

# MÉTHODES ET OUTILS DE DÉVELOPPEMENT LOGICIEL



## Chapitre 1

# INTRODUCTION AUX DÉMARCHES DE DÉVELOPPEMENT LOGICIEL

**EN INSISTANT ON OBTIENT TOUJOURS CE QU' ON MÉRITE**



Ce qu'a promis  
l'ingénieur commercial



Ce que voulait  
le client



Ce qu'a spécifié  
le client



Ce qu'a compris  
le chef de projet



Ce qu'a compris  
le concepteur



Ce qui a été livré  
Version 1



Ce qui fonctionne  
actuellement  
Version 1 + patch  
Rebaptisée Version Béta

# DÉVELOPPEMENT LOGICIEL

## DÉMARCHES

- Démarches de recueil des besoins, d'analyse et de conception.
- Méthodes de construction et d'obtention de la qualité.

## OUTILS

- Outils de modélisation,
- Environnements de développement, AGL, prototypage, gestion de versions et configurations.
- Langages.



# GÉNIE LOGICIEL



# DÉFINITIONS

## OUTIL

- UML : Unified Modelling Language, unification (std OMG) des notations de modélisation objet (J. Rumbaugh, G. Booch, I. Jacobson).



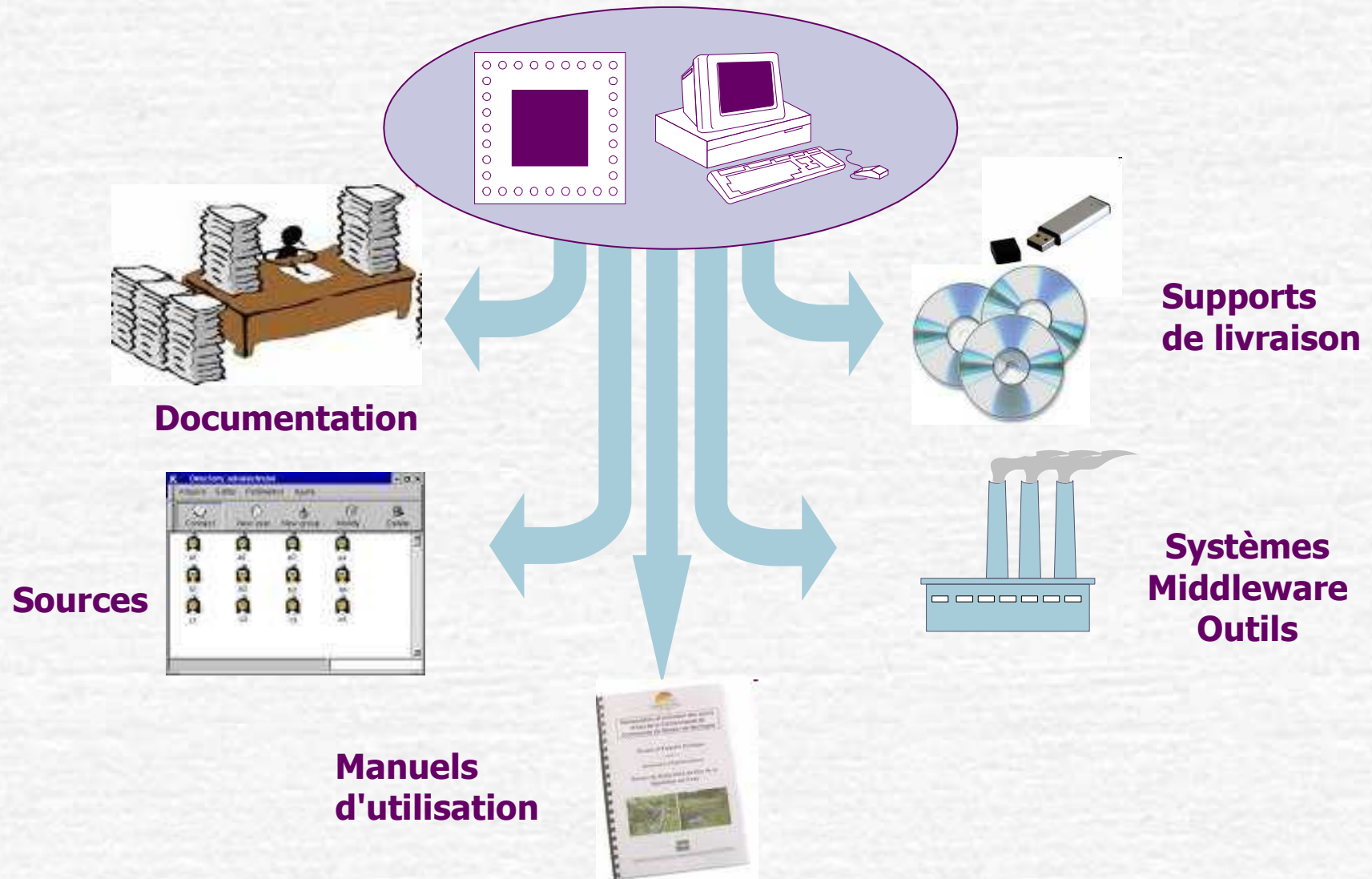
## DÉMARCHES

- UP : Unified Process (Unified software Development Process) : une démarche générique de développement de logiciel.
- SCRUM, XP (eXtreme Programming) : dérivées de UP, développement itératif et adaptatif par étapes courtes (méthodes agiles...).





# UN LOGICIEL CE N'EST PAS QUE DES PROGRAMMES



# GÉNIE LOGICIEL : LES ADVERSAIRES



les décideurs

Contrats



Besoins



les utilisateurs

Logiciel



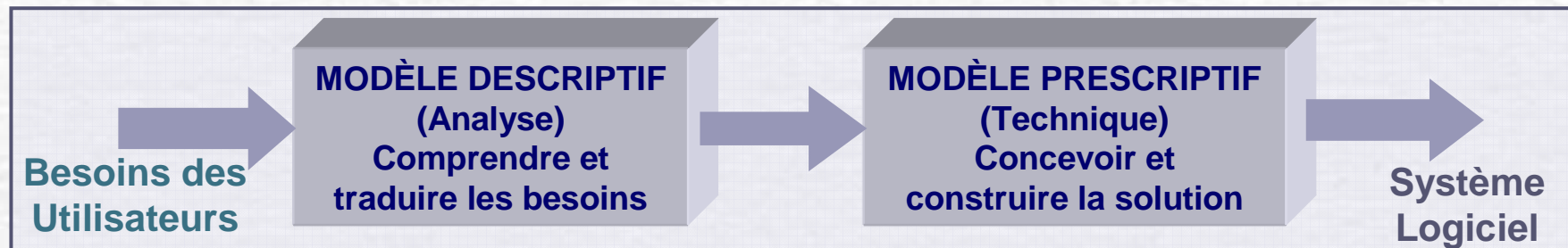
Besoins



l'informaticien



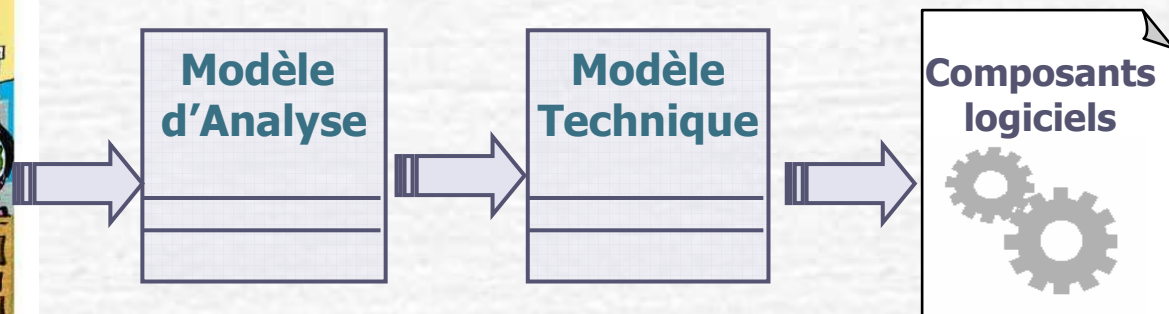
# DEMARCHE DE DEVELOPPEMENT



## Le monde est rempli de choses



Le Comptoir du Dessin



Christine FORCE

# PROCESSUS DE DÉVELOPPEMENT LOGICIEL (Cycle de vie des applications)

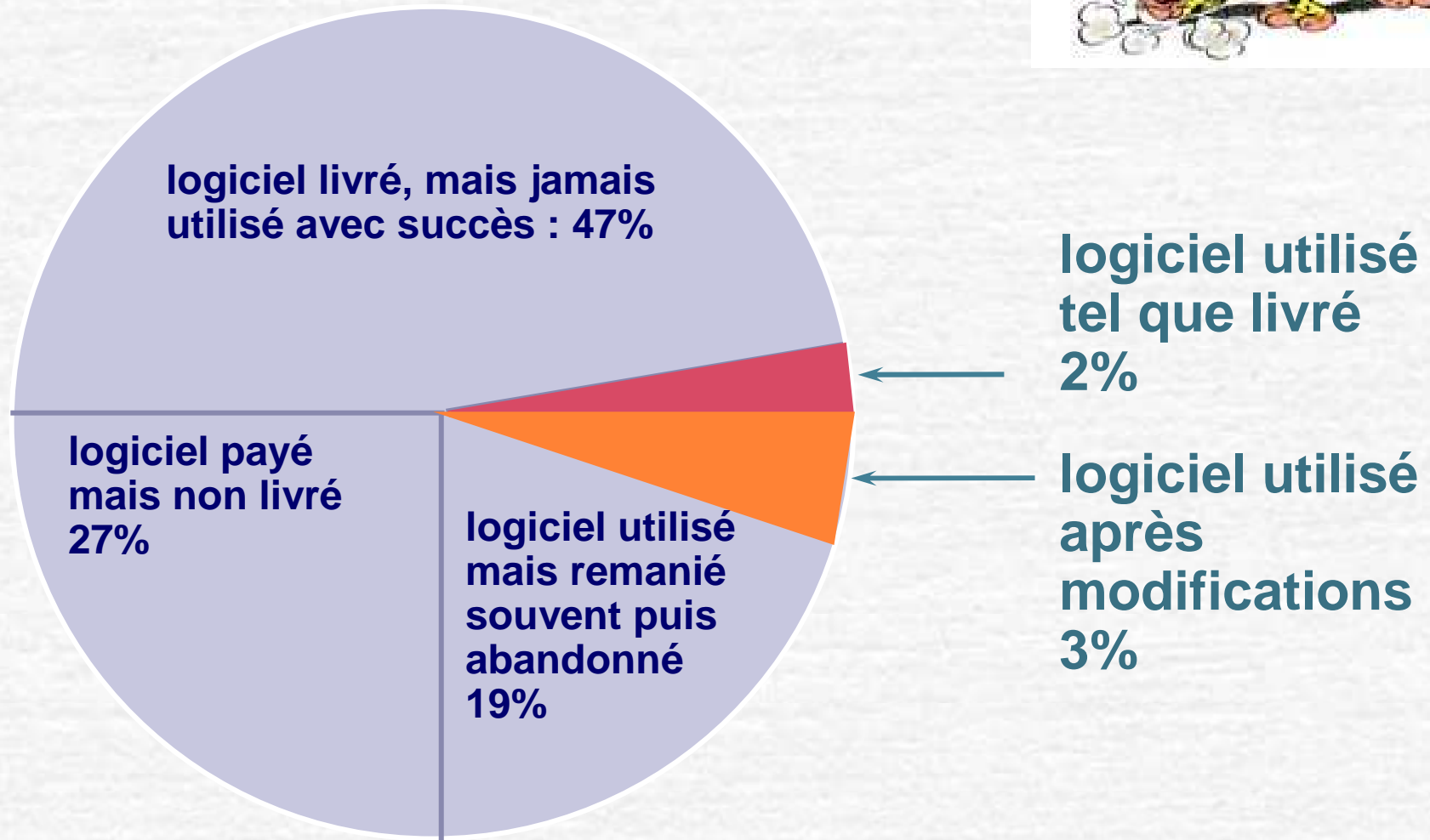


**Processus (démarche) : description des différentes manières d'organiser les activités du développement logiciel.**

- Définir l'ordre des travaux pour un projet,
- Spécifier les modèles et documents (artefacts) à développer et leurs échéances,
- Attribuer les responsabilités et les rôles dans les équipes,
- Permettre de suivre et évaluer les produits et les activités du projet.



## LA CRISE !



# LA NOUVELLE CRISE



## • Première crise : 1965

### • Solutions? (NATO Summer School 1967) :

- « Software engineering »
- « Structured programming »
- « Top-down approaches »

## • Seconde crise : 2000

### • Solutions? (OMG novembre 2000) :

- « La Programmation Objets ou la Programmation par Composant ne constituent que des réponses très partielles à la montée en complexité. Seules elles sont aujourd'hui insuffisantes ».
- « MDA : Decouple neutral business models from variable platforms »
- « Transformations as assets »



# LES MÉTHODES DE CONCEPTION DE SYSTÈME

- On distingue deux grandes familles de méthodes :
  - Les méthodes procédurales ou structurées :
    - Séparation des données (base de données) et des traitements qui les manipulent (les programmes)
  - Les méthodes orientées objets :
    - Intègrent au sein d'un objet (instance d'une classe) les données (attributs) et les comportements (méthodes ou opérations).
    - Les méthodes objets ont remplacé progressivement les méthodes dites structurées.
    - Elles apparaissent à partir des années 1988.





## **LES MÉTHODES ORIENTÉES OBJET**

### **Chronologie des méthodes**

- ❧ **OOA: Object Oriented Analysis - 1979 - Shlaer et Mellors**
- ❧ **OOD: Object Oriented Design - 1981 - Booch**
- ❧ **OMT : Object Modeling Technique - 1987 - Rumbaugh**
- ❧ **HOOD : Hierarchical Object Oriented Design - 1987 - Agence Spaciale Européenne**
- ❧ **OOSE : Object Oriented Software Engineering - 1990 - Jacobson**
- ❧ **OOA/OOD : Object Oriented Analysis/Object Oriented Design – 1991 - Codd et Yourdon**
- ❧ **OOM : Orientations Objet dans Merise-1993-Rochfeld**

## TOUT N'EST PAS SI SIMPLE...



- Même si les concepts objets sont stables et éprouvés,
- Et qu'ils bénéficient d'outils et de langages performants,
- Même si l'approche objets permet de concevoir les logiciels complexes et résistants aux évolutions :
  - L'approche objets est moins intuitive que l'approche fonctionnelle,
  - L'application des concepts nécessite une très grande rigueur,
  - Les langages objets ne guident pas l'utilisation des concepts,
  - Comment distinguer un "bon objet" d'un mauvais ?

## UML , UN LANGAGE POUR :

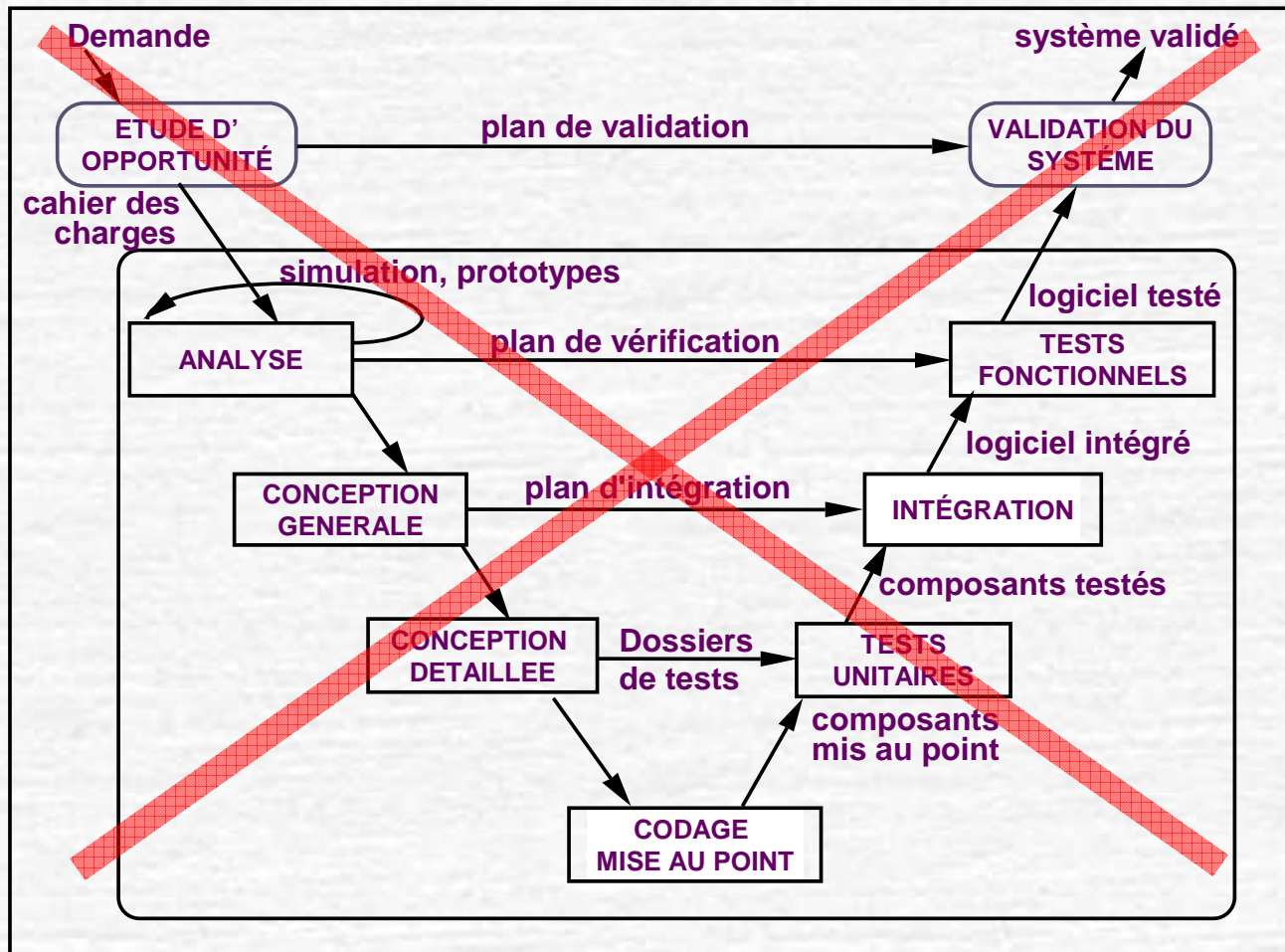
- Décrire, visualiser et comprendre le problème,
- Capturer et utiliser des connaissances pour la résolution du problème,
- Communiquer avec les utilisateurs et les experts des autres disciplines
- Spécifier, montrer et construire la solution,
- Documenter la solution,



**Quel que soit le domaine d'application.**



# CYCLE DE VIE EN V DU LOGICIEL



Dans ce modèle le principe est simple : chaque phase se termine à une date précise par la production de certains documents ou logiciels. Les résultats sont soumis à une revue approfondie, on ne passe à la phase suivante que s'ils sont jugés satisfaisants. Une étape ne remet en cause que la précédente, ce qui s'avère insuffisant en pratique.

## CYCLE EN V

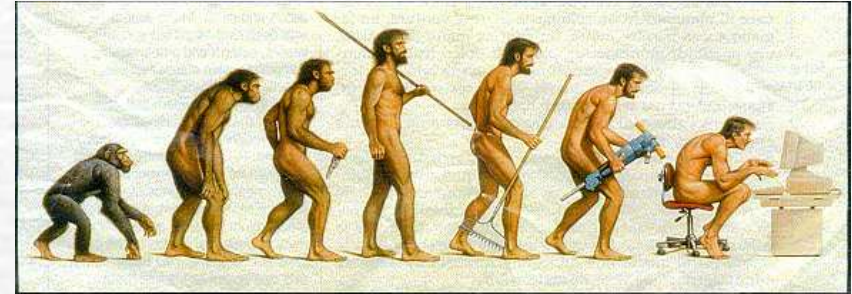
### ☛ Pour :

- Organisation simple et directe,
- Décomposition du système en sous-systèmes,
- Vérification ascendante.

### ☛ Contre :

- But de chaque étape = fabriquer un produit intermédiaire (document papier), soumis à évaluation et utilisé en l'état comme point de départ pour l'étape suivante.
- ⇒ Nombreux documents créés dans un environnement bureaucratique,
- Rigidité : planification à long terme détaillée,
- Hypothèse erronée : les exigences sont stables,
- Validation en fin de cycle (erreurs coûteuses),
- **Effet tunnel** (le client ne voit rien avant la fin de la réalisation).

# ÉVOLUTIONS



Somewhere, something went terribly wrong

- ❧ Processus incrémental : développer par étape, en commençant les fonctions clés.
- ❧ Processus itératif : réaliser les phases du développement en plusieurs itérations.
- ❧ Prototypage : valider les besoins en permettant à l'utilisateur de se faire une idée de ce que sera le produit final.
- ❧ Management des exigences : gestion des changements des besoins des utilisateurs.
- ❧ Réutilisation de composants : code, conceptions, descriptions, documents.
- ❧ Gestion des risques : prévoir et éliminer (diminuer) les risques d'échecs ou de dysfonctionnement.



# PROCESSUS UNIFIÉ EN VITESSE

## LES ACTIVITES



**SPECIFICATION  
des exigences**

Qu'est-ce qu'ILS veulent  
faire avec ce logiciel ?

**ANALYSE  
du domaine**

Avec quoi ILS travaillent ?

**DEPLOIEMENT**

Comment  
JE vais faire  
ce logiciel ?

**CONCEPTION  
de l'architecture**

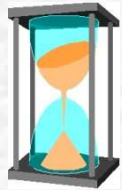
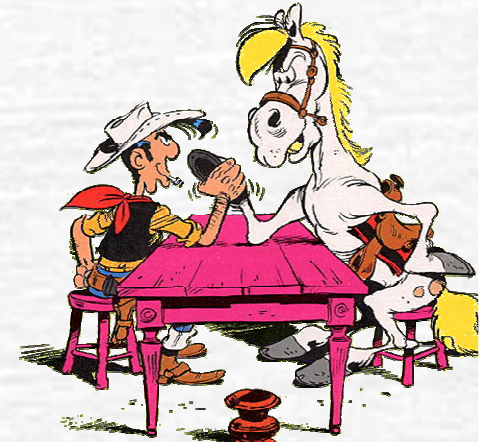
**TESTS**

Super, ça marche !

Chic, JE programme !

**IMPLEMENTATION**

# PROCESSUS UNIFIE LES PHASES



(s o u s) P R O J E T

	Inception (étude d'opportunité)	Élaboration	Construction	Transition
Exigences	★	★★★	★	
Analyse	★	★★★	★	
Conception	★	★★	★★★	★
Implémentation	★	★★	★★★	★★
Test	★	★	★★★	★★

Comportement global   Raffinement de l'architecture   Satisfaction des utilisateurs

# MÉTHODES AGILES

## (Scrum, eXtreme Programming, RAD)



- Processus plus léger et flexible,
- Focalisé sur la réalisation,
- Avec des pratiques réduisant les coûts du changement,
- Basé sur l'expérience des chefs de projet :
  - Travail en équipe,
  - Livraisons fréquentes,
  - Communication accrue avec le client,
  - Relecture et nettoyage du code.
- Simplicité avec pour règle : "Y're not gonna need it !"
- Boutons (des pratiques actuelles) tournés jusqu'à 10.



## Chapitre 2

# UML: DIAGRAMMES DE CLASSES



# DIAGRAMMES DE CLASSES

**SPECIFICATION  
DES BESOINS**

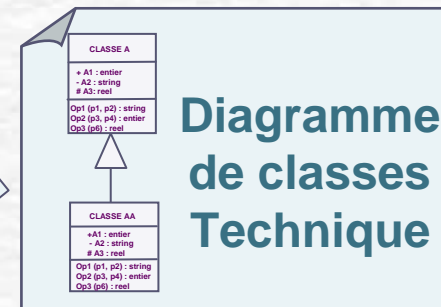
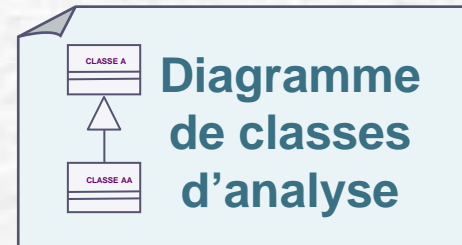
**ANALYSE**

**CONCEPTION  
de l'architecture**

**IMPLEMENTATION**

**TESTS**

Comme la plupart des modèles UML, les diagrammes de classes s'utilisent à 2 niveaux



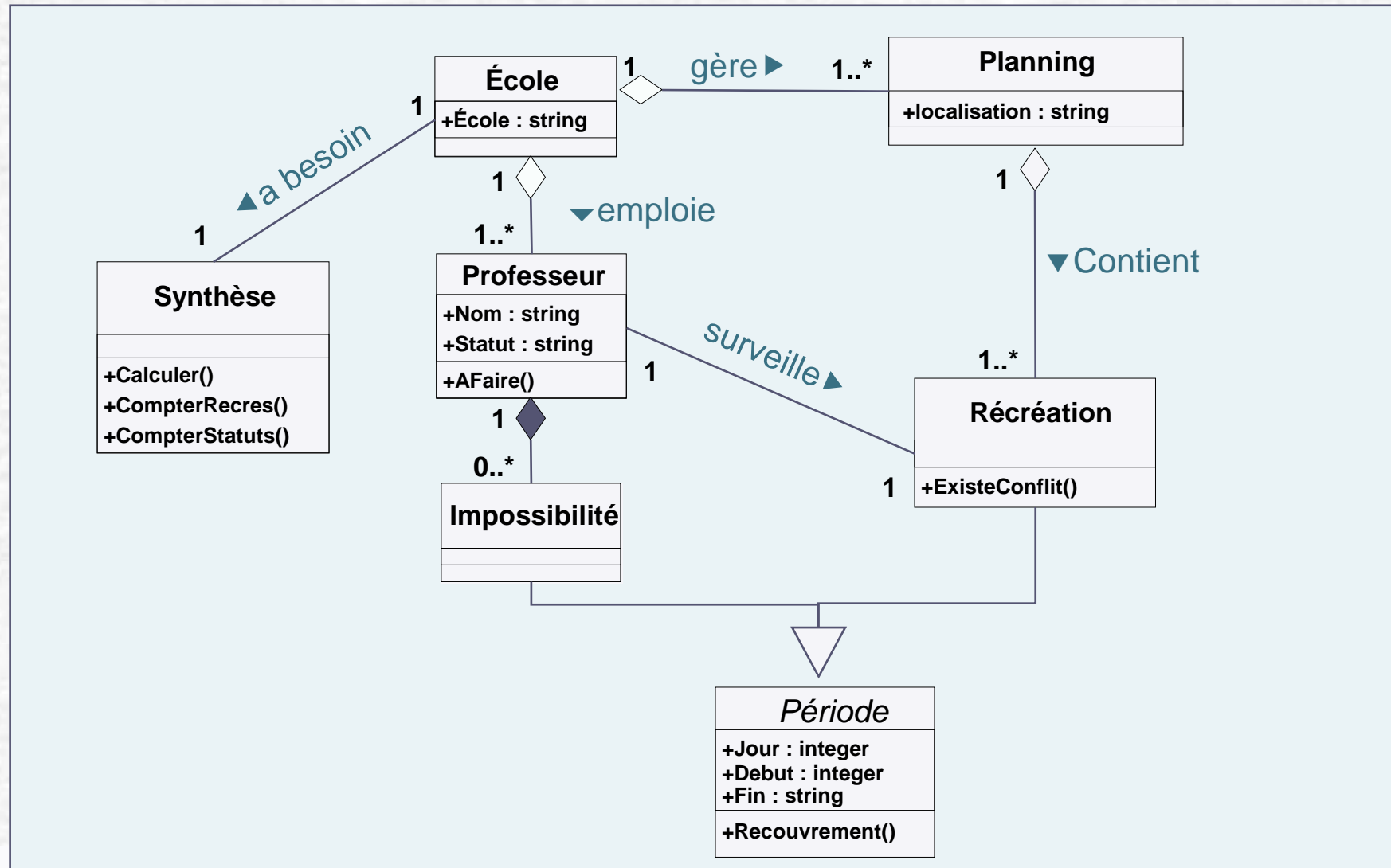
## DIAGRAMME DE CLASSES D'ANALYSE



- ❧ Décrire de façon abstraite un ensemble d'objets,
- ❧ Factoriser des éléments communs à un ensemble d'objets,
- ❧ Décrire le domaine de définition d'un ensemble d'objets,
- ❧ Identifier les concepts : dresser une liste de choses ou d'idées présentes dans l'environnement (le métier) des utilisateurs,
- ❧ Décrire les relations entre ces concepts (inclusion, utilisation, communication, dérivation...).



## EXEMPLE : PLANIFICATION

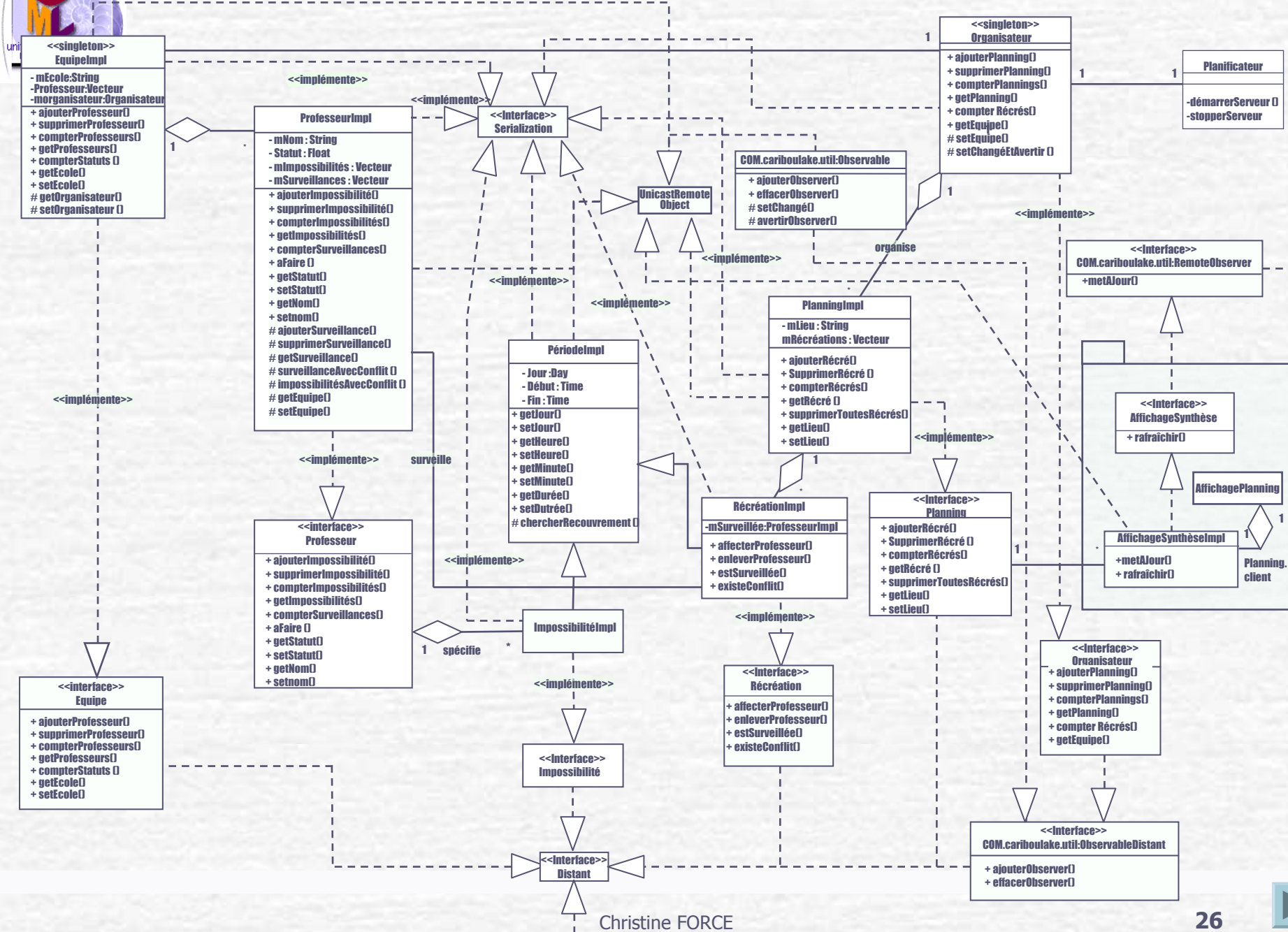


## DIAGRAMME DE CLASSES TECHNIQUE



- ☞ Il traduit le diagramme du domaine en architecture du logiciel.
- ☞ Il est beaucoup plus détaillé (cf. page suivante).
- ☞ Il donne des informations techniques (méthodes, attributs)
- ☞ Il contient souvent plusieurs modèles :
  - Modèle d'Interface Humain Ordinateur (présentation).
  - Modèle application (objets de contrôle, pilotage).
  - Modèle métier (les objets provenant de l'analyse).
  - Modèle d'accès aux données (requêtes).
  - Modèle des classes persistantes (modèle relationnel ?).
- ☞ Voir un exemple en fin de l'étude de cas

# SYSTÈME DE PLANIFICATION : DIAGRAMME DE CLASSES IMPLÉMENTÉ





# PLAN DU CHAPITRE



- ❧ I - CLASSES D'ANALYSE
- ❧ II - CLASSES TECHNIQUES
- ❧ III - EXERCICES
- ❧ IV - DIAGRAMMES D'OBJETS

# I - CLASSES D'ANALYSE REPRÉSENTATION



Classe
attributs
opérations

en analyse on peut indiquer  
(mais on ne le fait pas toujours)  
les attributs et les opérations  
fondamentaux de la classe.

Gaulois
nom fonction dateNaissance
combattre

exemple

Il s'agit de décrire les objets du métier de l'utilisateur,  
avec le point de vue et le vocabulaire de cet utilisateur.

But : comprendre comment le client travaille et  
communiquer avec lui (pour vérifier que l'on a bien compris).

# ASSOCIATIONS



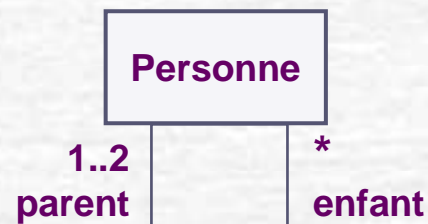
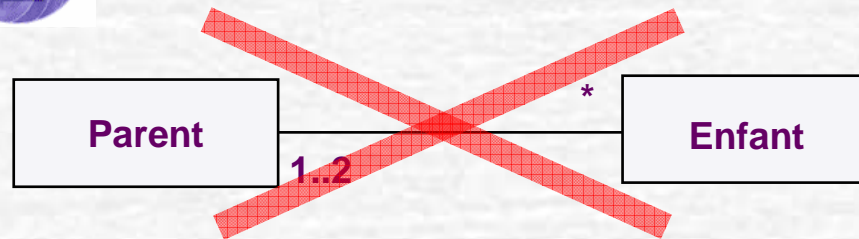
## Au choix : nommage de l'association ou nommage des rôles (extrémités)

Le nom de l'association apparaît en italique sur la ligne qui la symbolise, l'usage recommande de choisir une forme verbale active (*travaille pour*) ou passive (*est employé par*). Le sens de lecture du nom est indiqué par un triangle ◀ ▶ ou un signe < ou >.

