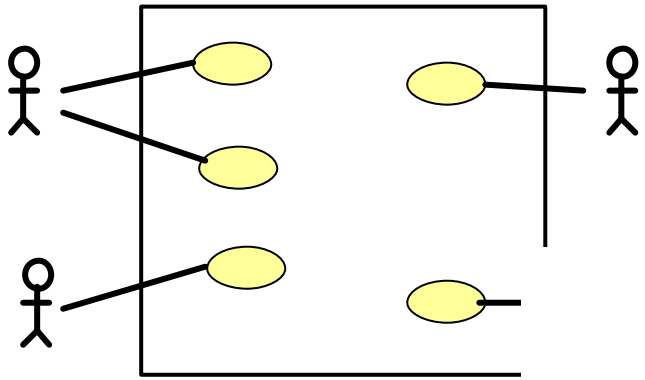
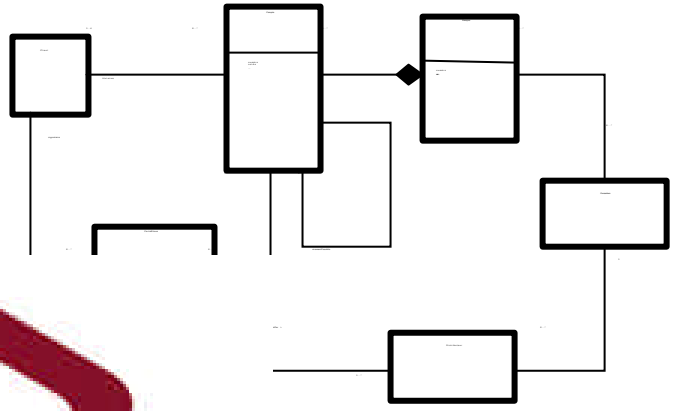
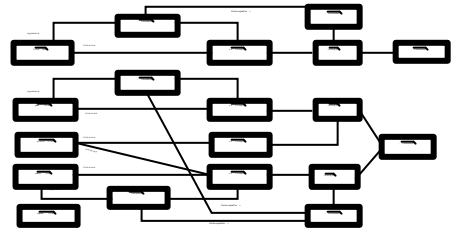
Le langage UML : une vision globale

Objectifs, Historique
Diagrammes, Modèles, Paquetages

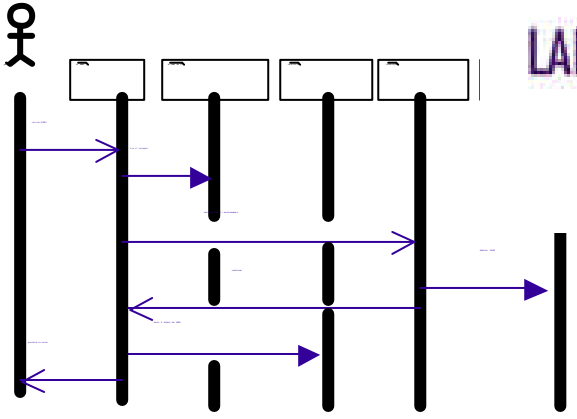
Jean-Marie Favre, Ioannis Parissis ©

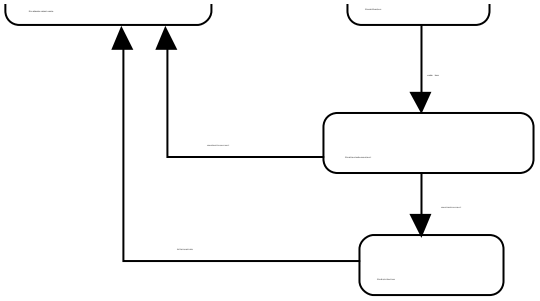


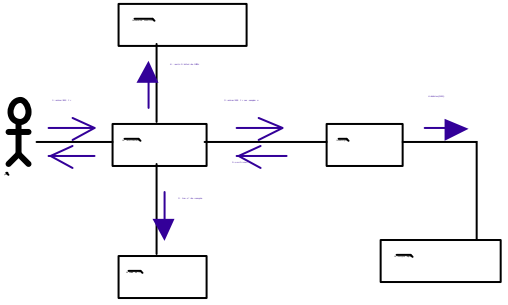




UNIFIED MODELING LANGUAGE™







UML = Unified Modeling Language

- UML = « Unified Modeling **Language** »
- Analyse et conception orientée-objet

- ==> Une notation <==

- Des méthodes
  - " The Rational Unified Process"
  - "The Unified Software Development Process",
  - ...

- Des outils
  - Rational Rose, Objecteering, Together J, ArgoUML,...

Historique

- 80-90 : apparition de nombreuses méthodes
- 90-95 : + de 50 méthodes orientées objets…
- 95 : première version UML (0.8)
- 97 : standardisation OMG
  - (Dec, Rational, Microsoft, HP, Oracle…)
- 99 : proposition UML 1.3
- Septembre 2001 : UML 1.4
- Mai 2003 : UML 1.5
- …

- Version majeure en cours de finalisation : UML 2.0

Standardisation via concensus

- Stabilisation de pratiques industrielles éprouvées

- Différents groupes de pressions et lobbys industriels
- Intérets multiples et variés, enjeux très importants
- Vendeurs d'outils, de méthodes, de savoir-faire
- Utilisateurs individuels, industriels, institutionnels, …

- UML ne vise pas l'innovation mais la consensualité

Deux vision du consensus

- A minima
  - Intersection
  - Mène à la simplicité (ou à l'appauvrissement)
  - Préférable pour les utilisateurs mais pas forcément pour les fournisseurs
  - Suppose une grande maturité
- A maxima
  - Union
  - Mène à la complexité et l'instabilité (ou à la complémentarité)
  - Difficultés pour les utilisateurs, différentiateur pour les fournisseurs
  - Solution de facilité en l'absence de maturité
- Le consensus à maxima est souvent dominant dans l'histoire d'UML

Innovation vs. Industrialisation

"In short:
the time for experimentation is past;
the time for stability and use is now."

Grady Booch
Chief Scientist
Rational Software Corporation

- Evolution continue et incrémentale
- UML n'est pas un projet de recherche
- Nombreuses critiques de la part des chercheurs
- … mais standard de facto dans l'industrie (et dans la recherche…)

Impact d'UML sur l'industrie ?

- Standard de facto lorsqu'un méthode est utilisée

- Conversion des vendeurs d'outils de génie logiciel
- Intégration forte dans le processus de certaines entreprises
- Projets de production ou projets pilotes
- Offres d'emplois
- Impact majeur sur l'industrie dans le futur

Impact sur la recherche ?

- Très critiqué …
  … mais aussi très utilisé
- Notation dominante dans les articles de recherche
- UML est là, il faut chercher comment en profiter
- Point d'accroche avec le monde industriel
- Peut faciliter la transition de techniques innovantes

Impact sur la formation et l'enseignement ?

- Enseignement dans la grande majorité des formations universitaires
  - De bac+2 à bac+ 5
  - Standard international
- Utilisation dans de nombreux cours transversaux
  - Base de données
  - Programmation objets
  - Modélisation
  - Systèmes d'exploitation
  - …
- Nombreuses formations dans l'industrie

- L'un des points les plus sensibles pour l'avenir d'UML

Livres

Livres

En anglais

Livres

En anglais

- 1997: 9+

Livres

En anglais

- 1997: 9+
- 1998: 16+

Livres

En anglais

- 1997: 9+
- 1998: 16+
- 1999: 23+

Livres

En anglais

- 1997: 9+
- 1998: 16+
- 1999: 23+
- 2000: 18+

Livres

En anglais

- 1997: 9+
- 1998: 16+
- 1999: 23+
- 2000: 18+
- 2001: 27+

Livres

En anglais

- 1997: 9+
- 1998: 16+
- 1999: 23+
- 2000: 18+
- 2001: 27+
- 2002: 26+

Livres

En anglais

- 1997: 9+
- 1998: 16+
- 1999: 23+
- 2000: 18+
- 2001: 27+
- 2002: 26+
- Jan-Avril 2003 : 19

Livres

En anglais

- 1997: 9+
- 1998: 16+
- 1999: 23+
- 2000: 18+
- 2001: 27+
- 2002: 26+
- Jan-Avril 2003 : 19
- Prévu 2004 : 6

Livres

En anglais

- 1997: 9+
- 1998: 16+
- 1999: 23+
- 2000: 18+
- 2001: 27+
- 2002: 26+
- Jan-Avril 2003 : 19
- Prévu 2004 : 6

En francais

- 1999: 3+
- 2000: 7+
- 2001: 5+
- 2002: 6+
- Jan-Avril 2003: 3
- …

Livres

En anglais

- 1997: 9+
- 1998: 16+
- 1999: 23+
- 2000: 18+
- 2001: 27+
- 2002: 26+
- Jan-Avril 2003 : 19
- Prévu 2004 : 6

En francais

- 1999: 3+
- 2000: 7+
- 2001: 5+
- 2002: 6+
- Jan-Avril 2003: 3
- …

Plus de 170 livres !

Exemples de titres …

- Developing Web Systems with UML
- Building Web Applications with UML
- Uml for Database Design
- Database Design for Smarties: Using UML for Data Modeling
- Using UML : Software Engineering With Objects and Components
- UML Components: A Simple Process for Specifying Component-Based Software
- Mobile UML: Developing Wireless and Voice Applications
- Real-Time UML: Developing Efficient Objects for Embedded Systems
- Designing Concurrent, Distributed, and Real-Time Applications with UML
- Doing Hard Time: Developing Real-Time Systems with UML
- Process Quality Assurance for UML-Based Projects
- Object-Oriented Project Management with UML
- Software Design: With Patterns, UML, and Components
- Developing Enterprise Java Applications with J2EE and UML
- Convergent Architecture: Building Model-Driven J2Ee Systems With UML
- Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML, 2nd Edition, Volume 1
- Large-Scale Software Architecture : A Practical Guide using UML
- Service- and Component-based Development: Using the Select Perspective and UML

Outils UML ?

