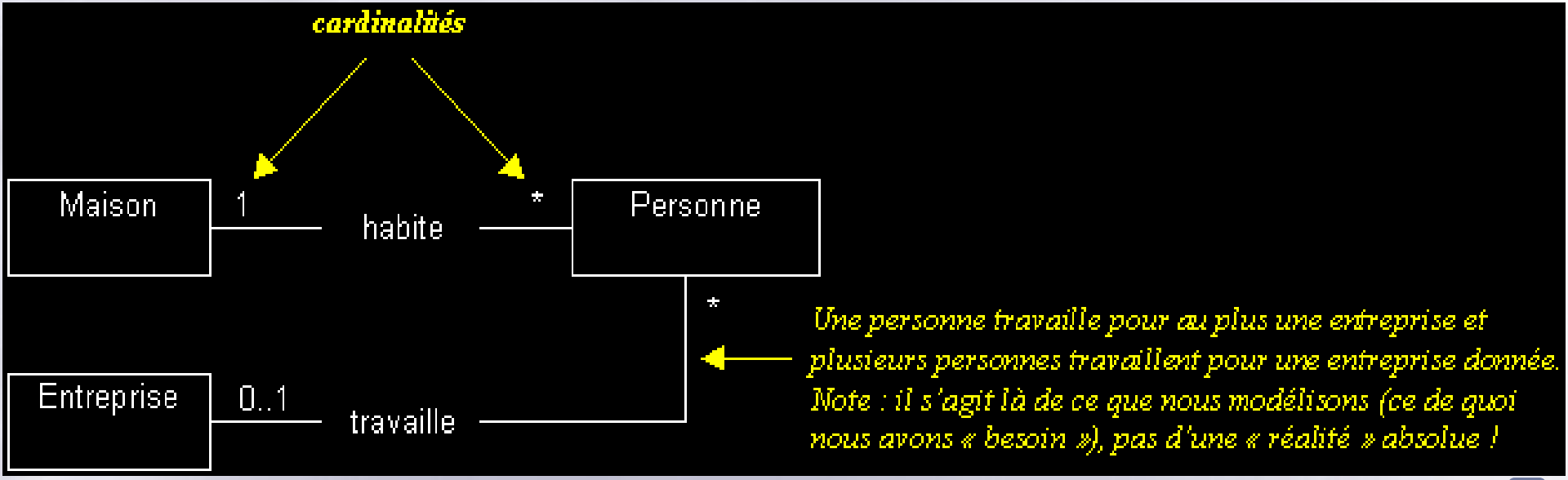
## Associations entre les classes (généralités)



# Diagramme de classes (4)

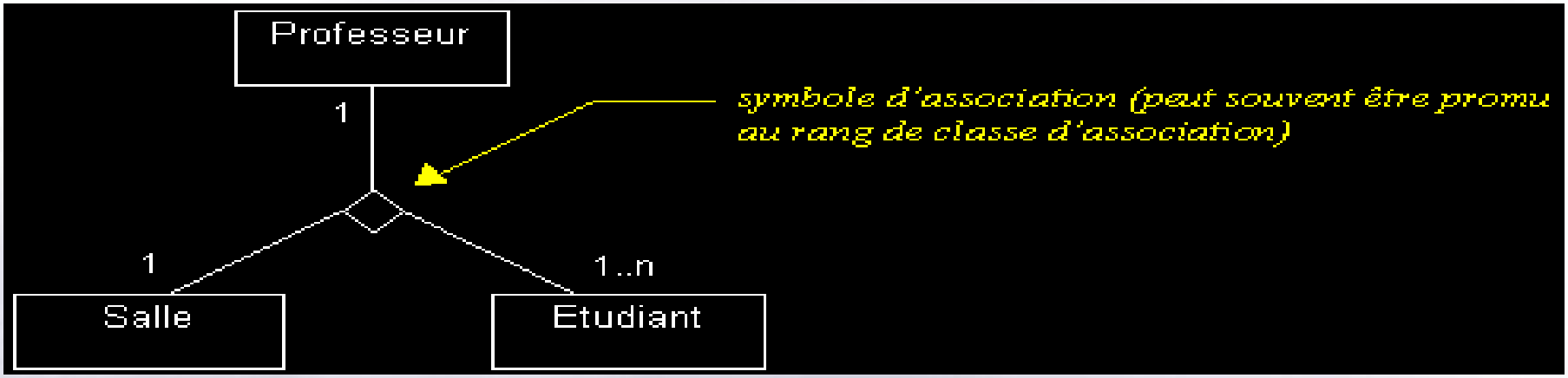
## Associations entre les classes (cardinalités)

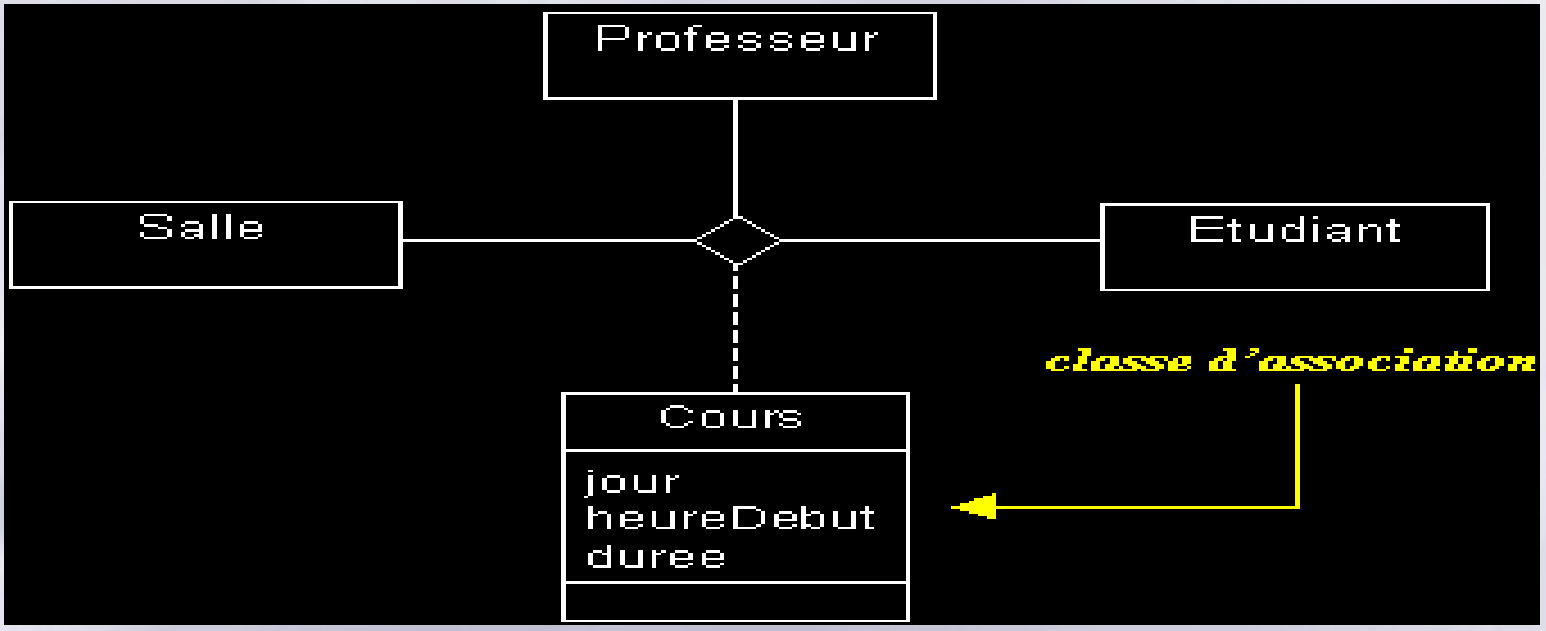
- n : exactement n instances de la classe (n entier >= 0)
- 0..1 : zéro ou une instance
- m..n : de m à n instances
- * : zéro à plusieurs instances
- 1..* : un à plusieurs instances (maximum non fixé)



# Diagramme de classes (5)

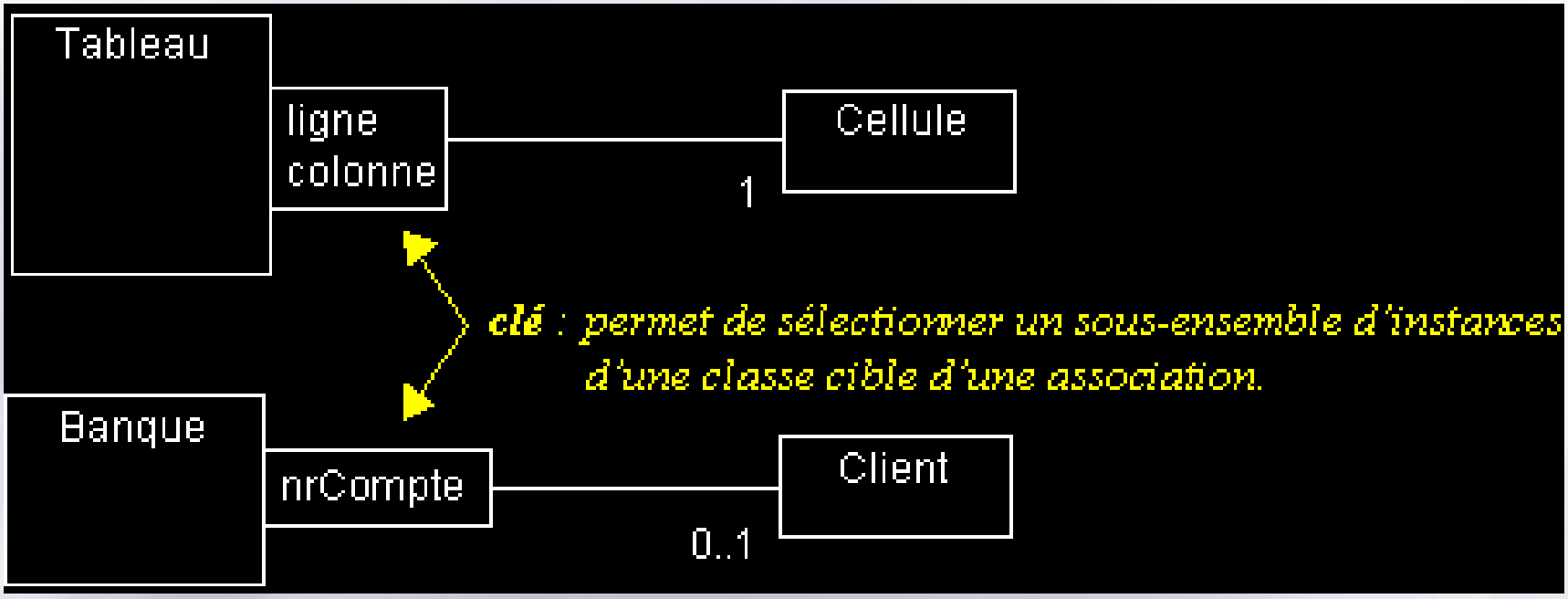
## Associations entre les classes (classes d'association)