Le rôle décrit comment une classe voit une autre classe, au travers d'une association (forme nominale).

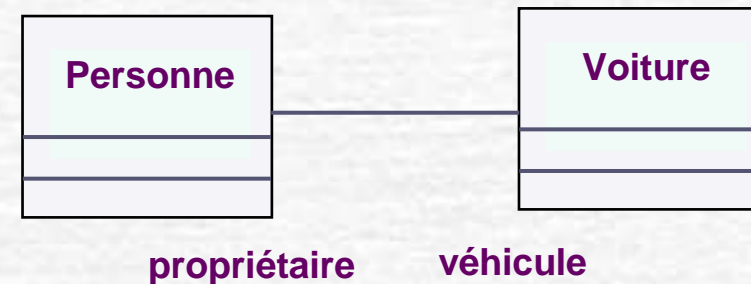
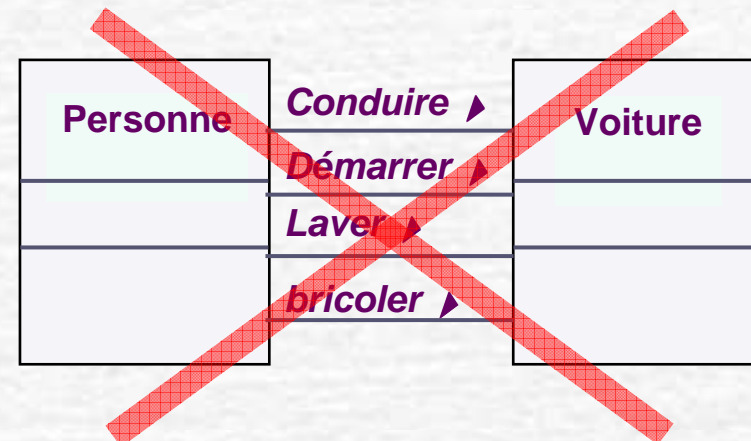
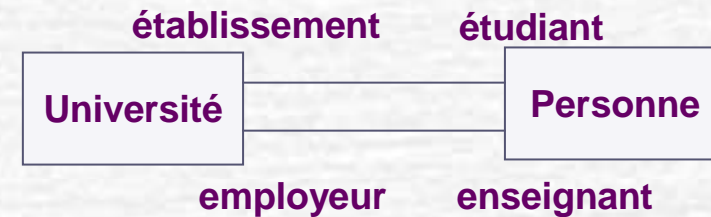


# ASSOCIATIONS



Association réflexive

On peut avoir plusieurs associations entre 2 classes à condition qu'elles représentent des concepts différents. Le nommage des rôles prend tout son intérêt lorsque plusieurs associations relient deux classes.

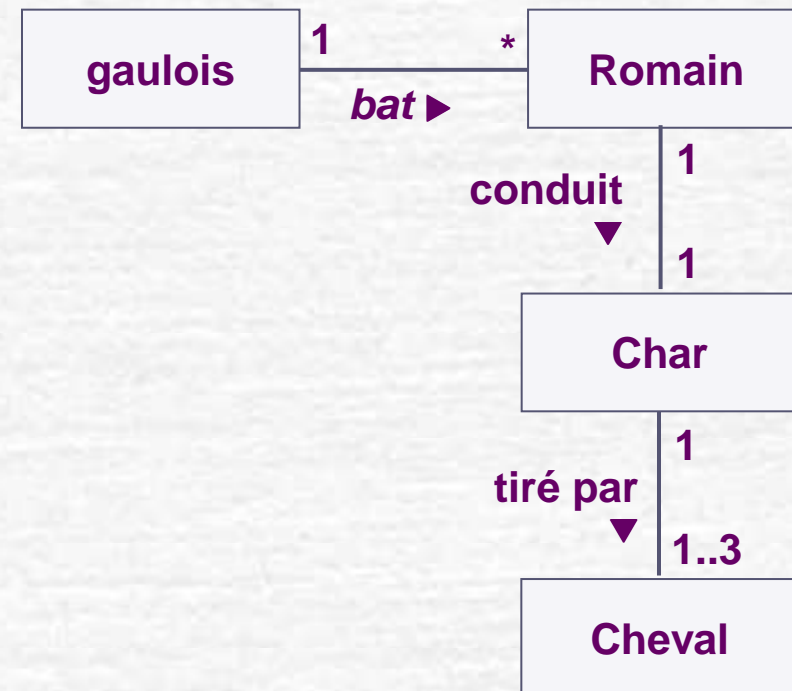


# MULTIPLICITES



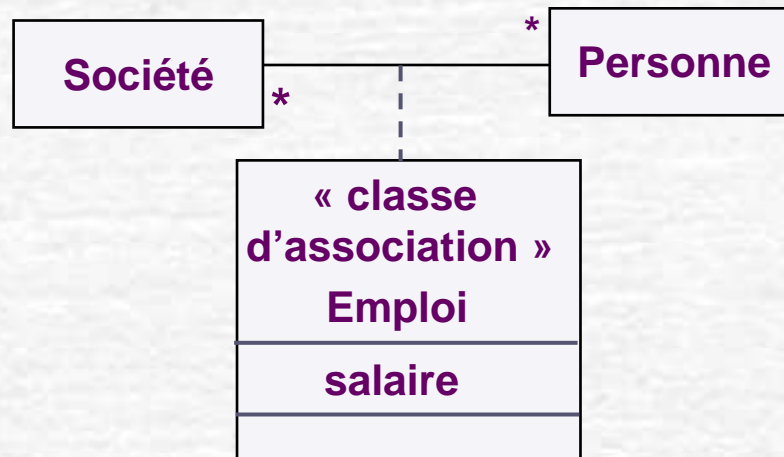
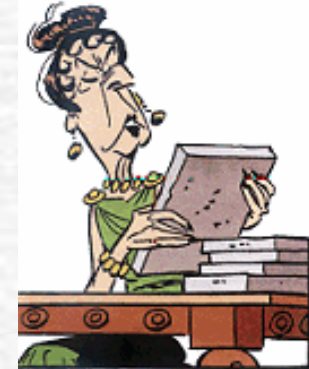
1	Classe	Exactement 1
*	Classe	Plusieurs (de 0 à n)
1..*	Classe	De 1 à n
0..1	Classe	optionnel
m..n	Classe	Spécifié numériquement

## Multiplicités



## Exemple

# CLASSES D'ASSOCIATION



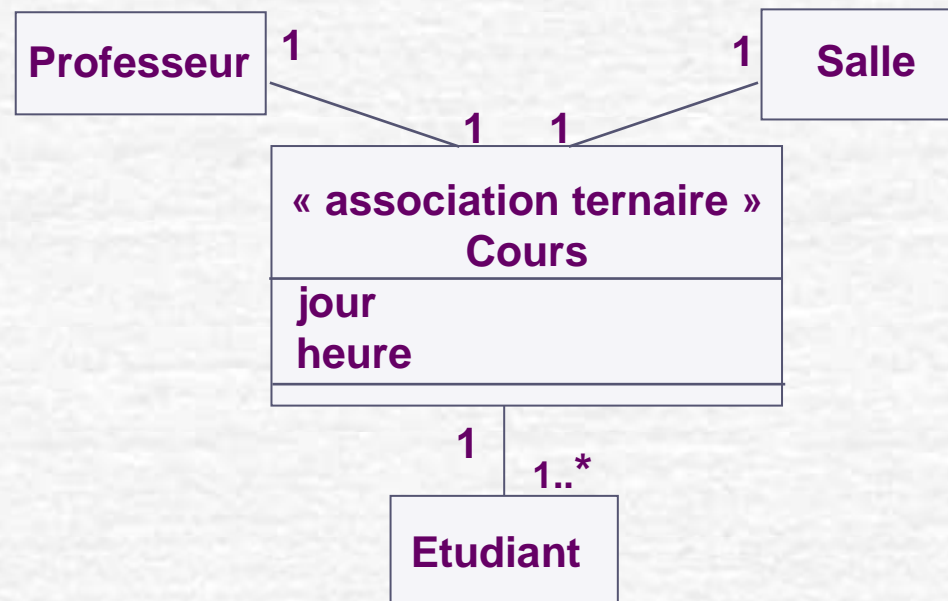
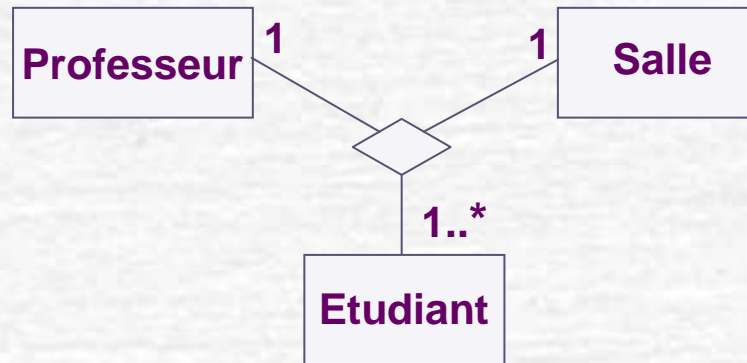
Une société emploie plusieurs personnes. Une personne peut être employée par plusieurs sociétés. L'attribut "salaire" est porté par l'association

## Exemple de classe d'association

Les associations plusieurs à plusieurs (\*-\*) se réifient par des classes d'associations qui permettent de décrire des attributs caractérisant le lien (elles ne modélisent pas des objets).



# ASSOCIATIONS TERNAIRES



Les associations ternaires sont souvent très ambiguës, mais elles servent pour esquisser le modèle, par exemple au début de l'analyse. Il faut les transformer en plusieurs associations binaires (en créant des classes d'association).

# ASSOCIATIONS

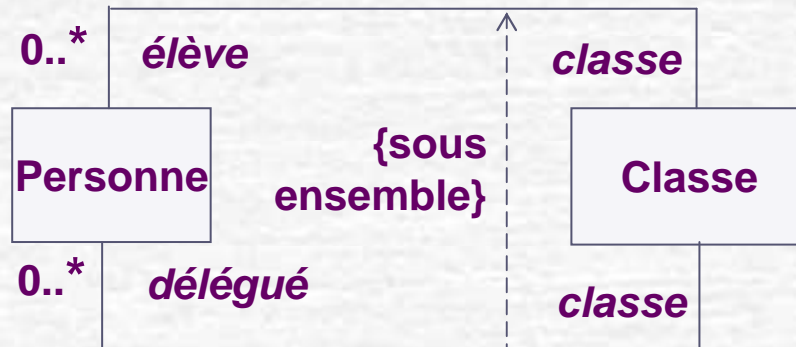
Les contraintes expriment les règles de validité, de cohérence ou de sémantique. Une contrainte est une expression entre accolades :

*{description de la contrainte}.*

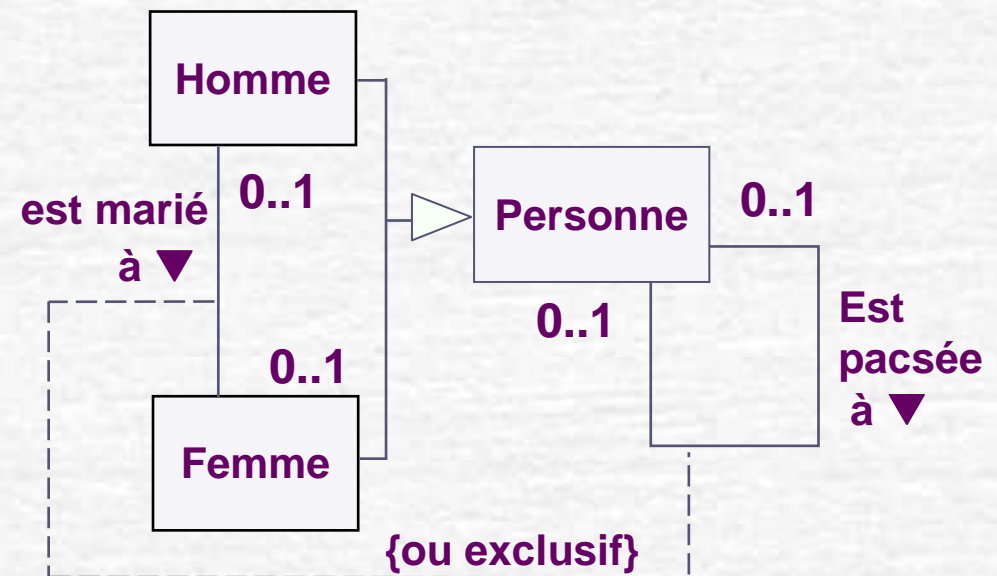
Il existe des contraintes prédéfinies par UML :

ordonné, sous-ensemble, ou exclusif (xor).

OCL (Object Constraint Language) permet de décrire des contraintes complexes.



les délégués sont aussi des élèves



un personne peut être mariée ou pacsée (mais pas les 2)

## ASSOCIATIONS

- Remplacer une association (agrégation) par un attribut relève de l'implémentation, non de la modélisation.
- On évitera de le faire en analyse pour :
  - Préserver le caractère bidirectionnel de l'association.
  - Révéler les cardinalités.
  - Faciliter la compréhension du modèle par les non spécialistes.
  - Préserver l'aspect visuel convivial de la représentation.
  - Identifier, grâce au diagramme, l'impact que peut avoir le retrait d'une classe.
- Critère : si l'on ne peut demander à un élément que sa valeur il s'agit d'un simple attribut, si l'on peut lui poser plusieurs questions, c'est un objet.

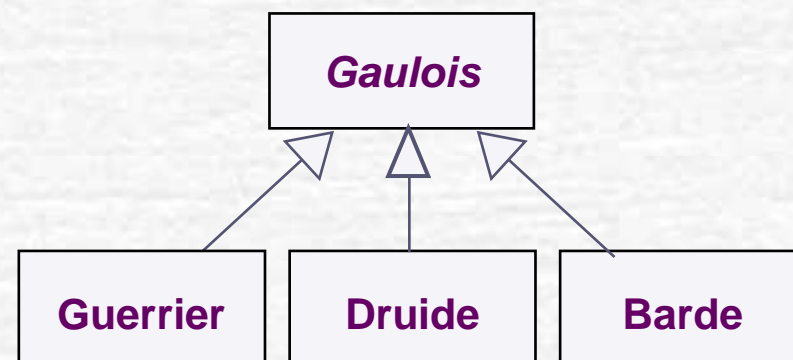
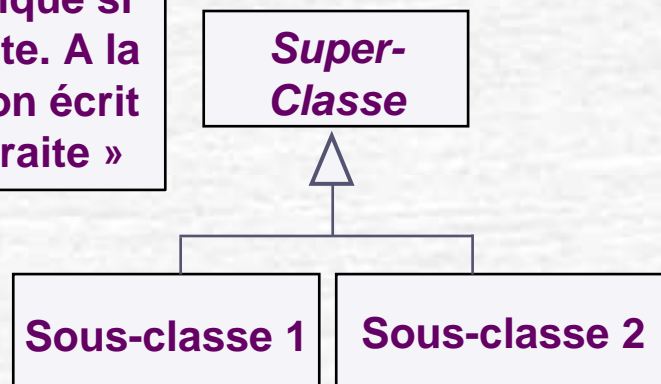


# GÉNÉRALISATION/SPÉCIALISATION

UML emploie le terme de généralisation pour désigner la relation de classification entre un élément général et un élément plus spécifique. La généralisation est souvent réalisée en utilisant la relation d'héritage des langages objets. C'est une manière de réaliser la classification, mais ce n'est pas la seule. La généralisation UML est plus abstraite que l'héritage.

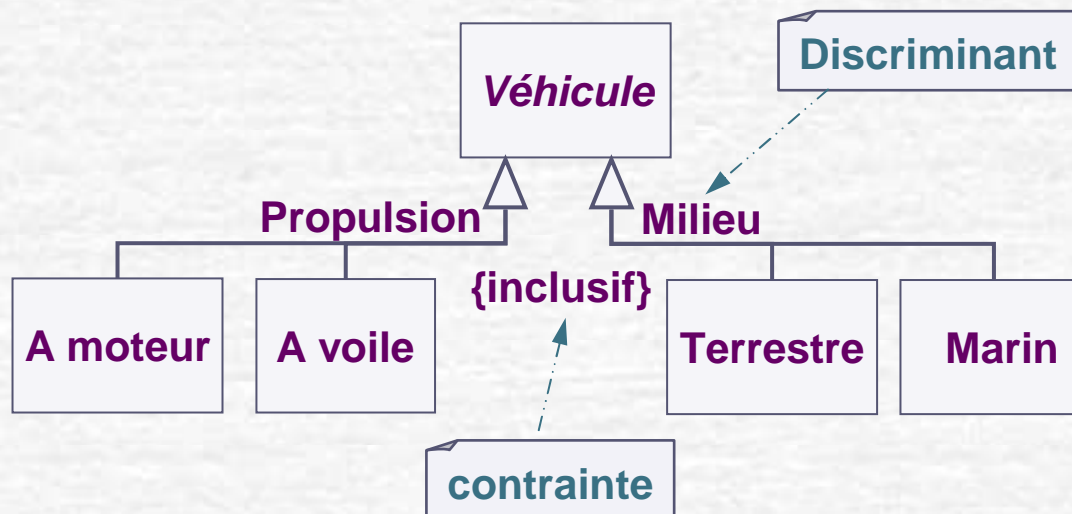


En italique si abstraite. A la main, on écrit « abstraite »

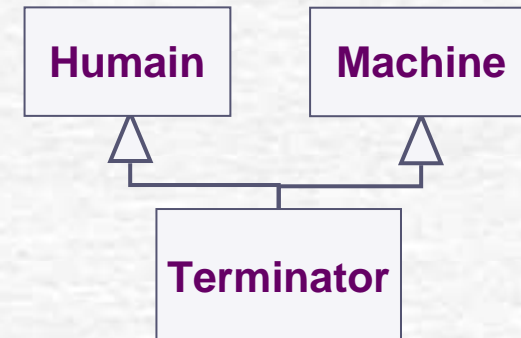


# CLASSIFICATION/SPECIALISATION MULTIPLE

Une classe peut être spécialisée selon plusieurs critères simultanément.  
Chaque critère de la généralisation est indiqué par un discriminant.  
Il permet de spécifier les combinaisons cohérentes de sous classes.  
Plusieurs sous-classes peuvent partager un même discriminant, elles sont alors disjointes. Une instance dérivée d'une super-classe ne peut être instance que d'une seule sous classe avec le discriminant commun



classification multiple :  
un objet est instance de plusieurs classes



Spécialisation multiple :  
une classe hérite de plusieurs classes

# CONTRAINTES

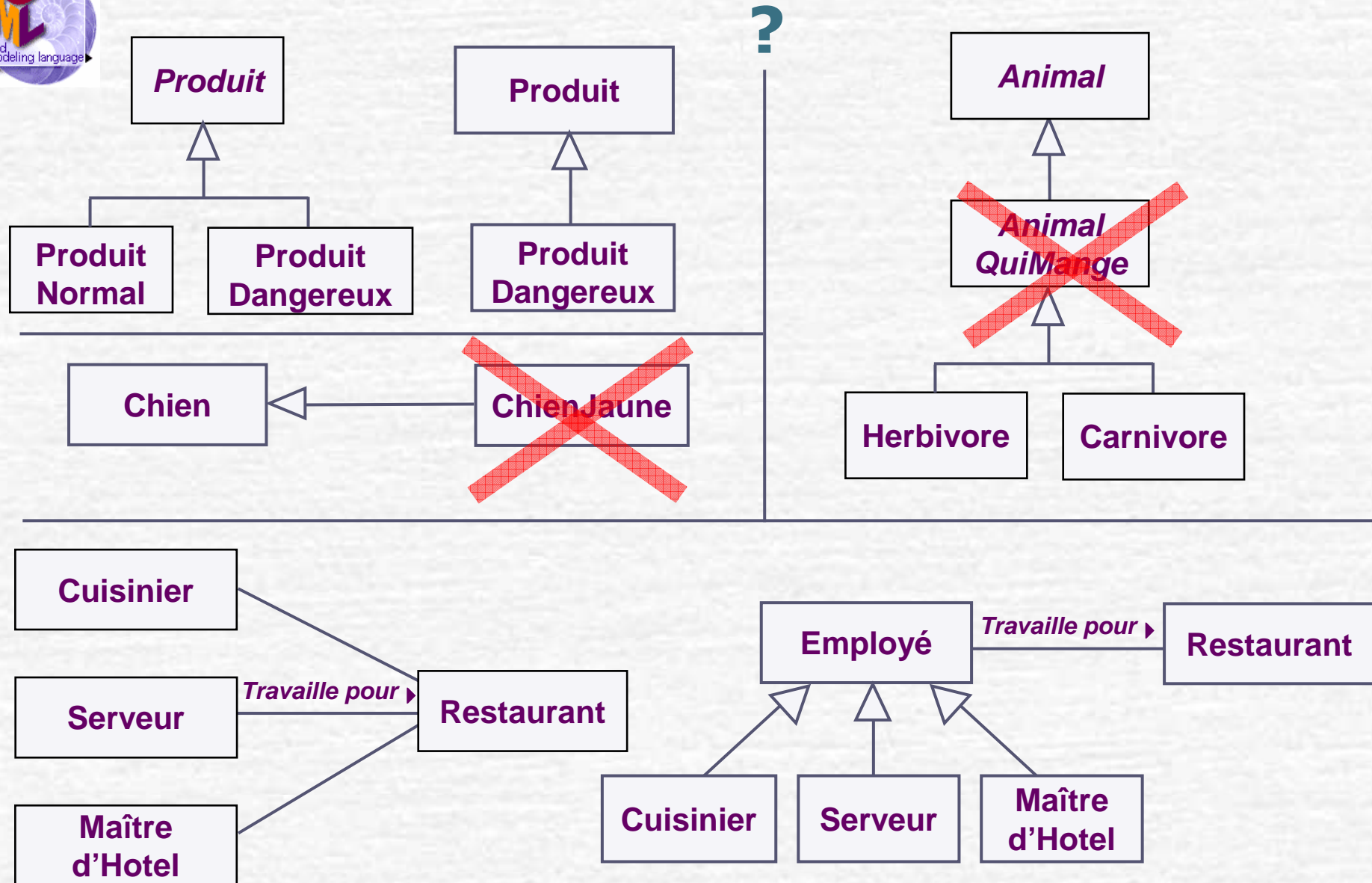


- Plusieurs contraintes peuvent être appliquées aux relations de généralisation :
- La contrainte {disjoint} ou {exclusif} indique qu'une classe descendante d'une classe A ne peut être descendante que d'une seule sous-classe de A (défaut).
- La contrainte {chevauchement} ou {inclusif} indique qu'une classe descendante d'une classe A appartient au produit cartésien des sous-classes de la classe A. Un objet concret est alors construit à partir d'une classe obtenue par mélange de plusieurs super-classes.
- {incomplète} indique une généralisation extensible.
- {complète} indique qu'une instance est forcément d'une des sous classes (la super classe est alors abstraite).





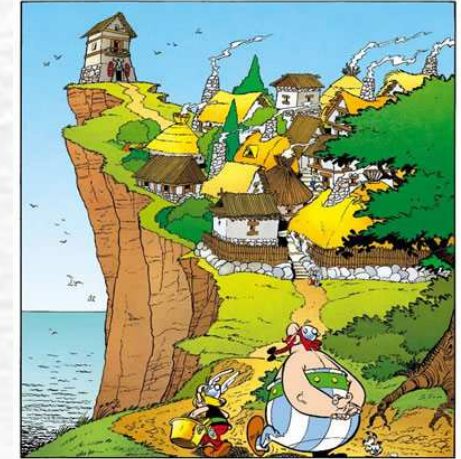
## I - CLASSES D'ANALYSE



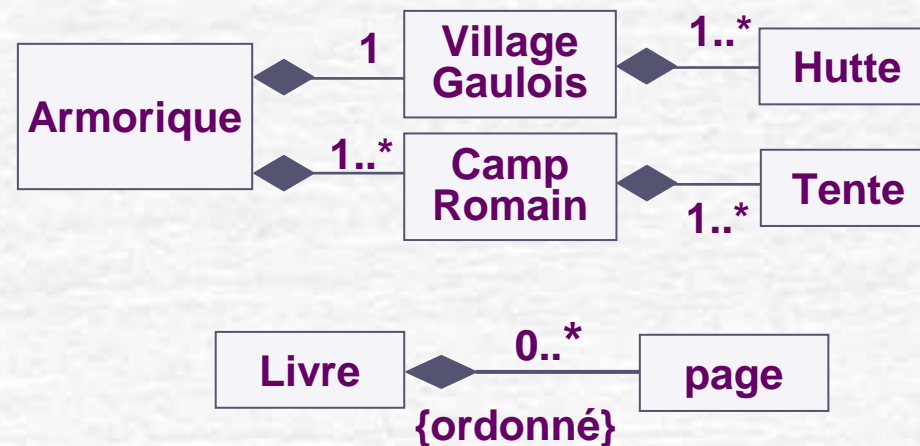
# PRÉCISIONS

- ☞ En UML, sauf si le contraire est spécifié explicitement, les hypothèses par défaut sont :
  - Héritage multiple : une classe peut hériter de plusieurs super-classes,
  - Classification simple : un objet est instance d'une seule classe,
  - Classification statique : un objet est créé à partir d'une classe et n'en change pas.
- ☞ Toutes les associations de la classe mère s'appliquent par défaut aux classes dérivées.
- ☞ Un héritage ne se justifie que si la classe dérivée possède au moins un attribut, une méthode ou une association spécifique.

# COMPOSITION



**Composition :**  
inclusion physique  
d'un objet dans un autre.  
La durée de vie des  
composants est  
identique à celle du  
composite (si le  
composite est détruit,  
les composants aussi).



L'élément composite est responsable de la création, de la copie et la destruction de ses composants.



# AGRÉGATION



L'agrégation ou composition par référence  
est une forme dégénérée  
de la composition.

C'est un lien entre deux  
classes dont les durées de vie  
peuvent être indépendantes.  
L'agrégation indique souvent  
une association impliquant  
une subordination



◀ *est affecté*



◀ *appartient*

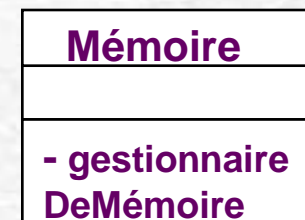
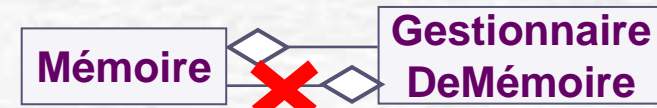
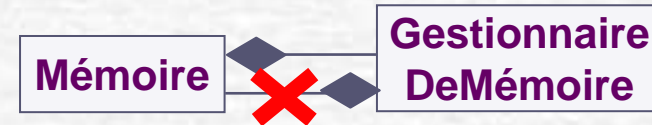
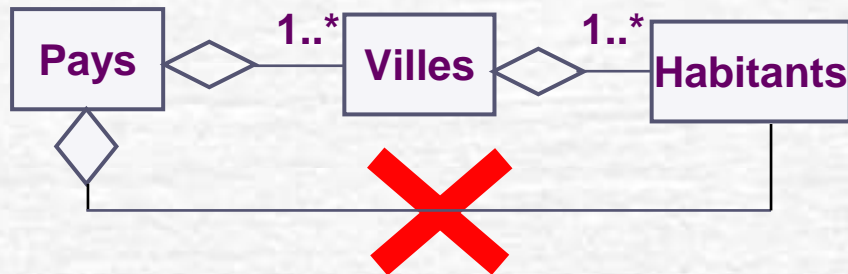
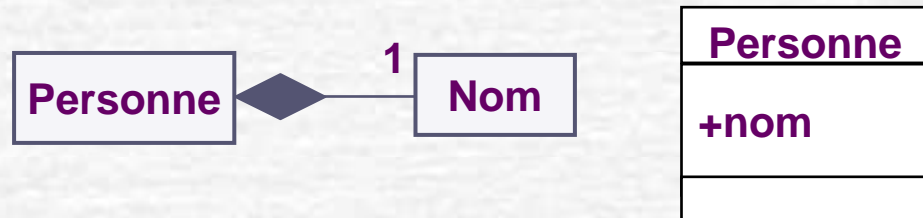
Propriétaire



L'agrégation est une  
association transitive

?

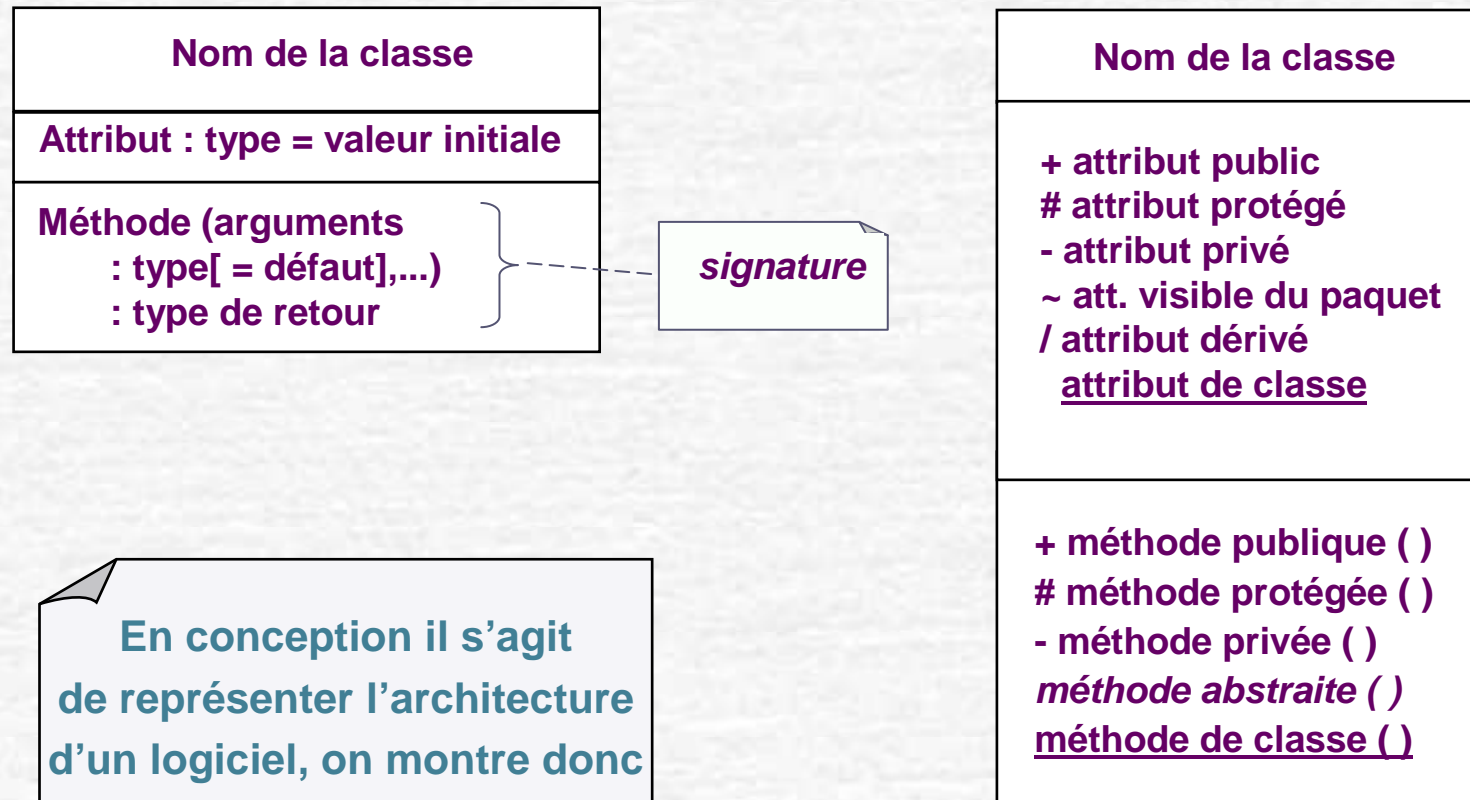
## I - CLASSES D'ANALYSE



ou



## II - CLASSES DE CONCEPTION

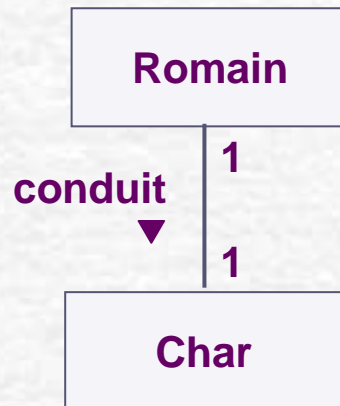


### Visibilité des attributs et des opérations



## CONCEPTION : TRADUCTION DES ASSOCIATIONS

Les associations se transforment en références  
(échanges d'attributs entre classes)

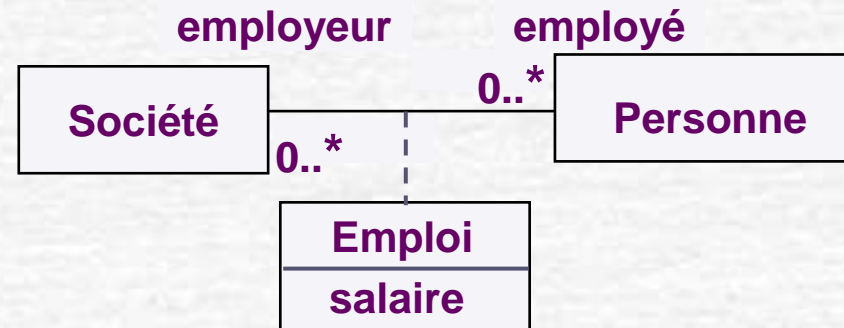


Classe d'analyse

Romain	Char
<ul style="list-style-type: none"> <li>- identifiantRomain : String</li> <li>- nom : String</li> <li>- fonction : String</li> <li>- dateNaissance : Date</li> <li>- véhicule : Char</li> </ul>	<ul style="list-style-type: none"> <li>- numVéhicule : int</li> <li>- marque : String</li> <li>- dateAchat : Date</li> <li>- propriétaire : Romain</li> </ul>
<ul style="list-style-type: none"> <li>+ calculAge(dn:Date) : int</li> <li>+ get...</li> <li>+ set...</li> </ul>	<ul style="list-style-type: none"> <li>+ addTuning ( )</li> <li>+ get...</li> <li>+ set...</li> </ul>

Classe technique  
(Conception du logiciel)

## CLASSE D'ASSOCIATION



### Analyse

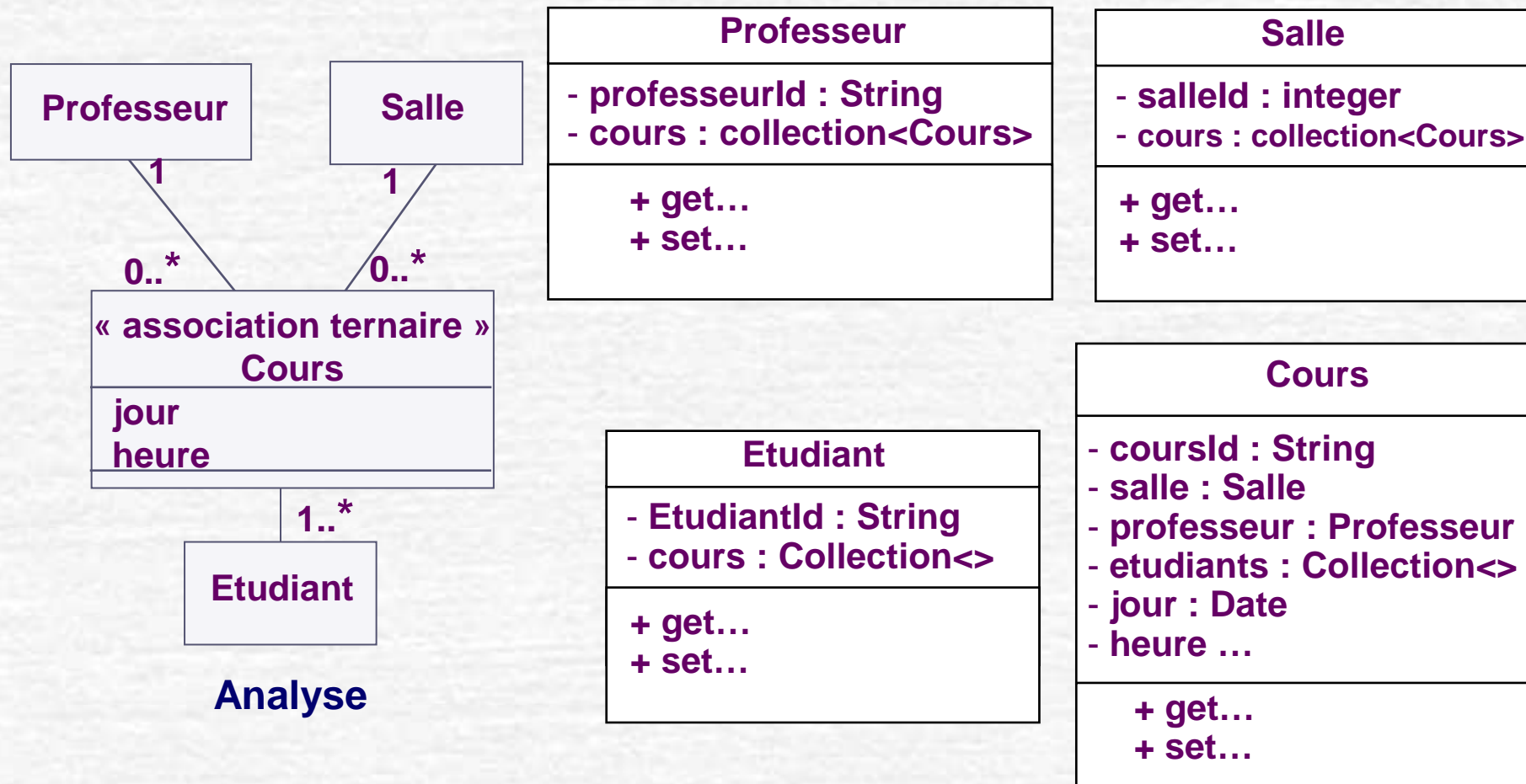
Société
- sociétéId : String - emplois : collection<Emploi>
+ get... + set...