- Plus de 77 organisations proposant des outils supportant UML en 2003

- Outils de modélisation
- Génération de code, de documentation, validation, test, …
- Gestion de données
- Transformation de modèles
- …

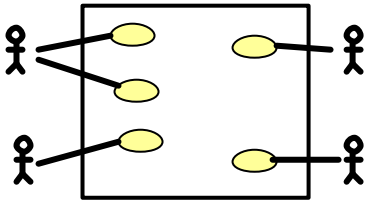
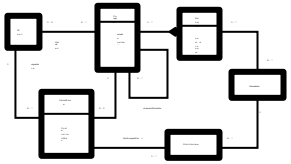
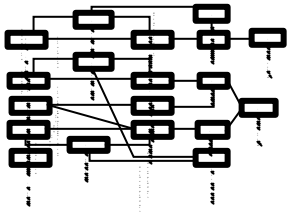
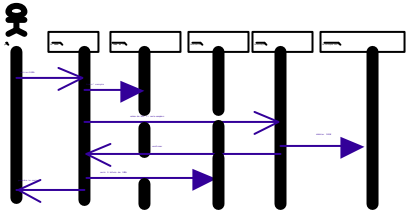
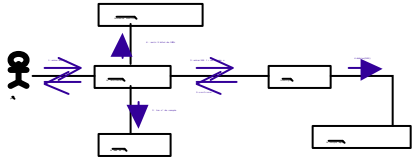
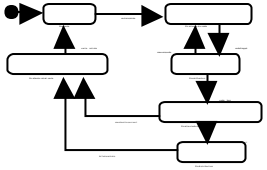
La notation UML ?

- LeS notationS UML : plusieurs notations

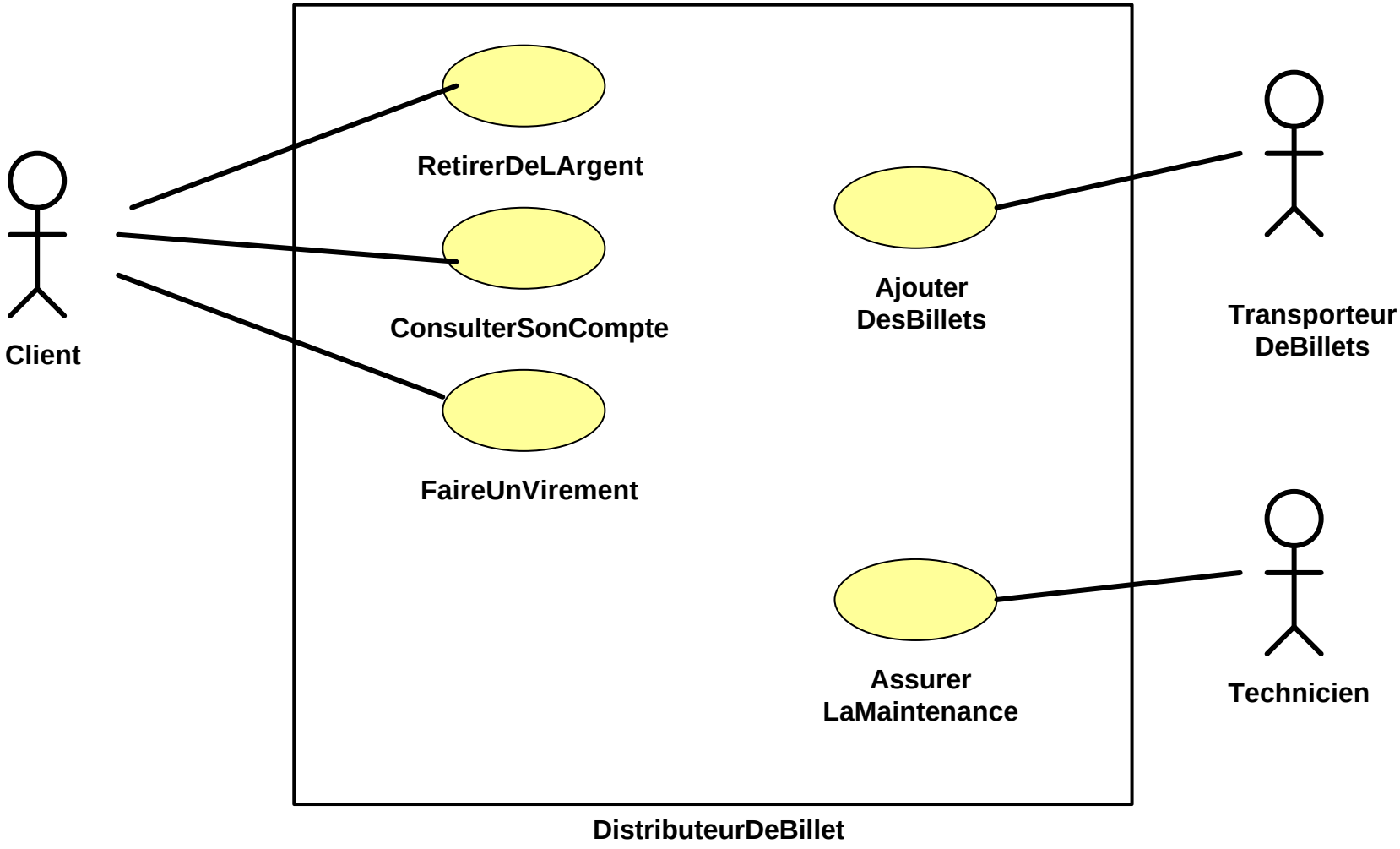
- Notations graphiques
- Signification précise
- Notation standard
- Notation très générale
- Notation extensible

LeS notationS UML

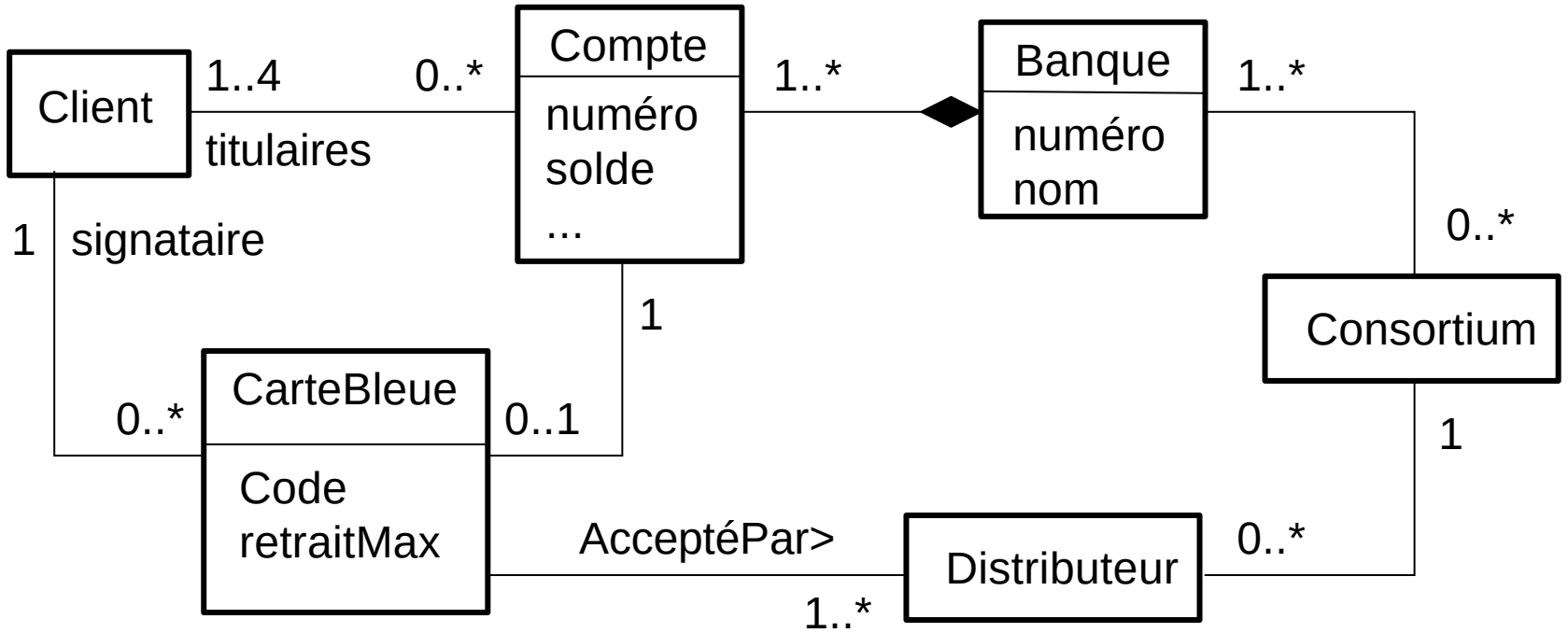
- Diagrammes des cas d 'utilisation
- Diagrammes de classes
- Diagrammes d'objets
- Diagrammes de séquences
- Diagrammes de collaboration
- Diagrammes d'états
- Diagrammes d'activités
- Diagrammes de composants
- Diagrammes de déploiement
- Langage de contraintes (OCL)
- Langage d'actions
- ...