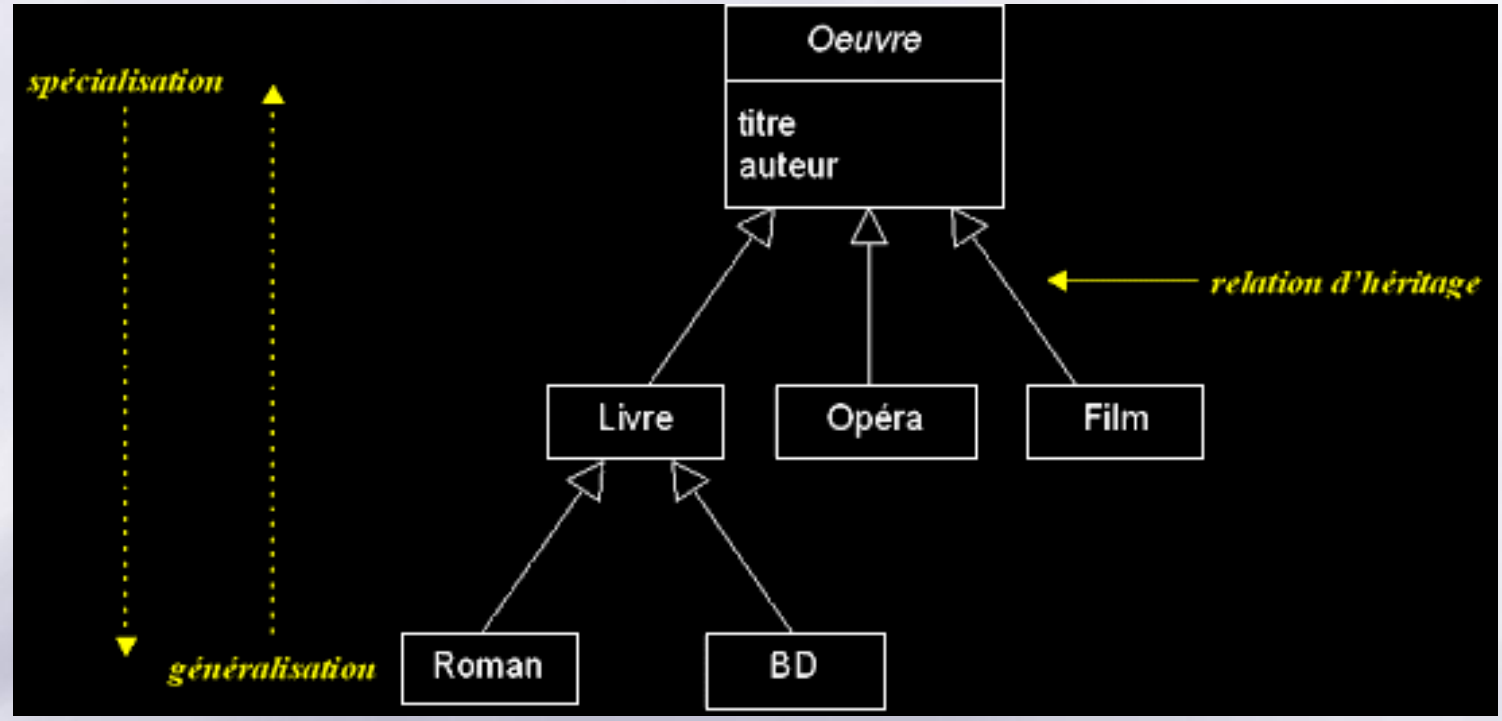
# Diagramme de classes (6)

## Associations entre les classes (qualifications)



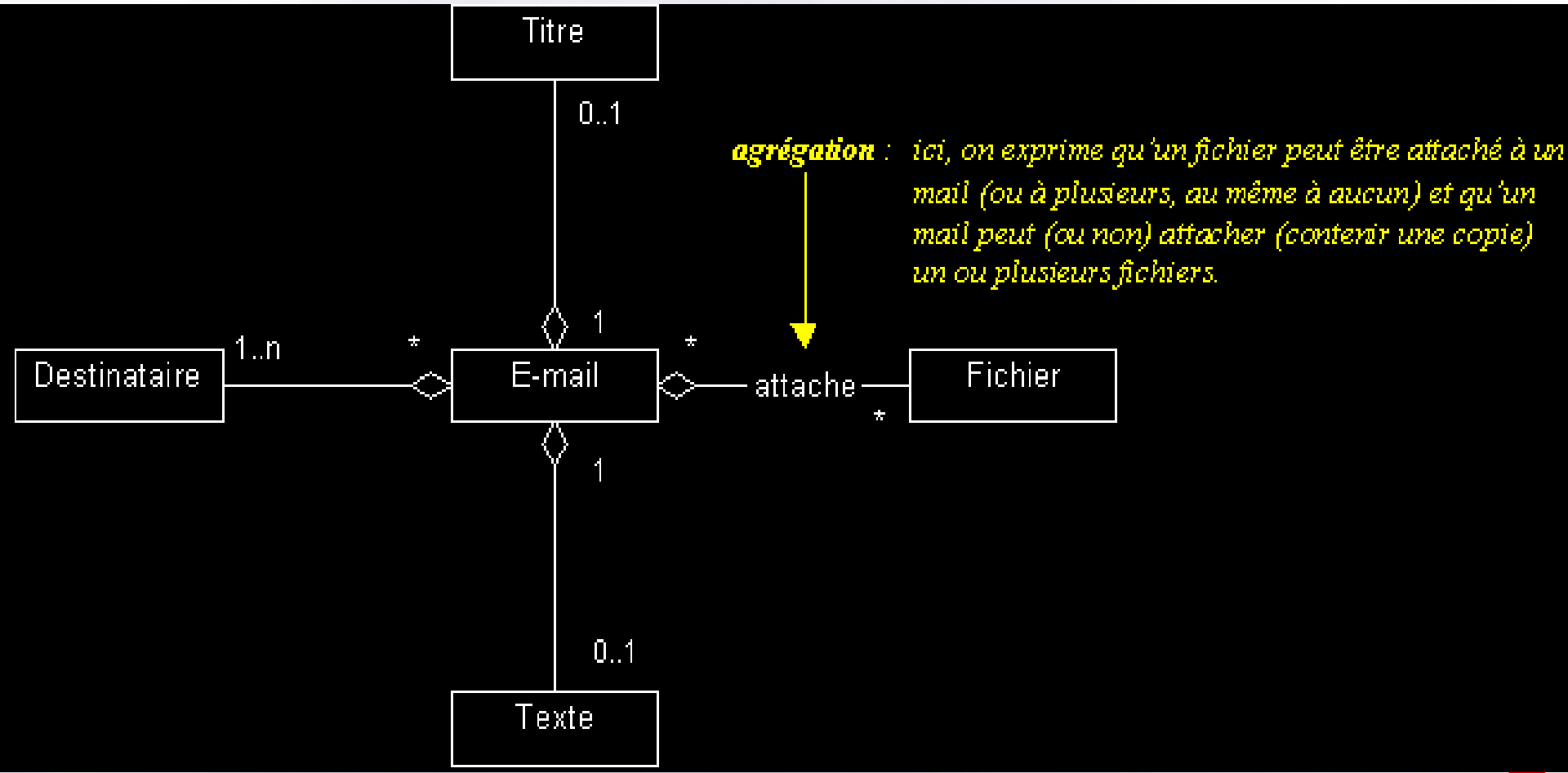
# Diagramme de classes (7)

## Associations entre les classes (héritage)



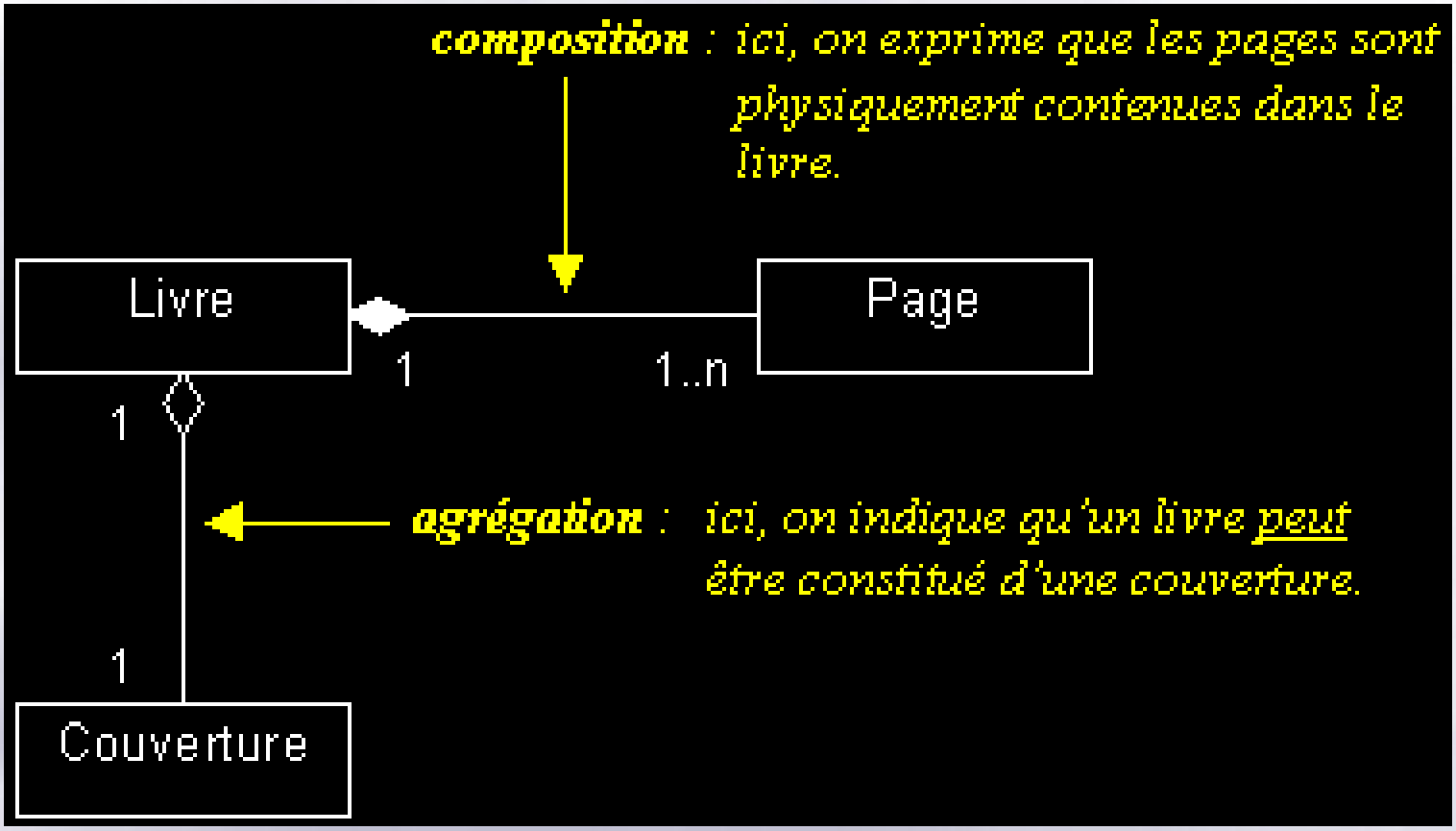
# Diagramme de classes (8)