Emploi
- emploird : String - personne : Personne - société : Société - salaire : Integer
+ get... + set...

Personne
- personneld : String - emplois : Collection<Emploi>
+ get... + set...

### Conception

### ASSOCIATION TERNAIRE



Analyse

Conception



## Associations navigables dans un seul sens

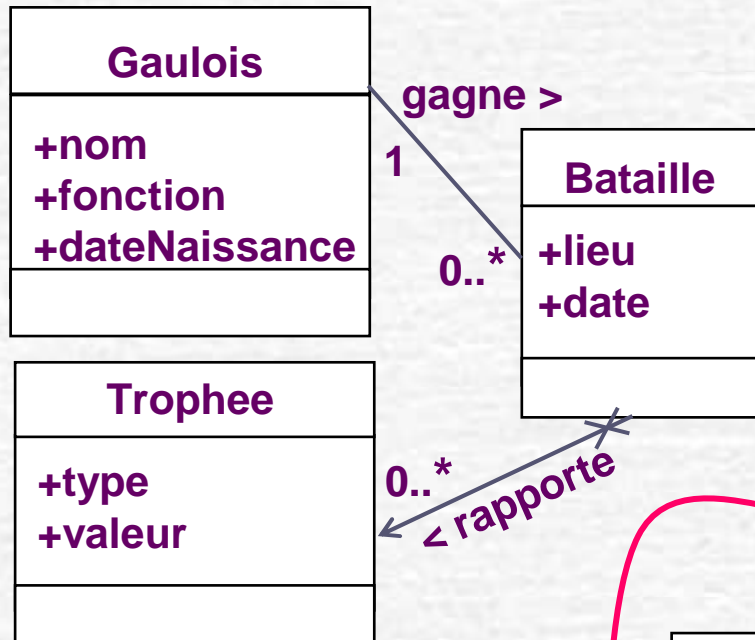


### Navigabilité restreinte

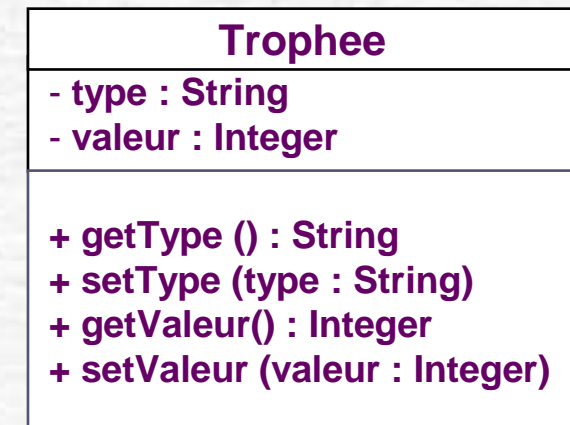
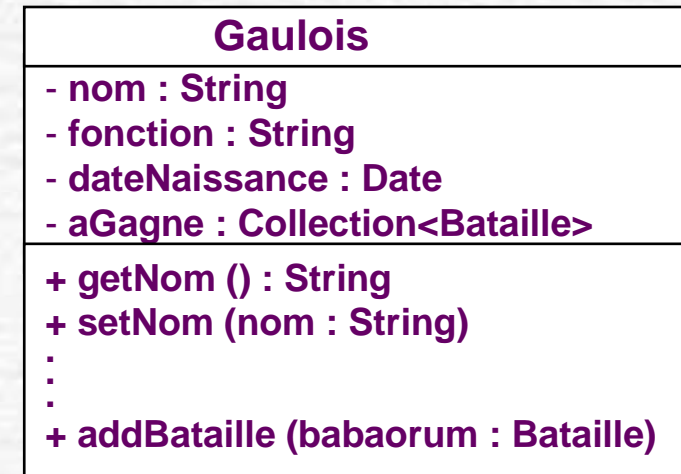
(on peut aller de la source vers la cible, mais non le contraire)

L'extrémité du côté de la classe Centurion n'est pas navigable : cela signifie que les instances de la classe CampRomain ne stockent d'objet du type Centurion. Inversement, la terminaison du côté de la classe CampRomain est navigable : chaque objet centurion possède un attribut de classe CampRomain.

## EXEMPLE DE NAVIGABILITÉ RESTREINTE



Analyse



Conception

### III – EXERCICES BRAIN STORMING



Préparer un diagramme de classes montrant au moins 10 relations entre les objets suivants. Inclure les associations, les agrégations et les généralisations. Placer les multiplicités.

(a) école, terrain de jeu, proviseur, conseil de classe, salle de classe, livre, élève, professeur, cafétéria, ordinateur, bureau, chaise, porte.

(b) château, douve, pont-levis, tour, fantôme, escalier, donjon, plancher, couloir, salle, fenêtre, pierre, seigneur, dame, cuisinier.

(c) Automobile, roue, frein, moteur, porte, batterie, silencieux, pot d'échappement.



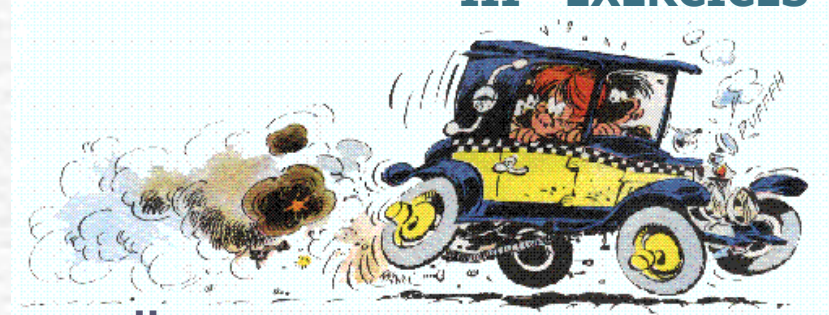
## PROBLÈME 1 : MEDUSE



- La médiathèque de l'Université des Schtroumfs Erudits (USE) possède des ouvrages. Ils peuvent être des livres, des CD ou des DVD qui existent en plusieurs exemplaires. Les informations à stocker sur un ouvrage dépendent de son type.
- La médiathèque est gérée par des documentalistes et est fréquentée par des lecteurs. Ces lecteurs sont des étudiants de l'USE, des enseignants ou des extérieurs. Ils peuvent emprunter 5 exemplaires au maximum lorsqu'ils sont inscrits. Un lecteur ayant commis des abus (rendus en retard, détériorations...) peut être interdit d'emprunt (durée décidée par le documentaliste).
- Pour pouvoir être emprunté, un exemplaire doit être disponible (non emprunté, non réservé). Chaque emprunt a une durée limitée à 4 semaines.
- Un exemplaire emprunté peut être réservé par un autre lecteur, cette réservation reste effective pendant 2 semaines après la date de retour du livre. Passé ce délai la réservation est annulée.
- Le documentaliste peut ajouter des ouvrages ou des exemplaires d'un ouvrage dans MEDUSE, il peut également en supprimer (perte, vieillissement...).
- Seul le documentaliste peut exécuter les mises à jour des ouvrages, des lecteurs et des emprunts. Pour chaque exemplaire on conserve l'emprunteur courant.
- Les lecteurs effectuent des recherches sur la discipline, des mots clés, l'auteur, le titre, la date de parution... Pour cela ils utilisent MEDUSE en consultation.

## PROBLÈME 2 : RALLYE CLERMONT IRKOUTSK

### III - EXERCICES



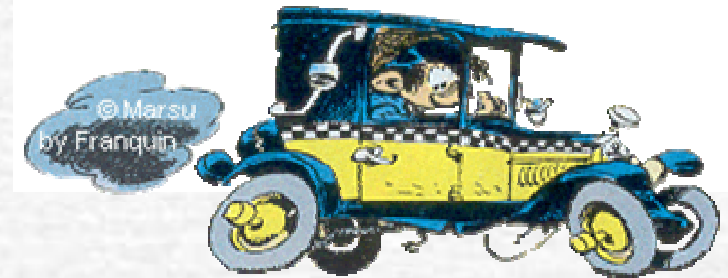
On souhaite construire une application pour un rallye :

Un véhicule est caractérisé par la marque, le modèle et la cylindrée. Une voiture est conduite par un pilote aidé d'un navigateur. Une moto est conduite par un pilote. Le pilote et le navigateur ont un numéro de licence (affiliation à la fédération) et un numéro de dossard (inscription à la course).

De plus, chaque véhicule est rattaché à une équipe (caractéristiques : nom, budget) qui lui fournit assistance et ravitaillement (une même équipe peut gérer plusieurs véhicules de catégories différentes). L'équipe est composée de membres officiellement inscrits auxquels on attribue un numéro de badge et contient un responsable (joint à tout moment grâce à son numéro de téléphone mobile), des assistants techniques (spécialités : mécanique, logistique, etc.) et bien sûr les concurrents en charge d'un véhicule de l'équipe (toujours le même).



## PROBLÈME 2 : RALLYE CLERMONT IRKOUTSK



La course se déroule par étapes. Chaque étape est caractérisée par un numéro, le nom de la ville de départ et de la ville d'arrivée.

Types d'étapes : étapes de transition, étapes en ligne et étapes spéciales.

Les étapes de transition ne servent qu'à permettre aux véhicules de transiter du point d'arrivée d'une étape au point de départ d'une autre. Elles ne rapportent pas de points.

Dans une étape en ligne tous les véhicules d'une même catégorie partent en même temps et doivent effectuer le même parcours.

Une spéciale est un groupe de courtes étapes contre la montre. Le temps du véhicule est obtenu par la somme des temps de chaque étape contre la montre formant la spéciale.

Le classement des véhicules est fonction du temps de chaque étape.



## SOLUTION



- ☞ Trouver les classes,
  - éliminer les classes superflues (attributs),
  - trouver un nom,
- ☞ Trouver les associations (verbes ou rôles),
- ☞ Factoriser (généraliser),
- ☞ Tester les chemins d'accès aux classes,
- ☞ Itérer et affiner.



## LES ERREURS A EVITER

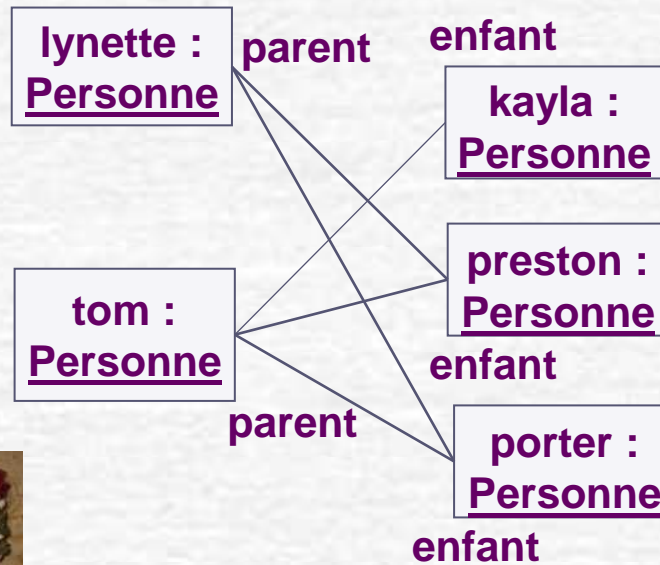
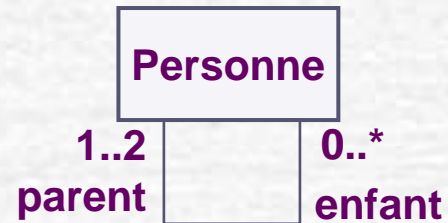
- ❏ Les classes redondantes (celles qui modélisent des concepts similaires).
- ❏ Les classes qui modélisent des implémentations possibles des éléments du domaine (par exemple un conteneur).
- ❏ Les propriétés non factorisées (on construit souvent le graphe d'héritage vers le haut).
- ❏ La multiplication des associations qui crée des chemins redondants entre classes.

## V - DIAGRAMMES D'OBJETS

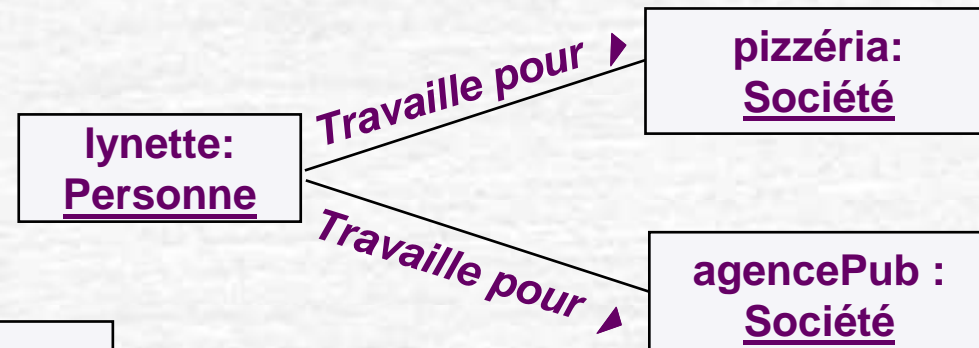
- Décrivent un état possible à un instant  $t$ , un cas particulier, une situation concrète, un cas réel.
- Doivent être conformes au diagramme de classes.
- Sont souvent établis en parallèle avec les diagrammes de classes.
- Peuvent être construit avant les diagrammes de classes afin de « découvrir » les classes.
- Peuvent être utilisés pour :
  - Expliquer un diagramme de classes (donner un exemple).
  - Valider un diagramme de classes (le tester).



## EXEMPLES



Parker, Penny...





# Chapitre 3

## LES DIAGRAMMES DE PAQUET

(Package Diagrams)



# DIAGRAMME DE PAQUET

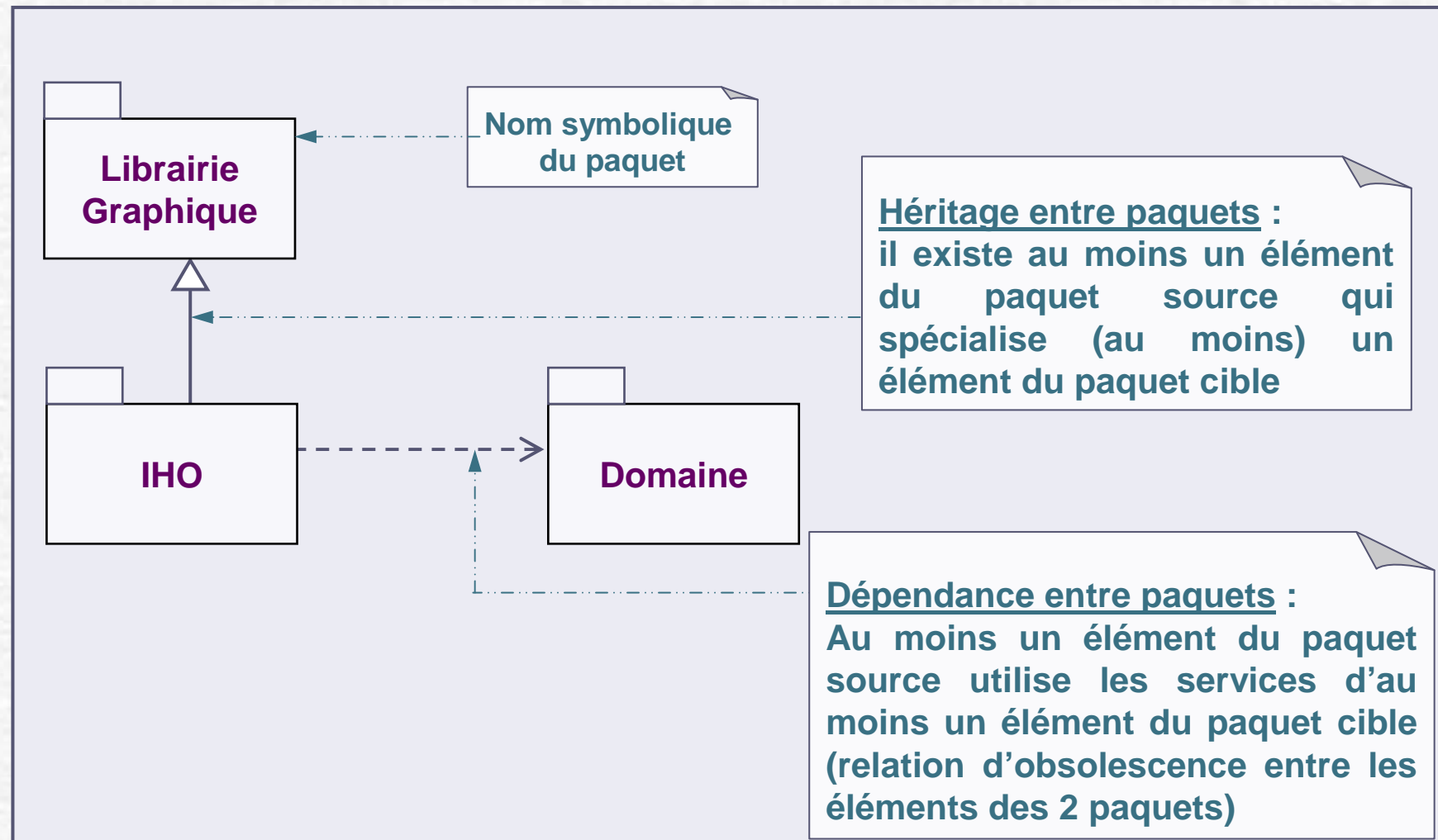
- ✎ Un paquet (package) UML est une mécanique de groupement.
- ✎ Les classes peuvent être groupées au sein d'une même unité, sujet, sous-système d'objets en raison de leurs objectifs communs.
- ✎ La notion de paquet peut être appliquée à n'importe quel élément du modèle, pas seulement aux classes.
- ✎ Les critères de découpage dépendent de la phase du processus de développement.

## QU'EST-CE QU'ON MET DANS UN PAQUET ?



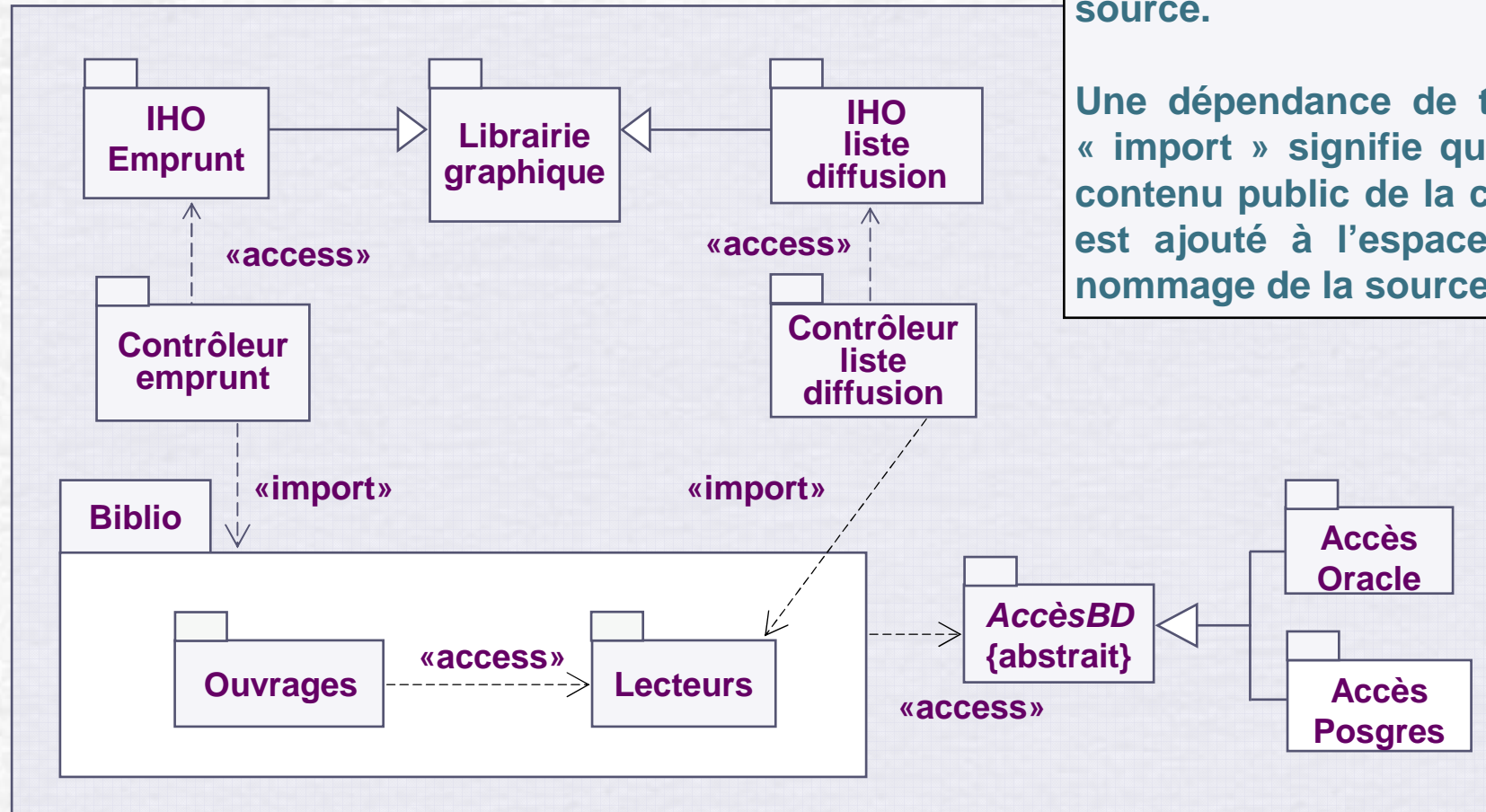
- **Découpage fonctionnel** : pour attribuer les objectifs et les responsabilités à chaque équipe de réalisation, il est possible de définir des paquets a priori, représentant chacun un sous-système fonctionnel du système à réaliser (Cas d'Utilisation).
- **Découpage structurel** : classes ayant un lien logique ou une définition voisine (ex : les classes reliées par une relation d'héritage).
- **Découpage “ pratique ”** : pour la lisibilité du modèle ou la facilité d'utilisation des AGL.

# DIAGRAMME DE PAQUET





## EXEMPLE

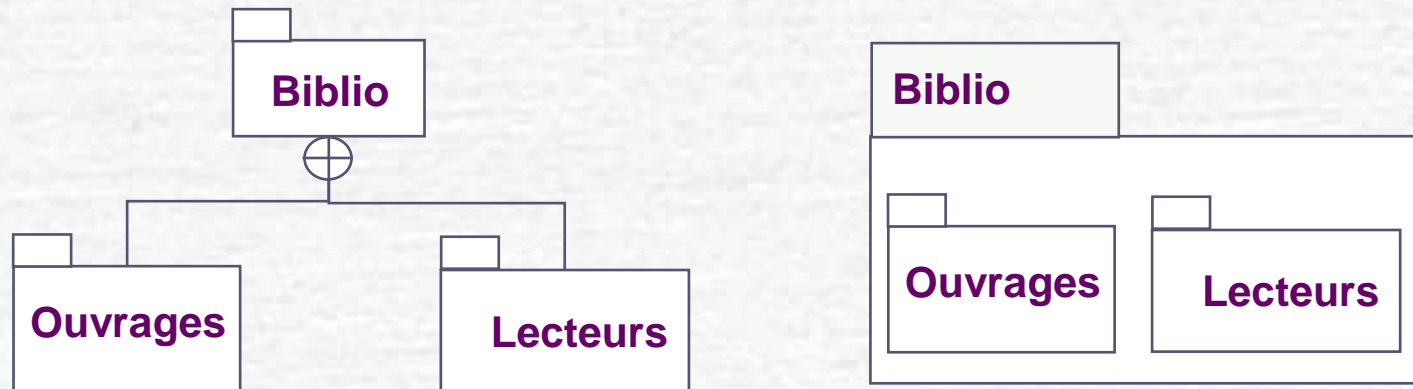


Une dépendance de type « access » signifie que le contenu public de la cible est accessible au paquet source.

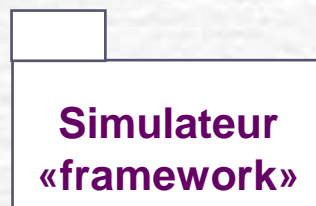
Une dépendance de type « import » signifie que le contenu public de la cible est ajouté à l'espace de nommage de la source.



# DIAGRAMME DE PAQUET



Description hiérarchique des paquets : sous forme d'arbre ou en montrant les paquets inclus.



Stéréotype d'un paquet  
(framework, toplevel...)

## CONCEPTION DES PAQUETS



- ☛ Minimiser le couplage inter paquets :
  - Le moins de dépendances possibles.
    - « on ne parle pas aux étrangers » !
- ☛ Maximiser la cohésion intra paquet :
  - Regrouper les éléments en forte relation,
  - Regrouper les classes qui rendent des services de même nature aux utilisateurs,
  - Isoler les classes réellement stables de celles qui risquent d'évoluer au cours du projet,
  - Isoler les classes métiers des classes applicatives,
  - Distinguer les classes dont les objets ont des durées de vie différentes.
- ☛ Vérifier les dépendances circulaires.

# INTERFACES ENTRE PAQUETS

- ☛ L'interface d'un paquet est l'ensemble des classes publiques du paquet.
- ☛ Les interfaces doivent contenir :
  - Seulement l'information nécessaire :
    - L'interface doit révéler le moins d'information possible...
  - Toute l'information nécessaire :
    - L'interface doit donner aux autres modules l'information suffisante pour pouvoir utiliser les ressources offertes.
  - Pour favoriser l'évolution du système.
    - Cacher les détails de bas niveau (algorithmes, SDD, etc.).



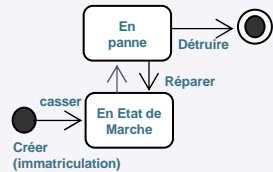
# Chapitre 4

## LES DIAGRAMMES D'ÉTATS

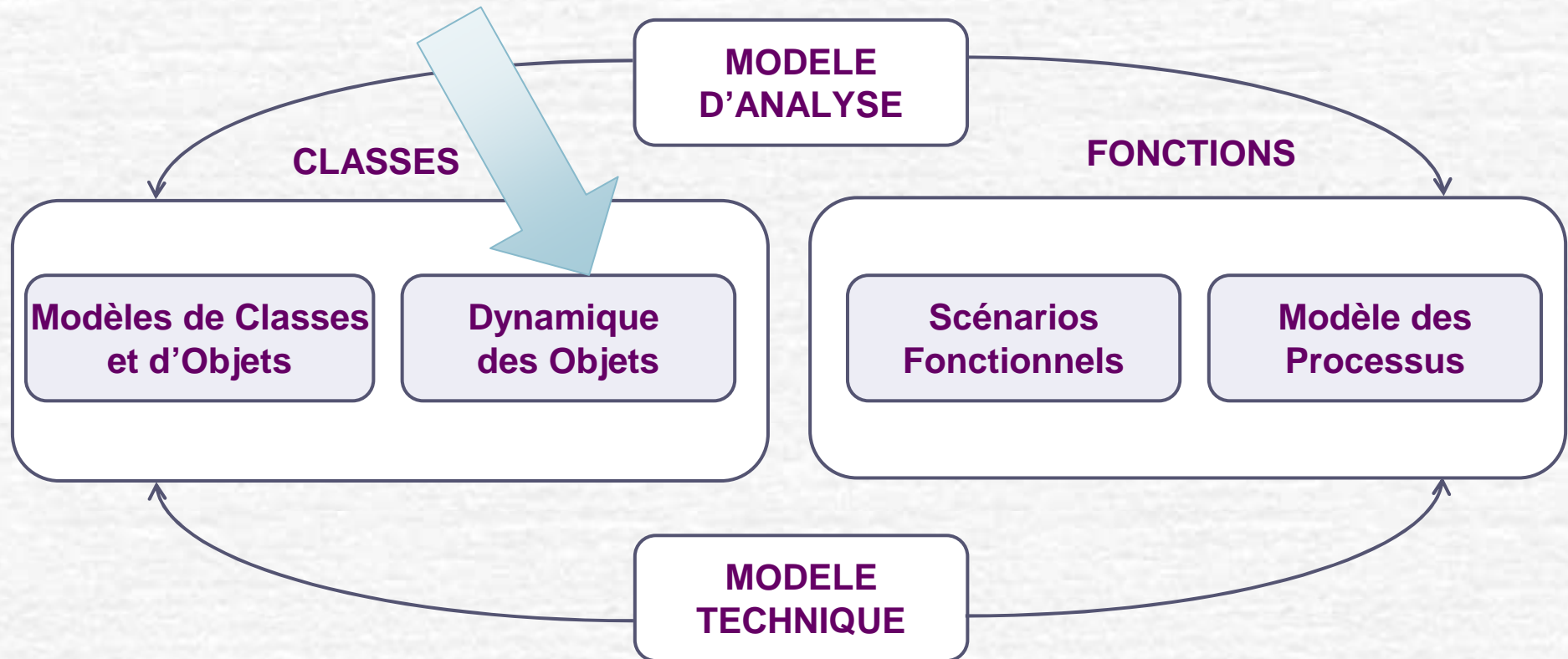
(State Machine Diagrams ou  
diagrammes états-transitions)



# DIAGRAMMES D'ÉTATS



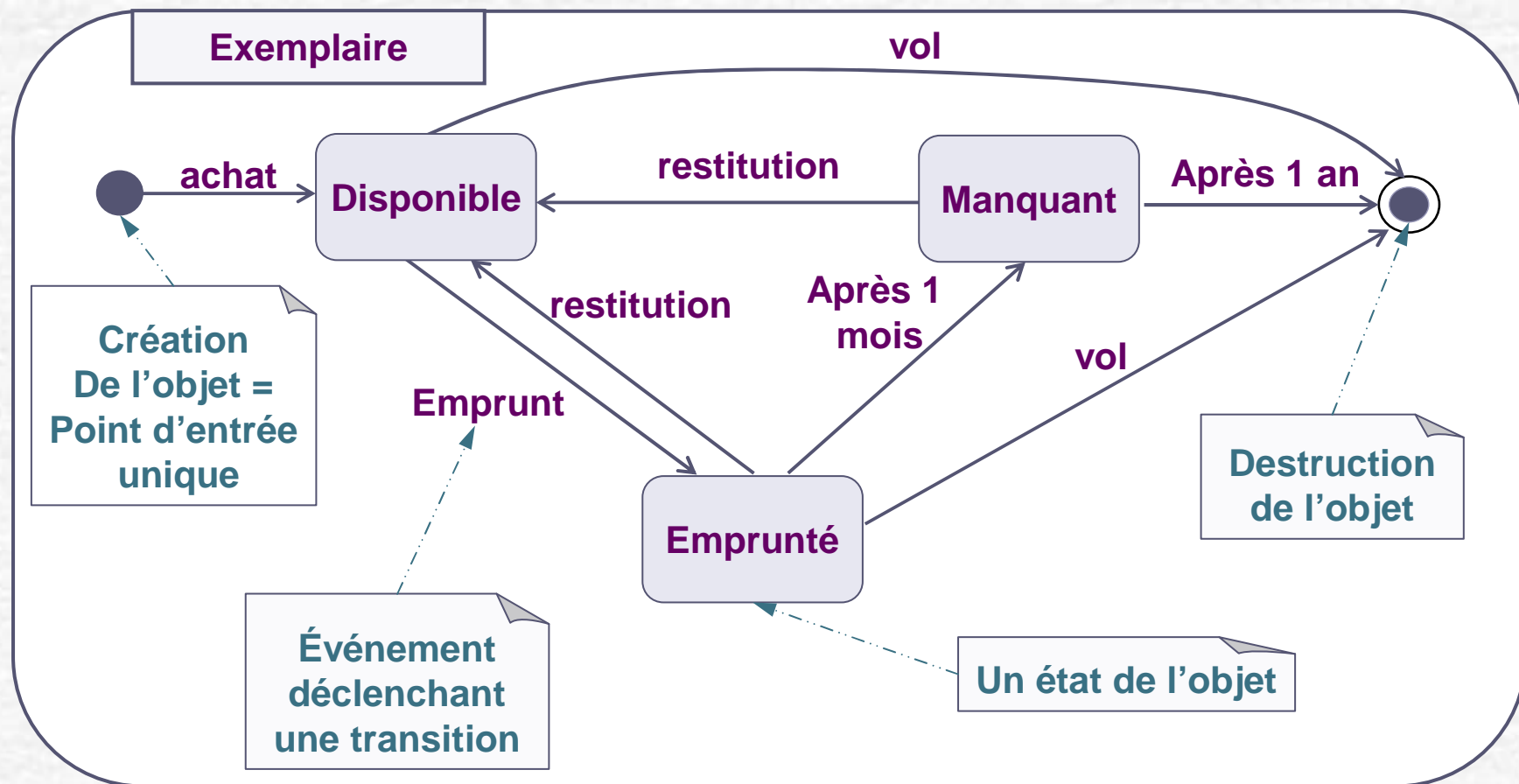
Les diagrammes d'états sont utilisés pour compléter aussi bien le diagramme de classes d'analyse que le diagramme de classes technique. Ils sont élaborés **pour une classe** afin de visualiser le comportement d'un objet au cours du temps.