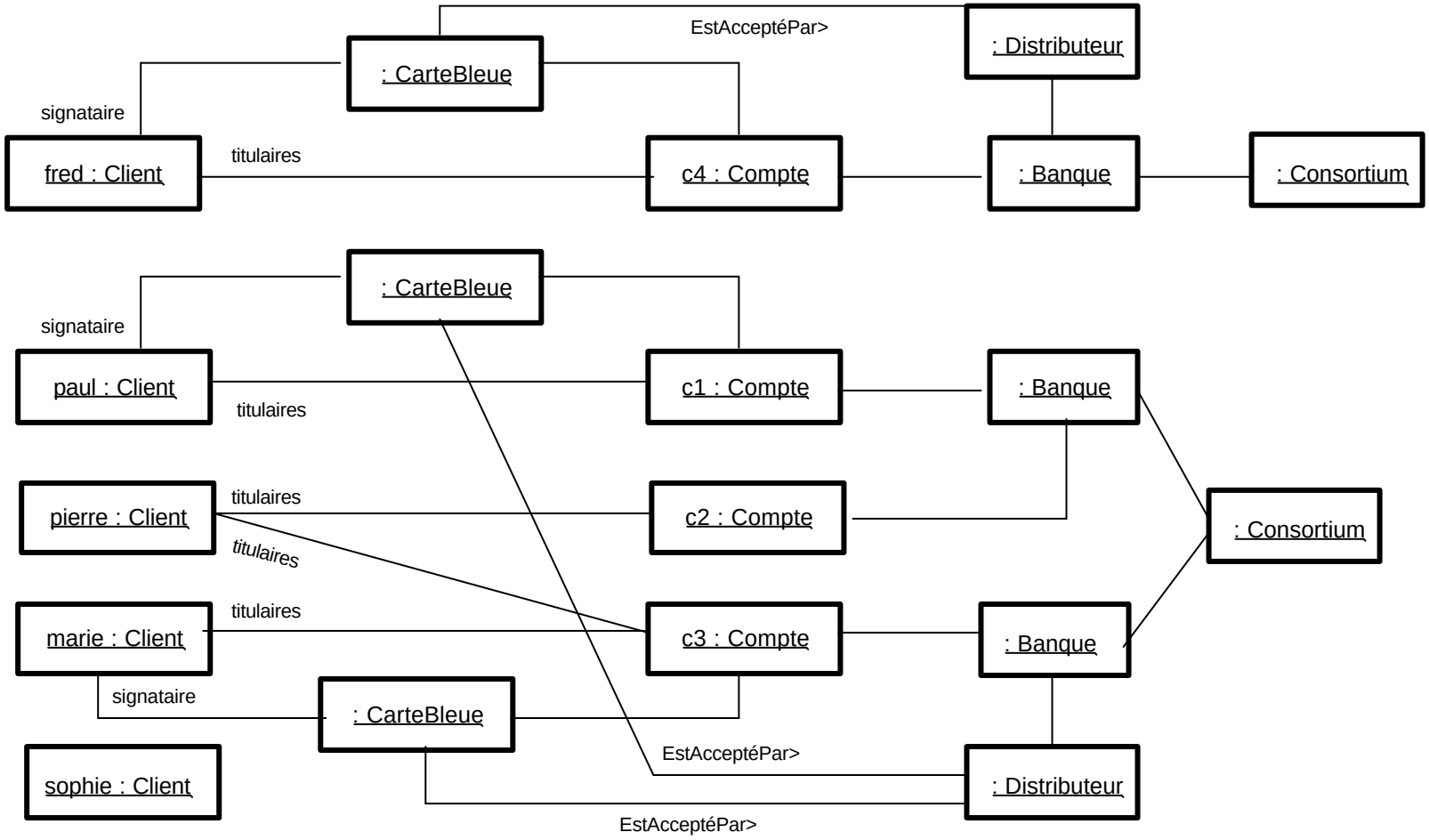
Diagrammes des cas d 'utilisation



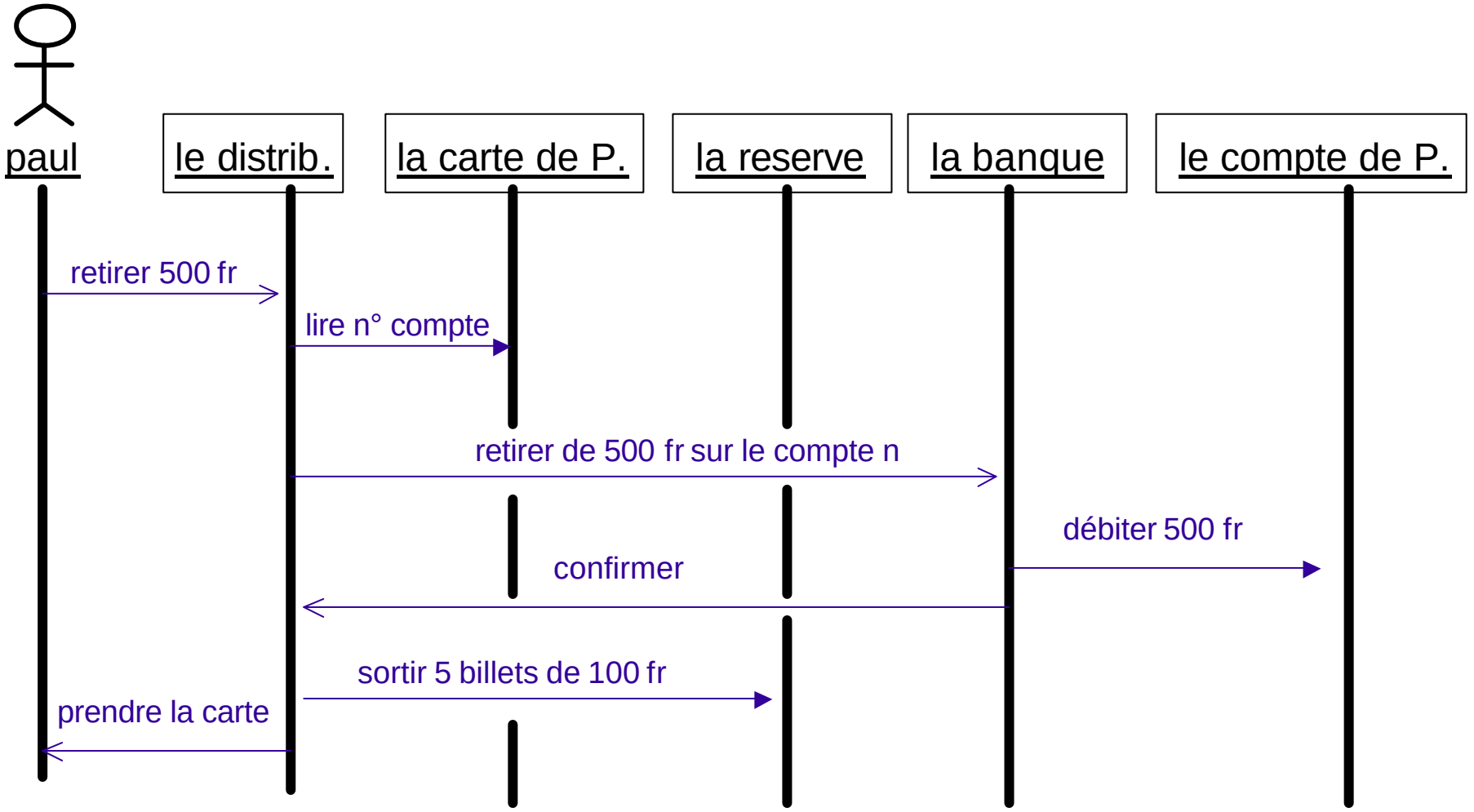
Diagrammes de classes



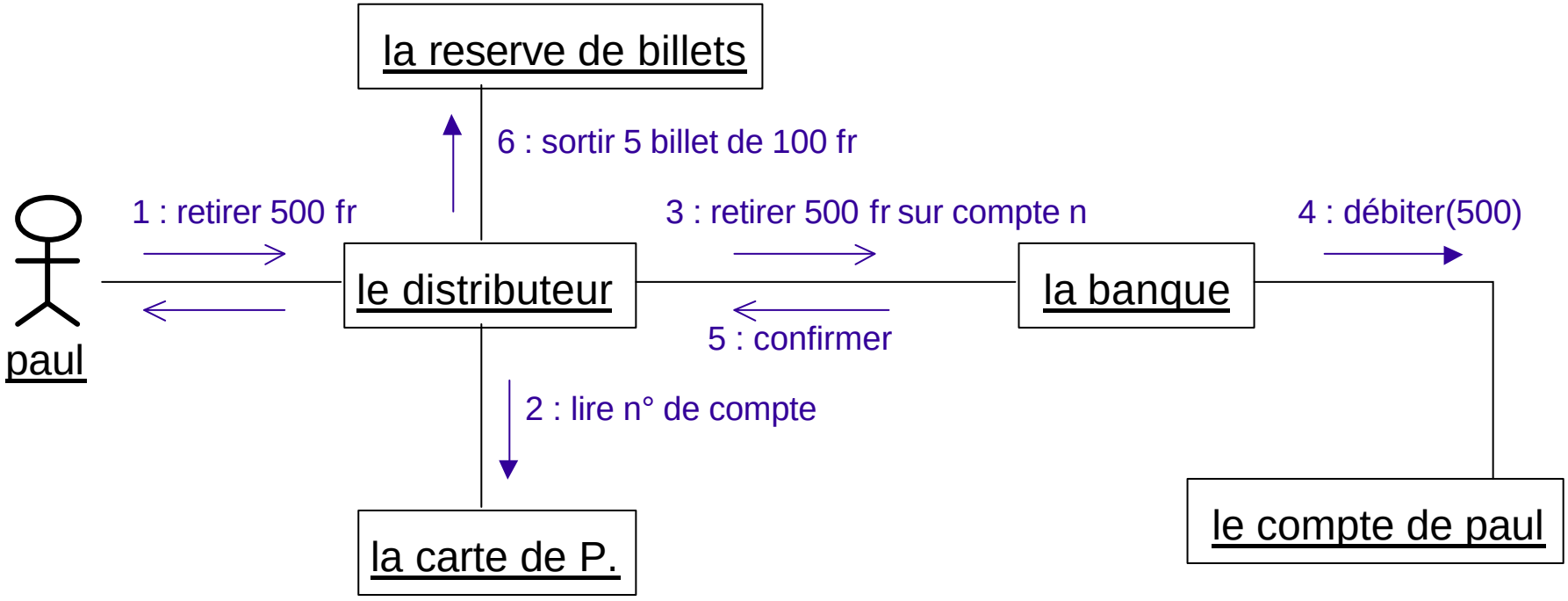
Diagrammes d 'objets



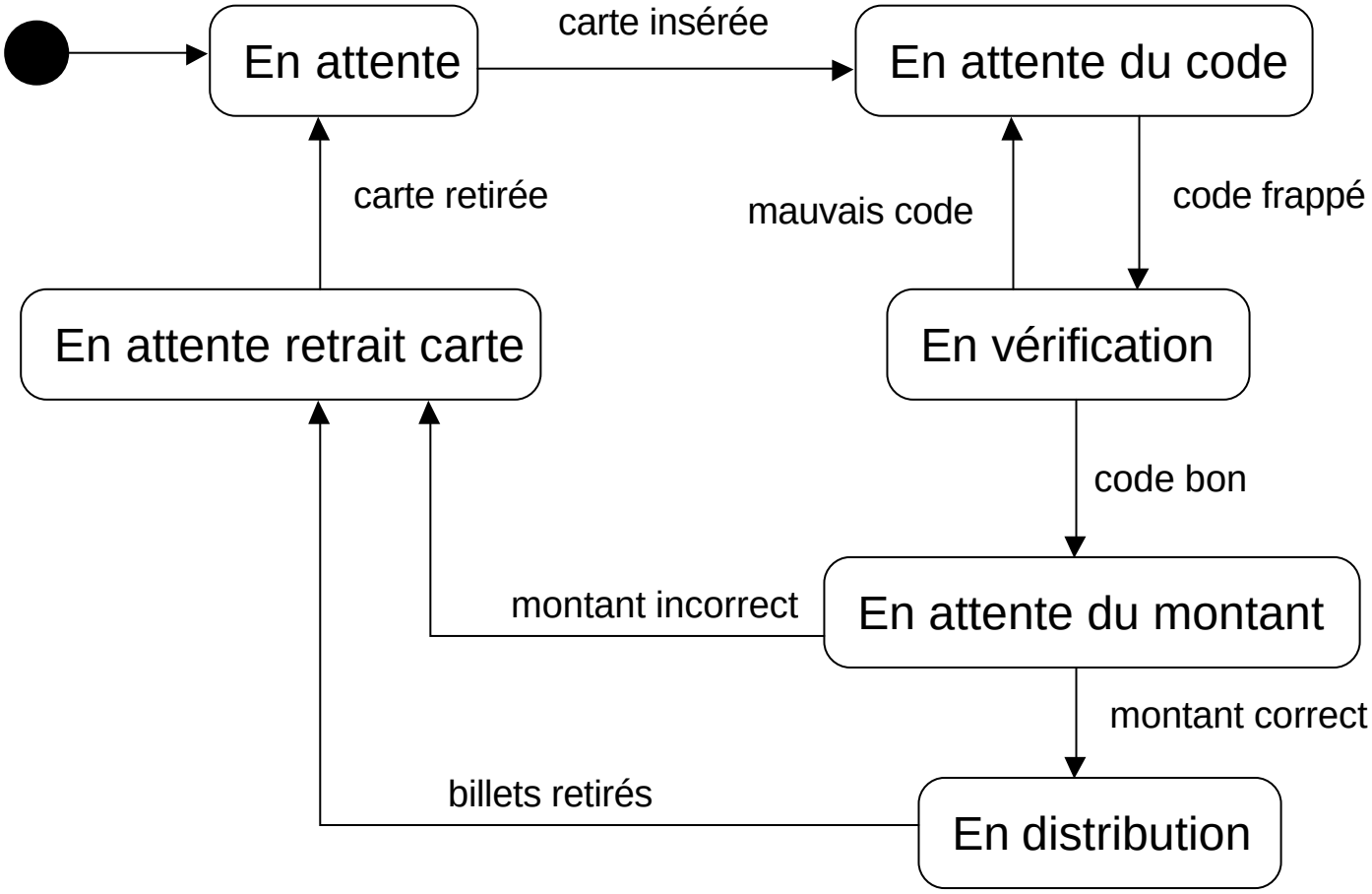
Diagrammes de séquence