## Associations entre les classes (agrégation)



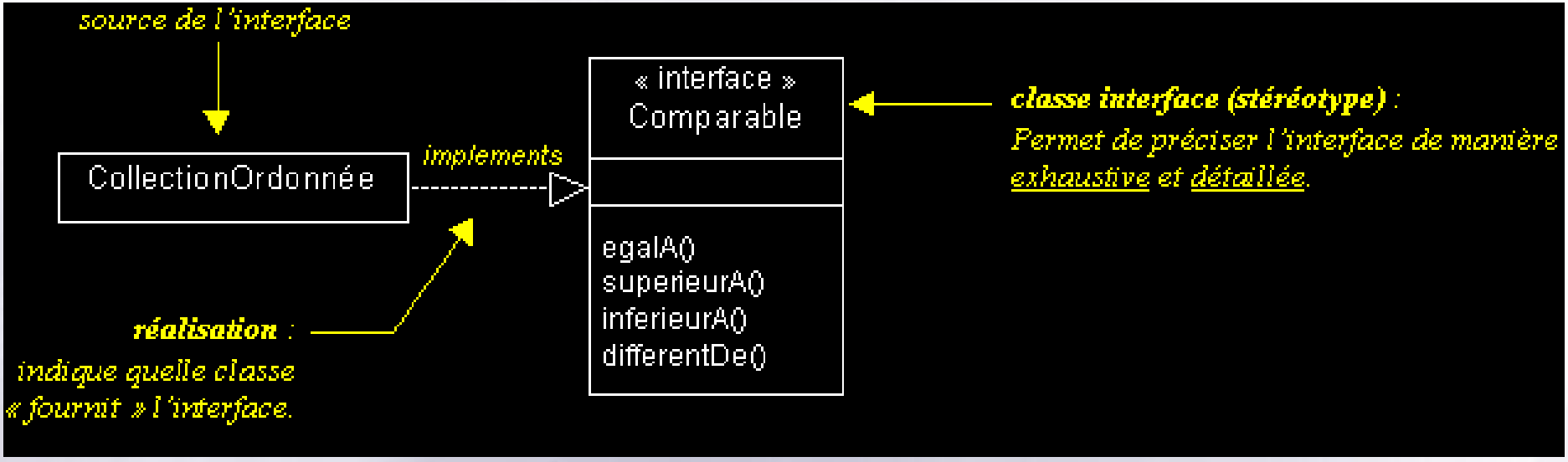
# Diagramme de classes (9)

## Associations entre les classes (composition)



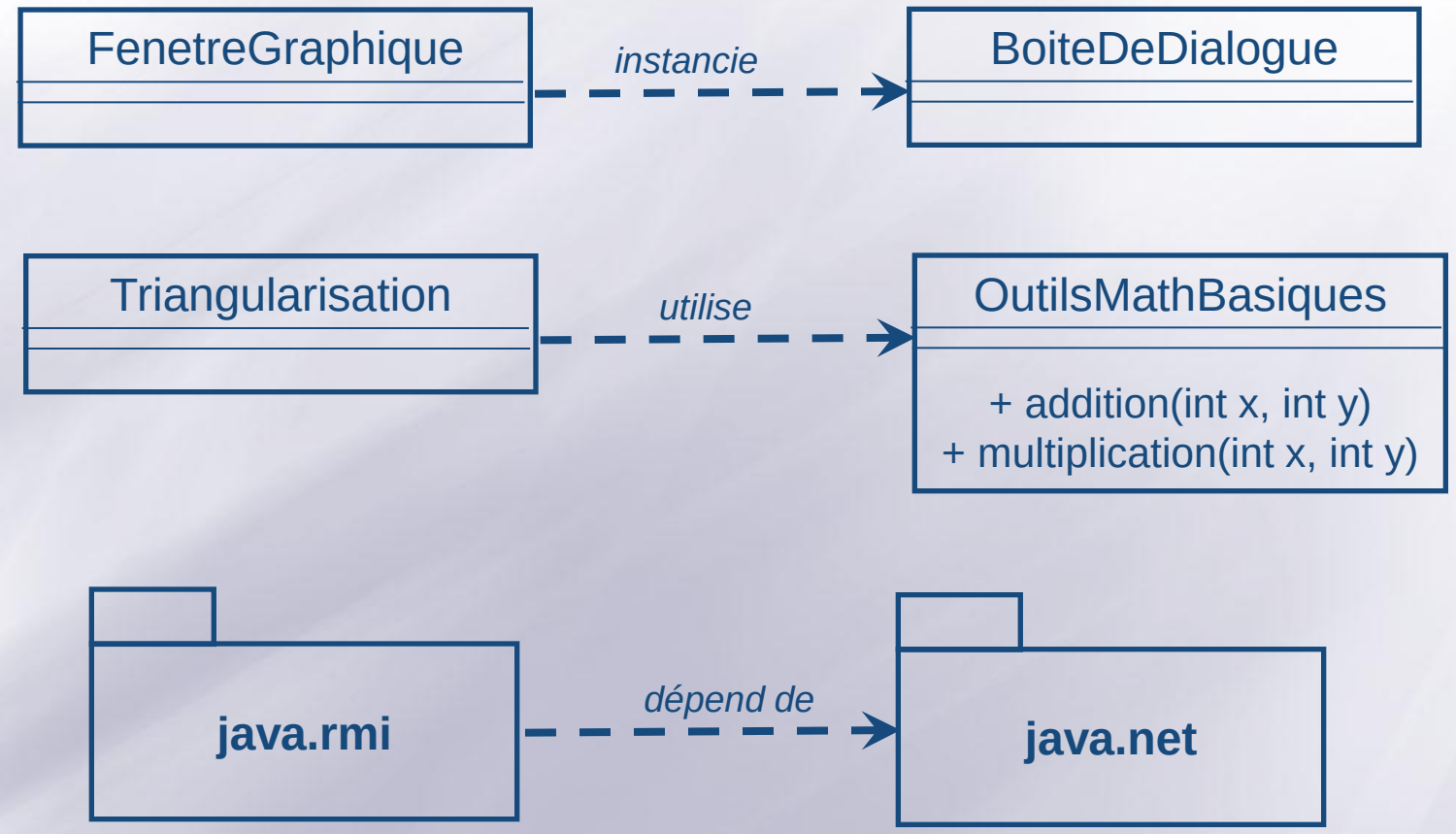
# Diagramme de classes (10)