# LES DIAGRAMMES D'ÉTATS

- **Modéliser les objets réactifs : ceux dont le comportement est caractérisé par leur réaction à des événements issus de l'extérieur de leur contexte (envoyés par d'autres objets).**
- **Spécifier :**
  - **Les états stables des objets ayant un comportement dynamique complexe,**
  - **Les événements qui déclenchent les transitions d'états,**
  - **Les actions qui se réalisent à chaque changement d'état ou à l'intérieur d'un état.**

# ETATS D'UN EXEMPLAIRE DE MEDUSE





## QUAND FAIT-ON UN DIAGRAMME D'ÉTATS ?

- ☛ Lorsque les objets d'une classe réagissent différemment à l'occurrence du même événement et que chaque type de réaction caractérise un état particulier.
- ☛ Un état représente une situation de la vie d'un objet pendant laquelle :
  - Il satisfait une certaine condition,
  - Il exécute une certaine activité,
  - Il attend un certain événement.
- ☛ Un état a une durée non négligeable mais finie, variable selon la vie de l'objet, en fonction des événements qui lui arrivent.

## QU'EST-CE QU'UN ÉVÉNEMENT ?



- Signal : événement nommé, déclenché explicitement (exemple : un envoi de message asynchrone).



- Appel de méthode : invocation synchrone d'une opération.
- Temporisation : passage du temps
  - exemple : après (after) 10 sec.
- Changement : modification de conditions
  - exemple : quand (when) température > 100°.

## LES TRANSITIONS

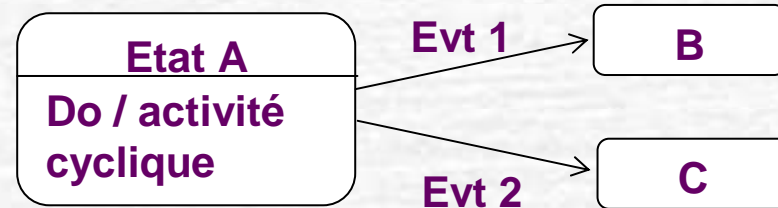


- Transition gardée : la transition se réalise si l'événement se produit et si la garde (ou condition) est vraie.
  - La condition porte sur des informations visibles et accessibles de l'objet (paramètres de l'evt, valeurs internes, données globales).
- Gardes :
  - Doivent être mutuellement exclusives pour chaque événement,
  - Doivent couvrir tous les cas.
- Action : opérations dont le temps d'exécution est négligeable ou nul (une action est instantanée). Une action est :
  - non sécable : toute action commencée se termine,
  - déclenchée après l'évaluation de la garde,
  - une méthode de la classe de l'objet destinataire de l'événement.

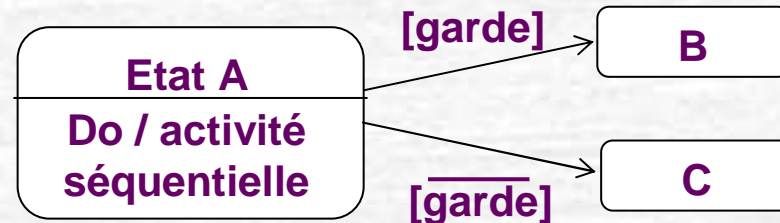
# LES ACTIVITÉS

Les *activités* s'exécutent à l'intérieur des états et peuvent prendre du temps, les activités peuvent être interrompues par l'occurrence d'un événement.

**Exemple 1** : une activité cyclique est interrompue par l'occurrence d'événements.



**Exemple 2** : une activité séquentielle déclenche une transition automatique (avec garde ici) : une des 2 transitions est déclenchée dès que l'activité se termine, selon la valeur de la condition.





# LES ÉVÉNEMENTS INTERNES

## Etat A

entry / action d'entrée  
on evt / action  
exit / action de sortie

## Saisie mot de passe

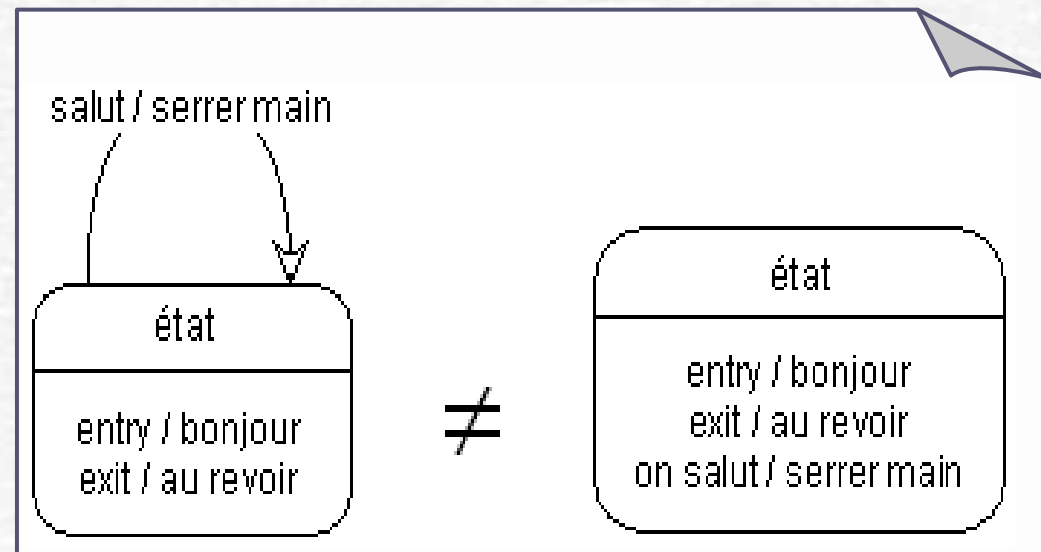
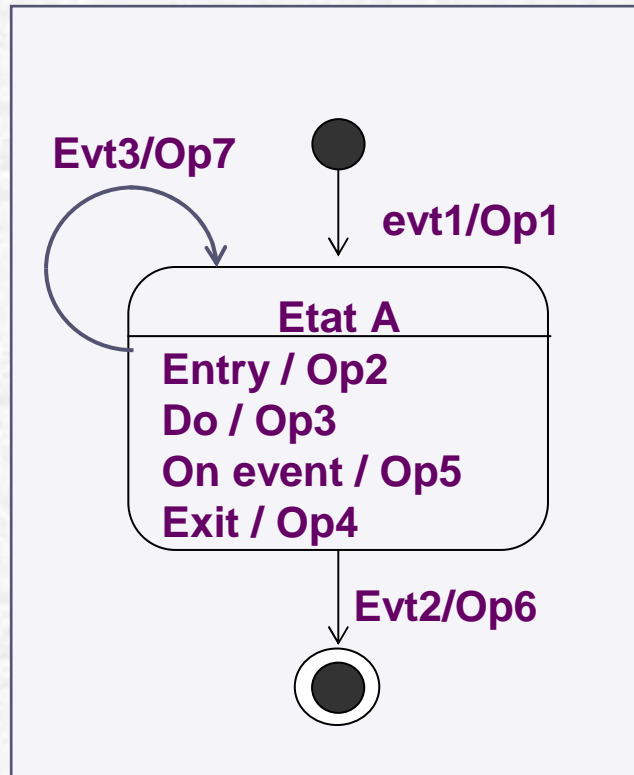
entry / bloquer les entrées  
clavier  
do / gérer caractères saisis  
on aide / afficher l'aide  
exit / afficher entrées  
clavier

Événement interne (introduit par le mot clé « on ») : génère une action qui ne déclenche pas de transition.

Evénements spéciaux entry et exit : déclenchent des actions en entrée et en sortie de l'état.

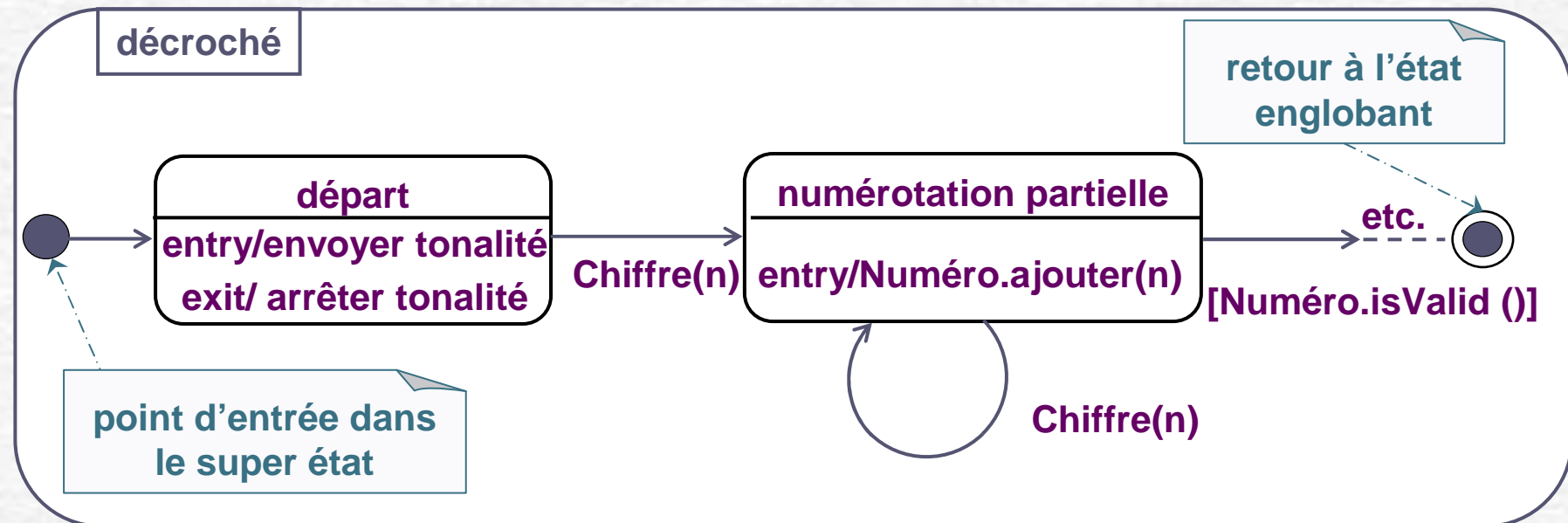
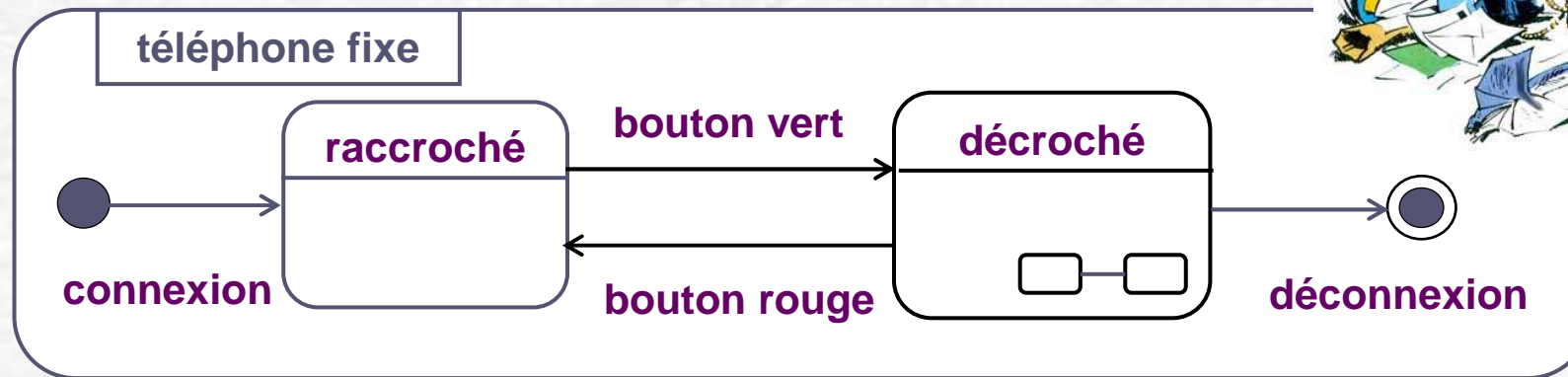
## Exemple

# INDIQUER LES OPÉRATIONS

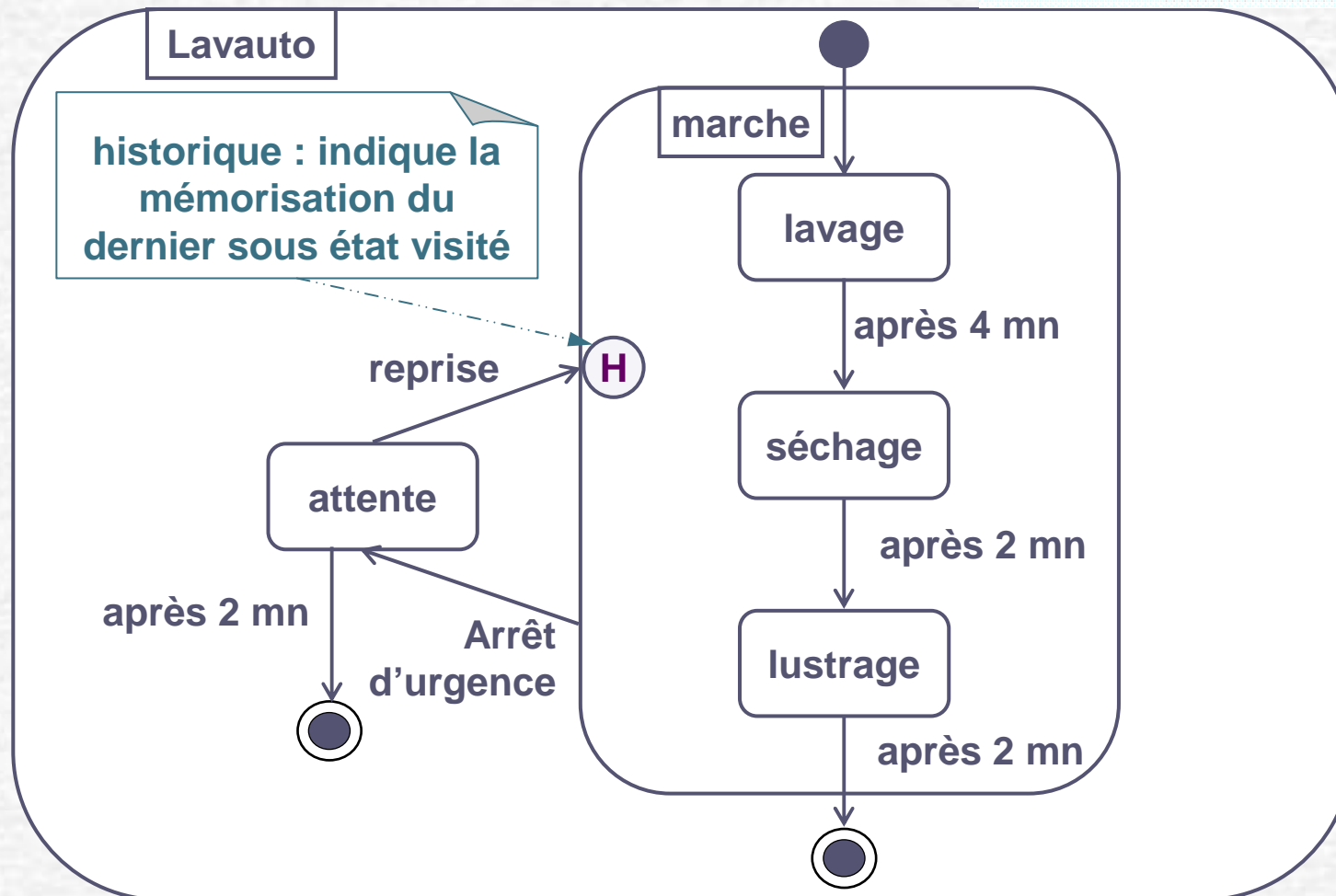
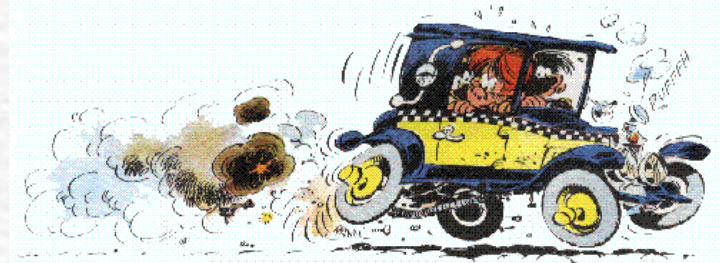


Il y a 7 manières différentes d'indiquer « des choses à faire » dans un diagramme d'états

# ÉTAT COMPOSITE (SUPER ÉTAT)



# HISTORIQUE



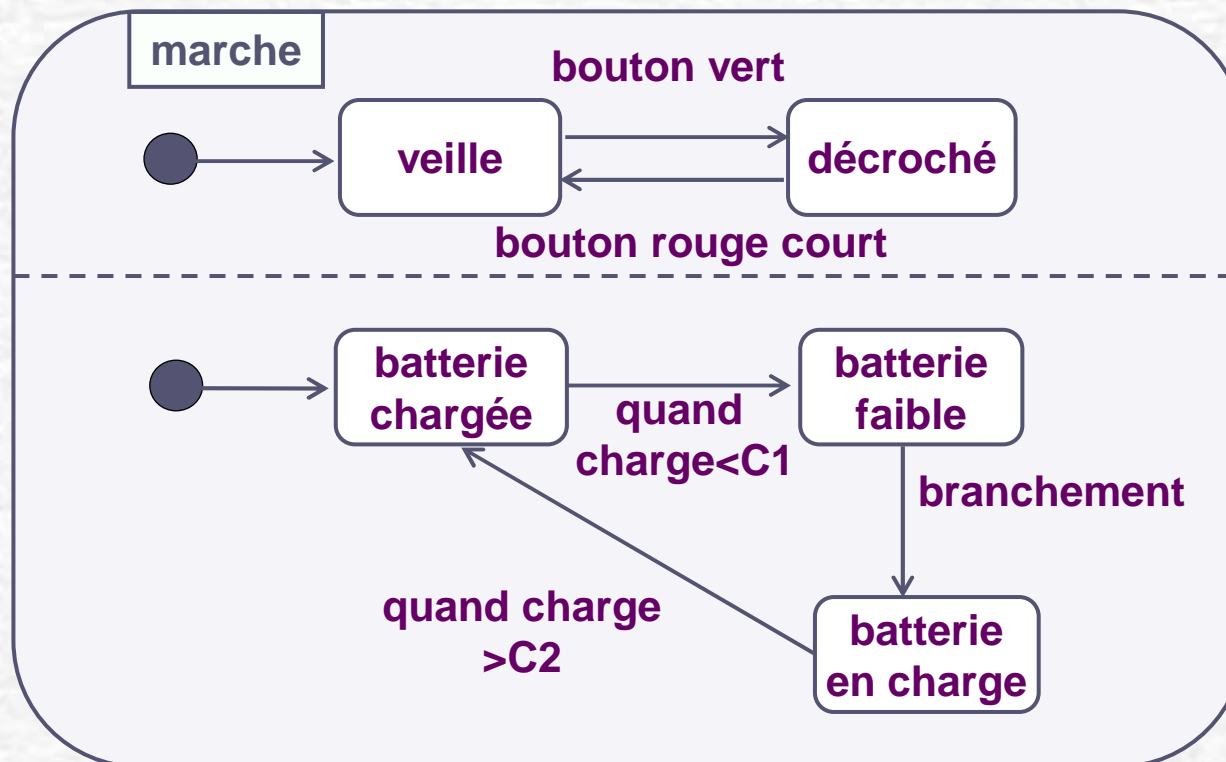
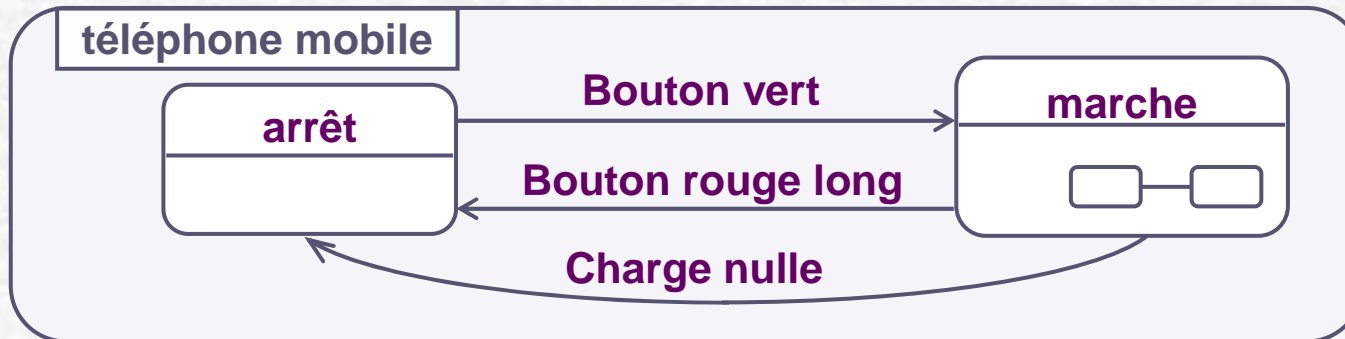
H\* indique que le dernier sous-état visité est mémorisé, quelle que soit la profondeur d'emboîtement des sous-états.



## ÉTATS ORTHOGONAUX

- ☛ Les diagrammes d'états permettent de décrire efficacement les mécanismes concurrents grâce à l'utilisation d'*états orthogonaux*.
  - Un état orthogonal est un état composite comportant plus d'une région, chaque région représentant un flot d'exécution.
  - Graphiquement, dans un état orthogonal, les différentes régions sont séparées par un trait horizontal en pointillé allant du bord gauche au bord droit de l'état composite.
- ☛ Chaque région peut posséder un état initial et final. Une transition qui atteint la bordure d'un état composite orthogonal est équivalente à une transition qui atteint les états initiaux de toutes ses régions concurrentes.
- ☛ Toutes les régions concurrentes d'un état composite orthogonal doivent atteindre leur état final pour que l'état composite soit considéré comme terminé.

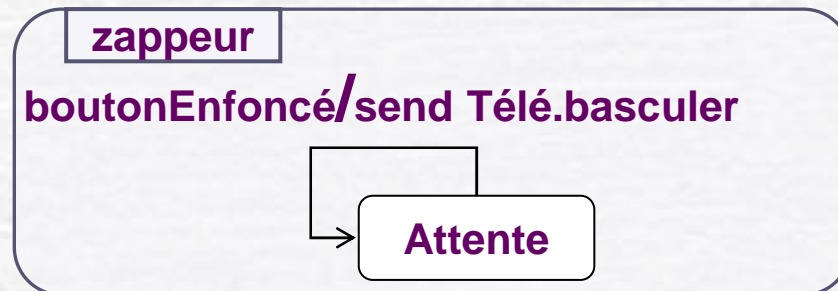
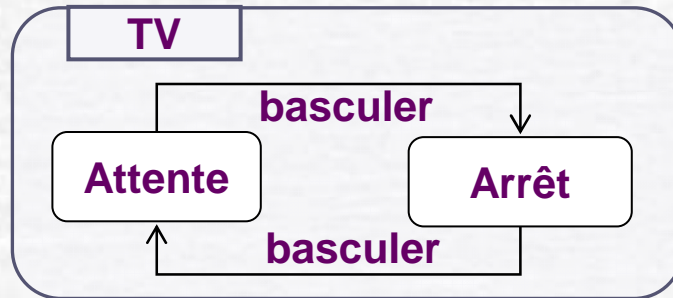
# ÉTATS ORTHOGONAUX



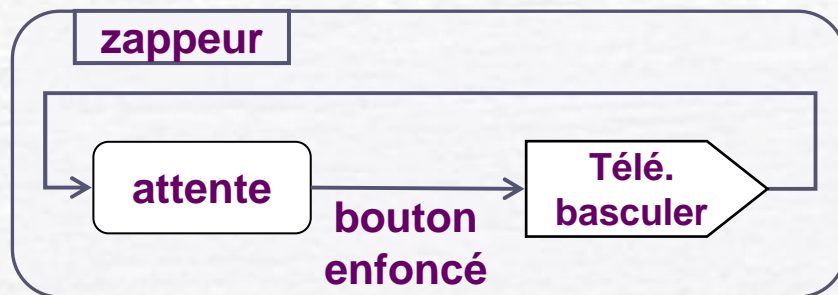
L'objet se trouve dans plusieurs sous-états simultanément



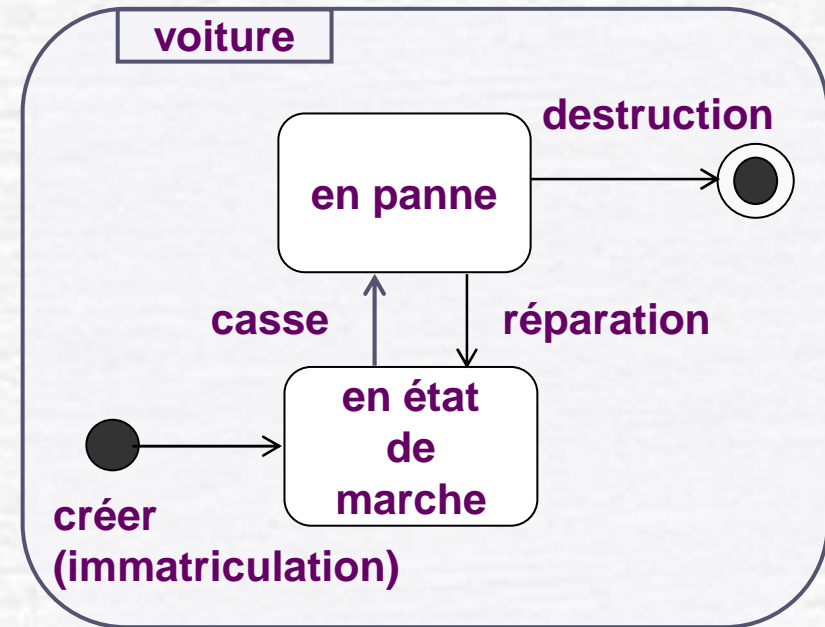
# VIE DES OBJETS



OU



Communication entre objets



Création et de destruction d'objet



# DIAGRAMMES D'ÉTATS

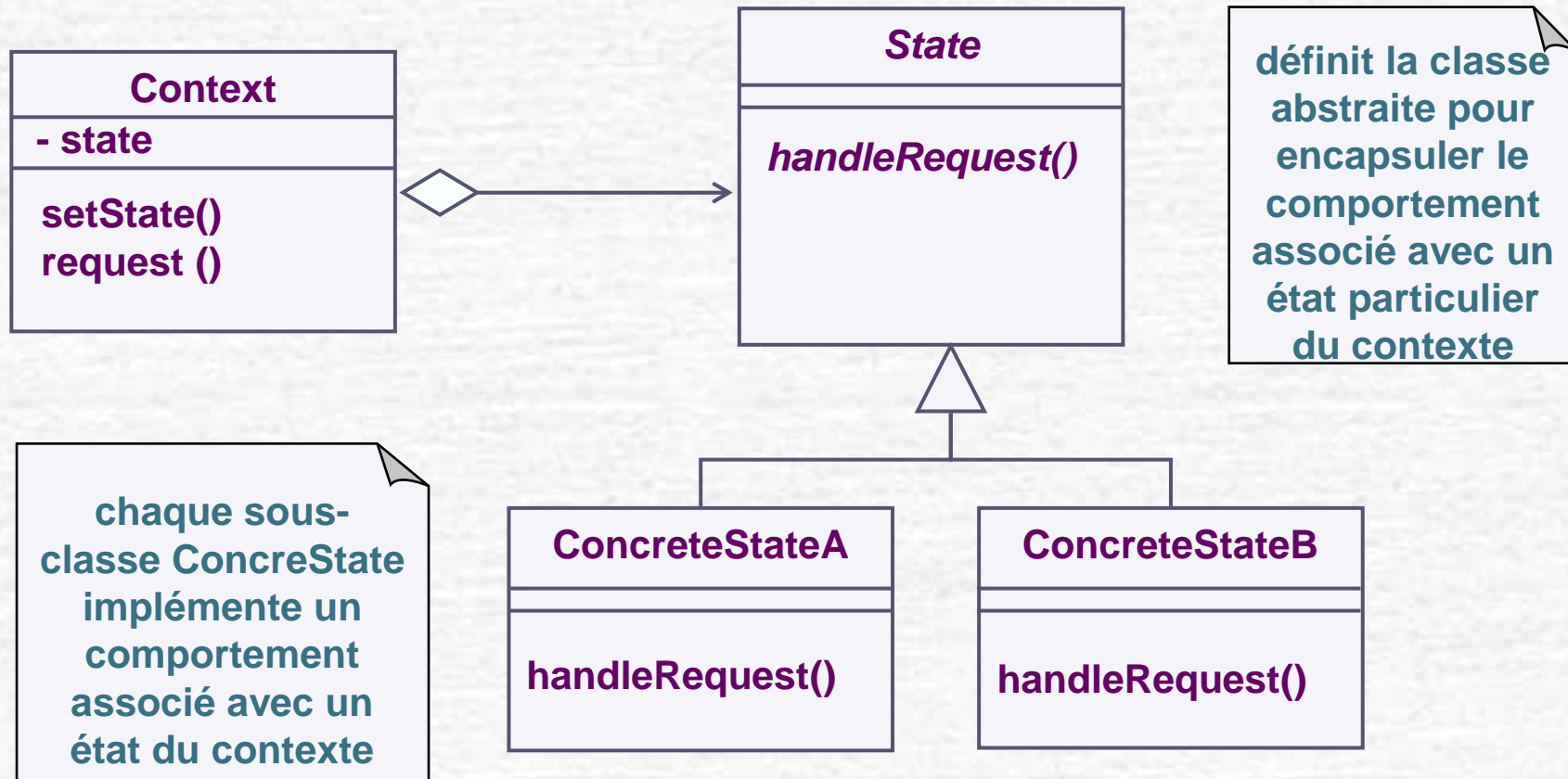
- ☛ Représentation de comportements complexes :
  - Analyse : comportement d'un objet métier,
  - Modélisation d'un comportement prévu,
  - Conception : comportement des objets d'une classe du programme.
- ☛ Représentation de tous les états et seulement les états et les transitions **valides** des objets de la classe.
  - Si pas de transition pour un événement X, l'objet ne réagit pas à cet événement dans cet état.
- ☛ **Pas de diagramme à moins de 3 états.**



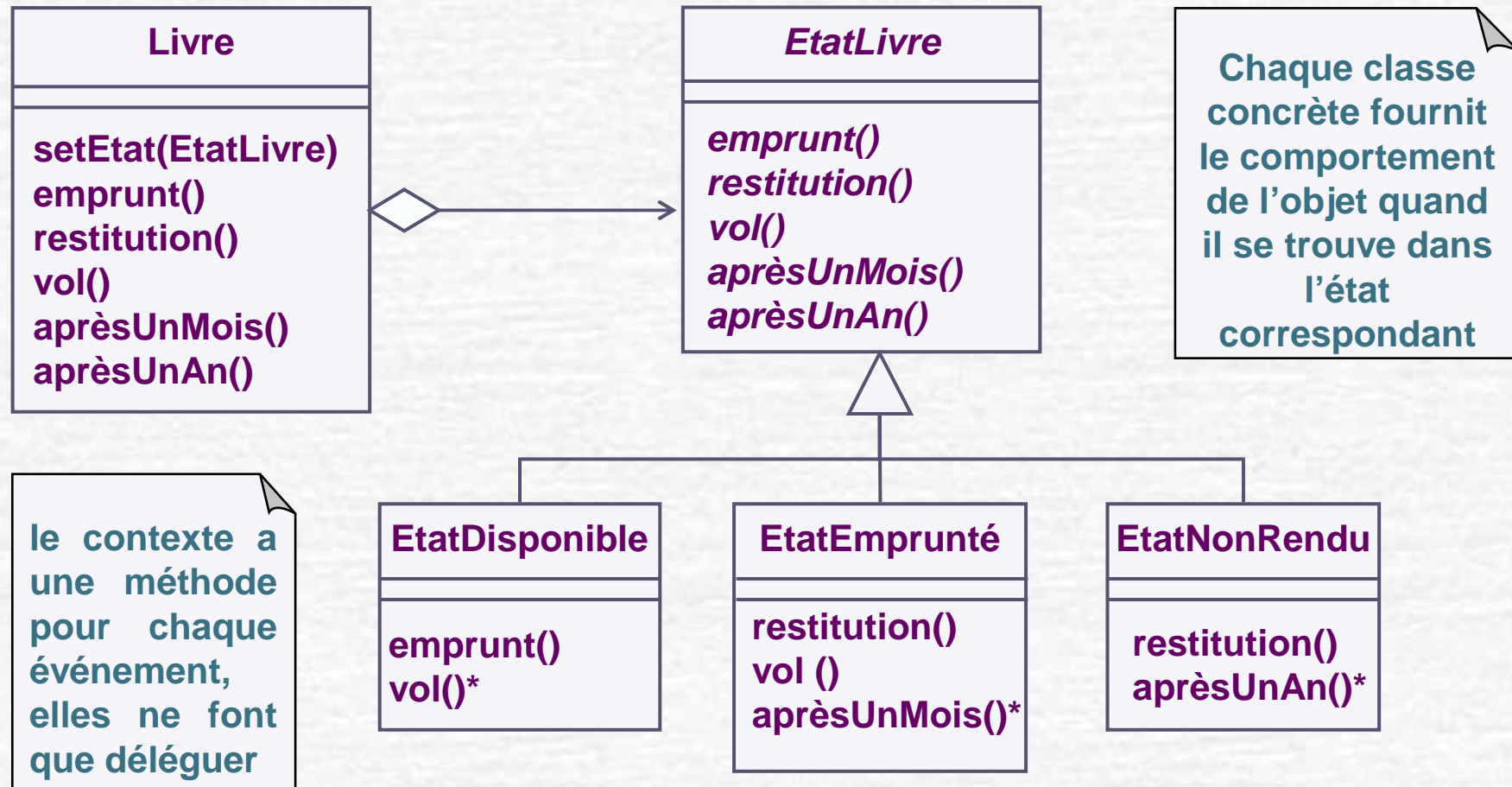
# IMPLÉMENTATION

- ❏ Instructions de type selon ("switch") imbriquées : manière directe qui devient vite peu lisible.
- ❏ Patron de conception State : une classe pour chaque état, un contrôleur délègue le traitement de chaque événement à la classe Etat.
- ❏ Table contenant les états et les actions à effectuer lors des transitions.