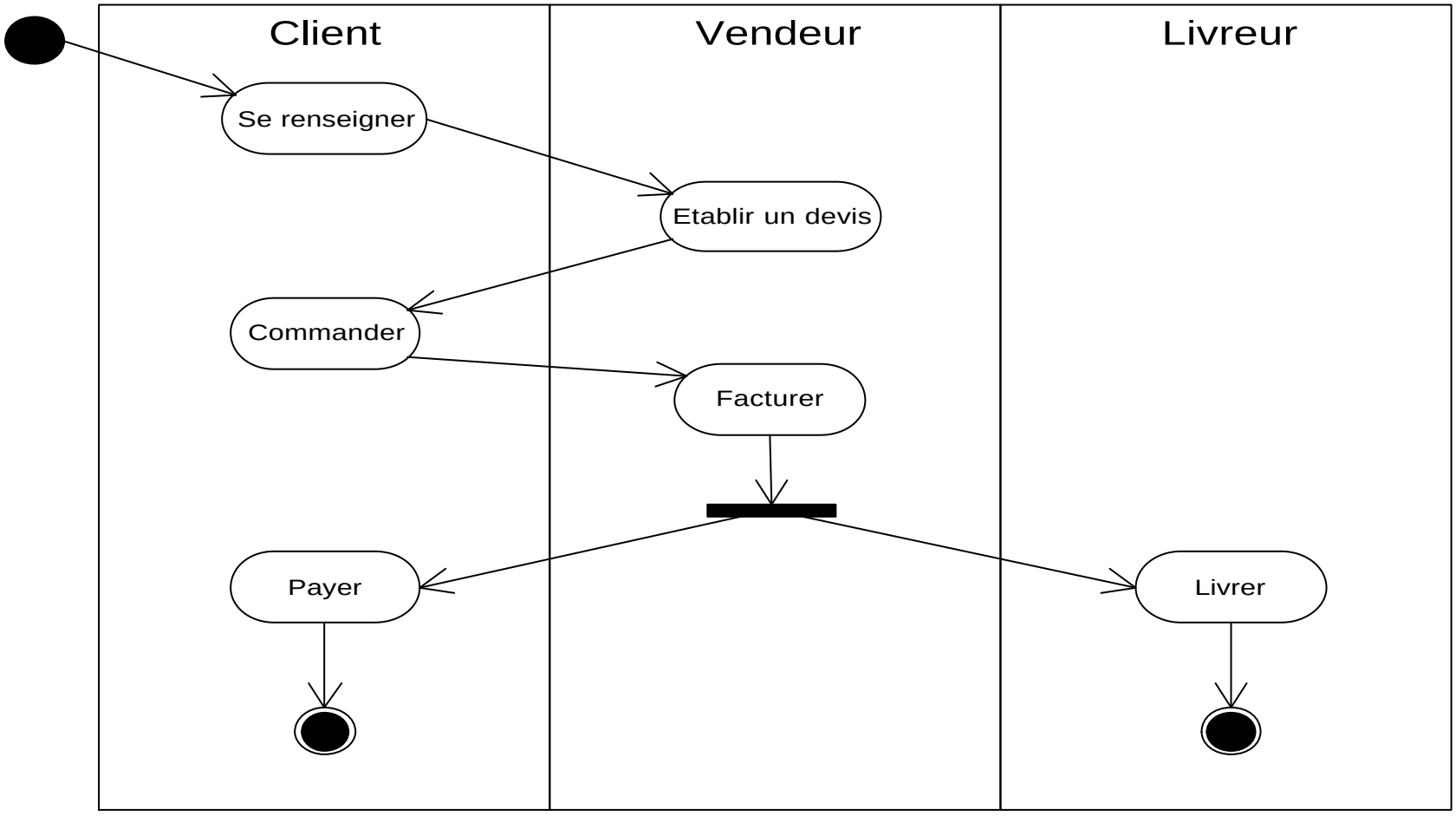
Diagrammes de collaboration



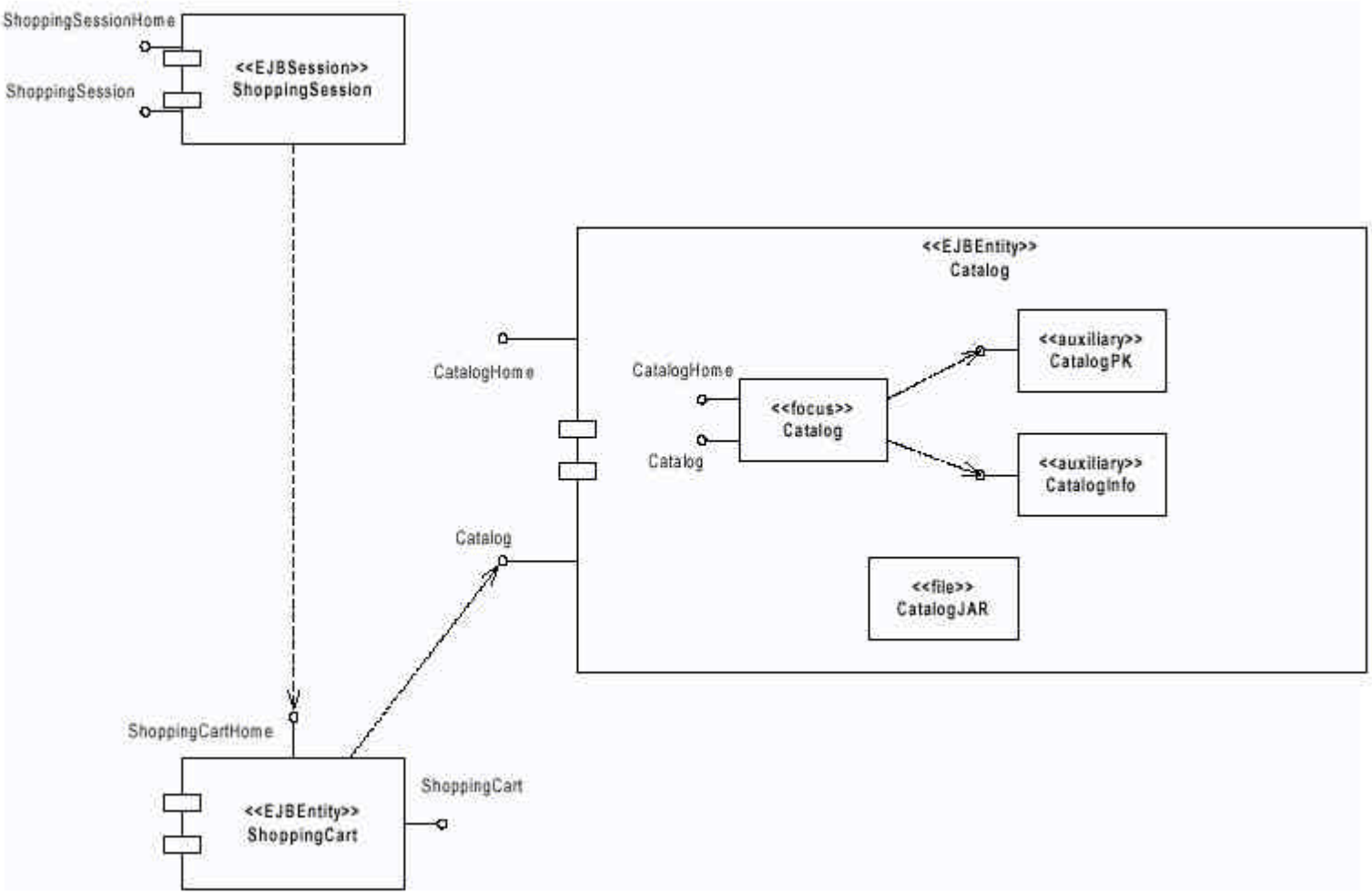
Diagrammes d 'états



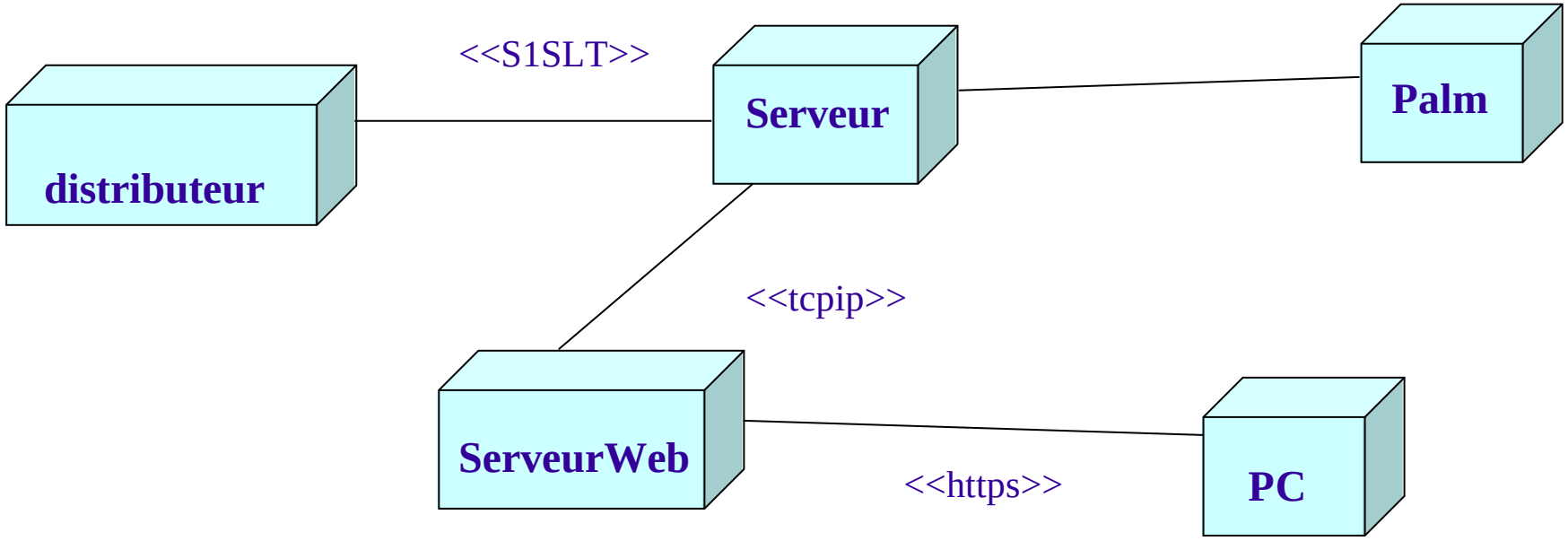
Diagrammes d'activités



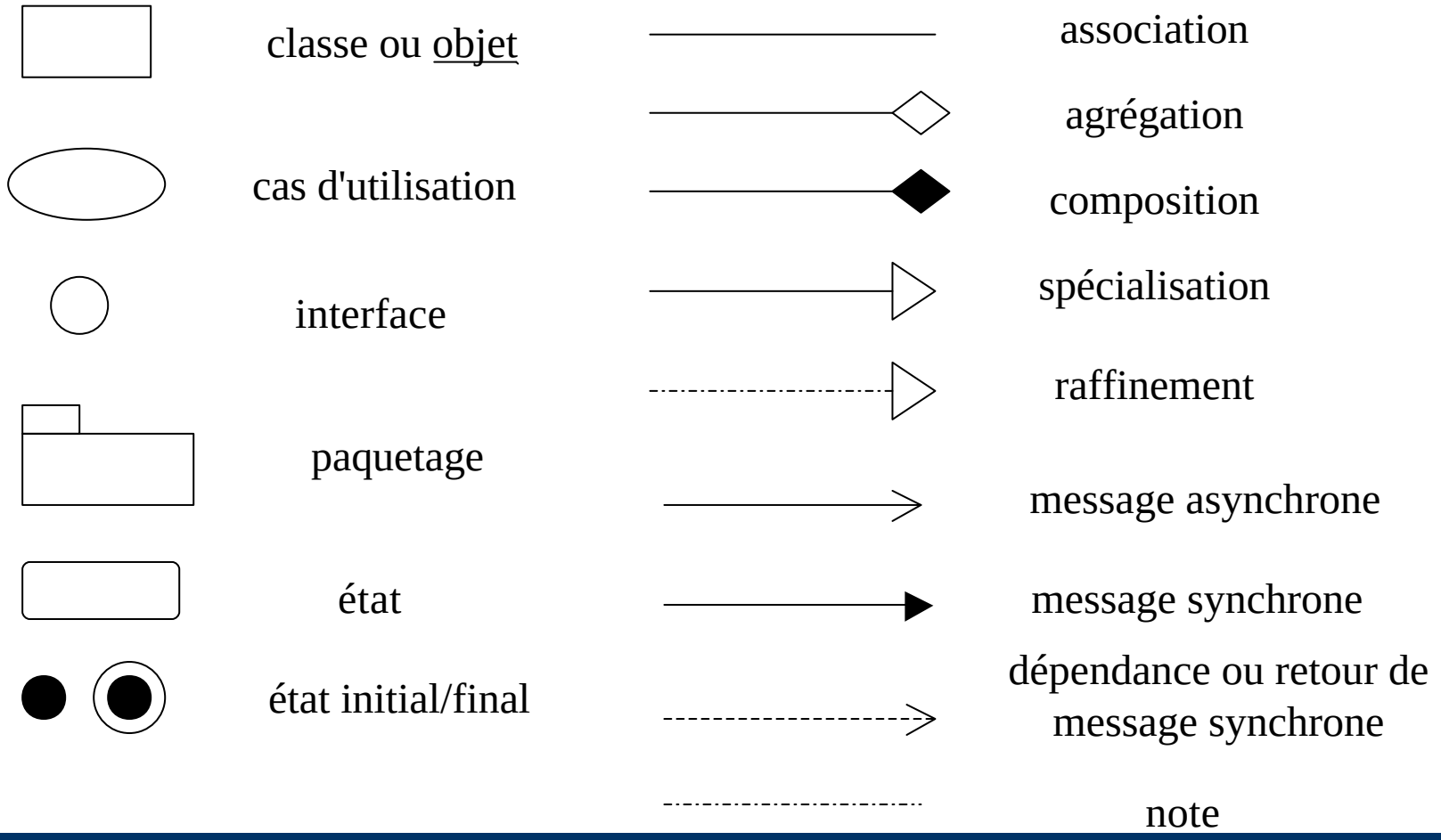
Diagrammes de composants



Diagrammes de deploiement

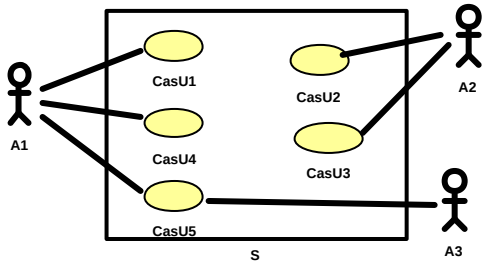


Principaux éléments graphiques



UML :
Diagrammes de Cas d'Utilisation

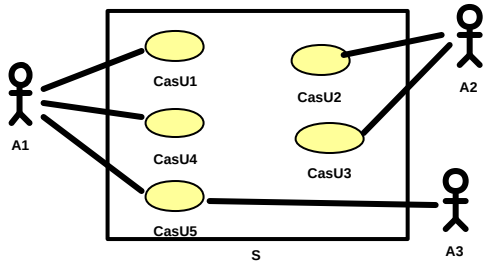
Acteurs, Cas d'utilisation, Système
Diagrammes de Cas d'utilisation
Modèle de Cas d'utilisation
Scénarii