## Associations entre les classes (interfaces)



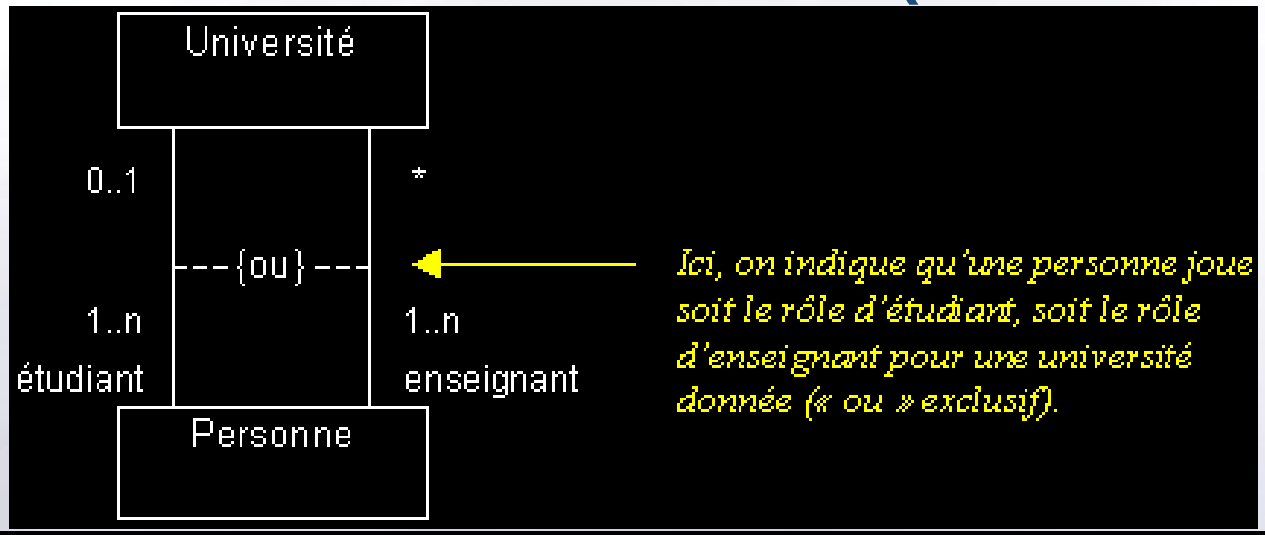

# Diagramme de classes (11)

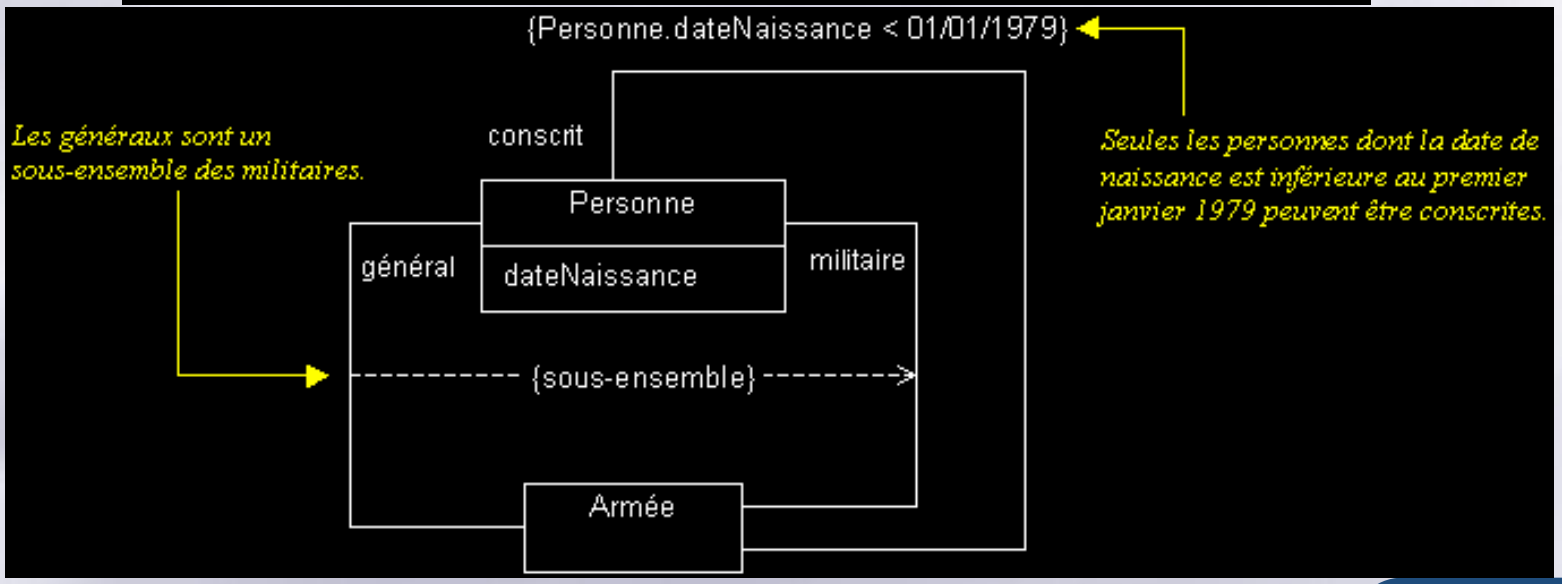
## Associations entre les classes (dépendances)



# Diagramme de classes (12)

## Associations entre les classes (contraintes)



# Diagramme de classes (Synthèse)

- Un diagramme de classes est une collection d'éléments de modélisation statiques (classes, paquetages...), qui montre la structure d'un modèle de données
- Un diagramme de classes fait abstraction des aspects dynamiques et temporels
- Pour un modèle complexe, plusieurs diagrammes de classes complémentaires doivent être construits
- Pour réussir un diagramme de classes:
  - identifier les entités (ou classes) pertinentes
  - identifier leurs interactions (relations et cardinalités)
  - utiliser les designs patterns (singleton, héritage…)

# A vous de jouer (1)

- **Gestion de la cité U (diagramme de classes)**

**Une Cité U est constituée d'un ensemble de bâtiments. Un bâtiment comporte un certain nombre de chambres. *La cité peut employer du personnel et est dirigé par l'un des employés*. Chaque chambre de la cité se loue à un prix donné (suivant ses prestations).**

**L'accès aux salles de bain est compris dans le prix de la location d'une chambre. Certaines chambres comportent une salle de bain, mais pas toutes. Les hôtes de chambres sans salle de bain peuvent utiliser une salle de bain sur le palier. Ces dernières peuvent être utilisées par plusieurs hôtes.**

**Une personne peut louer une et une seule chambre et une chambre peut être loué par une ou deux personnes.**

**Les pièces de la Cité U qui ne sont ni des chambres, ni des salles de bain (hall d'accueil, cuisine...) ne font pas partie de l'étude (hors sujet).**

# A vous de jouer (2)

- élaborer le diagramme de classes impliquant les entités suivantes:
  - Forme graphique (toute forme graphique est dessinable)
  - Forme euclidienne (on peut calculer son aire)
  - Ellipse, trapèze, cercle, rectangle

# A vous de jouer (3)

## Gestion d'une bibliothèque (diagramme de classes)

- Un gérant d'une bibliothèque désire automatiser la gestion des prêts
- Il commande un logiciel permettant aux utilisateurs de connaître les livres présents, d'en réserver jusqu'à 2
- l'adhérent possède un mot de passe qui lui est donné à son inscription
- L'emprunt est toujours réalisé par les employés qui travaillent à la bibliothèque. Après avoir identifié l'emprunteur, ils savent si le prêt est possible (nombre max de prêts = 5), et s'il a la priorité (il est celui qui a réservé le livre)
- Ce sont les employés qui mettent en bibliothèque les livres rendus et les nouveaux livres. Il leur est possible de connaître l'ensemble des prêts réalisés dans la bibliothèque

# Complément: Du modèle de classes au modèle relationnel

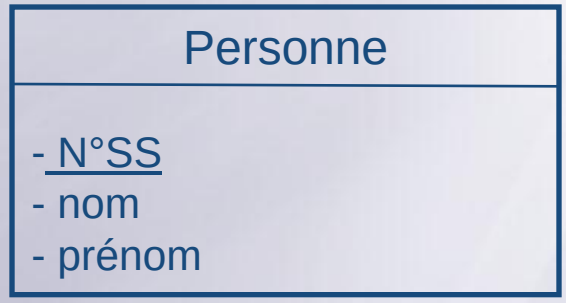
# Complément: Passage du modèle de classes au modèle relationnel (0)

- Identifier les classes
- Préparer le dictionnaire de données
- Identifier les associations entre objets (en incluant les agrégations)
- Identifier les attributs des objets et leurs liens
- Organiser et simplifier les classes en utilisant l'héritage
- Vérifier que le diagramme inclue toutes les demandes du cahier des charges
- Itérer et affiner le modèle
- Grouper les classes en modules

## Traduction des classes:

- Chaque classe devient une relation
- Chaque attribut de la classe devient un attribut de la relation
- L'identifiant de la classe devient une clé primaire de la relation

***Personne (N°SS, nom, prénom)***



## Traduction des associations 1- 1

- Inclure la clé primaire d'une des relations comme clé étrangère dans l'autre relation

- Exemple:
  - Un hôte habite dans une et une seule chambre
  - Une chambre héberge un et un seul hôte

*Chambre(N°chambre, superficie, type)*

*Hôte(ID, nom, prénom, N°chambre#)*

## Traduction des associations 1- N

- Inclure la clé primaire de la relation dont la multiplicité est N comme clé étrangère dans l'autre relation

- Exemple:
  - Un hôtel contient un ensemble de chambres

*Hôtel(ID_Hotel, nom, adresse)*

*Chambre(ID_chambre, superficie, type, ID_Hotel#)*

## Traduction des associations M – N

- créer une nouvelle relation dont la clé primaire est la concaténation des clés primaires des 2 relations participantes
- Les attributs de la classe d'association sont ajoutés à la nouvelle relation

- **Exemple:**

  *Etudiant(ID_Etudiant, nom, prenom)*

  *Enseignant(ID_Enseignant, nom, prenom)*

  *Enseignement(ID_Etudiant#, ID_Enseignant#, heure, durée)*

# Traduction de l'héritage

- Les sous classes ont la même clé primaire que la superclasse
- Un attribut « type » est ajouté à la superclasse dans le cas d'une spécialisation complète

- Exemple:
  - L'étudiant et l'enseignant sont des personnes
  - une personne est caractérisé par un ID, un nom et un prénom
  - Un étudiant est caractérisé par son niveau (bac+1, bac+2…)
  - Un enseignant est caractérisé par son grade (Vacataire, ATER, Maitre de conférences, professeur…)

*Personne(ID_Personne, nom, prenom, type)*

*type = « Etudiant » ou « enseignant »*

*Etudiant(ID_Personne#, niveau)*

*Enseignant(ID_Personne#, grade)*

# A vous de jouer

- Les étudiants sont caractérisés par un numéro unique, leur nom, prénom et date de naissance
- Ils passent des épreuves et obtiennent une note pour chacune
- Les épreuves sont caractérisées par un code ainsi que la date et le lieu auxquels elles se déroulent
- Chaque épreuve relève d'une matière unique
- Les matières sont caractérisées par un code et un intitulé. Chaque matière peut donner lieu à plusieurs épreuves

*Donnez le modèle de classes et le modèle relationnel correspondant à ce problème*

# Diagramme d’états - transitions

# Diagramme d’état – transition (1)

- Un diagramme d’états – transitions décrit l’évolution dans le temps d’un objet et son comportement en réponse aux interactions avec les objets qui peuplent son environnement

- Les diagrammes d'états - transitions permettent de décrire les changements d'états d'un objet, en réponse aux interactions avec d'autres objets ou avec des acteurs

- Un état se caractérise par sa durée et sa stabilité, il représente une conjonction instantanée des valeurs des attributs d'un objet

- Une transition représente le passage instantané d'un état vers un autre

# Diagramme d'état – transition (2)

- Associé à chaque classe « intéressante » du diagramme de classes
- Il permet de visualiser l'évolution d'un objet au cours de sa vie sous la forme d'un automate.
- Il est une abstraction des comportements possibles à l'image des diagrammes de classes qui sont les abstractions de la structure statique

# Diagramme d'état – transition (3)

- Chaque état possède un nom qui l'identifie
- Les états se caractérisent par la notion de durée et de stabilité. Un Objet est toujours dans un état donné pour un certains temps et un objet ne peut pas être dans un état inconnu ou non défini
- Un état est toujours l'image de la conjonction instantané des valeurs contenues par les attributs de l'objet, et de la présence ou non de ses liens avec d'autres objets

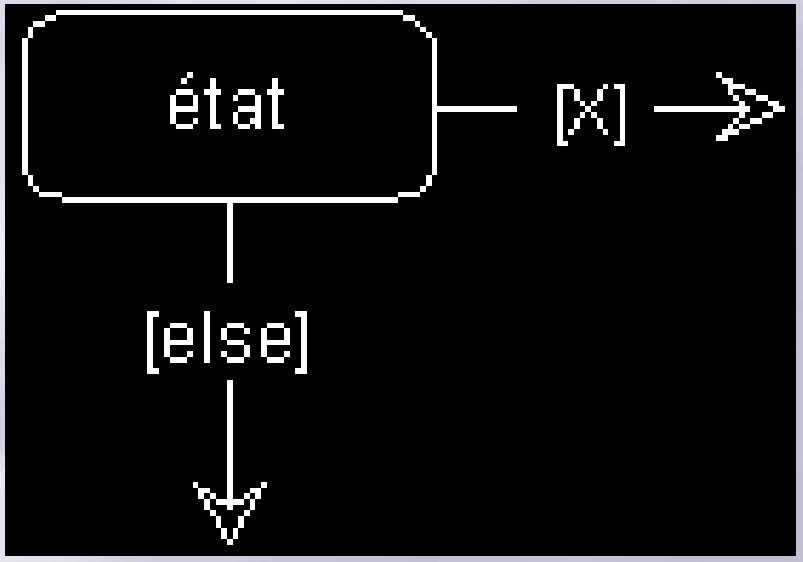
# Diagramme d’état – transition (4)