# LE PATRON STATE



## EXEMPLE : LE LIVRE



Le patron État (State)



(\*) les autres méthodes sont vides ou lèvent une exception

## EXEMPLE : LE LIVRE

État source	État cible	Événement	Garde	Action
disponible	emprunté	emprunt		
disponible	Non rendu	vol		
emprunté	disponible	restitution		
emprunté	non rendu	après 1 mois		
non rendu	disponible	restitution		
non rendu	État final	après 1 an		

**Table des états : on peut construire un interpréteur qui utilise la table à l'exécution : plus de travail au départ, mais plus de souplesse.**



## EXERCICE



Logiciel pour gérer une station service :

- Avant de pouvoir être utilisée par le client, chaque pompe à essence doit être armée par le pompiste. La pompe est alors prête, mais ce n'est que lorsque le client appuie sur la gâchette du pistolet que l'essence est pompée (servie). Si le pistolet est sur son support, même si la gâchette est pressée, l'essence n'est pas pompée.
- La distribution d'essence au client est terminée quand celui-ci remet le pistolet sur son support. Le débitmètre fournit alors la quantité d'essence distribuée.
- Le client peut payer en liquide, par chèque ou par carte. Le système enregistre le montant et le mode de paiement. En fin de journée les transactions sont transmises au système comptable de la société.
- Si le niveau de la cuve correspondant à une pompe est inférieur à 5% de la capacité maximale, la pompe ne peut plus être armée.

# Appareil Photo

- On souhaite modéliser les fonctions d'un appareil photo numérique:
- Lorsque l'appareil est allumé, il se trouve en mode normal avec flash désactivé.
- Un bouton mode permet de passer du mode normal, au mode paysage, au mode portrait puis à nouveau au mode normal (boucle).
- Le bouton flash permet d'activer le flash automatique, si on appuie à nouveau sur ce bouton flash, l'appareil supprimera également les yeux rouges, puis sur une nouvelle pression, le flash se trouve désactivé (boucle).
- Pour prendre une photo, il y a un gros bouton spécifique, lorsqu'il est appuyé l'appareil réalise la photo (la stocke en mémoire).
- Il existe aussi un bouton pour la mise au point, lorsque celui-ci est appuyé l'appareil effectue une phase de mise au point qui dure une seconde.
- L'appareil dispose d'un bouton de zoom avant et d'un bouton de zoom arrière. En restant appuyé sur le bouton de zoom avant, l'appareil avance son zoom (jusqu'à une position maximale), on relâche ce bouton dès que le zoom est satisfaisant (ou lorsque le zoom ne peut plus avancer). Le bouton de zoom arrière reprend le même principe.

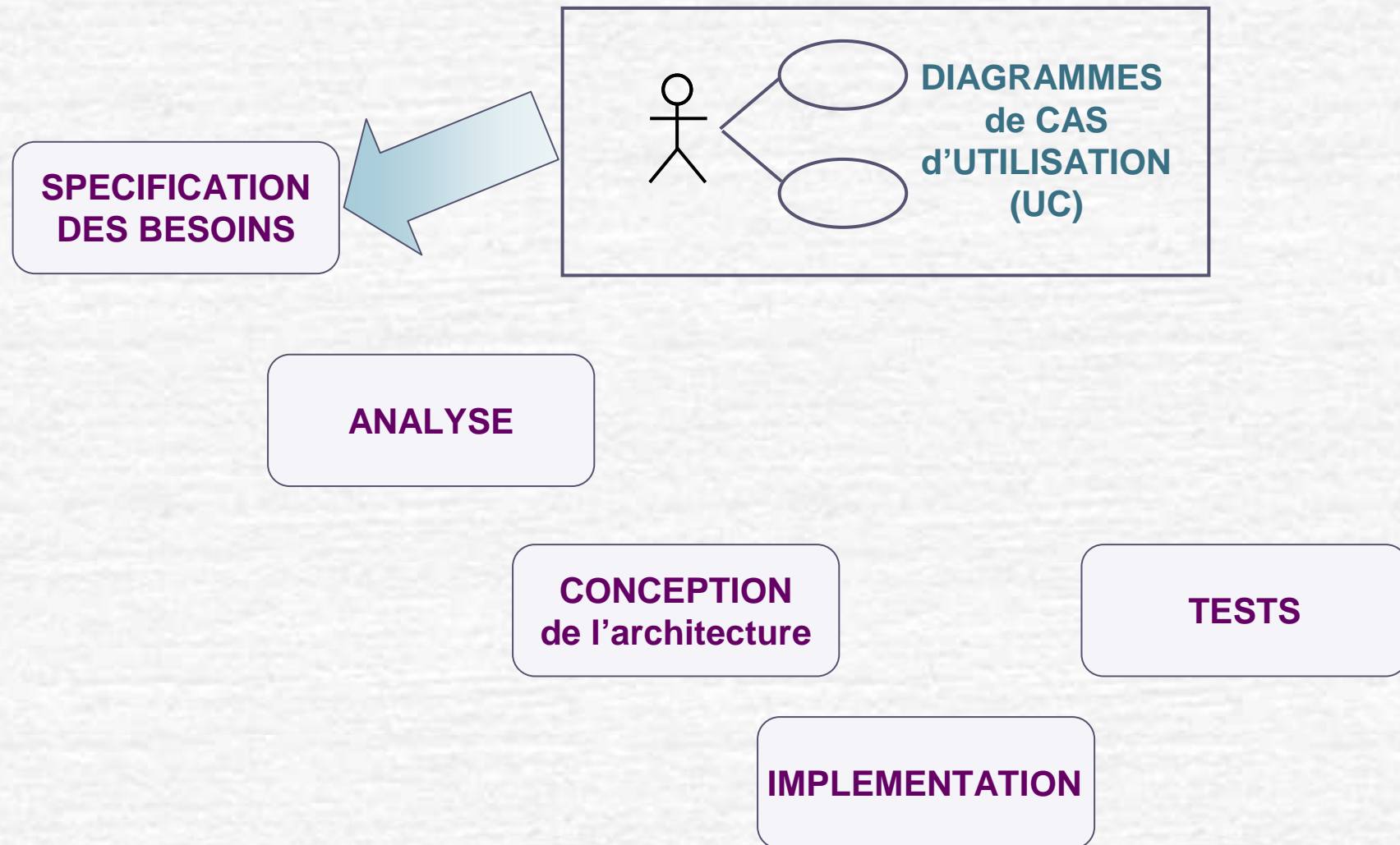
# Chapitre 5

## LES DIAGRAMMES DE CAS D'UTILISATION

(Use Case Diagrams ou UC)



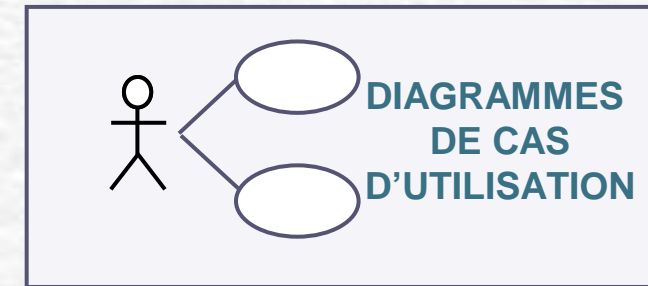
# DIAGRAMMES DE CAS D'UTILISATION





# RÔLE DES CAS D'UTILISATION

Les UC servent à recueillir la définition des besoins exprimés par les utilisateurs



**MODÈLE du  
DOMAINE**

**CLASSES**

**FONCTIONS**

**Modèles de Classes  
et d'Objets**

**Dynamique  
des Objets**

**Scénarios  
Fonctionnels**

**Modèle des  
Processus**

**MODÈLE  
TECHNIQUE**

## L'EXPRESSION DES BESOINS EST UNE TACHE DIFFICILE

L'utilisateur au  
moment de  
l'expression  
des besoins



L'utilisateur lors  
de la revue  
d'analyse



L'utilisateur  
après lecture  
des documents  
de conception



L'utilisateur  
le jour de  
la livraison  
et installation

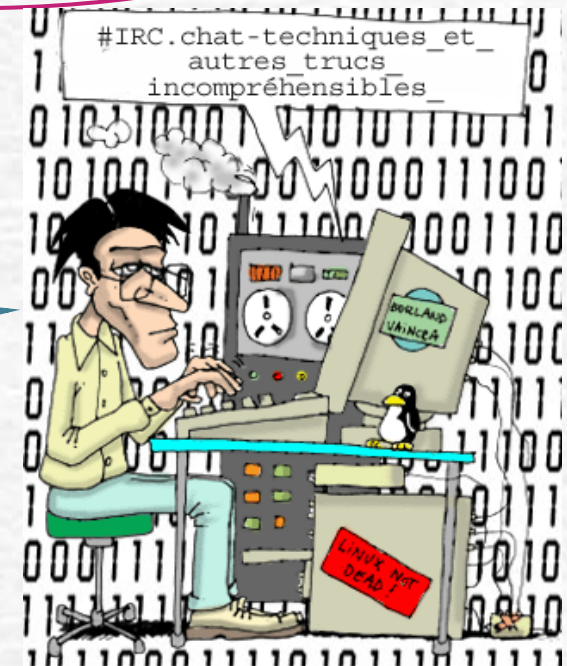


# UN PROBLÈME DE COMMUNICATION



- Expertise, jargon du métier
- Indécision, opinion changeante
- Besoins ambigus,
- Éléments manquants

- ❑ Schémas souvent incompréhensibles pour les non initiés.
- ❑ Langage énigmatique
- ❑ Descriptions longues et fastidieuses





# DIFFICULTÉS

- ☞ Définir de **QUOI** les utilisateurs ont vraiment besoin :
  - Poser le bon problème,
  - Ne pas laisser les utilisateurs se mêler de réalisation,
  - Ne pas inventer des fonctions pour le plaisir.
- ☞ Bannir toute considération de réalisation lors des premières rencontres,
- ☞ Comprendre le besoin de manière globale :
  - Flot incommensurable d'informations (il faut que...)
  - Contradictions entre les utilisateurs.
- ☞ Besoins mouvants.

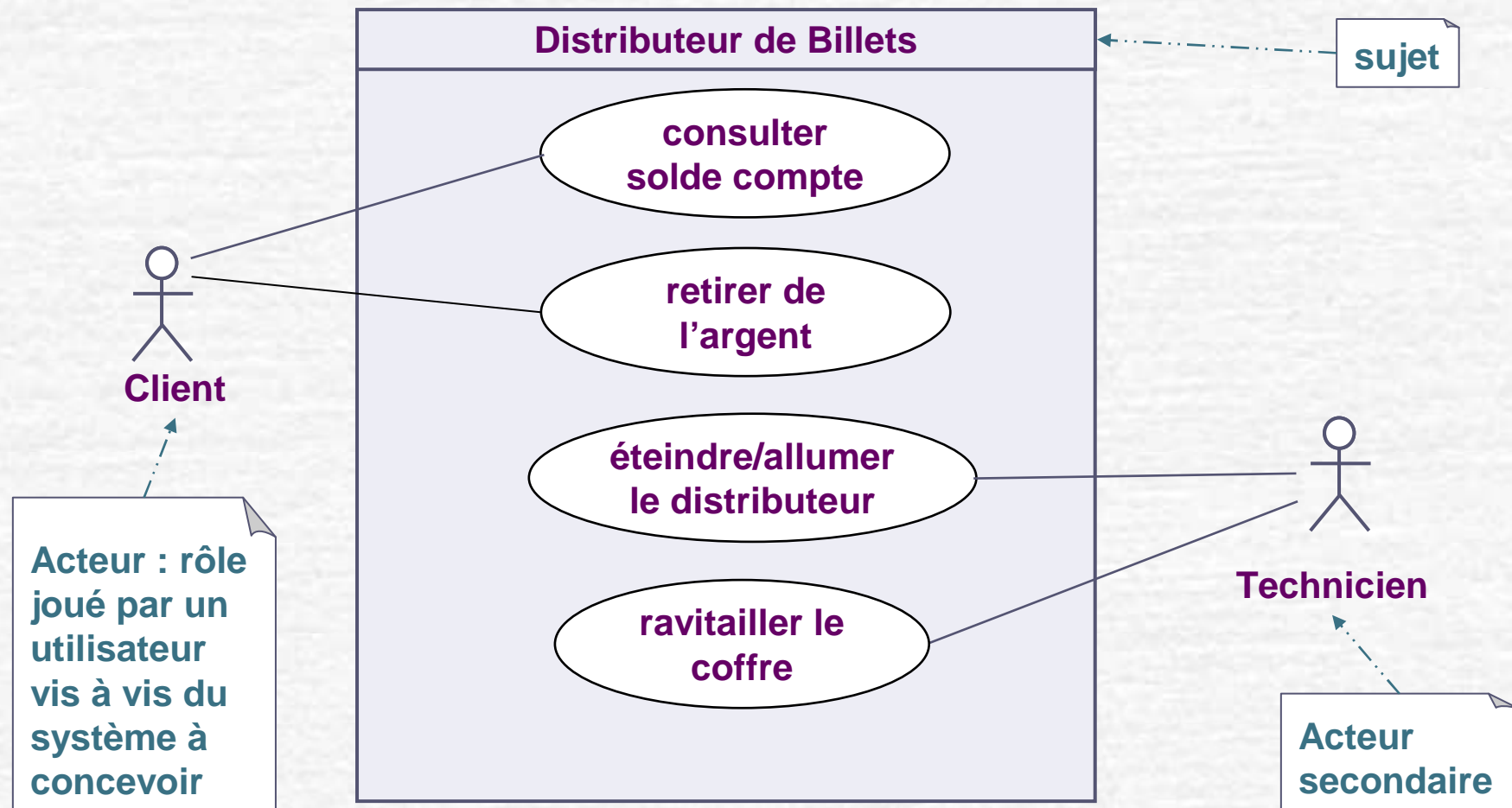


## LES CAS D'UTILISATION

- Les cas d'utilisation constituent une approche concrète.
- Les analystes ont souvent utilisé des scénarios pour comprendre les besoins.
- Ces scénarios étaient traités de manière informelle et rarement documentés.
- L'approche par les Cas d'Utilisation place les scénarios au premier plan dans les projets.
- La démarche est commune aux méthodes objets et aux méthodes traditionnelles.

## EXEMPLE

UC - Définition : objectif ou service que le système doit remplir, motivé par un besoin d'un ou plusieurs acteurs.



## QU'EST CE QU'UN CAS D'UTILISATION ?

- Représente une communication typique entre un utilisateur et un système informatique,
- Identifie une fonctionnalité visible de l'utilisateur.
- S'occupe d'un objectif élémentaire de l'utilisateur.
- Décrit la communication (données, événements) entre les entités extérieures et le système à concevoir.
- Le diagramme n'est qu'une "table des matières".



## ADOPTER LE POINT DE VUE DE L'UTILISATEUR

- Un Cas d'Utilisation modélise un service rendu à un utilisateur par le système.
  - Exemple, un traitement de texte, objectifs de l'utilisateur :
    - “Réaliser une présentation agréable pour un document ”.
    - “Rendre une présentation similaire à une autre”.
  - Solutions de l'informaticien (appelées interactions système) :
    - “ Définir un style ”
    - “ Changer un style ”.
    - “ Copier le style d'un document vers un autre ”.
- Une interaction (interne au) système correspond à une fonction du logiciel du point de vue de l'informaticien,
  - Ce sont des décisions à reporter le plus tard possible.



# SCÉNARIO



**Un cas d'utilisation est un ensemble de scénarios reliés par un but commun du point de vue de l'utilisateur**

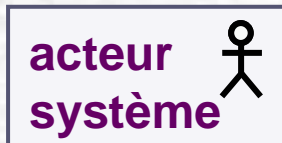
- ☛ Un scénario est une séquence particulière d'enchaînements s'exécutant du début à la fin du cas d'utilisation.
- ☛ Un cas d'utilisation contient en général un scénario nominal (le cas général) et plusieurs :
  - Scénarios alternatifs qui se terminent normalement (les cas particuliers),
  - Scénarios d'erreur qui se terminent en échec.

# ACTEURS

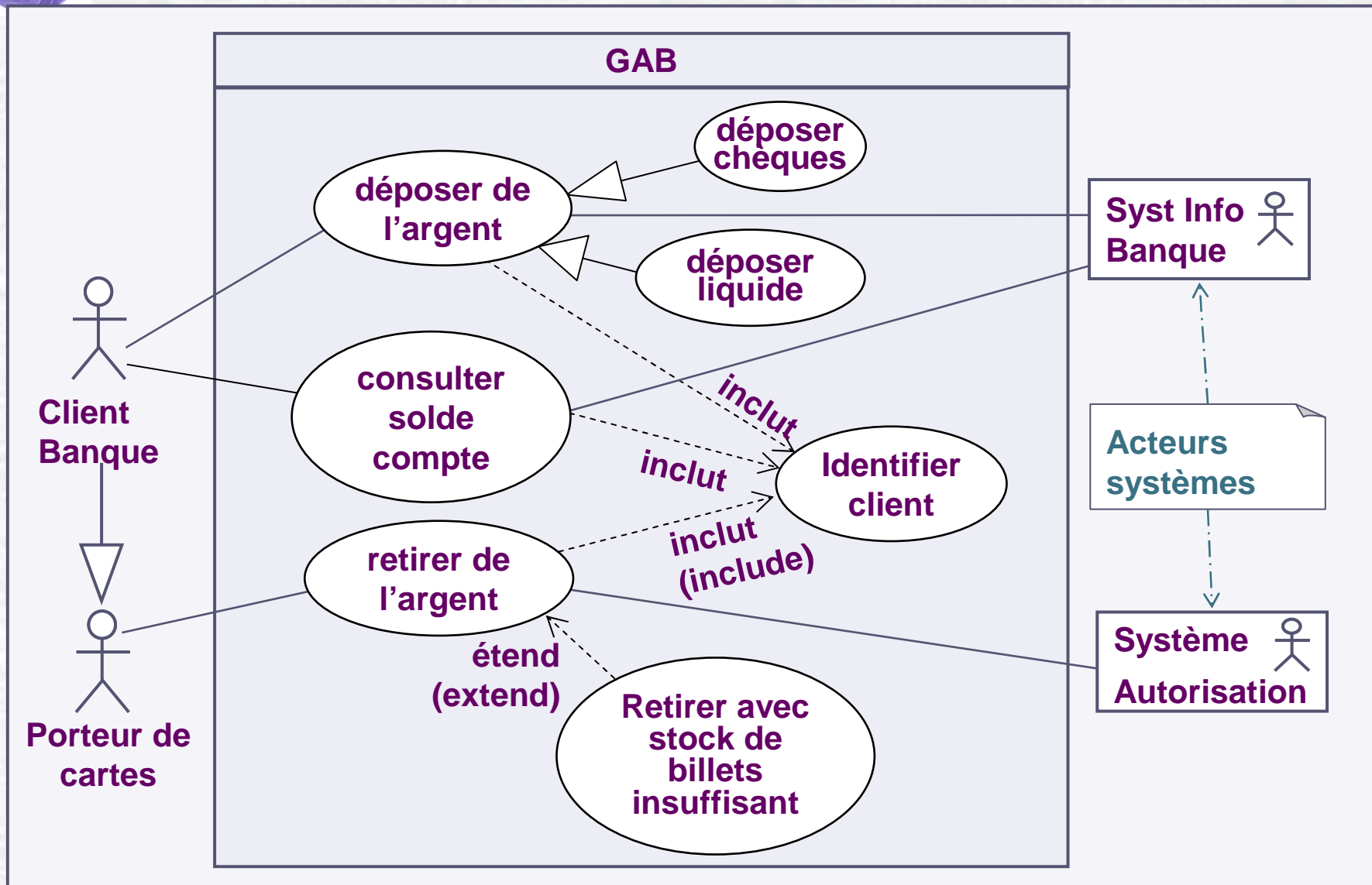
on associe un cas d'utilisation à chaque suite d'événements initiée par un acteur.



- ☛ Pour trouver les acteurs, on cherche :
- Quelles sont les principales tâches de chaque acteur ?
  - L'acteur a-t-il à lire, écrire, modifier une information du système ?
  - L'acteur doit-t-il informer le système de modifications extérieures ?
  - L'acteur doit-t-il être informé de modifications internes ?
  - Quels sont les acteurs secondaires ?
  - Y a-t-il des acteurs systèmes (d'autres logiciels en interaction) ?



## RELATIONS ENTRE UC



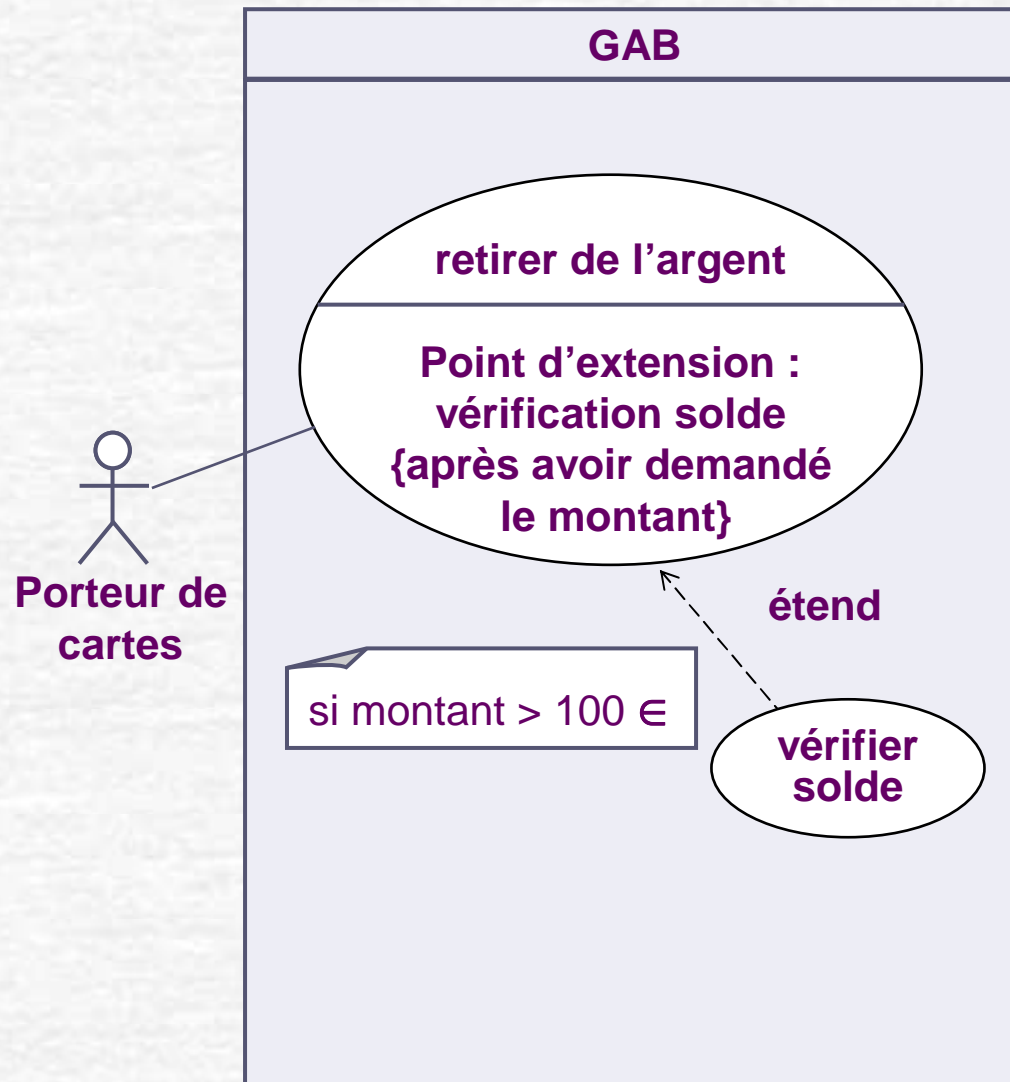


## RELATIONS ENTRE UC

- ☞ **Factorisation de comportements communs à plusieurs cas d'utilisation mais :**
  - intention différente,
  - liaisons différentes avec les acteurs .
- ☞ **U1 inclut U2 : U2 est un sous cas appelé en plusieurs points.**
- ☞ **U2 étend U1 : le comportement de U2 contient celui de U1 :**
  - U2 réalise quelque chose de plus que U1.
  - un acteur réalise le cas d'utilisation de base et toutes les extensions qui se présentent.
- ☞ **U1 généralise U2 : le comportement de U2 hérite celui de U1 (vrai aussi pour les acteurs).**
- ☞ **Etend : indique les variations d'un comportement normal (les cas particuliers importants).**
- ☞ **Inclut ou Généralise s'utilisent pour éviter les répétitions dans la description détaillée des UC.**



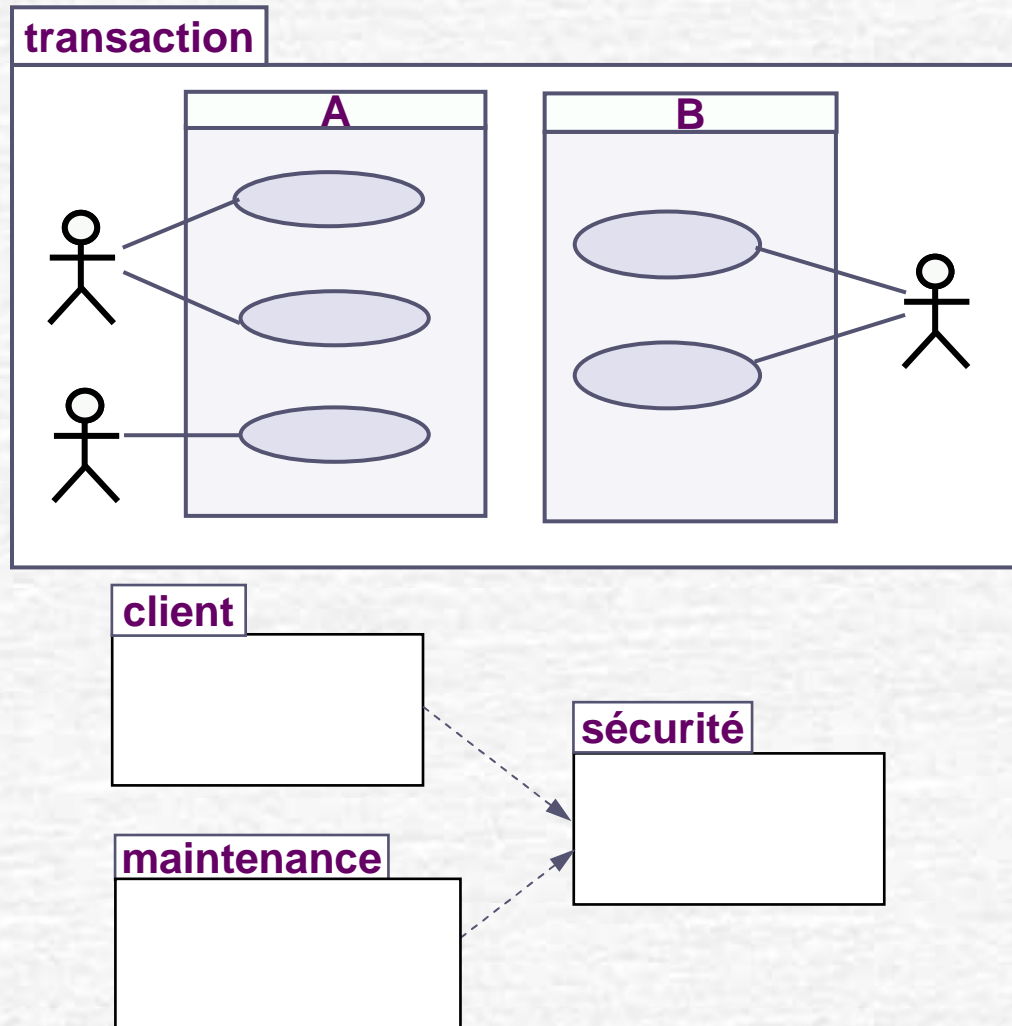
## POINT D'EXTENSION



- Une extension peut intervenir à un point précis du cas étendu.
- On l'indique dans un compartiment du cas d'utilisation,
- avec une contrainte indiquant le moment où il intervient.
- On peut indiquer une condition dans une note.

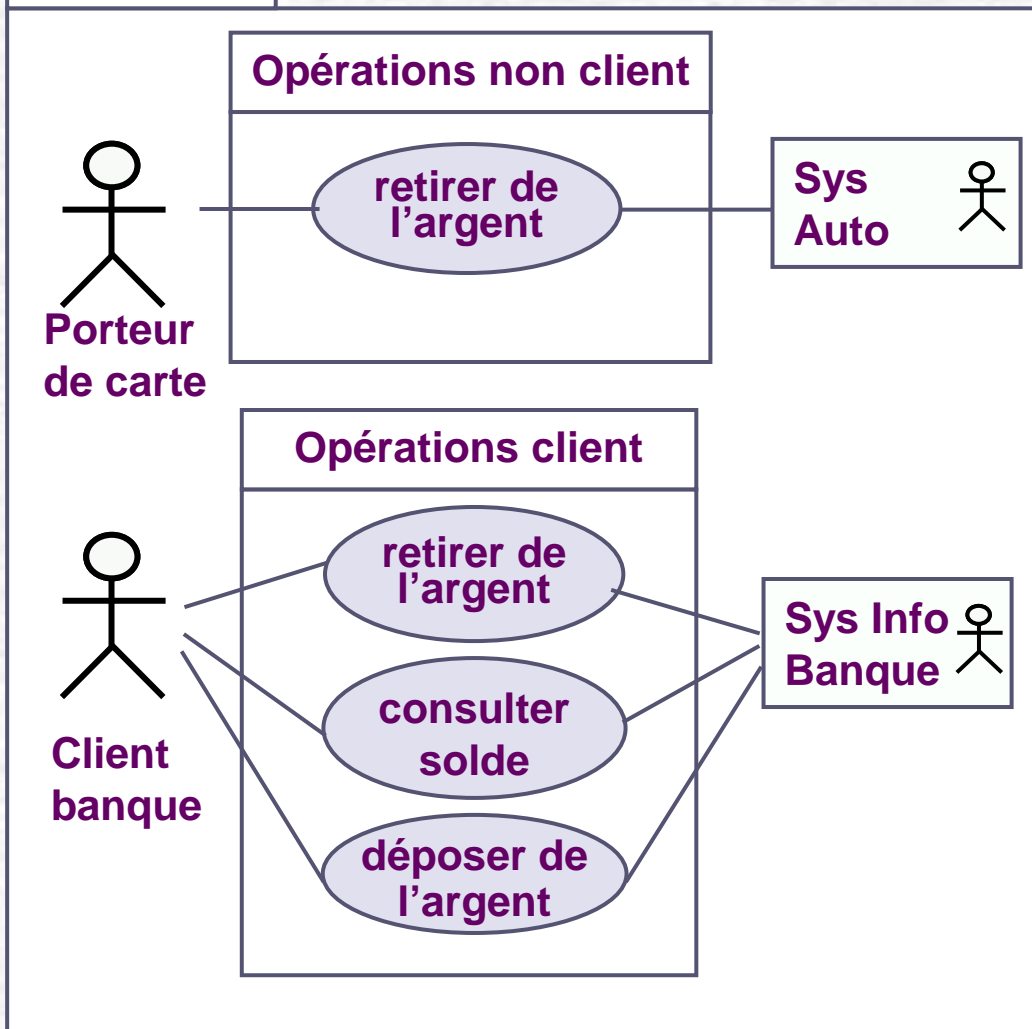
# GROUPEMENT DES CAS D'UTILISATION

Pour maîtriser la quantité et la complexité des UC, on les répartit en plusieurs conteneur, que l'on regroupe en paquets.

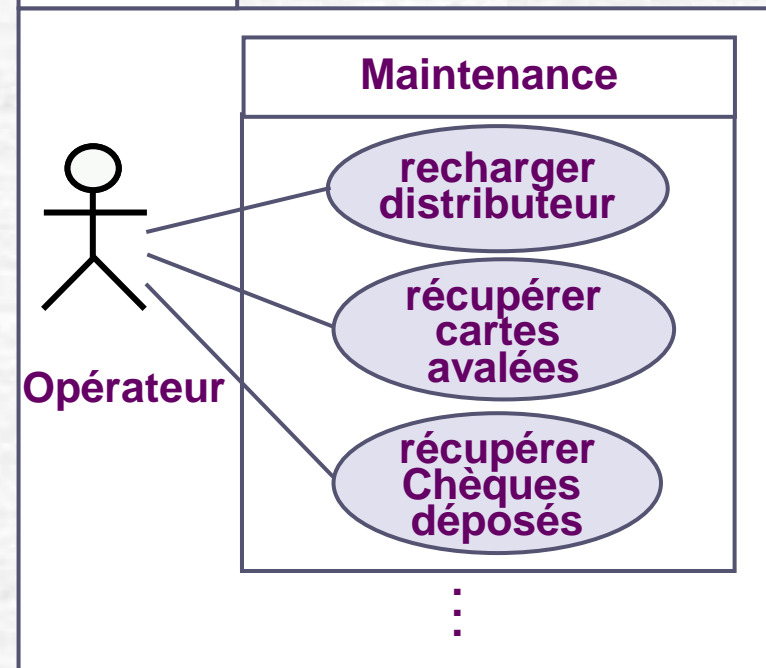


## EXEMPLE

### transactions



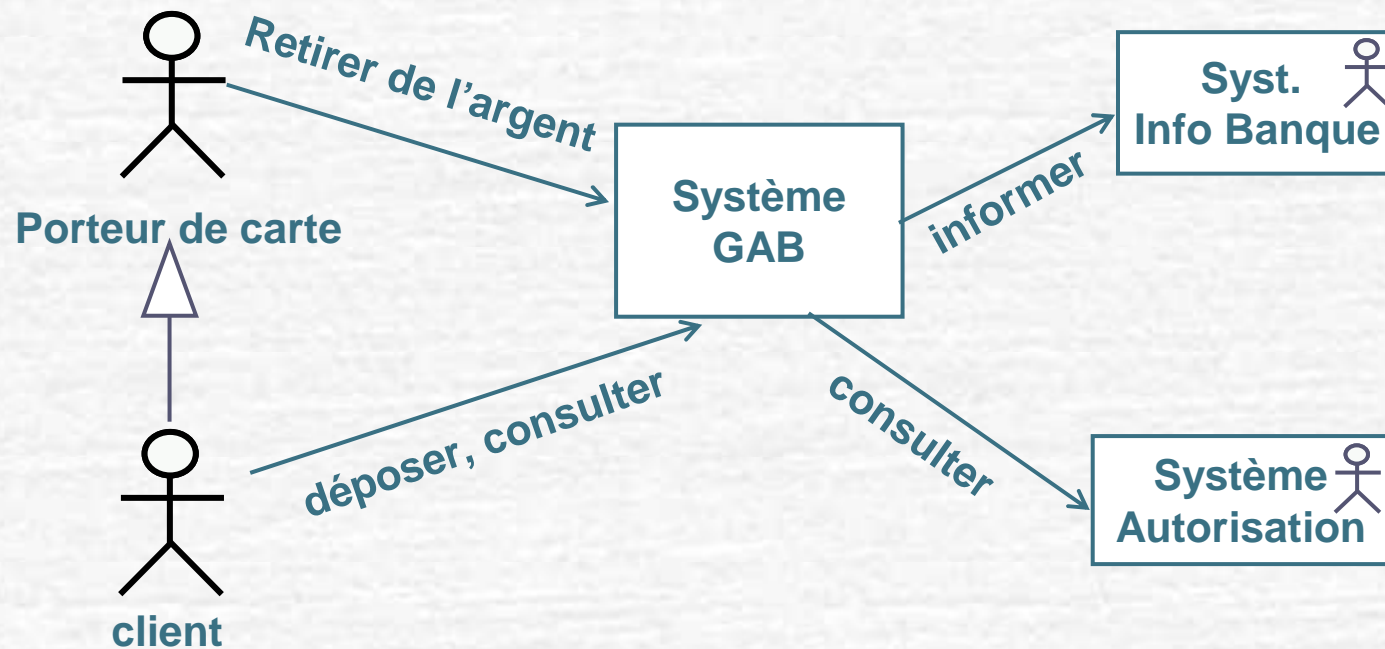
### service



On peut donner le même nom à deux cas d'utilisation différents s'ils n'appartiennent pas au même paquet.