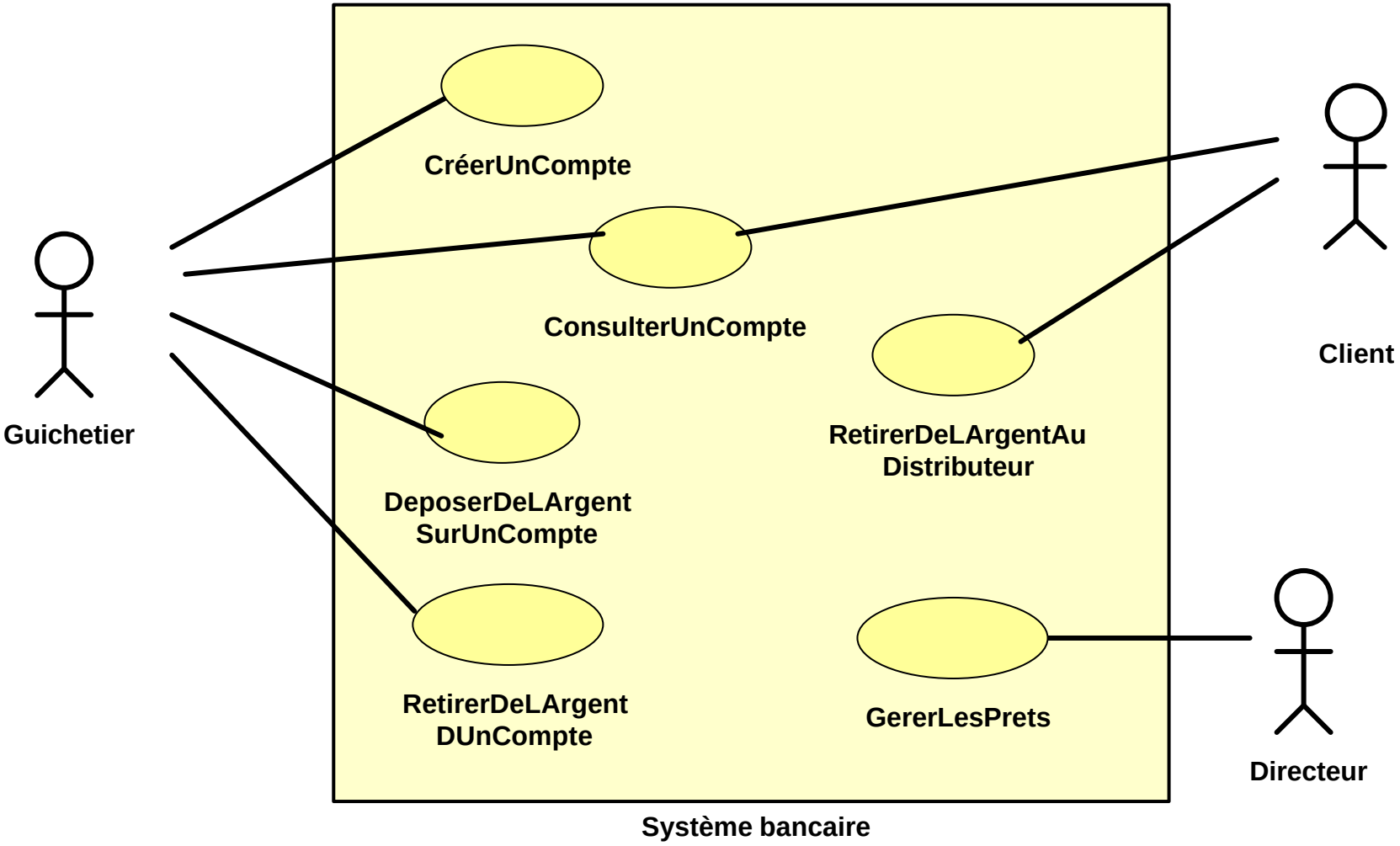
Jean-Marie Favre, Ioannis Parissis ©

Les diagrammes de cas d 'utilisation

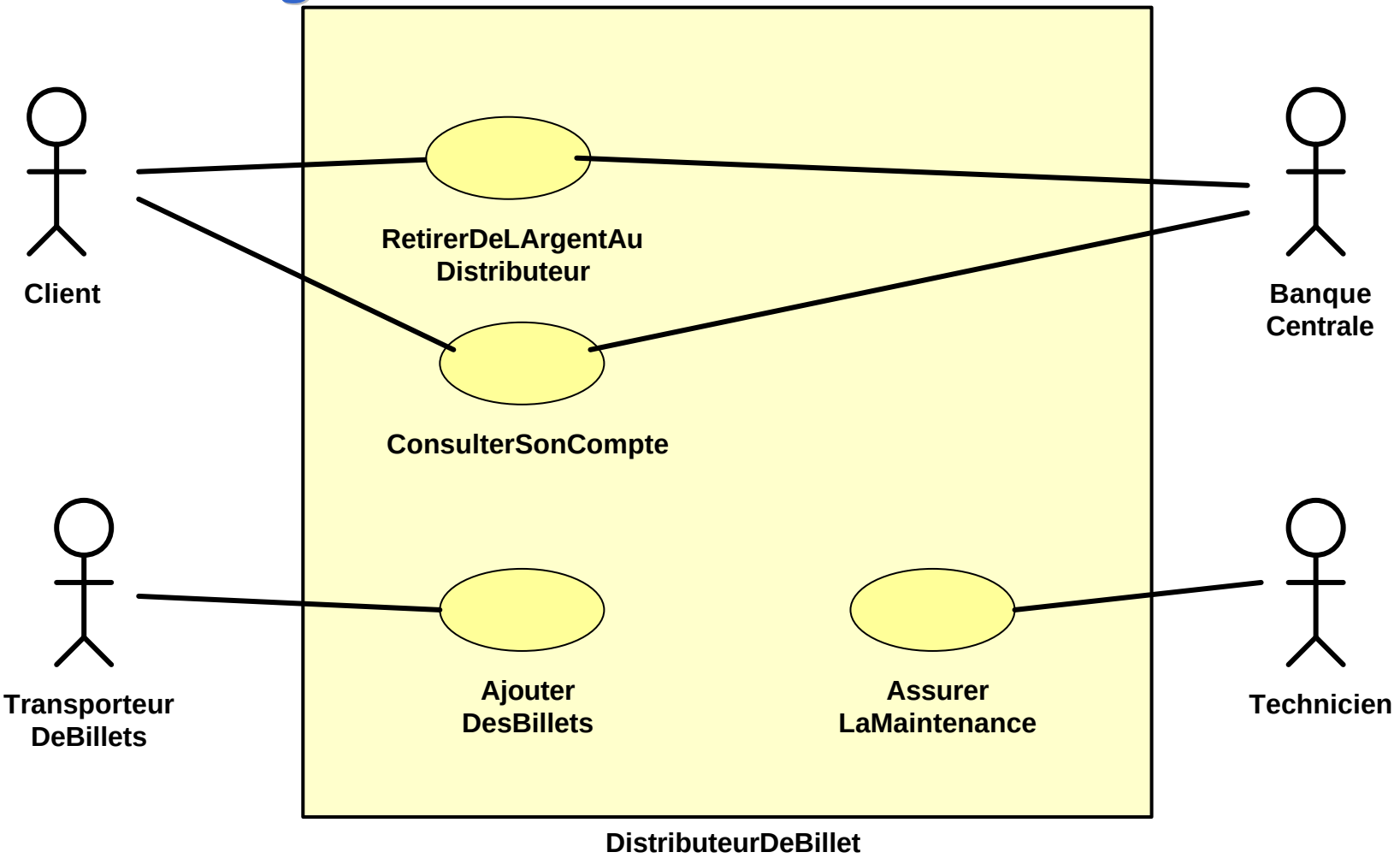
- Une des notations d 'UML (use-cases)
- But :
  - définir le système du point de vue des utilisateurs
  - définir les limites précises du système
- Notation très simple, compréhensible par tous
- Permet de structurer :
  - les besoins (cahier des charges)
  - le reste du développement



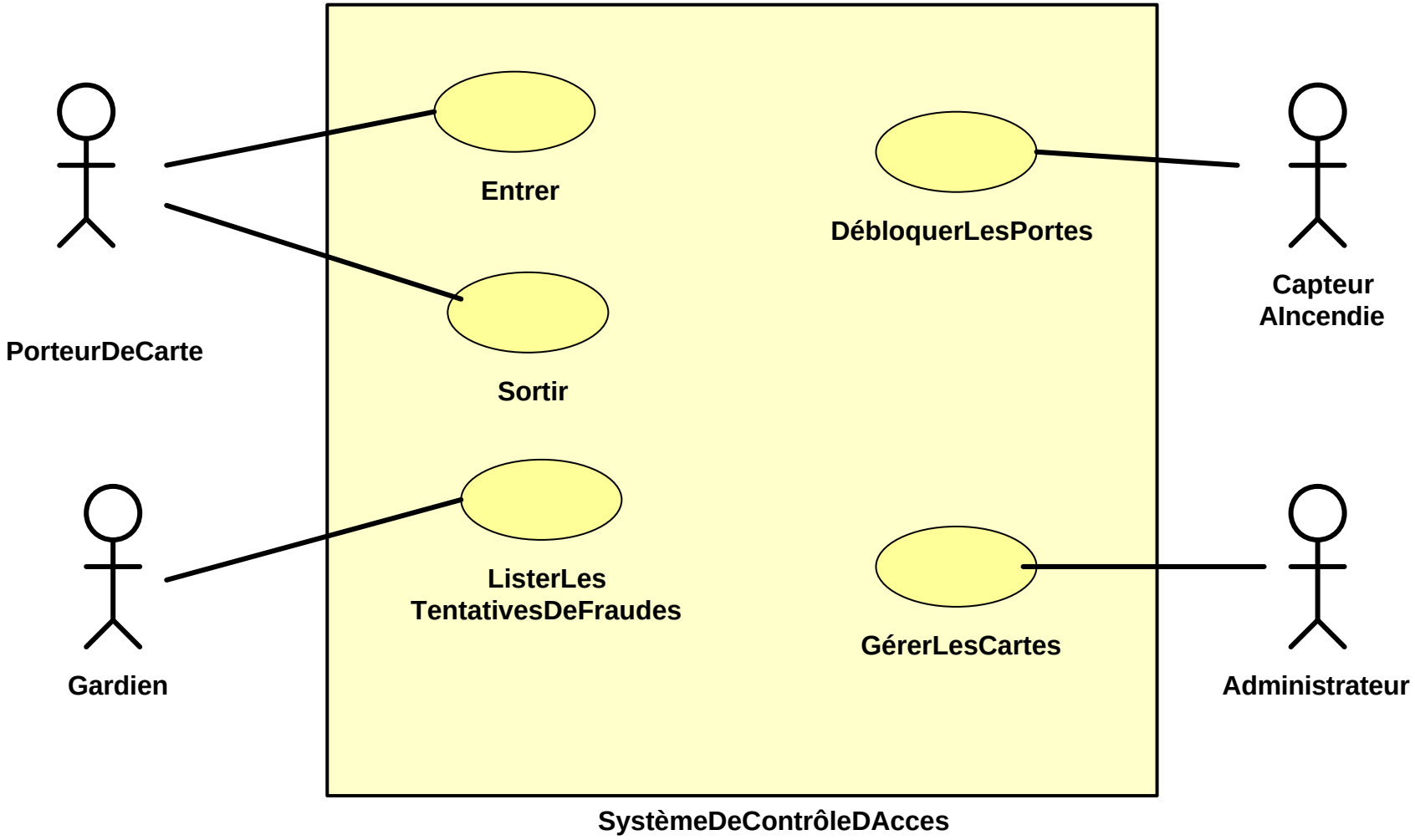
Exemple de diagrammes de cas d 'utilisation



Exemple de diagrammes de cas d 'utilisation

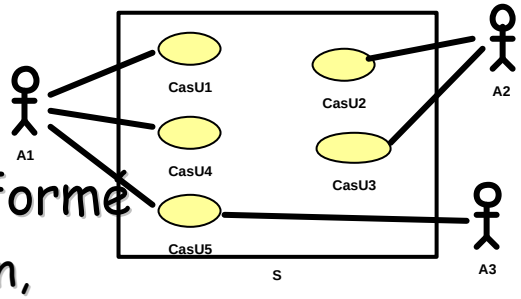


Exemple de diagrammes de cas d 'utilisation

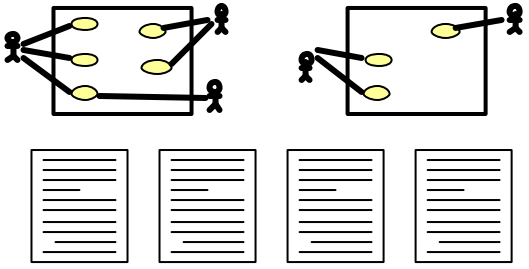


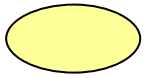
Modèle des cas d'utilisation

- Un **diagramme de cas d'utilisation** décrit :
  - les **acteurs**
  - les **cas d'utilisation**
  - le **système**



- Un **modèle de cas d'utilisation** peut être formé
  - de plusieurs diagrammes de cas d'utilisation,
  - de descriptions textuelles,
  - de diagrammes de séquences
  - ...





CasDUtilisationX

Cas d'utilisation (CU)

- Cas d'utilisation (CU)
  - une manière d'utiliser le système
  - une suite d'interactions entre un acteur et le système

    ex: le guichetier peut créer un nouveau compte

- Correspond à une fonction visible par l'utilisateur
- Permet d 'atteindre un but pour un utilisateur
- Doit être utile en soi

Le système

- Le système est un ensemble de cas d'utilisation
- Le système contient :
  - les cas d 'utilisation,
  - mais pas les acteurs.
- Un modèle de cas d 'utilisation permet de définir :
  - les fonctions essentielles du système,
  - les limites du système,
  - le système par rapport à son environnement.