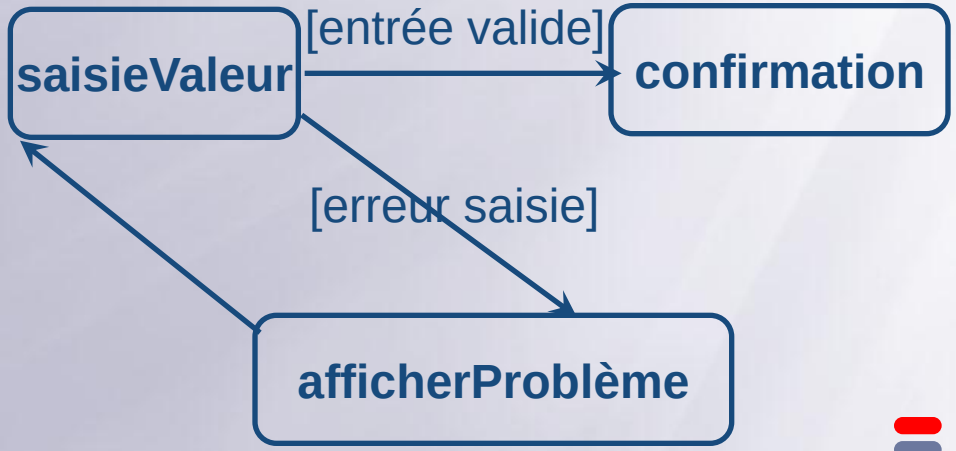
- Une transition est déclenchée par un événement. En d'autres termes : c'est l'arrivée d'un événement qui conditionne la transition
- Les transitions peuvent aussi être automatiques, lorsqu'on ne spécifie pas l'événement qui la déclenche
- En plus de spécifier un événement précis, il est aussi possible de conditionner une transition, à l'aide de "gardes" : il s'agit d'expressions booléennes, exprimées en langage naturel (et encadrées de crochets)

# Diagramme d'état – transition (6)

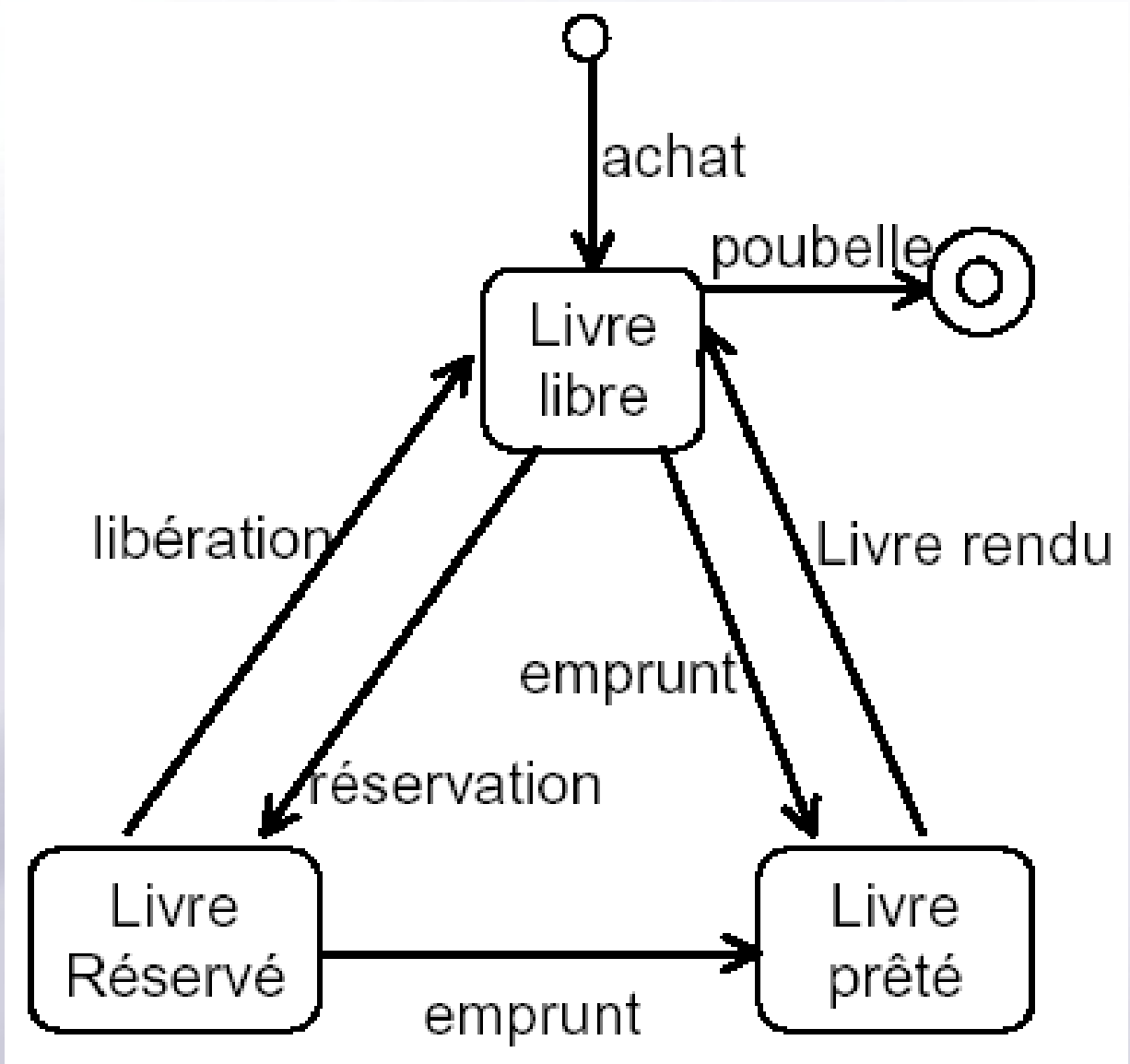
- **états, transition et événement, notation :**


- **transition conditionnelle :**



# Diagramme d'état – transition (7)



# A vous de jouer

- Représentez par un diagramme d'états – transitions les états que peu prendre un individu vis-à-vis la sécurité sociale: vivant, décédé, mineur, majeur, célibataire, marié, veuf et divorcé

- Une porte munie d'une serrure offre les opérations ouvrir, fermer, verrouiller (simple tour et double tour) et déverrouiller
  - représentez le diagramme états-transitions d'une serrure
  - représentez le diagramme états-transitions d'une porte avec verrou

# Diagramme d'activités

# Diagramme d’activités (1)

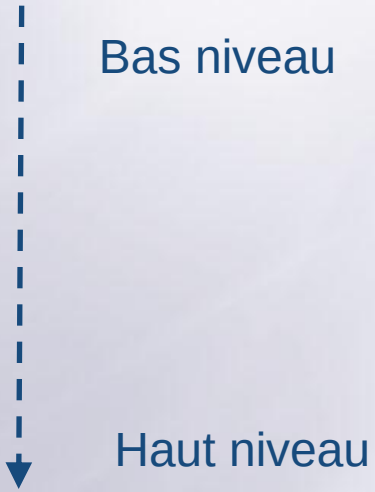
- UML permet de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation, à l'aide de diagrammes d'activités (une variante des diagrammes d'états - transitions).
- Une activité représente une exécution d'un mécanisme, un déroulement d'étapes séquentielles.
- Le passage d'une activité vers une autre est matérialisé par une transition.
- Les transitions sont déclenchées par la fin d'une activité et provoquent le début immédiat d'une autre (elles sont automatiques).

# Diagramme d’activités (2)

- Les diagrammes d’activités offrent un pouvoir d’expression très proche des langages de programmation objet
  - déclarations de variables, affectation…
  - structures de contrôles (conditionnelles, boucles…)
  - appel d’opérations, exceptions…
- Il peuvent aussi être utilisés pour des descriptions détaillées de cas d’utilisation
- En théorie, tous les mécanismes dynamiques pourraient être décrits par un diagramme d'activités, mais seuls les mécanismes complexes ou intéressants méritent d'être représentés.

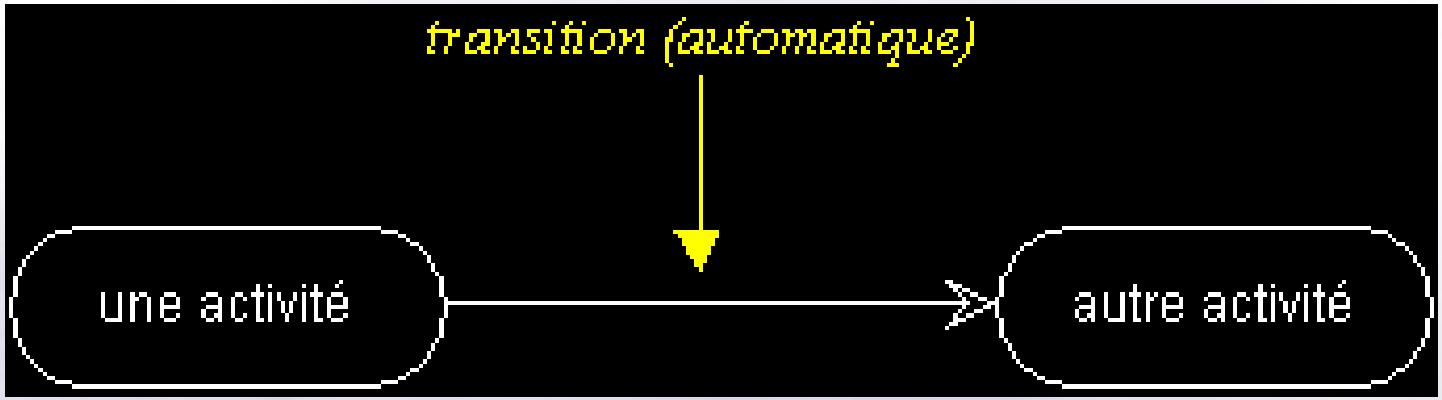
# Diagramme d'activités (3)