# DIAGRAMME DE CONTEXTE STATIQUE

L'ensemble des UC peut se représenter dans un diagramme d'activités (appelé diagramme de contexte).







## DESCRIPTION DES UC (sommaire d'identification)

### EXEMPLE

Titre : Retirer de l'argent

Résumé : ce cas d'utilisation permet à un porteur de carte (non client de la banque) de retirer de l'argent, si son crédit hebdomadaire le permet.

Acteurs : porteur de carte (principal), système d'autorisation (secondaire)

Date de création, date de mise à jour, version, responsable, etc.

### On peut ajouter :

Fréquence d'exécution, sécurité, caractéristiques des acteurs, besoins en ergonomie...

## DESCRIPTION DÉTAILLÉE (description des scénarios)

- **Un cas d'utilisation = un processus complet lié à l'acteur principal (déclencheur) :**
  - Pré-conditions : contraintes sous lesquelles le cas peut démarrer,
  - Enchaînement nominal :
    - ex : connexion + calcul + impression...
  - Enchaînements alternatifs et d'erreur,
  - Post-conditions : contraintes après la fin du cas d'utilisation,
- **Enchaînement = une liste où chaque ligne est :**
  - Un message entre un acteur et le système,
  - Une activité réalisée par le système.

## CAS D'UTILISATION "retirer de l'argent "



- pré-conditions : la caisse du GAB est alimentée.
  - aucune carte ne se trouve déjà dans le lecteur.

- Scénario nominal :

1. Le porteur de carte introduit sa carte dans le lecteur.
2. Le GAB vérifie si la carte est une carte bancaire.
3. Le GAB demande au porteur de saisir son code.
4. Le GAB compare le code saisi avec celui de la carte.
5. Le GAB demande une autorisation au Sys. d'autorisation global
6. Le Sys.Auto. Donne son accord et fournit le solde hebdomadaire.
7. Le GAB demande au porteur le montant du retrait...

- Post-conditions : la caisse du GAB contient moins de billets qu'au début du scénario et une transaction a été enregistrée.



## CAS D'UTILISATION "retirer de l'argent "



### Enchaînement alternatif :

A1 : code d'identification provisoirement erroné : démarre au point 5 du scénario nominal.

6. Le GAB indique au porteur de carte que le code est erroné pour la 1ère ou la seconde fois.
7. Le GAB enregistre l'échec sur la carte

Le scénario nominal reprend au point 3.

### Enchaînement d'erreur :

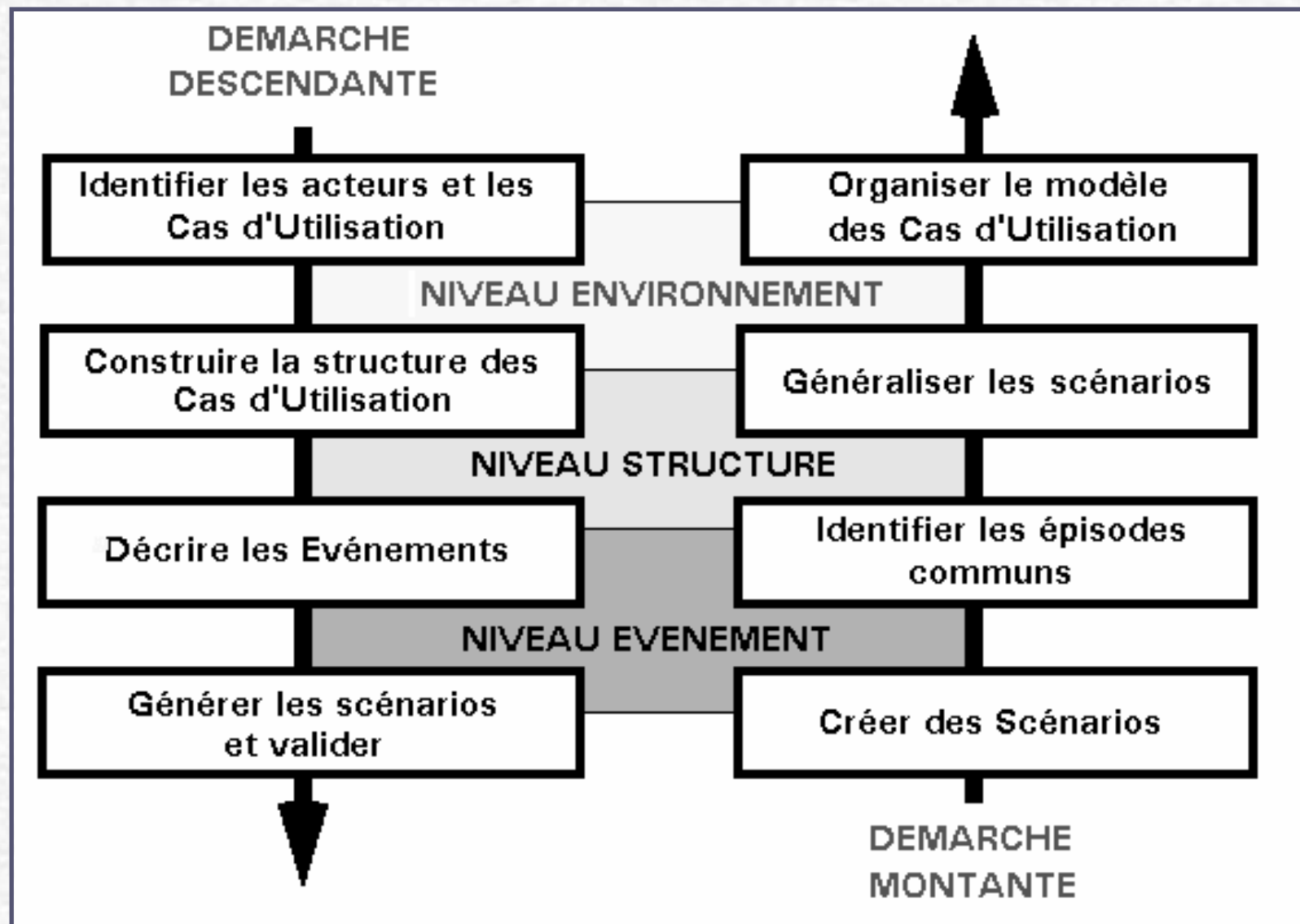
E1 : carte non valide : démarre au point 2 du scénario nominal.

3. Le GAB indique au porteur que la carte n'est pas valide (illisible, périmée...) et la confisque. Le cas d'utilisation est terminé.

E2 : code d'identification définitivement erroné ...



# DÉMARCHES DE CONSTRUCTION



## BESOINS NON FONCTIONNELS

- Les UC permettent de spécifier les besoins fonctionnels d'un système logiciel.
- Les besoins non fonctionnels sont des critères de qualité, pour chaque service et pour le système global (norme ISO 9126) :
  - **FACILITÉ D'UTILISATION** : ergonomie, esthétique, facilité d'apprentissage, cohérence de l'IHO, de la documentation et du matériel de formation.
  - **FIABILITÉ** : fréquence et sévérité des fautes, facilité de reprise après panne, prédictibilité et précision des résultats.
  - **PERFORMANCE** (rendement) : temps de réponse, charge, temps de récupération après erreur, quantité de mémoire.
  - **MAINTENABILITÉ** : testabilité, évolution du système après livraison.



## MEDUSE (LE RETOUR)

- ☞ La médiathèque de l'Université des Schtroumfs Erudits (USE) possède des ouvrages. Ils peuvent être des livres, des CD ou des DVD qui existent en plusieurs exemplaires. Les informations à stocker sur un ouvrage dépendent de son type.
- ☞ La médiathèque est gérée par des documentalistes et est fréquentée par des lecteurs. Ces lecteurs sont des étudiants de l'USE, des enseignants ou des extérieurs. Ils peuvent emprunter 5 ouvrages au maximum lorsqu'ils sont inscrits. Un lecteur ayant commis des abus (ouvrages rendus en retard, détériorations...) peut être interdit d'emprunt (durée décidée par le documentaliste).
- ☞ Pour pouvoir être emprunté, un exemplaire doit être disponible (non emprunté, non réservé). Chaque emprunt a une durée limitée à 4 semaines.
- ☞ Un exemplaire emprunté peut être réservé par un autre lecteur, cette réservation reste effective pendant 2 semaines après la date de retour du livre. Passé ce délai la réservation est annulée.
- ☞ Le documentaliste peut ajouter des ouvrages ou des exemplaires d'un ouvrage dans MEDUSE, il peut également en supprimer (perte, vieillissement...).
- ☞ Seul le documentaliste peut exécuter les mises à jour des ouvrages, des lecteurs et des emprunts. Pour chaque exemplaire on conserve l'emprunteur courant.
- ☞ Les lecteurs effectuent des recherches sur la discipline, des mots clés, l'auteur, le titre, la date de parution... Pour cela ils utilisent MEDUSE en consultation.

## EXERCICE 2 : POMPE À ESSENCE

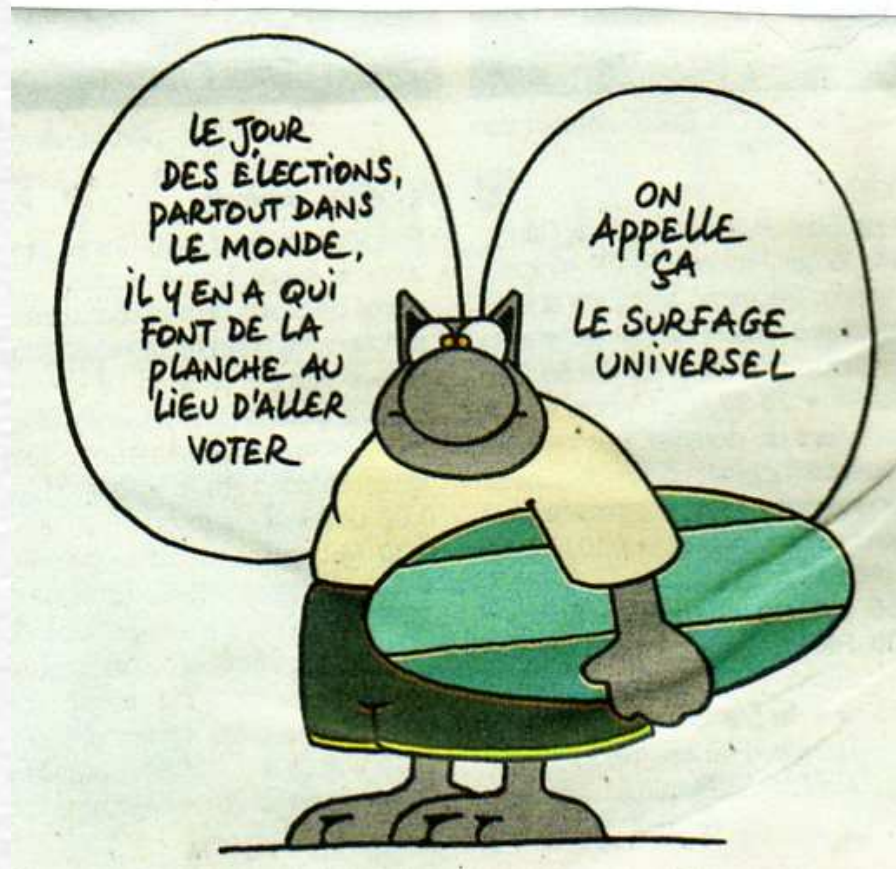
Un logiciel pour gérer une station service a le fonctionnement suivant :

- Avant de pouvoir être utilisée par le client, chaque pompe à essence doit être armée par le pompiste. La pompe est alors prête, mais ce n'est que lorsque le client appuie sur la gâchette du pistolet que l'essence est pompée (servie). Si le pistolet est sur son support, même si la gâchette est pressée, l'essence n'est pas pompée.
- La distribution d'essence au client est terminée quand celui-ci remet le pistolet sur son support. Le débitmètre fournit alors la quantité d'essence distribuée.
- Le client peut payer en liquide, par chèque ou par carte. Le système enregistre le montant et le mode de paiement. En fin de journée les transactions sont transmises au système comptable de la société.
- Si le niveau de la cuve correspondant à une pompe est inférieur à 5% de la capacité maximale, la pompe ne peut plus être armée.



## Chapitre 6

# LES DIAGRAMMES DE SEQUENCES

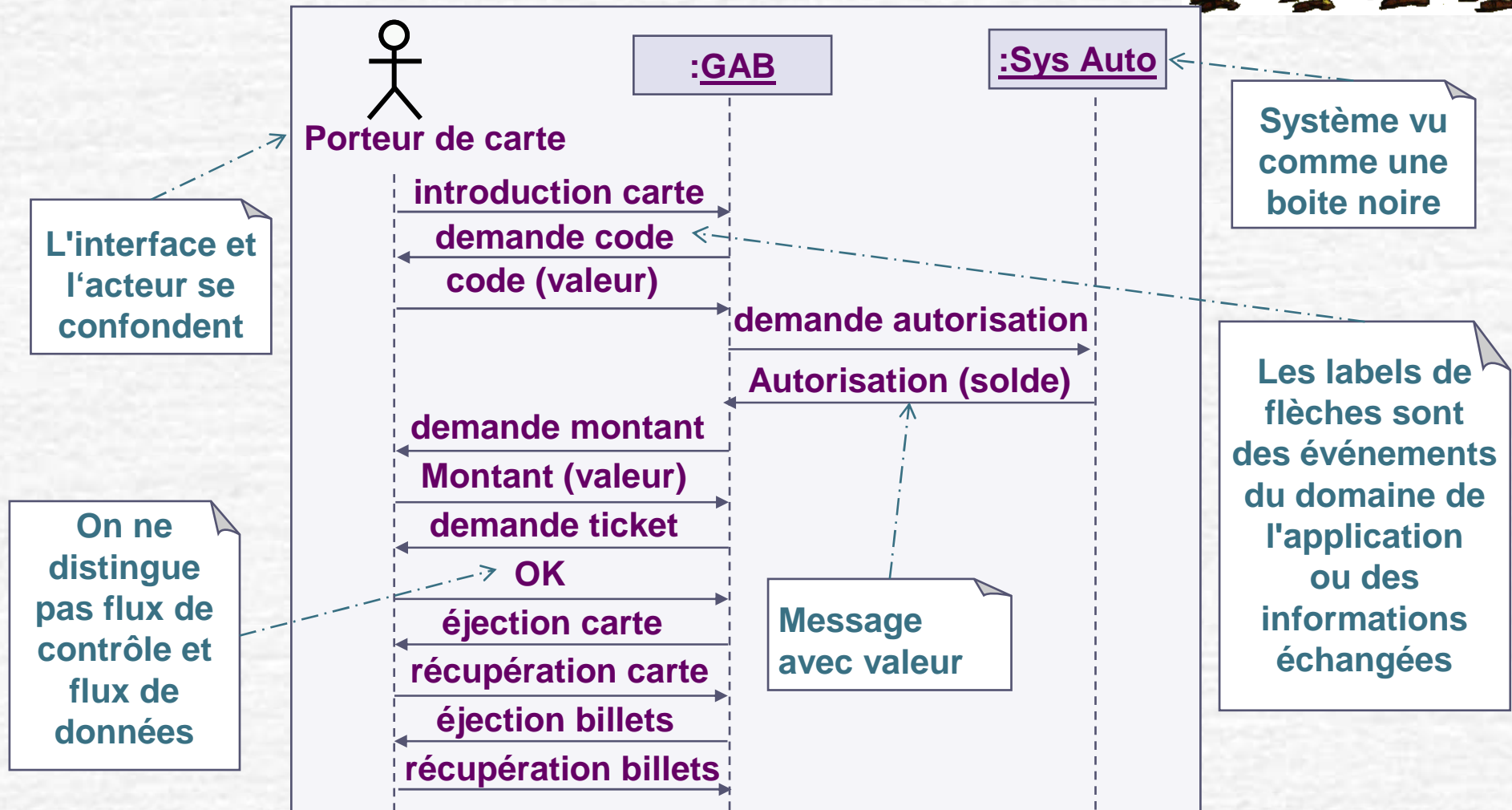


## COMPORTEMENT DES OBJETS

- ☞ Pour monter ou concevoir le comportement des objets, on a :
  - ☞ Les diagrammes d'États : comportement d'une classe.
  - ☞ Les diagrammes d'interactions : échanges de messages entre plusieurs objets, 2 types :
    - Diagramme de Séquences,
    - Diagramme de Communication,
  - ☞ Diagrammes d'Activités : description d'un processus.
- } F2 F3

**À UTILISER PARTOUT OÙ ON EN A BESOIN :  
domaine, exigences, conception détaillée...**

# DIAGRAMME DE SEQUENCE SYSTEME = DOCUMENTATION DES CAS D'UTILISATION

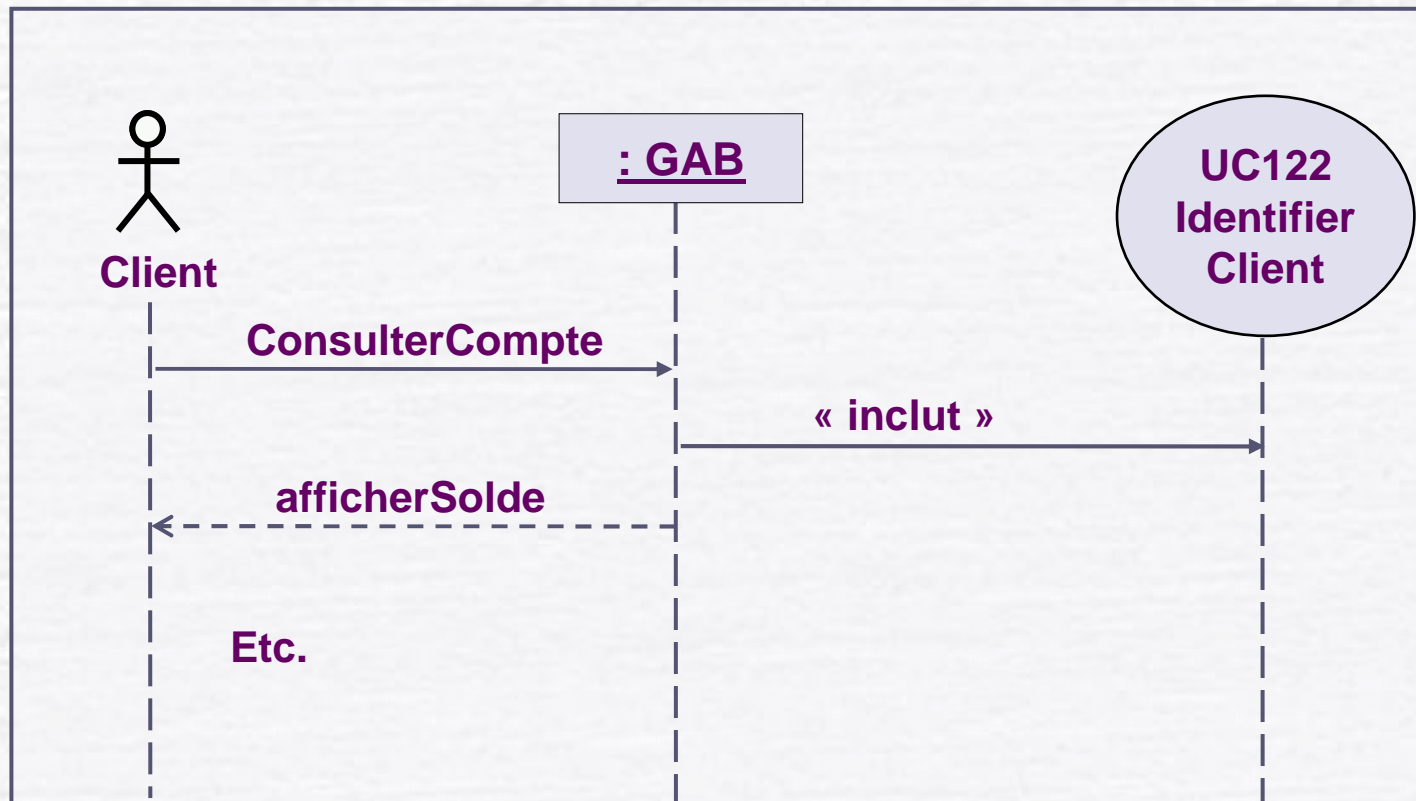


exemple du chapitre précédent





## INCLUSION D'UN CAS D'UTILISATION



La relation « inclut » des cas d'utilisation permet de structurer les diagrammes





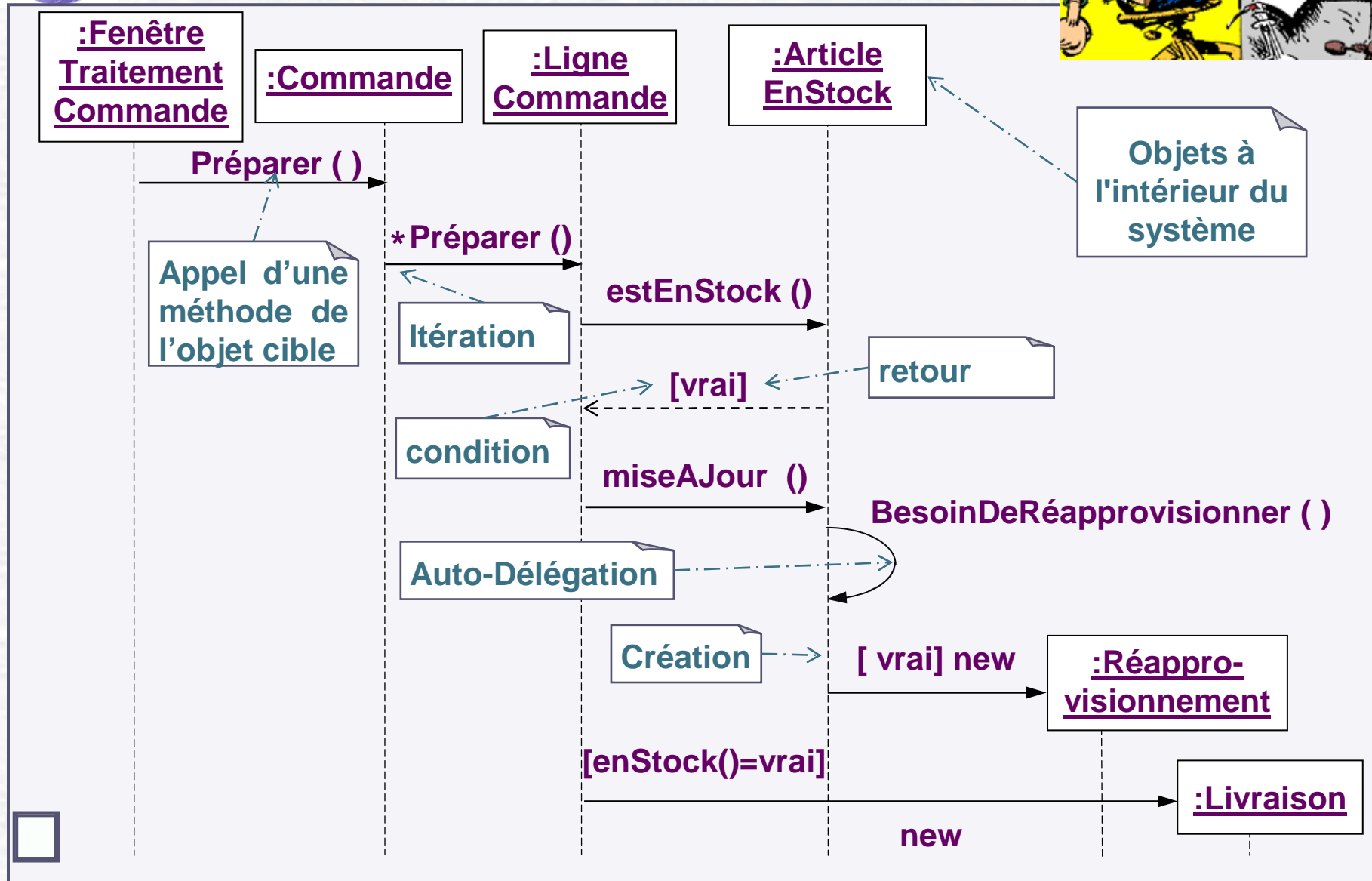
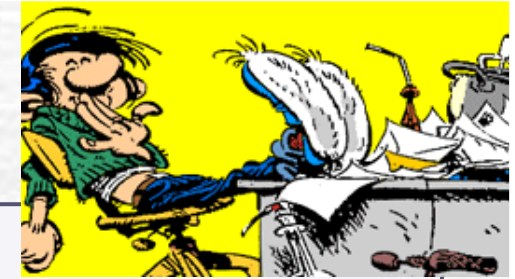
# DIAGRAMME DE SEQUENCES DE REALISATION

## EXEMPLE HYPER BASIQUE

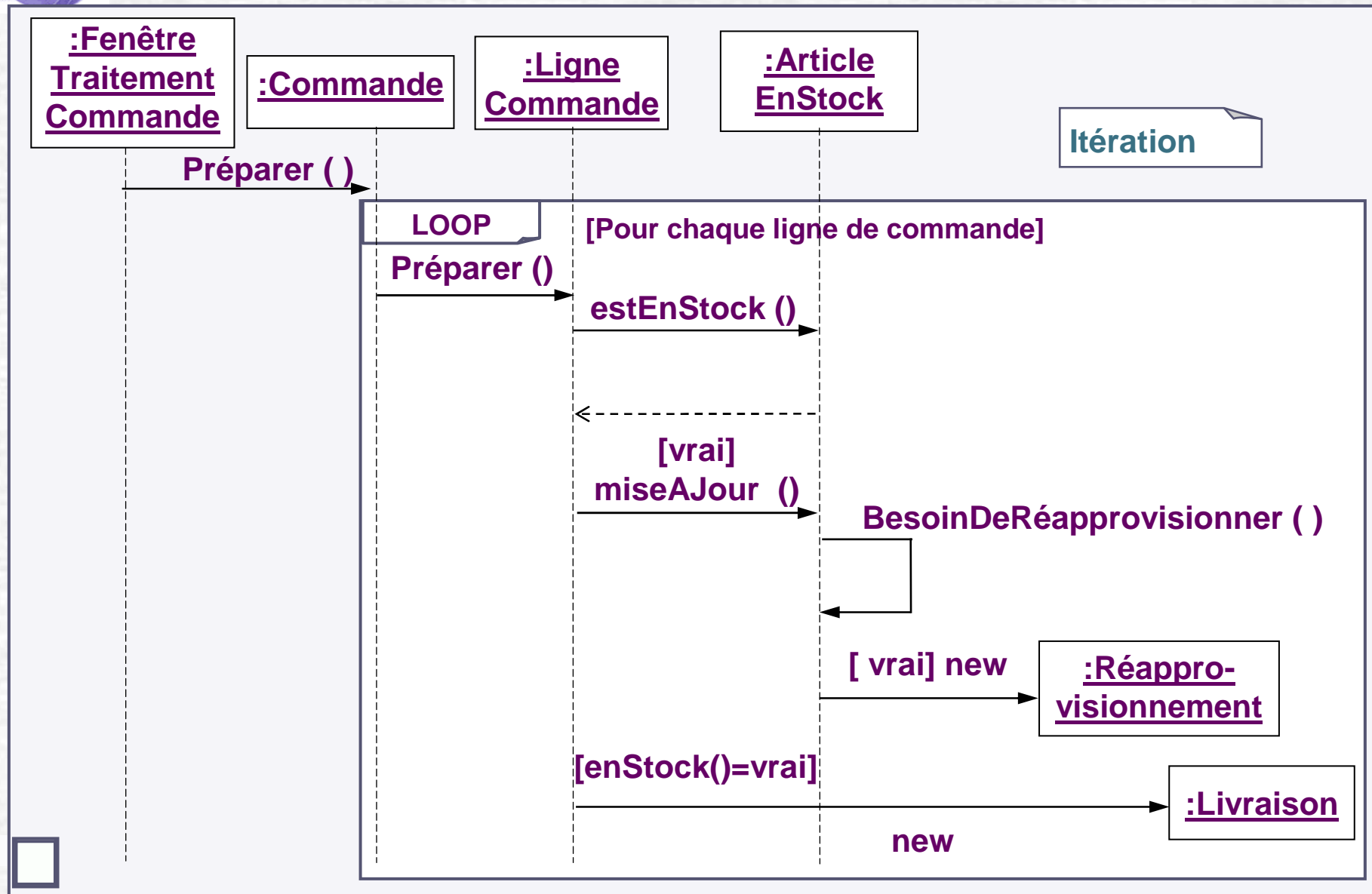
- Dans une application d'achats en ligne, la préparation des commandes est lancée via l'IHO de l'appli.
- Le client a commandé plusieurs articles : chaque article fait l'objet d'une ligne de commande.
- Pour chaque ligne de commande, il faut vérifier que l'article est en stock en quantité suffisante (le cas sinon n'est pas traité ici).
- Pour chaque article livré, si le stock est en dessous du seuil critique, on lance un réapprovisionnement.

- Les « rectangles » sont des **OBJETS**, c'est-à-dire des instances des classes de conception.
- Ces objets peuvent être nommés. On peut montrer plusieurs objets de la même classe.
- On peut utiliser une flèche de « retour » d'une méthode (seulement dans les cas où un retour implicite n'est pas clair).