Client

Acteurs

Un Acteur =

- élément externe qui interagit avec le système
- un acteur prend des décisions, des initiatives, il est "actif" (un élément logiciel ne prend pas de décision)
- rôle qu'un utilisateur joue par rapport au système

  ex: un client, un guichetier

Acteurs vs. utilisateurs

Ne pas confondre la notion d'Acteur
et de personne utilisant le système

- Une même personne peut jouer plusieurs rôles
  ex: Maurice est directeur mais peut jouer le rôle de guichetier
- Plusieurs personnes peuvent jouer un même rôle
  ex: Paul et Pierre sont deux clients
- Un acteur n'est pas forcément un être humain
  ex: un distributeur de billet peut être vu comme un acteur

Différents types d 'acteurs

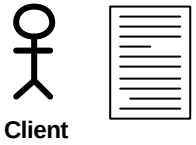
- Utilisateurs principaux

  ex: client, guichetier
- Utilisateurs secondaires

  ex: contrôleur, directeur, ingénieur système, ...
- Périphériques externes

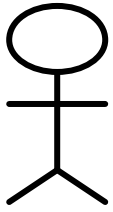
  ex: un capteur, une horloge externe, ...
- Systèmes externes

  ex: système interbancaires



Description des acteurs

- Pour chaque acteur :
  - choisir un identificateur représentatif de son rôle
  - donner une brève description textuelle



Guichetier

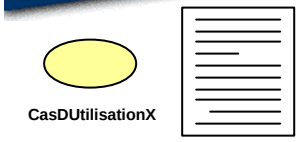
Un guichetier est un employé de la banque chargé de faire l'interface entre le système informatique et les clients qu'il reçoit au comptoir. Le guichetier peut réaliser les opérations courantes : création d 'un compte, dépôt et retrait d 'argent, etc.

Utilité des acteurs

- La définition d'acteurs permet surtout
  - de trouver les cas d'utilisation

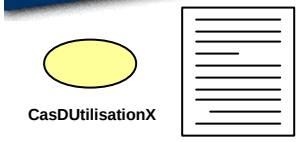
    ex: que peut faire un guichetier ? un client ? le directeur ?

- mais peut aussi être utilisée pour
  - définir différents points de vues sur le système,
  - déterminer des droits d'accès par type d'acteur,
  - fixer des ordres de priorités entre acteurs,
  - ...

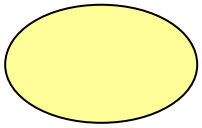


Description des cas d'utilisation

- Pour chaque cas d 'utilisation
  - choisir un identificateur représentatif
  - donner une description textuelle simple
  - la fonction réalisée doit être comprise de tous
  - pas trop de détails
  - préciser ce que fait le système, ce que fait l'acteur
- Les cas d'utilisation ne doivent pas se chevaucher



Exemple de description d 'un cas d 'utilisation

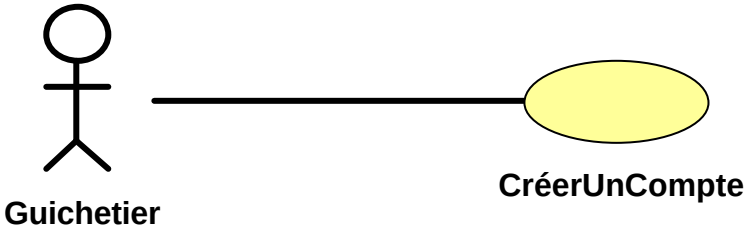


Retirer DeLArgent AuDistributeur

Lorsqu'un *client* à besoin de retirer du liquide il peut en utilisant un distributeur retirer de l 'argent de son compte. Pour cela
- le *client* insère sa carte bancaire dans le distributeur
- le *système* demande le code pour l 'identifier
- le *client* choisi le montant du retrait
- le *système* vérifie qu 'il y a suffisamment d 'argent
- si c 'est le cas, le *système* distribue les billets et retire l 'argent du compte du client
- le *client* prend les billets et retire sa carte

Relation acteur - cas d'utilisation : communication

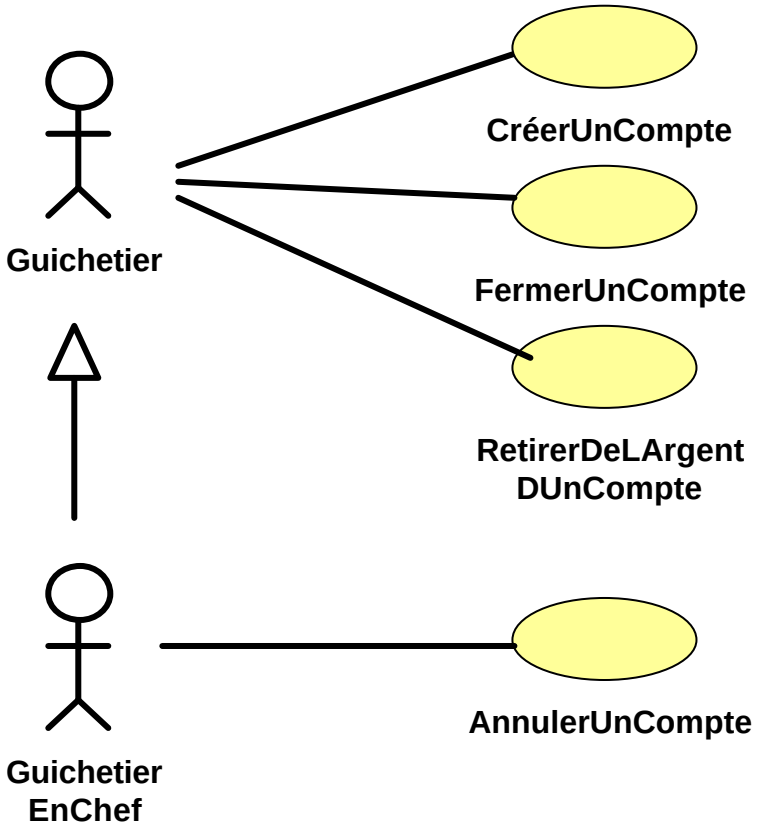
- Représente une communication (initiée par l'acteur)
- Echange de messages potentiellement dans les deux sens



- Sera raffiné par la suite (e.g. document de spécifications externes)
  - Si l'acteur est un humain : interface homme – système
  - Si l'acteur est un logiciel : interface logicielle

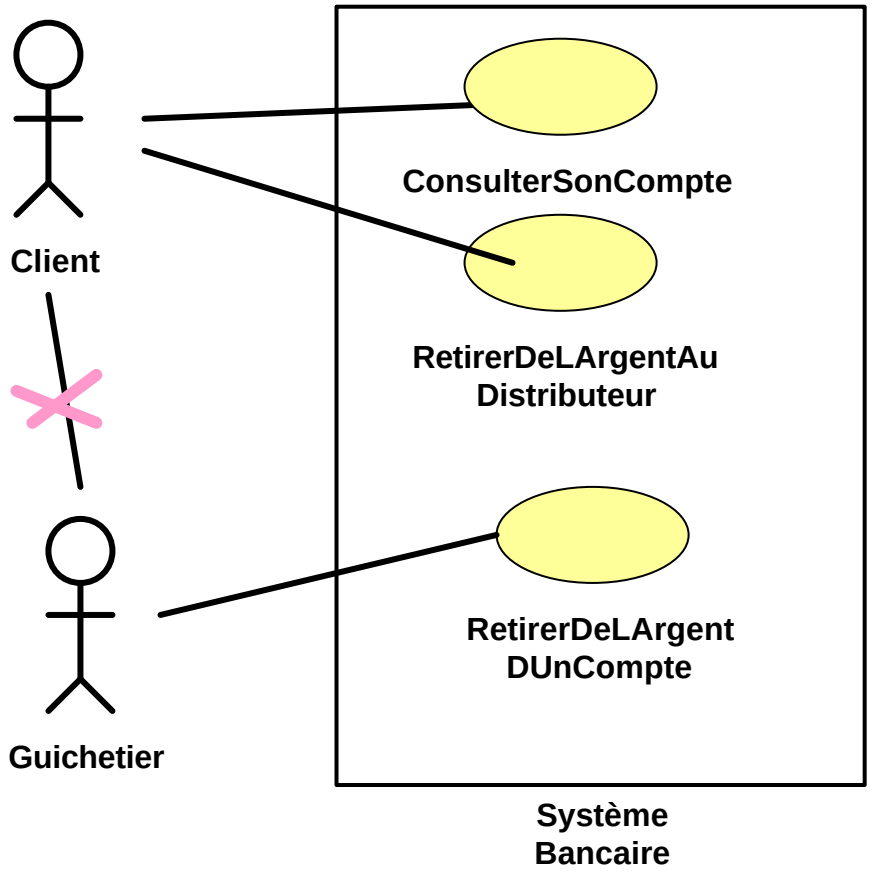
Relation entre acteurs : généralisation

- La seule relation entre acteurs est la relation de généralisation

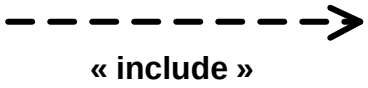


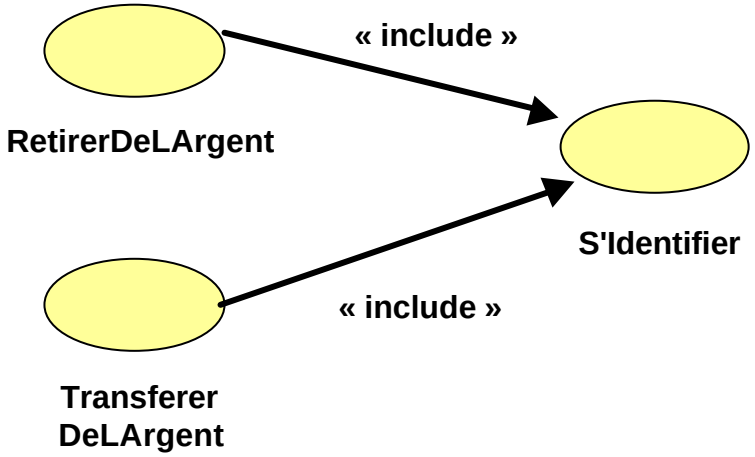
Remarque

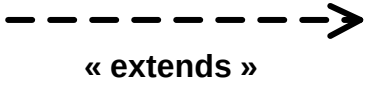
- Les communications externes entre acteurs n'est pas modélisée.
- UML se concentre sur la description du système et de ses interactions avec l'extérieur