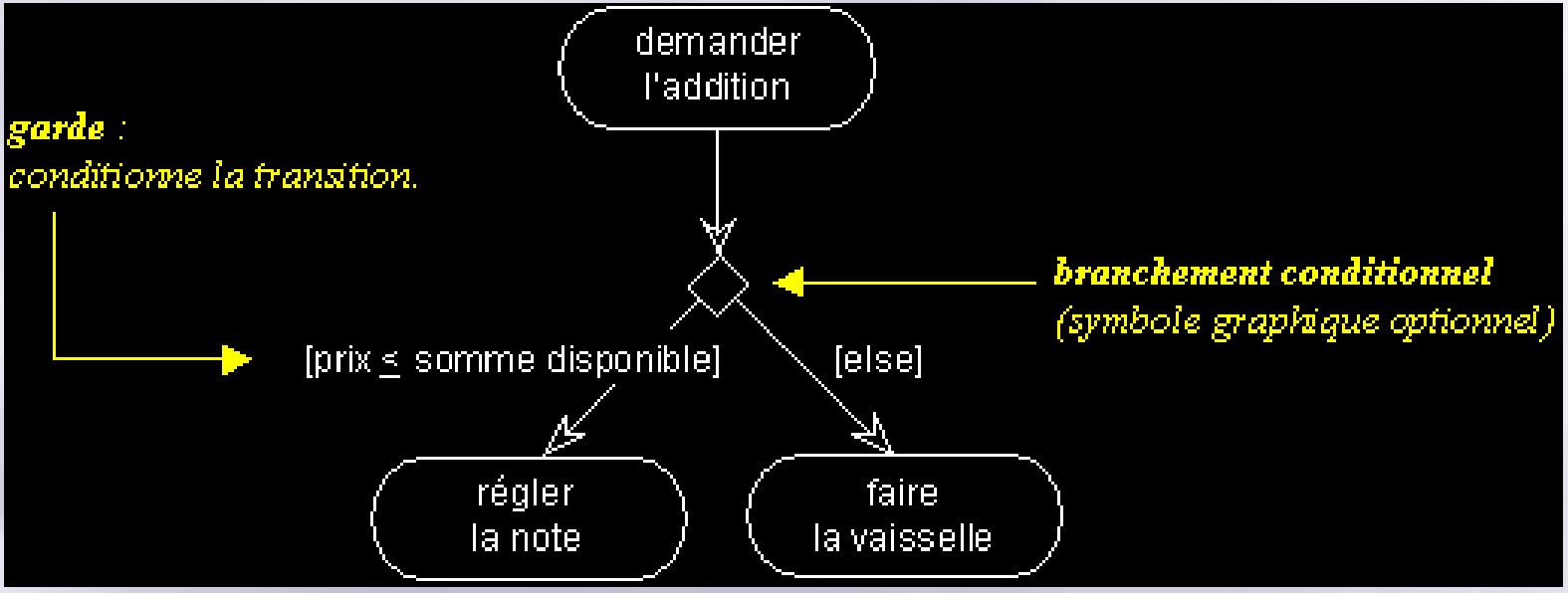
- **Les diagrammes d'activités permettent de décrire le comportement de:**
  - **une opération**
  - **une classe**
  - **cas d'utilisation**



# Diagramme d'activités (4)

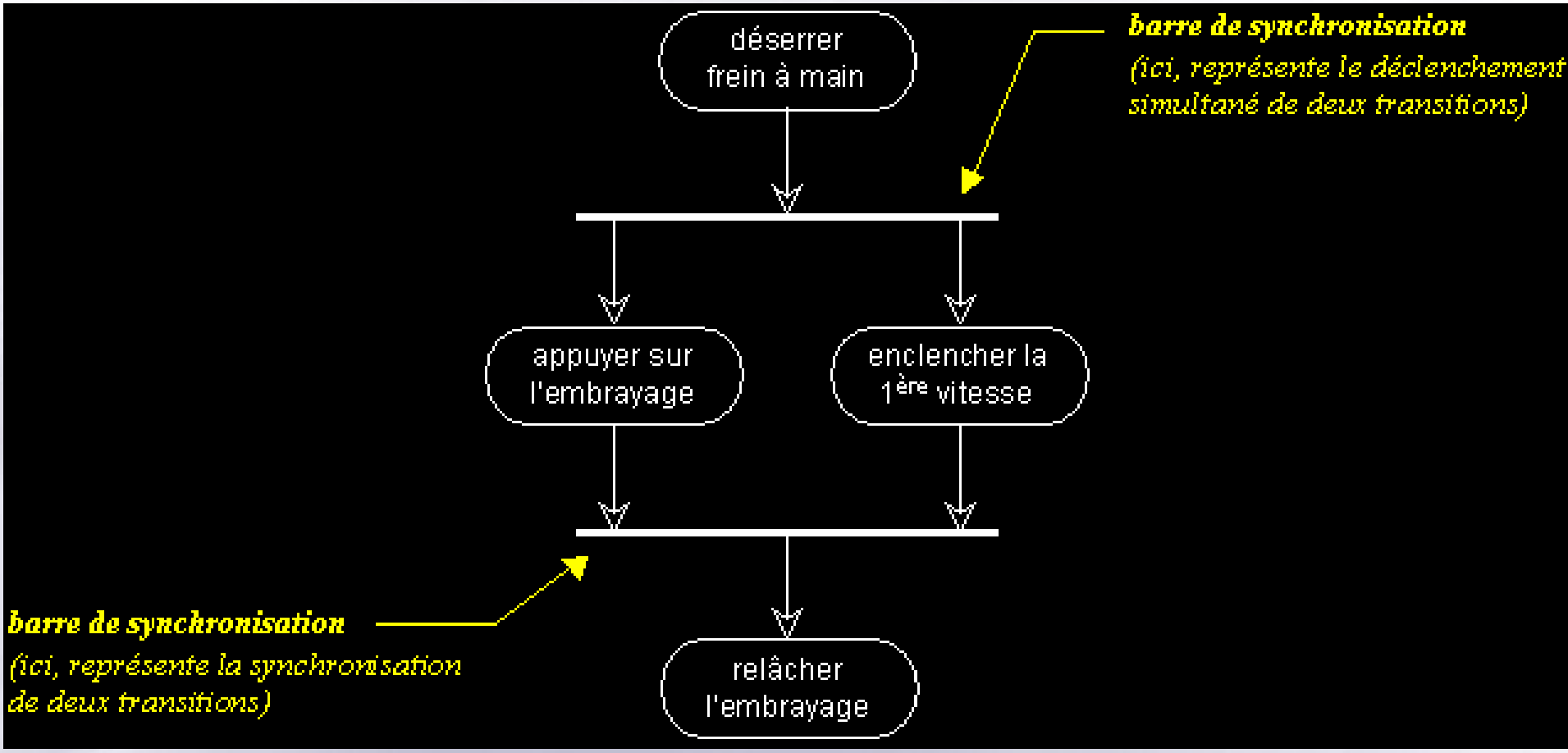
## Notations

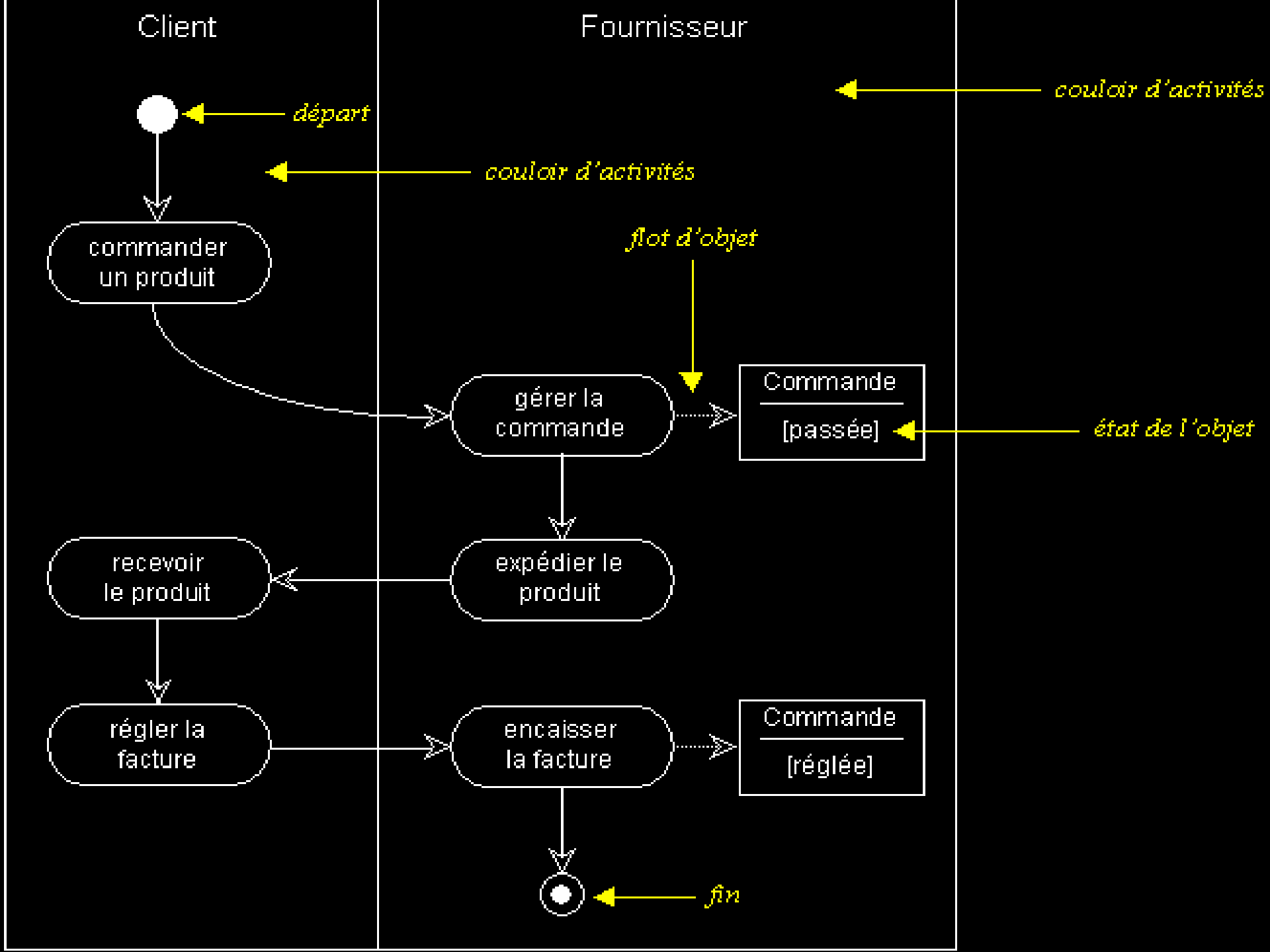


# Diagramme d'activités (5)

## Synchronisation




Client
Fournisseur
couloir d'activités
départ
couloir d'activités
commander un produit
flot d'objet
gérer la commande
Commande
[passée]
état de l'objet
recevoir le produit
expédier le produit
régler la facture
encaisser la facture
Commande
[réglée]
fin