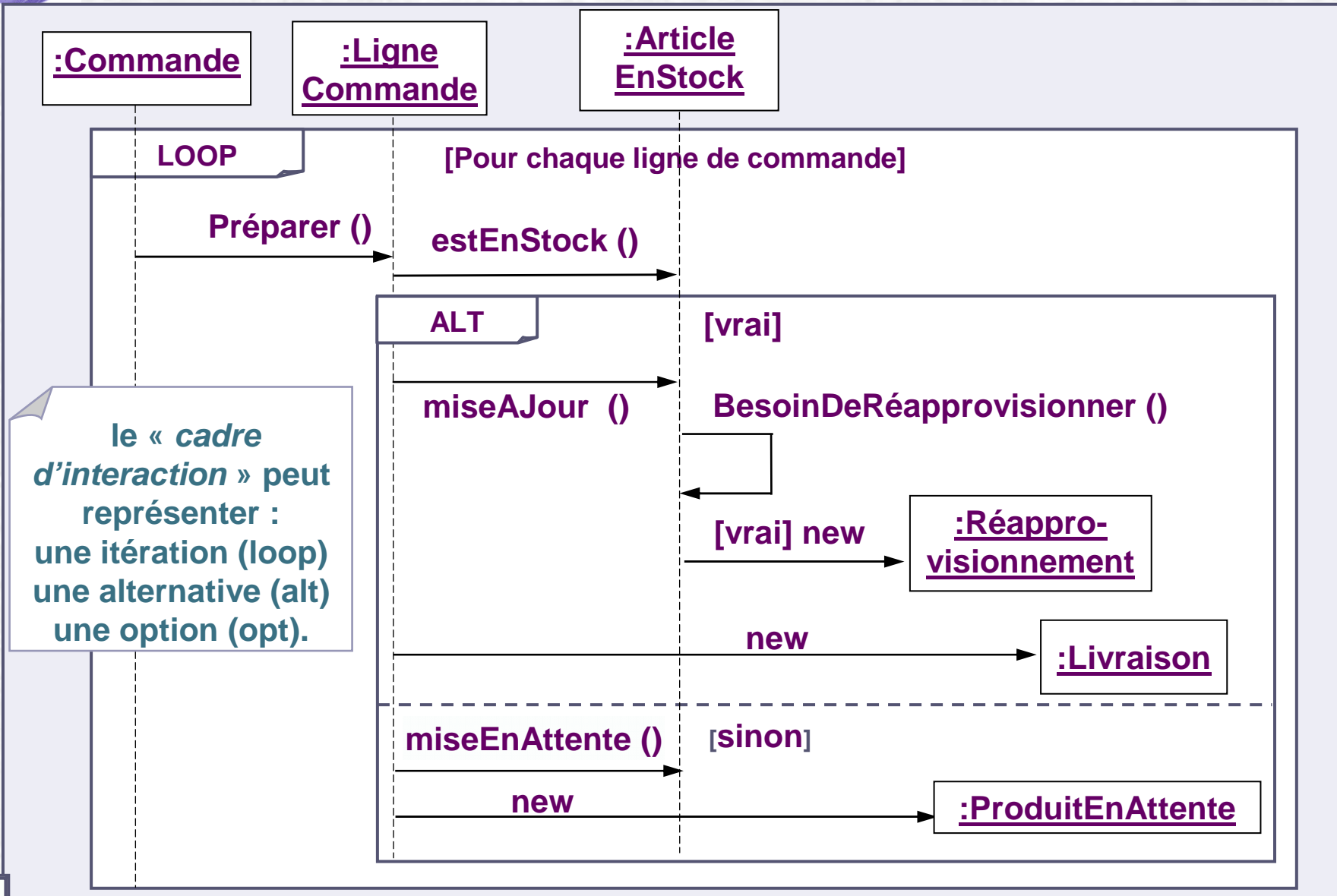
# INTERACTIONS ENTRE OBJETS



# ITÉRATIONS EN UML 2.0



## ALTERNATIVES EN UML 2.0



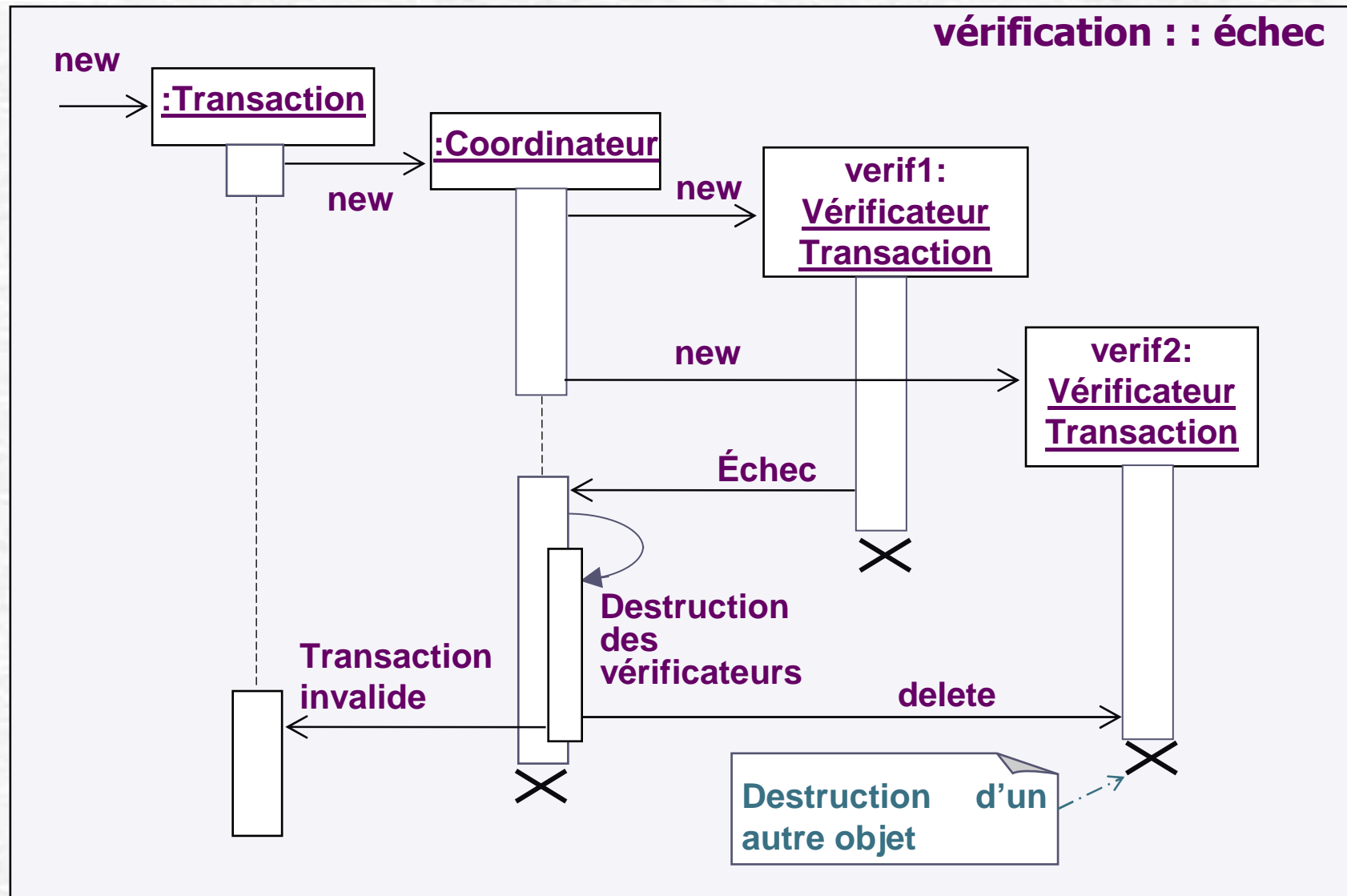


# DIAGRAMMES DE SÉQUENCES ET PROCESSUS CONCURRENTS

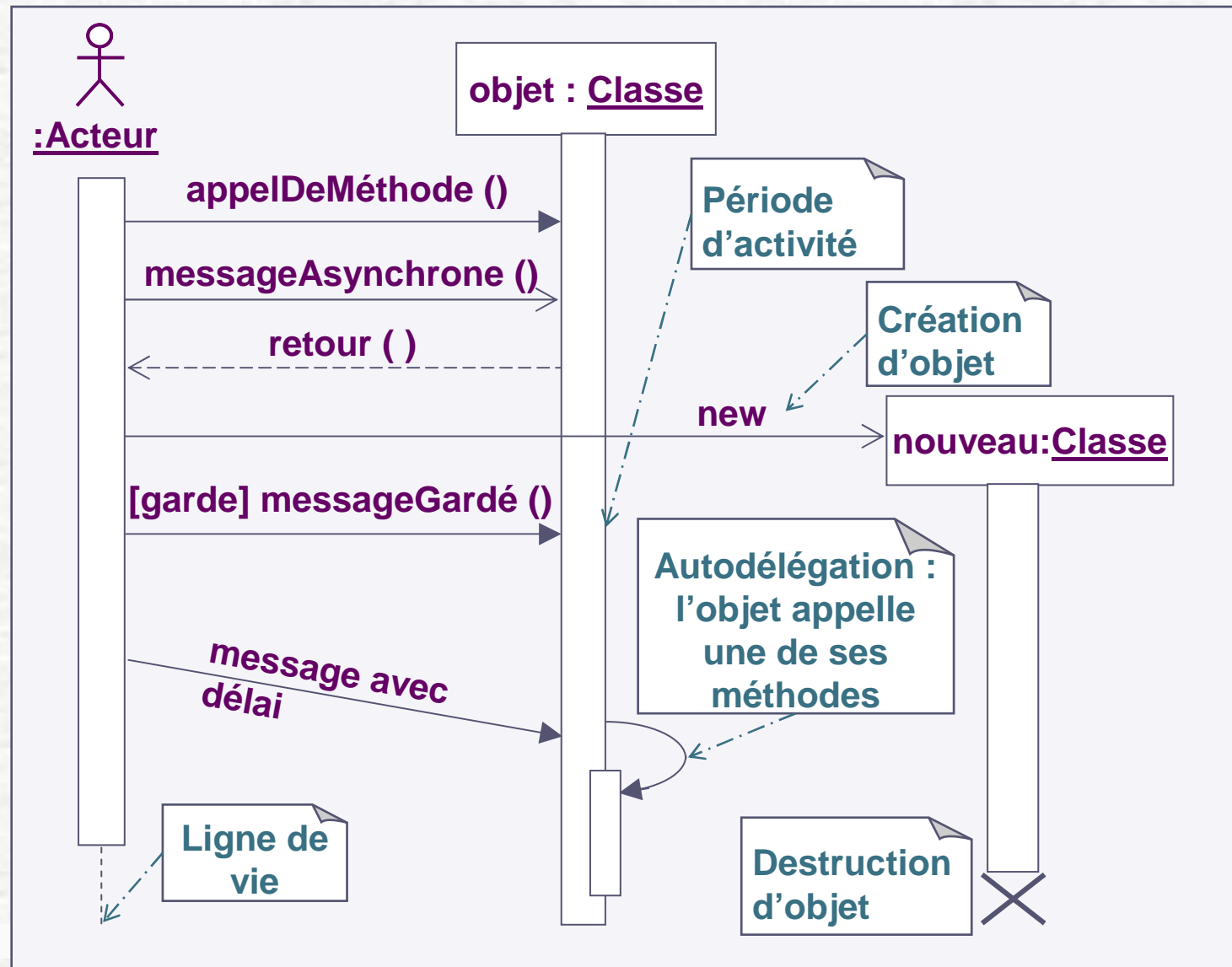
- ☛ Les diagrammes de séquences distinguent deux catégories d'envoi de messages :
  - Les envois synchrones pour lesquels l'émetteur est bloqué et attend que l'appelé ait fini de traiter le message (méthodes),
  - Les envois asynchrones, où l'émetteur n'est pas bloqué et peut continuer son exécution (processus communicants).
- ☛ Exemple : objets vérifiant une transaction bancaire :
  - Lorsqu'une transaction est créée, elle crée un coordinateur de transaction pour l'ensemble des vérifications.
  - Ce coordinateur crée un certain nombre (ici 2) d'objets vérificateurs, chacun responsable d'une vérification. Les objets vérificateurs peuvent être appelés de manière asynchrone et ils s'exécutent en parallèle.
  - Lorsqu'un vérificateur se termine correctement, il avertit le coordinateur qui s'assure que tous les vérificateurs sont terminés. lorsque tout est OK, un signal est envoyé à la transaction.



# PROCESSUS CONCURRENTS ET ACTIVATIONS

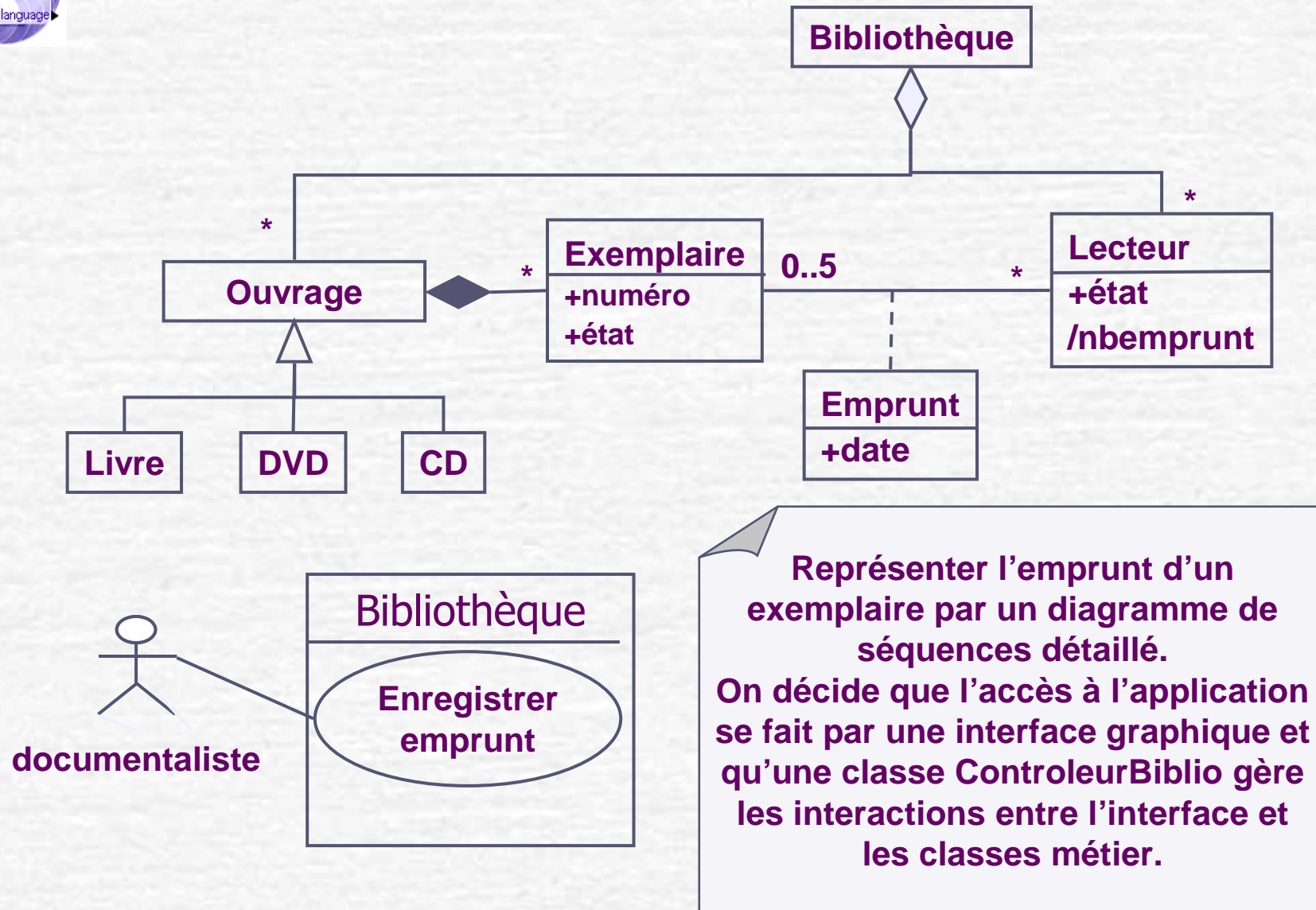


# SYNTAXE





# MEDUSE (LA VENGEANCE)

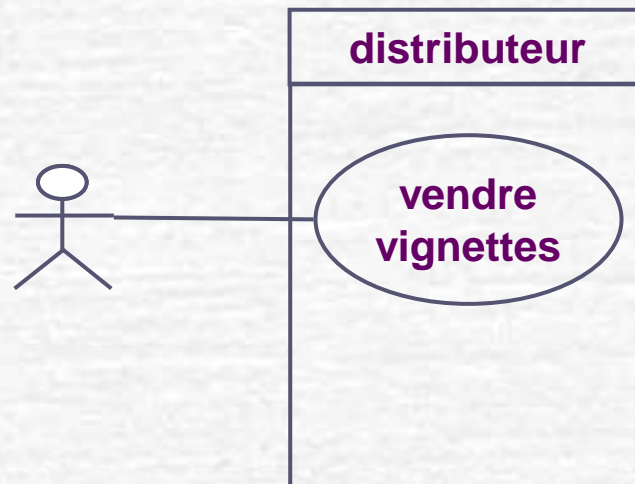
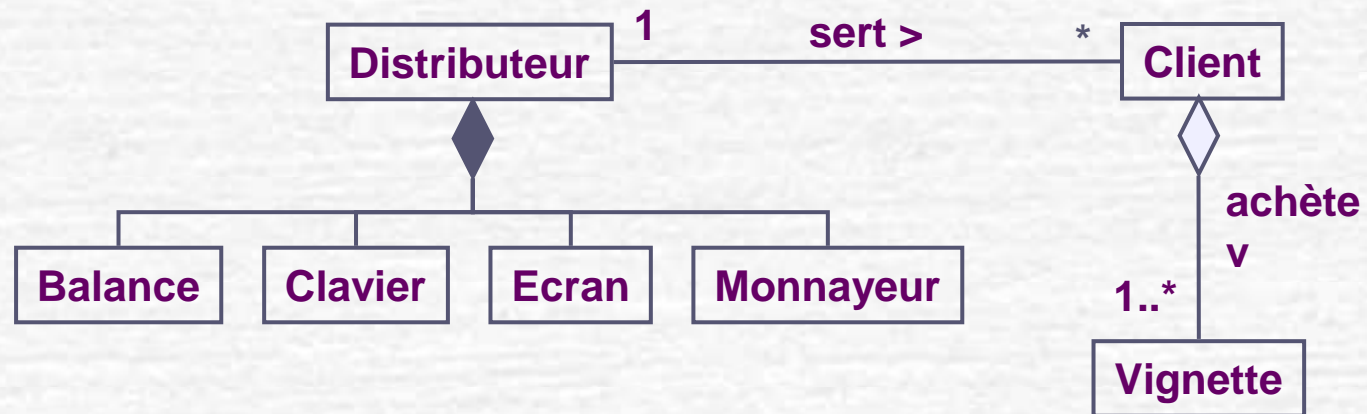


## EXERCICE

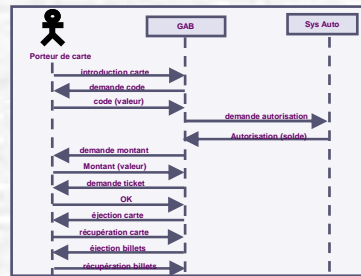
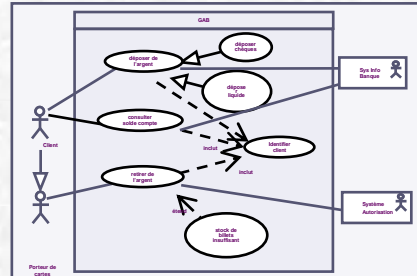
**Au lieu de faire la queue pour affranchir vos lettres, vous préférez utiliser le distributeur automatique, il faut :**

- Initialiser le distributeur (p. ex. un bouton sur l'écran tactile)
- Poser une lettre sur la balance,
- Choisir le tarif d'expédition sur l'écran tactile,
- L'écran affiche alors le prix et demande si d'autres lettres sont à affranchir.
- Si oui, le même scénario se répète (à partir de poser une lettre),
- Sinon il faut payer : le montant total s'affiche et vous devez introduire les pièces.
- La monnaie est rendue et les vignettes sont délivrées.

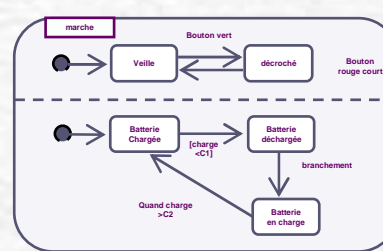
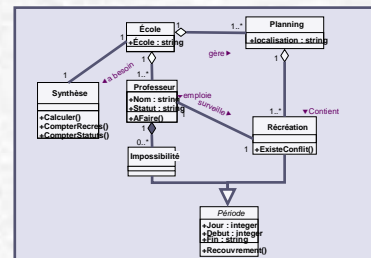
**NB : Représenter le distributeur sous forme de plusieurs objets.**



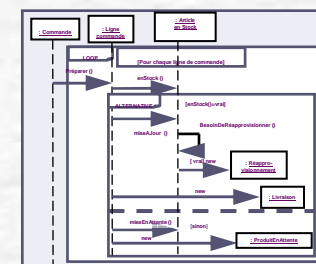
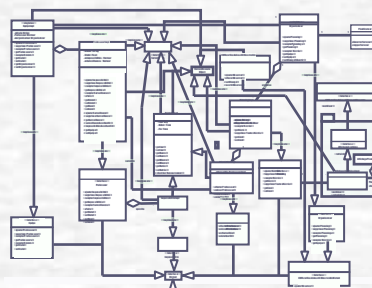
# OÙ SOMMES NOUS ??



**Spécification des besoins :**  
**Cas d'Utilisation +**  
**description détaillée**  
**(texte ou diag de Séquences système)**



**Analyse :**  
**Diag. de Classes +**  
**États-Transitions**



**Conception :**  
**Classes +**  
**Séquences +**  
**États-Transitions**

**Implémentation**



# Chapitre 10

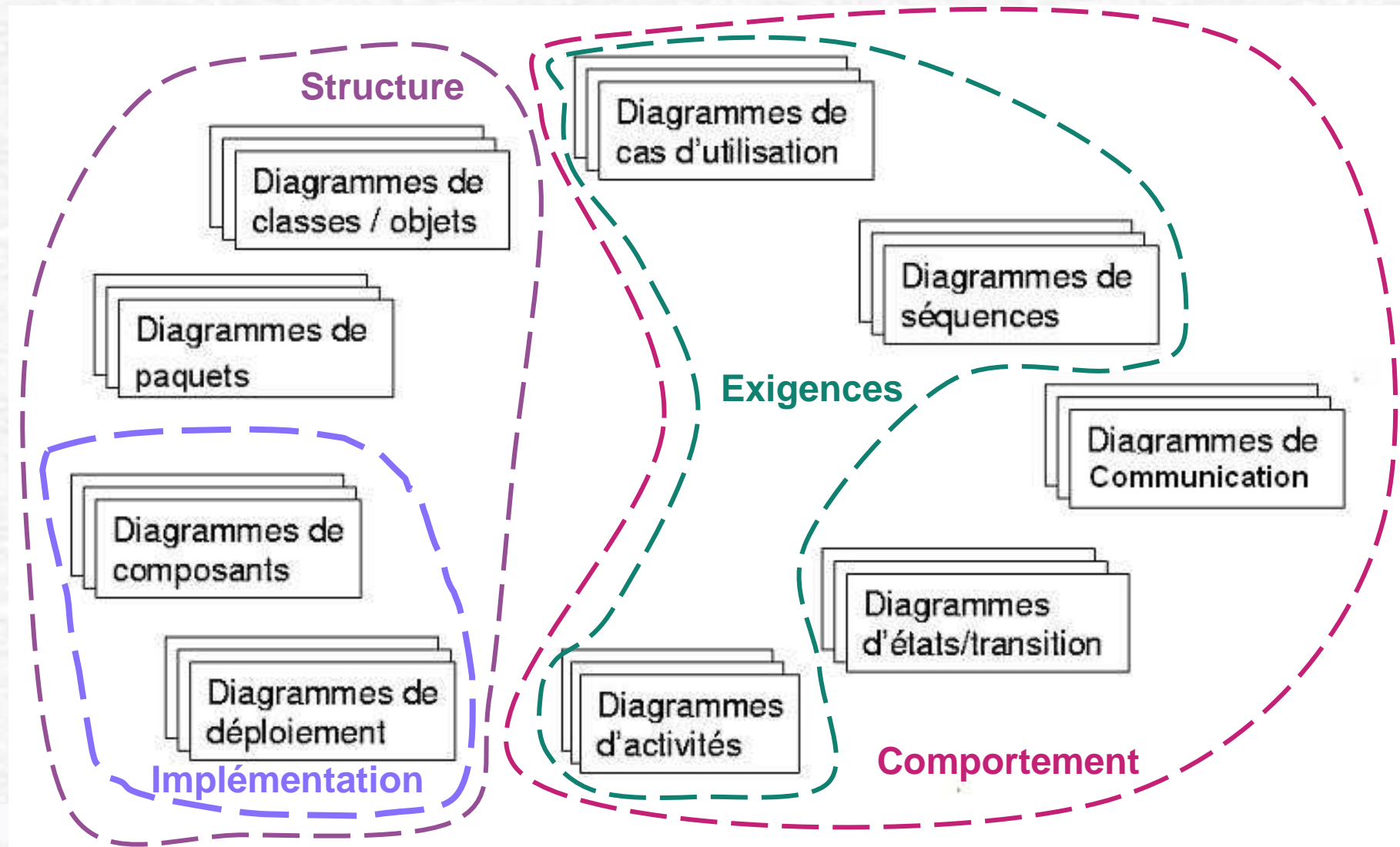
## RETOUR SUR LE DÉVELOPPEMENT DE LOGICIEL



## SYNTHÈSE : QU'EST-CE QU'ON A ?

DIAGRAMME	OBJECTIF
<b>Diagramme de Classes et d'Objets</b>	<b>Représenter les classes et les relations entre classes : Domaine : structure statique du système étudié ; Technique : architecture du logiciel.</b>
<b>Diagramme de Cas d'Utilisation</b>	<b>Décrire le comportement du système du point de vue de l'utilisateur : fonctionnalités du système, acteurs externes et leurs relations.</b>
<b>Diagramme de Séquences</b>	<b>Décrire la chronologie des messages échangés : objets, scénarios caractéristiques d'envoi de messages</b>
<b>Diagramme de Communication</b>	<b>Décrire la communication entre les objets par l'envoi et la réception de messages.</b>
<b>Diagramme d'Etats</b>	<b>Décrire le cycle de vie des objets : les états que traverse un objet pendant sa durée de vie.</b>
<b>Diagramme d'Activités</b>	<b>Décrire les processus métiers de haut niveau ou les actions d'une opération complexe</b>
<b>Diagramme de Composants</b>	<b>Décrire les composants de la STRUCTURE du logiciel</b>
<b>Diagramme de Déploiement</b>	<b>Représenter les différents sites qui supportent des composants</b>

## SYNTHESE : QU'EST-CE QU'ON A ?

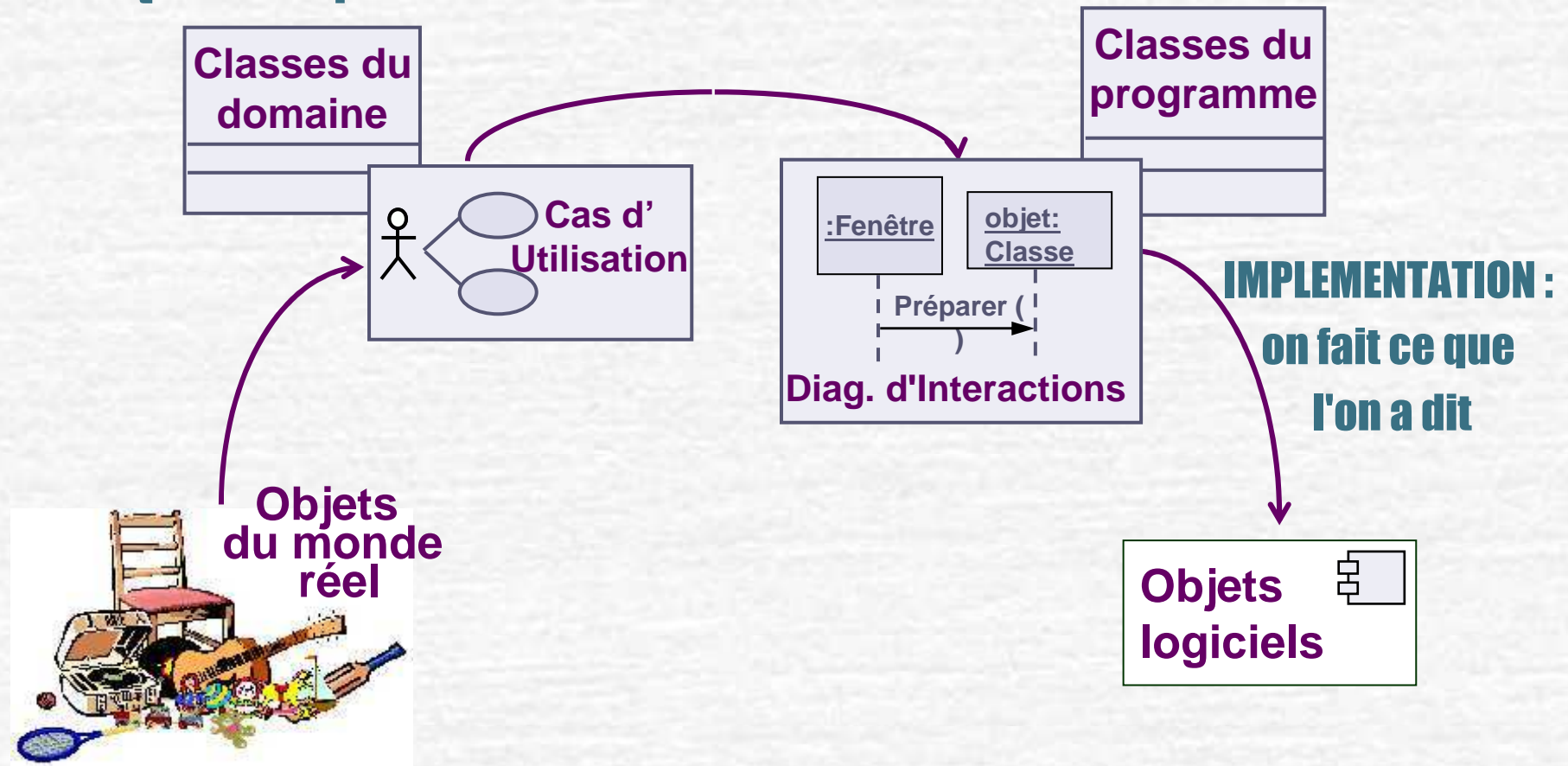




# SYNTHÈSE : QU'EST-CE QU'ON FAIT ?

**ANALYSE :**  
Quel est le problème ?

**CONCEPTION :** Quelle est la solution ?



Mais il y a aussi beaucoup de textes à écrire : définitions, commentaires, explications, glossaires, dictionnaires de données...☹



# LE PROCESSUS UNIFIE

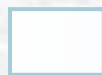
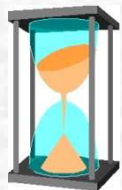
- ☞ **Piloté par les cas d'utilisation “ Use-case driven ”**
  - Un logiciel existe pour servir des utilisateurs, pour le construire on doit connaître ce que ses utilisateurs veulent et ont besoin.
  - Les cas d'utilisation pilotent les développements comme un conducteur pilote une voiture. On peut lâcher les mains quelques secondes, pas plus.
- ☞ **ITÉRATIF ET INCRÉMENTAL**
  - Les itérations sont des étapes dans le travail et les incréments des extensions du projet.
- ☞ **Centré sur l'architecture “ Architecture Centric ”**
  - L'architecture est la forme qu'a le système.
  - l'architecture doit permettre la réalisation des cas d'utilisation en tenant compte de l'environnement (matériel, OS, SGBD, réseau...)
- ☞ **Basé sur les composants “ Component Based ”**
- ☞ **Piloté par la prise en compte systématique et permanente des risques “ Risk driven”.**

# ANALYSE DES RISQUES

## Objectifs

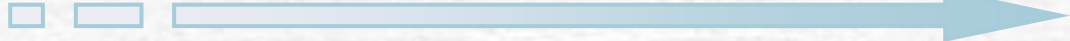
- Anticiper les aléas qui menacent le projet
- Mettre en place les mesures de préventions
  - Pour que l'occurrence n'ait pas lieu
- Prévoir les mesures de secours
  - Pour atténuer les impacts si l'occurrence a lieu
- Evaluer la gravité/occurrence :
  - Gravité moyenne/ occurrence assez forte
  - Gravité élevée / occurrence faible

# LE PROCESSUS UNIFIÉ (RAPPEL)



	Inception (étude d'opportunité)	Élaboration	Construction	Transition
Exigences	★	★★★	★	
Analyse	★	★★★	★	
Conception	★	★★	★★★	★
Implémentation	★	★★	★★★	★★
Test	★	★	★★★	★★





## INCEPTION (OPPORTUNITÉ)

	Inception (étude d'opportunité)	Élaboration	Construction	Transition
Exigences	*	***	*	
Analyse	*	***	*	
Conception	*	**	***	*
Implémentation	*	**	***	**
Test	*	*	***	**

- Quel système allons nous construire ? (10% des UC)
- Est-on capable de le construire ? (faisabilité, délais, coût)
- Est-il opportun de le construire ?
- Qu'est-ce qui pourrait mettre le projet en péril ? (risques)
- Quelles sont les frontières du système ?
- Quelle pourrait être l'architecture ?
- Comment planifier les itérations ?

***QUOI ? avec QUI ? pour QUAND ? pour COMBIEN ?***





# ÉLABORATION

proposer et argumenter des solutions au besoin exprimé

	Inception (étude d'opportunité)	Élaboration	Construction	Transition
Exigences	*	***	*	
Analyse	*	***	*	
Conception	*	**	***	*
Implémentation	*	**	***	**
Test	*	*	***	**

- ☞ Pour chaque nouvel incrément :
- Connaissance du métier des clients (contexte),
  - Majorité des cas d'utilisation (80%),
  - Analyse des risques,
  - Modèle conceptuel,
  - Prototypes,
  - Modèle Physique (architecture, solution retenue pour implémenter le modèle conceptuel et les UC).



## CONSTRUCTION

	Inception (étude d'opportunité)	Élaboration	Construction	Transition
Exigences	*	***	*	
Analyse	*	***	*	
Conception	*	**	***	*
Implémentation	*	**	***	**
Test	*	*	***	**

- Créer les composants : sources, scripts, puis exécutables.
- Fournir une version  $\beta$  de l'incrément en cours de développement en plusieurs itérations :
  - Chaque itération est un mini-projet, ce qui permet de remédier aux risques d'intégration et de tests en “ big bang ”.
  - Chaque itération fournit un produit de qualité, intégré et testé.



## TRANSITION

	Inception (étude d'opportunité)	Élaboration	Construction	Transition
Exigences	*	***	*	
Analyse	*	***	*	
Conception	*	**	***	*
Implémentation	*	**	***	**
Test	*	*	***	**

- Mettre le système entre les mains de la communauté des utilisateurs,
- Améliorer les performances si nécessaire,
- Réparer les défauts (mais pas de développement pour ajouter des fonctionnalités),
- Rédiger la documentation utilisateur,
- Former les utilisateurs et intégrer les retours d'expérience.



# JALONS



## Inception :

- définition des ambitions du projet,
- estimation des coûts et des délais,
- exigences décrites par les cas d'utilisation primaires,
- budget approuvé et fonds disponibles.

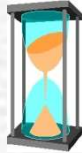
## Elaboration :

- architecture stable,
- résolution des risques majeurs,
- adhésion des décideurs (à l'architecture et la planification)
- évaluation des ressources consommées / prévisions.

## Construction :

- version assez stable pour être confiée aux utilisateurs (rédaction d'un manuel) et accord des décideurs.





## EXIGENCES

	Inception (étude d'opportunité)	Élaboration	Construction	Transition
Exigences	*	***	*	
Analyse	*	***	*	
Conception	*	**	***	*
Implémentation	*	**	***	**
Test	*	*	***	**

- Diagrammes de Cas d'Utilisation (réunions, interviews),
- Documentation détaillée des cas d'utilisation,
- Diagrammes de séquences systèmes (scénarios caractéristiques) ou diagrammes d'activités,
- Exigences non fonctionnelles (performance, ergonomie, portabilité, modifiabilité, fiabilité),
- Prototypes (IHO au minimum).



## ANALYSE

	Inception (étude d'opportunité)	Élaboration	Construction	Transition
Exigences	*	***	*	
Analyse	*	***	*	
Conception	*	**	***	*
Implémentation	*	**	***	**
Test	*	*	***	**

- **Modèle d'Activités Métier** : diagrammes d'activités, décrivant les processus de gestion et/ou industriels (existants, futurs).
- **Modèle de Classes métier**,
- **Dictionnaire** : responsabilité des classes, attributs, opérations,
- **Diagrammes d'états des classes d'analyse**,
- **Validation par les experts du domaine.**



## CONCEPTION

	Inception (étude d'opportunité)	Élaboration	Construction	Transition
Exigences	*	***	*	
Analyse	*	***	*	
Conception	*	**	***	*
Implémentation	*	**	***	**
Test	*	*	***	**

### Comment construire le logiciel ?

- Identifier les classes qui vont faire partie de l'architecture.
- Découpage de l'application en paquets (sous-systèmes).

### Quelle technologie utiliser ?

- Diagrammes de classes techniques : architecture du logiciel à construire (design patterns),
- Diagrammes d'Interactions : communication des classes pour l'implémentation des cas d'utilisation,
- Diagrammes d'États : comportement des objets du logiciel.

# CONCEPTION

- **Activité de raffinement progressif (du général aux détails) :**
  - **Conception de l'architecture : identifier les sous-systèmes**
  - **Spécification abstraite : identifier les services et les contraintes de chaque sous-système.**
  - **Conception de l'interface (de programmation) entre sous-systèmes.**
  - **Conception des composants.**
  - **Conception des SDD.**





## IMPLÉMENTATION

	Inception (étude d'opportunité)	Élaboration	Construction	Transition
Exigences	*	***	*	
Analyse	*	***	*	
Conception	*	**	***	*
Implémentation	*	**	***	**
Test	*	*	***	**

- Diagrammes de Paquets, de Composants montrant les classes fortement dépendantes,
- Diagrammes d'interaction (séquences ou communication)
- Diagrammes États-Transitions : description détaillée des comportements,
- Diagrammes de Déploiement.



# METHODES AGILES

- ☞ Scrum
  - <http://www.scrum.org/>
- ☞ Agile Modelling
  - <http://www.agilemodeling.com/>
- ☞ RAD : Rapid Application Modelling
  - <http://www.rad.fr/>
- ☞ eXtreme Programming
  - <http://xp-france.net/>
- ☞ Principes
  - organisation en équipes
  - retours rapides
  - gestion des changements
  - simplicité de conception
  - Qualité

# SCRUM

- Scrum n'est pas une méthodologie, c'est plutôt un cadre à l'intérieur duquel il est possible d'utiliser différents processus et techniques.
- Scrum adopte une approche itérative et incrémentale dans le but de
  - rendre le processus plus prévisible,
  - contrôler le risque.
- Une itération en Scrum s'appelle un **sprint**.
  - Un sprint dure environ 1 mois.

# Scrum : déroulement d'un sprint

- ☞ Avant : réunion de planification (sprint planning meeting),
  - l'équipe décide de ce qui va être fait
    - « Quoi ? » ➔ carnet de produit (product backlog)
  - l'équipe détermine comment elle va développer les fonctionnalités décidées
    - « Comment ? » ➔ carnet de sprint (sprint backlog)
- ☞ Pendant : la mêlée quotidienne (daily scrum)
  - L'équipe se rencontre chaque jour pour une réunion d'inspection et d'adaptation du processus (environ 15mn),
  - Chaque membre de l'équipe présente ce qui suit :
    - ce qu'il ou elle a accompli depuis la dernière mêlée,
    - ce qu'il ou elle va faire d'ici la prochaine mêlée,
    - les obstacles à surmonter, s'il y a lieu.
  - La mêlée quotidienne améliore la communication et favorise la prise de décision



# Scrum : déroulement d'un sprint

- **Après : revue du sprint (sprint review meeting),**
  - l'équipe Scrum et les parties prenantes discutent de ce qui a été fait pendant le sprint :
    - Ce qui a été complété
    - Ce qui n'a pas été complété
    - Les problèmes rencontrés et leur résolution.
  - elles discutent également de ce qui devra être fait au cours du prochain sprint.
- **Encore après : *rétrospective de sprint*,**
  - inspecter le déroulement du dernier sprint du point de vue des individus, des relations interpersonnelles, des processus et des outils.
    - Succès, améliorations...

# L'équipe Scrum (les rôles)

- ☞ **Le Scrum Master : « expert » Scrum**
  - Guide l'équipe dans les pratiques et règles de Scrum.
- ☞ **Le propriétaire de produit (product owner)**
  - Gère le carnet du produit,
  - Sélectionne et affecte les priorités aux fonctionnalités,
  - S'assure de la valeur du travail de l'équipe.
- ☞ **L'équipe (team) : 7 personnes  $\pm$  2**
  - transforme le contenu du carnet du produit en un sous-ensemble de fonctionnalités livrable à la fin du sprint.
  - Contient des membres aux compétences variées (architecture, développement, contrôle qualité, conception d'IHO, base de données...)
  - L'équipe s'organise elle-même.

# Artefacts de Scrum

- ☛ **Carnet du produit (product backlog) :**
  - liste des fonctionnalités, technologies, améliorations et correctifs qui correspondent aux changements à apporter au produit lors des livraisons futures.
  - Chaque élément possède : une description, un niveau de priorité et une estimation.
- ☛ **Graphique de progression de livraison (Burndown Chart) :**
  - mesure ce qui reste à accomplir dans le carnet au fil du temps.
- ☛ **Carnet de sprint (sprint backlog) :**
  - Contient toutes les tâches nécessaires pour réaliser un sous-ensemble potentiellement livrable du produit.
- ☛ **Graphique de progression de sprint (sprint Burndown Chart) :**
  - mesure l'avancement des tâches à réaliser dans le sprint.



## ACTIVITES DE SOUTIEN

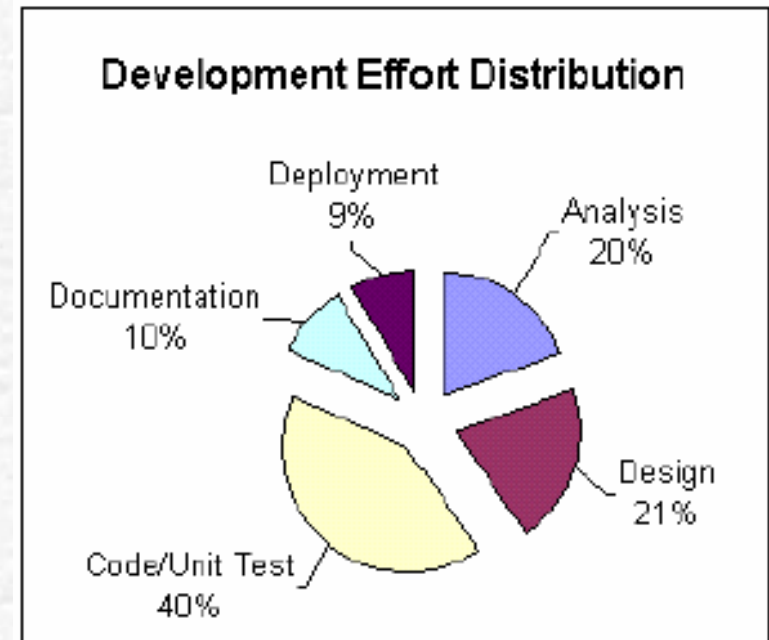
- ☞ **Gestion de projet :**
  - Planification, contrôle de la progression,
  - Gestion des ressources et du budget.
- ☞ **Gestion des configurations et des changements**
  - Trace des éléments du développement (artefacts),
  - Gestion des demandes de changements.
- ☞ **Gestion de l'environnement : adaptation du processus au projet et fourniture d'outils logiciels:**
  - Modélisation graphique et Développement (AGL),
  - Management des exigences,
  - Gestion des versions et des configurations,
  - Aide aux tests et Evaluation de la qualité.





## LE PROCESSUS DE TEST

- ☞ Test **unitaire**,
- ☞ Test **d'intégration** : cohésion,
- ☞ Test de **sous-système** :
  - recherche des erreurs d'interface,
- ☞ Test du **système** : vérification de l'adéquation aux besoins fonctionnels et non fonctionnels,
- ☞ Test **d'acceptation** ("α") : erreurs ou omissions dans la définition des besoins = **VALIDATION**
  - Test dans les conditions définies par le client.



# IMPORTANCE DU TEST

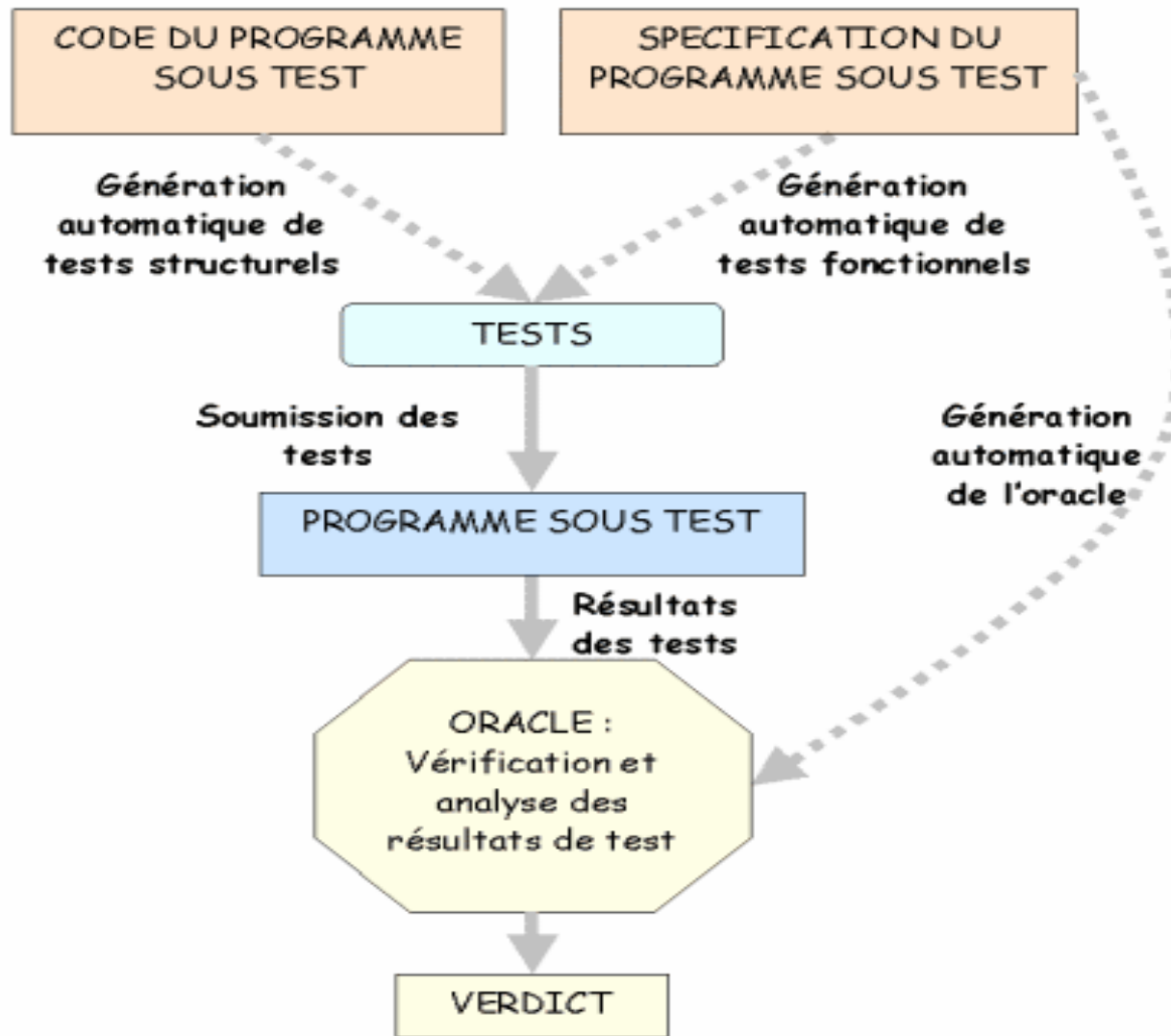


- La qualité du test manuel repose sur la pertinence du testeur.
- Même si le test de logiciel revient cher, l'absence de qualité peut être encore plus coûteuse pour l'entreprise.
- Le test n'ajoute pas de qualité, il permet de connaître l'état du produit aux différents stades de son développement.
- La création d'une équipe de test distincte exige un changement radical de culture : ce n'est pas une pratique courante de faire vérifier son travail par quelqu'un d'autre.

## TYPES DE TESTS

- ☞ Tests “boîte noire” ou **fonctionnels** : scénarios déduits des spécifications fonctionnelles.
  - Test aux **limites**,
  - Tests de non régression,
- ☞ Tests **statistiques** : données tirées aléatoirement dans le domaine des entrées en supposant une répartition statistique.
- ☞ Tests “boîte blanche” ou **structurels** : exploitation systématique des chemins du logiciel.
- ☞ Test de **charge** : performance d’un système sous une charge maximale (vérifier si le système peut supporter les trafics de pointe).
- ☞ Test de **stress** : capacité d’un système à récupérer lorsqu’il est poussé au-delà de ses limites

# PROCESSUS DE TEST





# TESTS FONCTIONNELS

## PRINCIPE

- Comparer les résultats obtenus avec les résultats attendus.
- montrer non seulement qu'un logiciel fait ce qu'on attend de lui, mais aussi qu'il ne fait pas ce qu'on en n'attend pas.

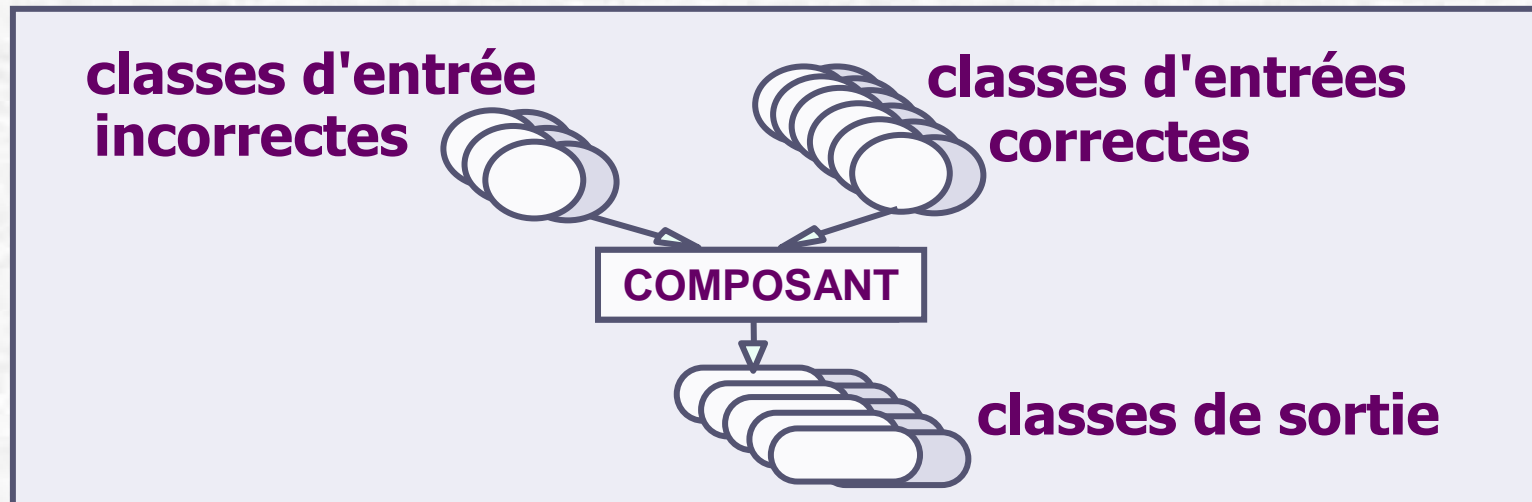
## MÉTHODE

- Déterminer des classes d'équivalence :
  - Valeurs valides non extrêmes,
  - Valeurs valides extrêmes (aux bornes),
  - Valeurs spéciales,
  - valeurs non valides...

## REGLE

- Archiver et Automatiser (tests de non régression).

## CLASSES D'ÉQUIVALENCE



- Une classe = les cas devant être traités identiquement (1 test par classe)
  - Un test peut combiner plusieurs classes valides,
  - Il est nécessaire de réaliser un test individuel par classe invalide.
- On identifie les classes en analysant la spécification.
- Pour chaque condition externe, on établit la liste des classes valides et invalides.

# CLASSES D'ÉQUIVALENCE

**exemple :**

- spécification : x doit être compris entre 0 et 999, et y entre 0 et 4 :

Condition externe	Classes valides	Classes invalides
X	(1) $0 \leq X \leq 999$	(2) $X < 0$ (3) $X > 999$
Y	(4) $0 \leq Y \leq 4$	(5) $Y \leq 0$ (6) $Y > 4$

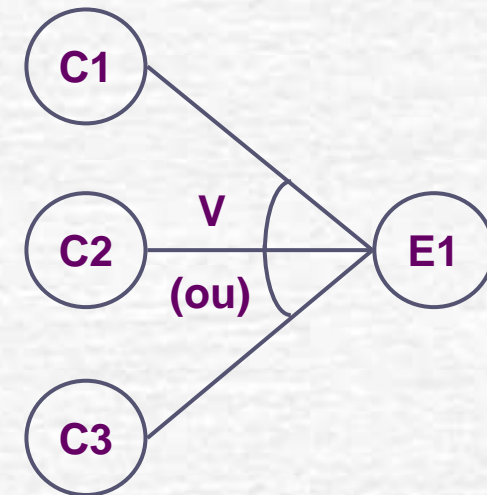
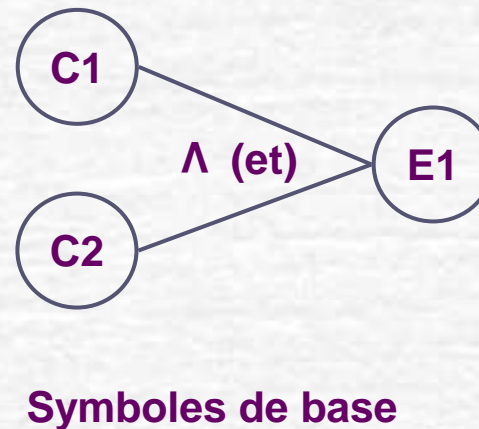
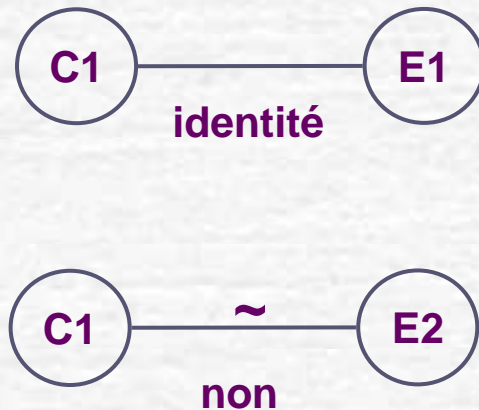
**Jeux de tests valides (1 & 4 : X=10,Y=2)**

**(non compris les tests aux bornes)**

**Invalides (2 : X=-4, Y=3), (3 : X=2000, Y=2),  
(5 : X=10, Y=-10), (6 : X=6, Y=6)**

## GRAPHE CAUSE - EFFET

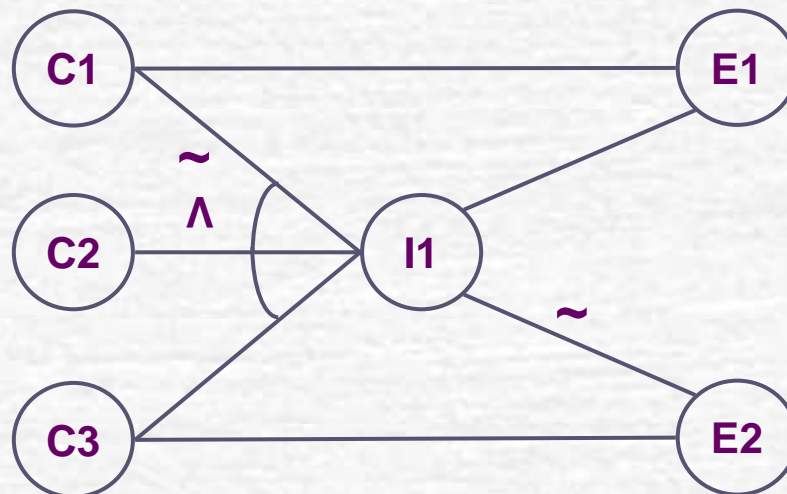
- On identifie des sous-ensembles indépendants de la spécification :
  - Un graphe représentant la totalité de la spécification serait trop complexe,
- Les conditions sur les entrées (ou les classes d'équivalence) sont identifiées et numérotées,
- Les sorties produites sont identifiées et numérotées,





## GRAPHE CAUSE - EFFET

- ☛ Soit  $M$  le montant d'une proposition et  $D$  sa durée :
  - si  $M < M1$  elle est recevable,
  - si  $M1 \leq M < M2$  et  $D < D1$ , elle est recevable.



- ☛ C1 :  $M < M1$
- ☛ C2 :  $M < M2$
- ☛ C3 :  $D < D1$
- ☛ E1 : proposition recevable
- ☛ E2 proposition non recevable

## GRAPHE CAUSE - EFFET

### Table de Décision

C1	C2	C3	E1	E2
0	0	0		X
0	0	1		X
0	1	0		X
0	1	1	X	
1	0	0	X	
1	0	1	X	
1	1	0	X	
1	1	1	X	

Pour extraire des cas de tests, il suffit de choisir pour chaque ligne contenant un X, des données correspondant aux combinaisons des causes associées.

## TESTS STRUCTURELS

- ☞ Ne permettent de tester que ce qui figure dans le programme, mais pas de trouver les oublis par rapport à la spécification.
- ☞ Tests unitaires : mesurer une couverture de test sur
  - Blocs d'Instructions (IB),
  - Chemins de Décision à Décision (CDD ou DDP ou branches) = transferts de contrôle qui résultent d'une décision,
  - Portions Linéaires de Code suivie d'un Saut (PLCS ou LCSAJ)
    - Saut = transfert de contrôle qui donne lieu à une rupture de séquence au cours de la lecture d'un source.
    - PLCS = suite de nœuds du graphe de contrôle commençant au point d'entrée ou à la cible d'un saut, finissant à la sortie ou à la cible d'un saut et ne comportant aucun saut.

## PLCS : exemple

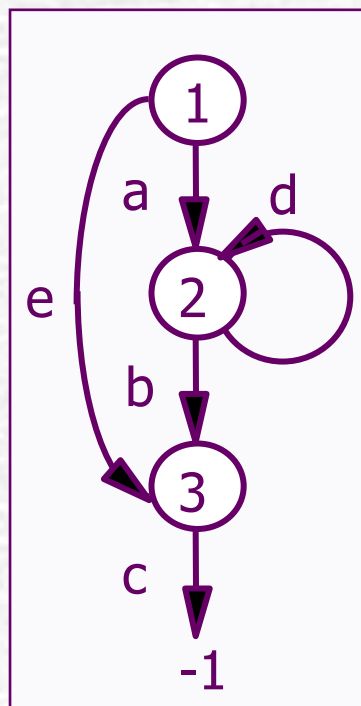
lire N

pour i de 1 à N

...

fait

fin



### Dénombrement :

- sautes : arcs e, d, c

- cibles d'un saut :  
(2) (3) (-1)

- PLCS :

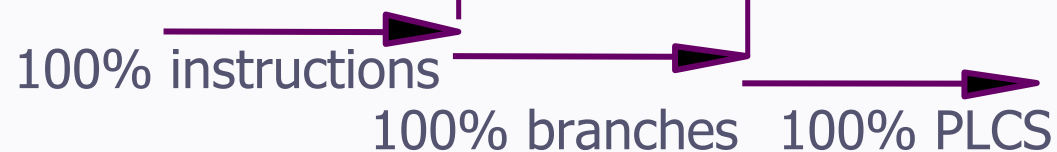
- p1 : (1) -> (3) e
- p2 : (1,2,3) -> (-1) c
- p3 : (2) -> (2) d
- p4 : (1,2) -> (2) d
- p5 : (3) -> (-1) c
- p6 : (2,3) -> (-1) c



## PLCS : exemple

	Jeux d'essai	Chem. d'exécution nœuds	Chem. d'exécution branches	Chem. d'exécution PLCS
D1	n=3	1,2,2,2,3,-1	a,d,d,d,b,c	p4,p3,p6
D2	$n \leq 0$	1,3,-1	e,c	p1,p6
D3	n=1	1,2,3,-1	a,b,c	p2

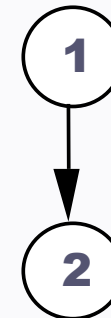
PLCS \ JE	D1	D2	D3
p1		X	
p2			X
p3	X		
p4	X		
p5		X	
p6	X		


 100% instructions      100% branches      100% PLCS

## LIMITES DU TEST STRUCTUREL

A	B	S
pair	pair	0
pair	impair	1
impair	pair	2
impair	impair	3

```
lire (A,B)
x := A mod 2;
y := B mod 2;
S := 2 * x + y;
```



- ❖ Incapacité de démontrer que certaines parties de code sont manquantes,
- ❖ Impossibilité de détecter certaines erreurs sur les données,
- ❖ mais indispensables dans les domaines critiques.

# TEST DU TEST

- ☞ **Le semage de défauts (fault seeding : capacité des tests à trouver les fautes)**
  - On « sème » des défauts et on effectue les tests,
  - On compte le nombre de défauts découverts,
  - On estime le nombre de défauts restants.
- ☞ **La concurrence :**
  - 2 équipes différentes effectuent des tests,
  - On compte les défauts non communs découverts,
- ☞ **Les mutants (variante du semage de défauts):**
  - On introduit des défauts pour créer des mutants,
  - Les mutants qui ont un comportement différent sont « tués par le tests ».



# TEST DES LOGICIELS OBJETS

- Premier problème, trouver une unité indépendante de test :
  - Dans les systèmes structurés, chaque procédure ou fonction peut être testée indépendamment.
  - En objet, une méthode n'existe que par rapport à la classe à laquelle elle est attachée.
  - On ne peut pas toujours tester une méthode sans l'appliquer à un objet.
  - Chaque objet possède un état, le contexte dans lequel une méthode est exécutée, est défini par ses paramètres et par l'objet auquel elle est appliquée.



## TESTS DE CLASSES

- ☞ On teste des séquences de méthodes appliquées à un objet.
- ☞ Graphe d'héritage :
  - Tester une classe générale, puis les classes qui la spécialisent, jusqu'aux classes feuilles.
    - Mais attention : ce n'est pas parce qu'une méthode fonctionne bien dans une classe donnée, qu'elle fonctionnera aussi bien une fois héritée.
    - Il est difficile de savoir, parmi les tests sélectionnés pour la classe parente, ceux qui peuvent être éliminés et ceux qui doivent être rejoués.
- ☞ Test des paires de fonctions : tous les enchaînements possibles de 2 méthodes.

# TESTS D'INTÉGRATION

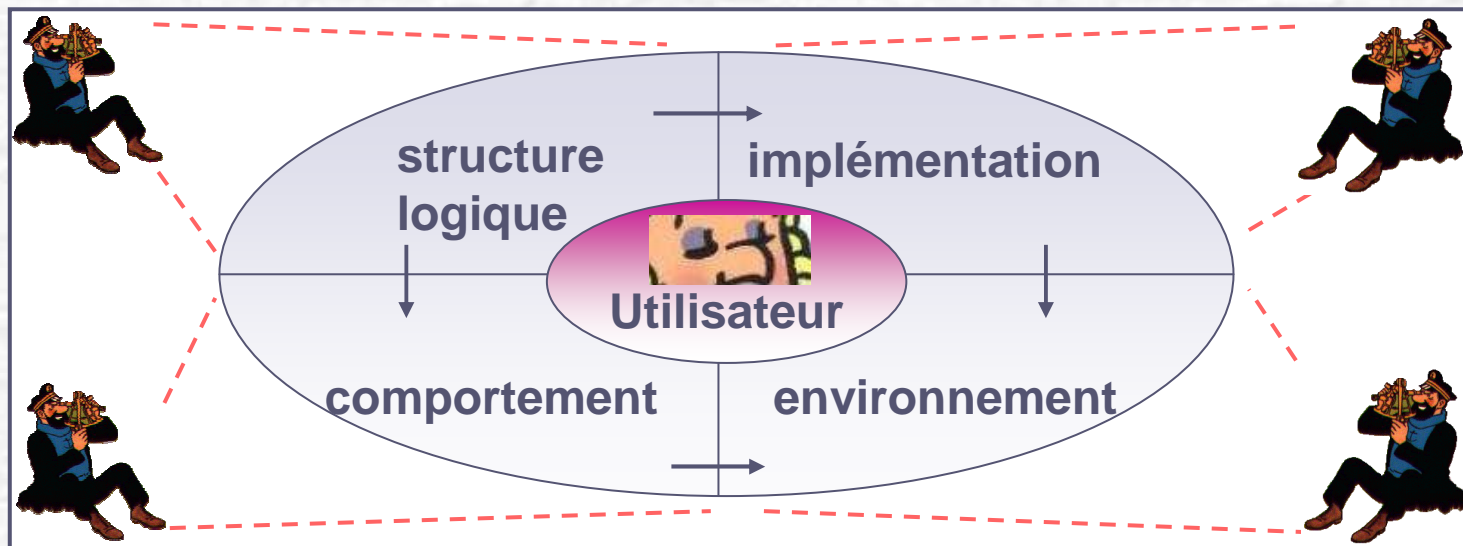
- Le test d'intégration consiste à vérifier que les classes clientes utilisent les classes serveurs en conformité avec l'interface offerte par la classe serveur.
- Il faut parfois simuler le comportement d'une classe pour pouvoir en tester une autre.
- On explore les diagrammes de dépendances,
- On exploite les diagrammes d'états :
  - Tester chaque transition d'état, et chaque méthode de la classe dans chacun des états où elle est sollicitée.

## CONCLUSION SUR LE TEST

- Les méthodes agiles préconisent une utilisation importante du test unitaire.
- Une famille de frameworks a été développée pour le test unitaire de classes : Junit pour Java, Nunit pour la plateforme .NET...
- On utilise les cas d'utilisation et les diagrammes d'interaction associés pour ordonner et dériver les cas de test.

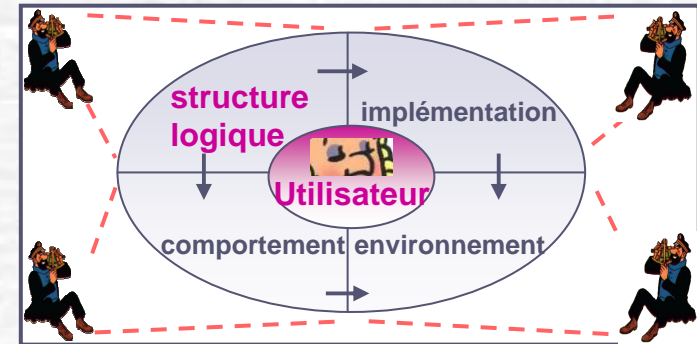
## DIAGRAMMES UML ET VUES

- Le modèle UML d'un système peut être étudié sous différentes perspectives (vues).
- Modèle UML = ensemble de diagrammes décrivant le système développé.
- Vue = angle particulier sous lequel un participant voit le système ou combinaison de diagrammes intéressant un participant.





# LES VUES D' UML



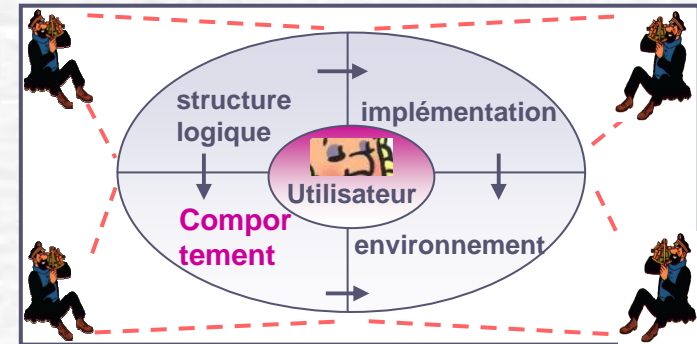
## Vue utilisateur :

- Définit les buts et objectifs des clients du système (services).
- Définit les besoins et contraintes de la solution
- Vue unificatrice des autres vues : elle sert de référence à leur validation.

## Vue structure logique

- Décrit les aspects statiques, représentant la structure du problème.
- Identification des éléments du domaine (classes, attributs, paquets, etc.) et de leurs relations.

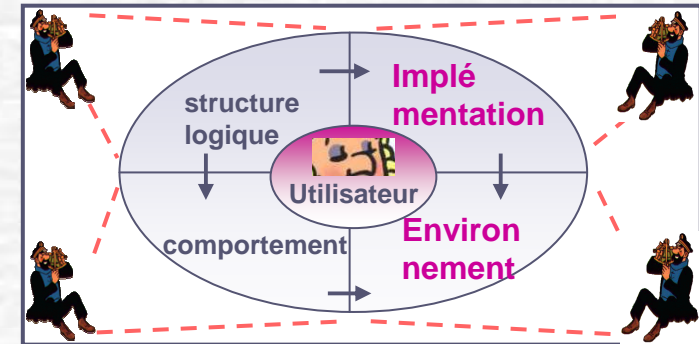
# LES VUES D' UML



## Vue comportement

- Décrit les aspects dynamiques, du comportement du problème et de sa solution.
- Spécifie les interactions et collaborations entre éléments de la solution.
- Montre la décomposition du système en termes de processus, d'interactions entre processus, de synchronisation et de communication entre activités.

# LES VUES D' UML



## Vue implémentation :

- aspects structure et comportement de la solution.
- réalisation, organisation en composants, contraintes de développement, etc.

## Vue environnement :

- aspects de structure et de comportement du contexte dans lequel la solution est réalisée.
- ressources matérielles (disposition, nature, performance, etc.) et leur utilisation par le logiciel.

## IL N'Y A PAS DE MIRACLE

- ❏ On n'a jamais toutes les informations du premier coup (les itérations dans une phase sont nécessaires).
- ❏ Les choses changent au cours du développement (surtout les exigences).
- ❏ On n'éliminera pas tous les risques comme on l'espérait et de nouveaux apparaissent.
- ❏ Il faudra détruire des lignes de codes écrites pendant l'itération précédente.



## IL NE FAUT PAS :



Penser que Inception = Spécification, Élaboration = Conception, Construction = Codage.



Penser que Élaboration = définir rigoureusement des modèles traduits en code pendant la Construction.



Essayer de définir la plupart des besoins avant d'entamer la conception ou l'implémentation, etc.



Croire que la durée normale d'une itération est de 4 mois et non de 4 semaines.



Penser que l'adoption d'UML implique beaucoup d'activités et beaucoup de documents.



Essayer de planifier un projet en détail du début à la fin et de prédire toutes les itérations.



## IL FAUT ALLER VOIR :

### En français :

- [uml.free.fr](http://uml.free.fr)
- [www.iro.umontreal.ca/~dift6803/](http://www.iro.umontreal.ca/~dift6803/) (voir les transparents)
- [dept-info.labri.fr/~aimar/Enseignement/UML/cours.pdf](http://dept-info.labri.fr/~aimar/Enseignement/UML/cours.pdf)
- <http://www.abrillant.com/doc/uml/>

### En anglais :

- [www.rational.com/uml](http://www.rational.com/uml)
- [www.omg.org/uml/](http://www.omg.org/uml/)
- [www.cetus-links.org/](http://www.cetus-links.org/) (lien UML)
- [www.sdmagazine.com/uml/](http://www.sdmagazine.com/uml/)
- [www.agilemodeling.com/essays/umlDiagrams.htm](http://www.agilemodeling.com/essays/umlDiagrams.htm)
- [bdn.borland.com/together/modeling/uml/](http://bdn.borland.com/together/modeling/uml/)
- <http://hillside.net/patterns/>



## BIBLIOGRAPHIE

- ☛ **Modélisation objet avec UML (2de édition) - Pierre-Alain Muller – Eyrolles 2000**
- ☛ **UML2 distilled: brief guide to the standard object modelling language (3<sup>ème</sup> édition) - Fowler - Addison-Wesley 2003.**
- ☛ **UML Guide de l'utilisateur - G. Booch, J. Rumbaugh, I. Jacobson - Eyrolles 2000.**
- ☛ **Le Processus Unifié de Développement Logiciel - I. Jacobson – Eyrolles 2000.**
- ☛ **Introduction au Rational Unified Process - Philippe Kruchten - Eyrolles 2000.**
- ☛ **UML2 et les Design Patterns - Craig Larman - Campus Press 2005**
- ☛ **UML2 en action - Pascal Roques, Frank Vallée - Eyrolles 2004**
- ☛ **UML2 par la pratique - Pascal Roques - Eyrolles 2004**
- ☛ **UML2 B. Charroux, A. Osmani et Y. Thierry-Mieg - Pearson Education 2005**
- ☛ **Tête la première : Design Patterns, E. Freeman – O'Reilly 2005**