# A vous de jouer

- **Diagramme d'activités de validation d'achat en ligne**
  - Un site de vente en ligne propose des produits placés dans un panier virtuel. Pour valider ses achats, l'utilisateur clique sur le bouton valider. On lui propose alors de se connecter à un compte existant ou d'en créer un s'il en a pas encore.
  - Pour créer un nouveau compte, l'utilisateur doit fournir une adresse de messagerie, qui sert également de login, son nom et son adresse, éventuellement une adresse de livraison et ses coordonnées bancaires. On prévoit le cas où l'adresse de messagerie est déjà associée à un compte.
  - Si la validation de ces information réussit, on propose à l'utilisateur une confirmation définitive de l'achat

# Diagramme de composants

# Diagramme de composants (1)

- Les diagrammes de composants permettent de décrire l'architecture physique et statique d'une application en terme de modules : fichiers sources, librairies, exécutables, etc

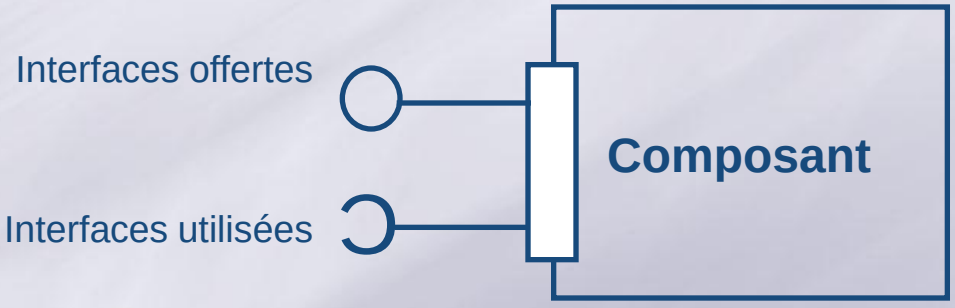
- Les dépendances entre composants permettent notamment d'identifier les contraintes de compilation et de mettre en évidence la réutilisation de composants

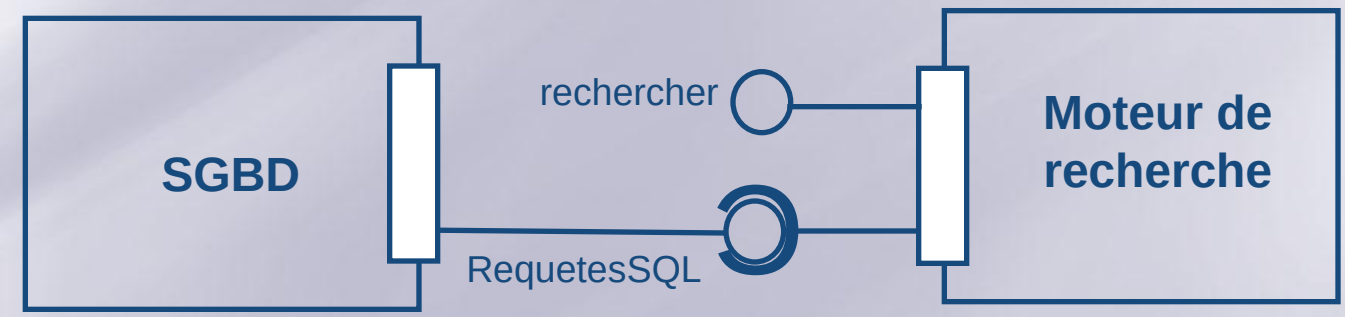
- Le composants peuvent être organisés en paquetages, qui définissent des sous-systèmes

# Diagramme de composants (2)

- **Les diagrammes de composants sont généralement utilisés pour identifier les différents modules d'un système d'information et leurs interactions**

- **Notations**



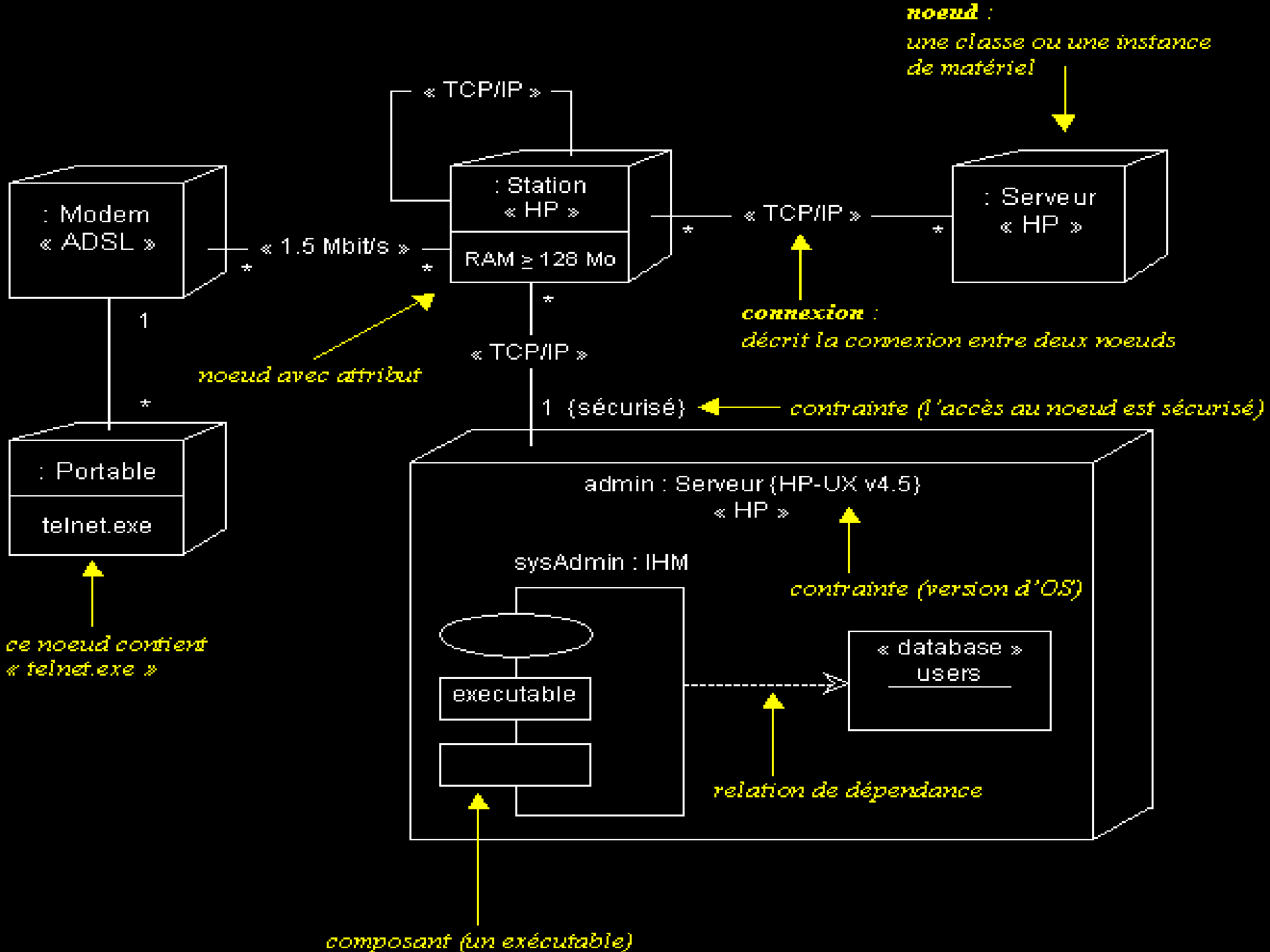
# Diagramme de déploiement

# Diagramme de déploiement (1)

- Les diagrammes de déploiement montrent la disposition physique des matériels qui composent le système et la répartition des composants sur ces matériels.
- Les ressources matérielles sont représentées sous forme de noeuds.
- Les noeuds sont connectés entre eux, à l'aide d'un support de communication.
- Les diagrammes de déploiement peuvent montrer des instances de noeuds (un matériel précis), ou des classes de noeuds.

# Diagramme de déploiement (2)

- Les diagrammes de déploiement servent à donner une idée sur l'architecture physique (matérielle) et logique (logicielle) d'un système d'information
- Ils décrivent sur quels dispositifs matériels on va déployer les composants logiciels (les différents modules de l'application)
- Les natures des lignes de communication entre les dispositifs matériels peuvent être précisées


noeud :
une classe ou une instance de matériel
« TCP/IP »
: Modem « ADSL »
« 1.5 Mbit/s »
: Station « HP »
RAM ≥ 128 Mo
« TCP/IP »
: Serveur « HP »
connexion :
décrit la connexion entre deux noeuds
noeud avec attribut
« TCP/IP »
1 {sécurisé}
contrainte (l'accès au noeud est sécurisé)
: Portable
telnet.exe
admin : Serveur {HP-UX v4.5}
« HP »
sysAdmin : IHM
contrainte (version d'OS)
« database »
users
executable
relation de dépendance
ce noeud contient « telnet.exe »
composant (un exécutable)

# A vous de jouer

- **Elaborez le diagramme de déploiement d'une application impliquant une machine (serv1) hébergeant un système de gestion de base de données (mysql), une machine (serv2) hébergeant un serveur HTTP (IIS) et une application en ASP et une machine cliente disposant d'un navigateur WEB (internet explorer). Les clients peuvent se connecter directement aux bases de données sans passer par le serveur HTTP en utilisant l'application (mysql control center) via le protocole TCP/IP.**

# La suite..

- Bon, ils deviennent nombreux ces diagrammes !!!
- On fait une synthèse,
- Puis, pourquoi pas un peu de ménage
- Et pour finir un peu d’ordre dans tout ça

# Synthèse rapide

- **Diagramme de cas d'utilisation**
  - ce qu'on attend du système
- **Diagramme de classes**
  - les entités du système
- **Diagrammes de séquence et de collaboration**
  - comment les entités interagissent
- **Diagrammes d'états - transitions et d'activités**
  - les états et les fonctionnalités de chaque entité
- **Diagrammes de composants et de déploiement**
  - l'architecture du système

# Une classification

- Vue statique du système
  - cas d'utilisation
  - classes
  - composants
  - déploiement
- Vue dynamique du système
  - séquence
  - états - transitions
  - activités

# Des redondances? Pas vraiment…

- **Diagrammes de composants et de déploiement**
  - **ils sont vraiment complémentaires (même inséparables)**
- **Diagrammes d'activités et diagramme d'états-transitions**
  - **niveaux d'expressions différents mais un diagramme d'états – transitions est généralement suffisant**

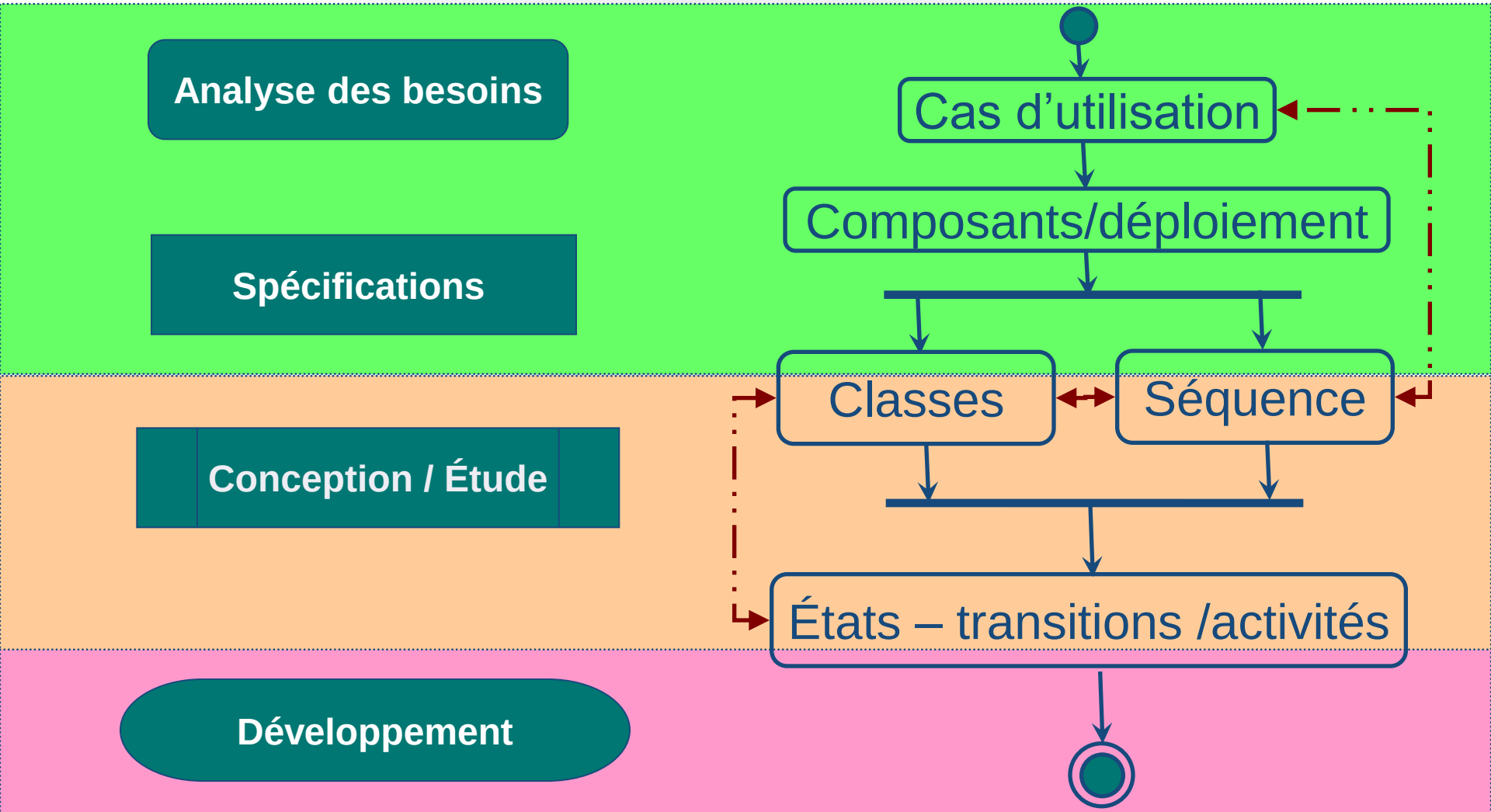
# Allez un peu de ménage, on en garde 5 seulement

- **Diagramme de cas d'utilisation**
  - **c'est là où on assimile les fonctionnalités demandées par le client**
- **Diagramme de classes**
  - **le cœur de la conception d'un système**
- **Diagramme de séquence**
  - **indispensable pour comprendre l'interaction entre les classes**
- **Diagramme états- transitions et diagramme d'activités**
  - **Toute la dynamique du système est là**

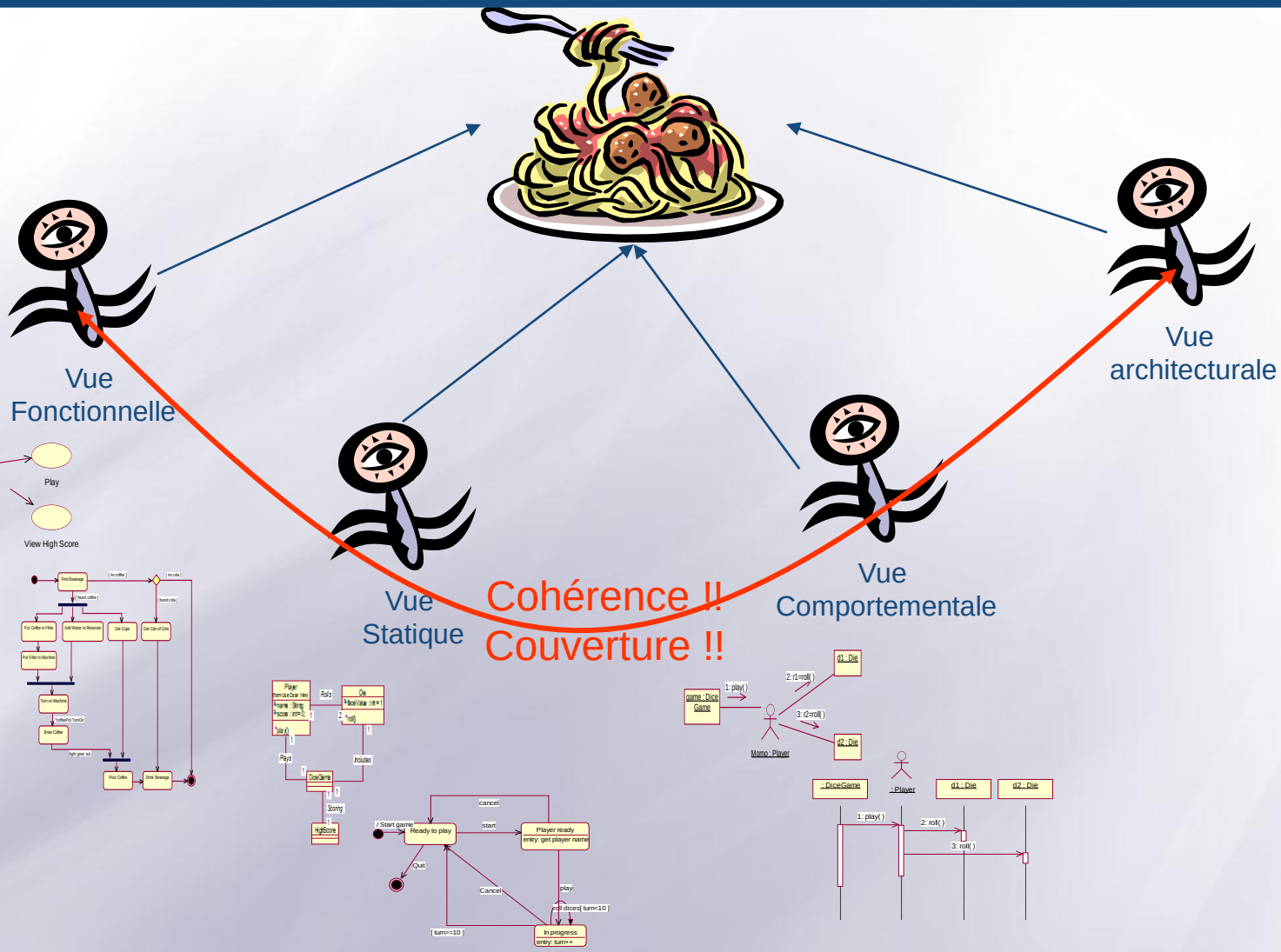
→ Mais le diagramme de déploiement reste très utile aussi !!!

# Quel diagramme avant l'autre

## Eh oui, le génie logiciel, on y revient toujours !



# Cohérence de conception (1)



# Cohérence de conception (2)

## Diagramme Use-cases/Activities

- Un diagramme de séquence représente un scénario associé à un cas d'utilisation
- Une activité doit toujours être assignable à un cas d'utilisation
- Tous les use-case doivent être réalisés dans des diagrammes de séquences (ou d'activités si on veut plus de détails)

# Cohérence de conception (3)

- Pour chaque classe se demander si son statut évolue au cours du temps ?
- Si oui, faire un diagramme d ’états - transitions
- Attention chaque transition du diagramme d’état doit être vérifiée !
- Pour chaque méthode « complexe » de la classe, penser à faire un diagramme d’activités s’il le faut

# Cohérence de conception (4)

- **Tous les objets d'un diagramme de séquence ont un type : Classe du diagramme de classe**
- **Les relations d'un diagramme de séquence doivent exister ou pouvoir être dérivées du diagramme de classe !**
- **Les messages échangés sont des méthodes du diagrammes de classes !**

# Est-ce suffisant?

- NON !!!
- Bonne documentation
- Utiliser des noms significatifs pour les éléments de conception
- On peu mettre autant de détails qu'on veut
  - c'est la force d'expression de UML !!!
- Clé de réussite = KISS (Keep It Small and Simple)
- Experience is the best teacher

# Des outils

- **Un éditeur très primitif mais gratuit**
  - UMLPAD: http://web.tiscali.it/ggbhome/umlpad/umlpad.htm
- **Quelque chose de plus évoluée**
  - VISIO: http://office.microsoft.com/fr-fr/FX010857981036.aspx
- **Un atelier de génie logiciel**
  - WINDESIGN http://www.win-design.com/fr/index.htm
- **L'un des meilleurs Atelier de génie logiciel les plus complets**
  - RATIONAL ROSE www.rational.com

# Bibliographie

- Absolument tout sur UML: www.uml.org
- IBM & Rational: www.rational.com/uml
- UML en français uml.free.fr
- Modélisation objet avec UML, *Pierre-Alain Muller*, *Eyroles*, *430 pages*.
- UML in a nutshell, *S. Alhir, oreilly, 234 pages.*