Relations entre cas d'utilisation : inclusion et extension

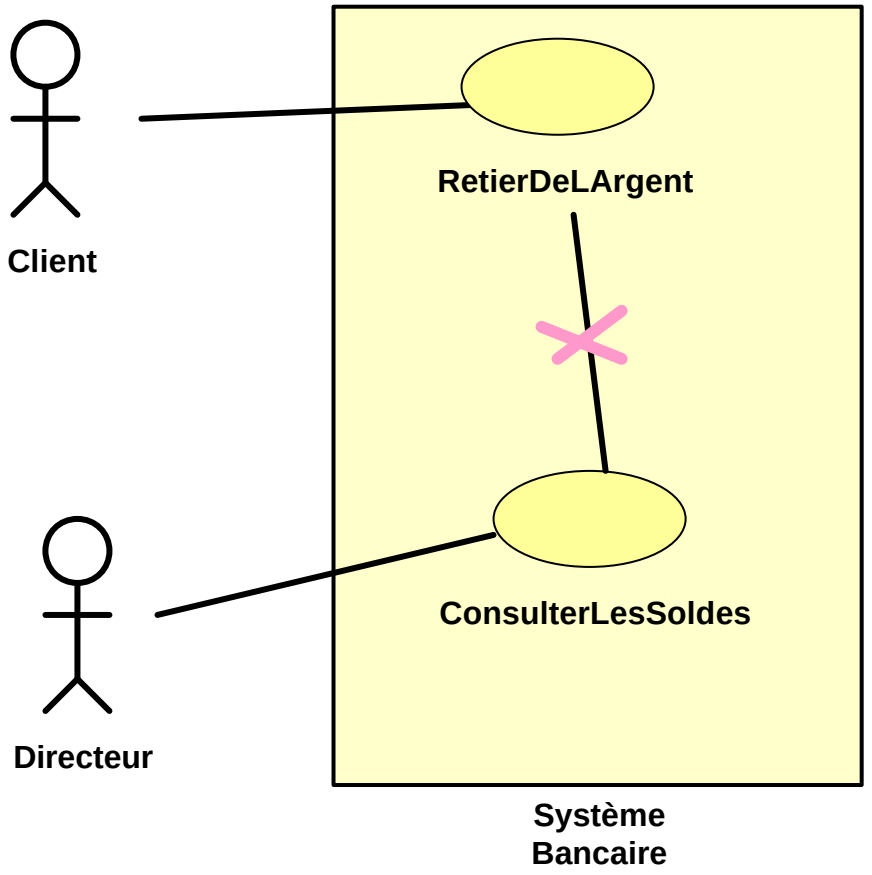


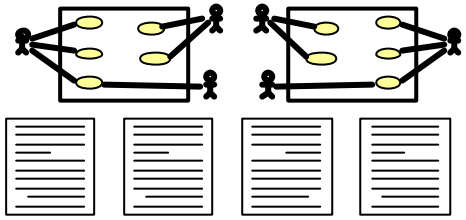




Remarque

- Pas de communications internes entre cas d'utilisation.
- UML se concentre sur la description du système et de ses interactions avec l'extérieur

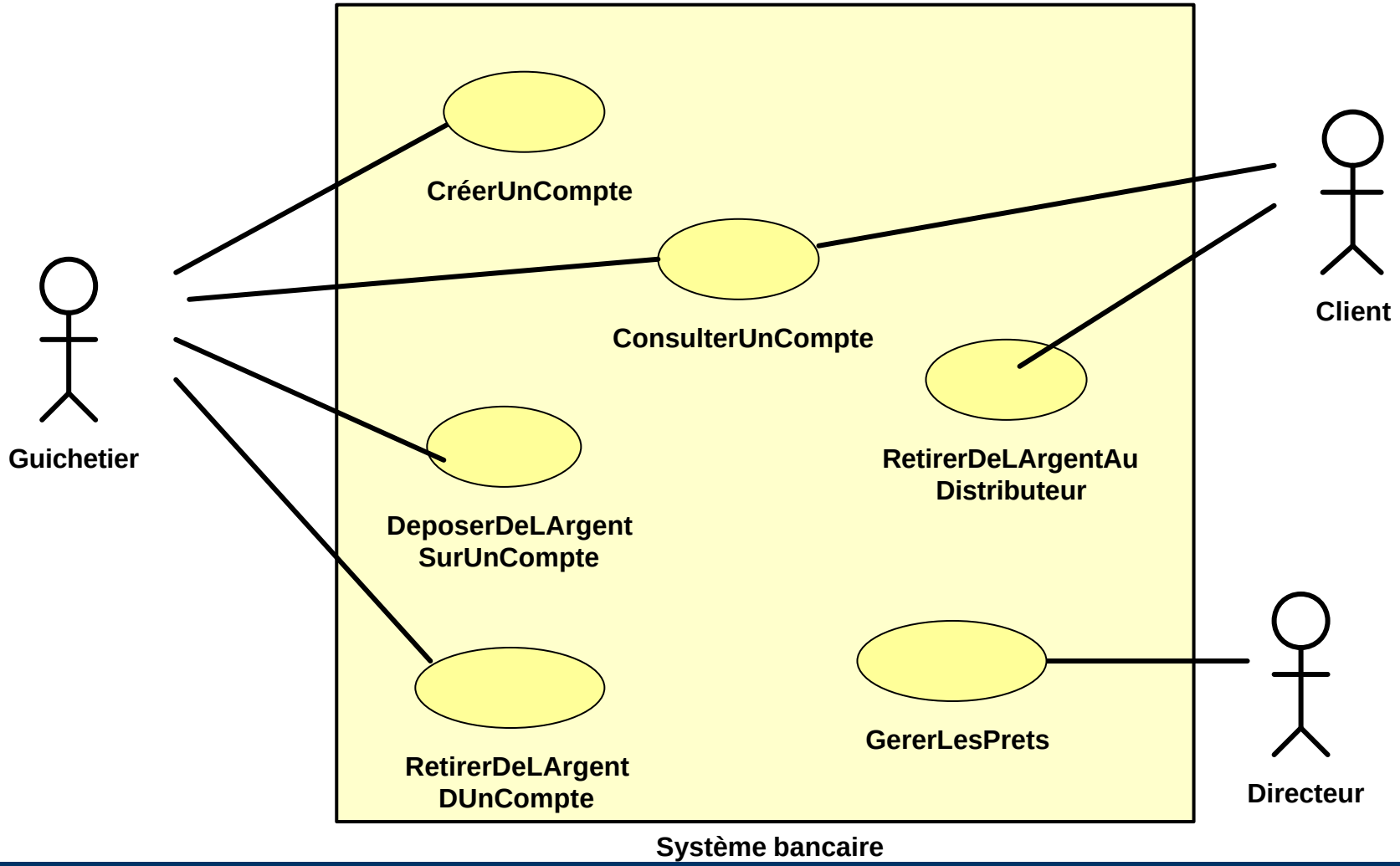




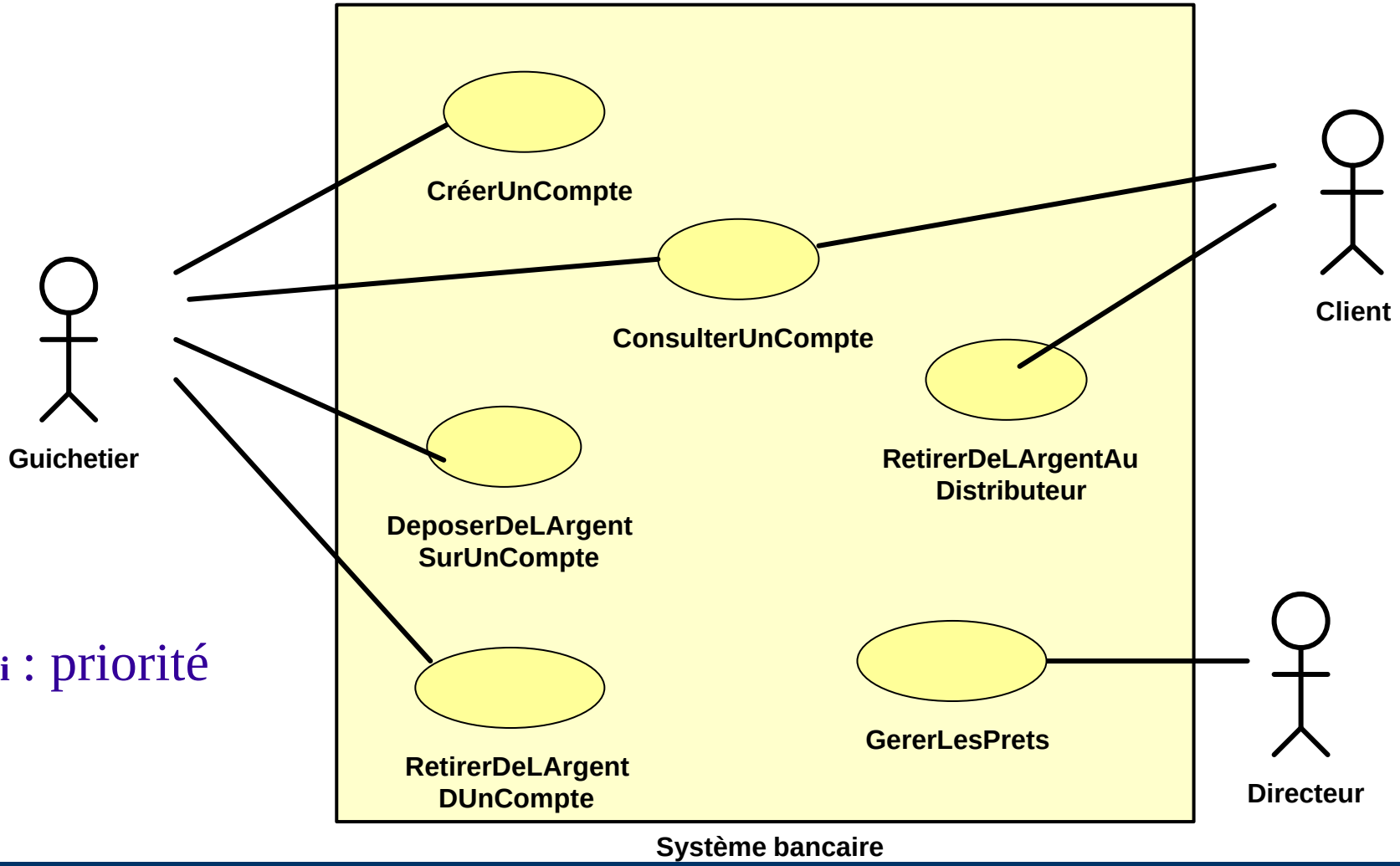
Description du modèle de cas d 'utilisation

- Un modèle de cas d'utilisation peut contenir
  - plusieurs diagrammes
  - plusieurs descriptions textuelles
- Structuration en termes de paquetages
- Permet d'avoir une vision globale du système
- Permet de définir des priorités entre CU
- Utile pour le client, pour la planification,...
- Trop général pour être utilisé par les développeurs

Exemple de diagramme de CU décoré

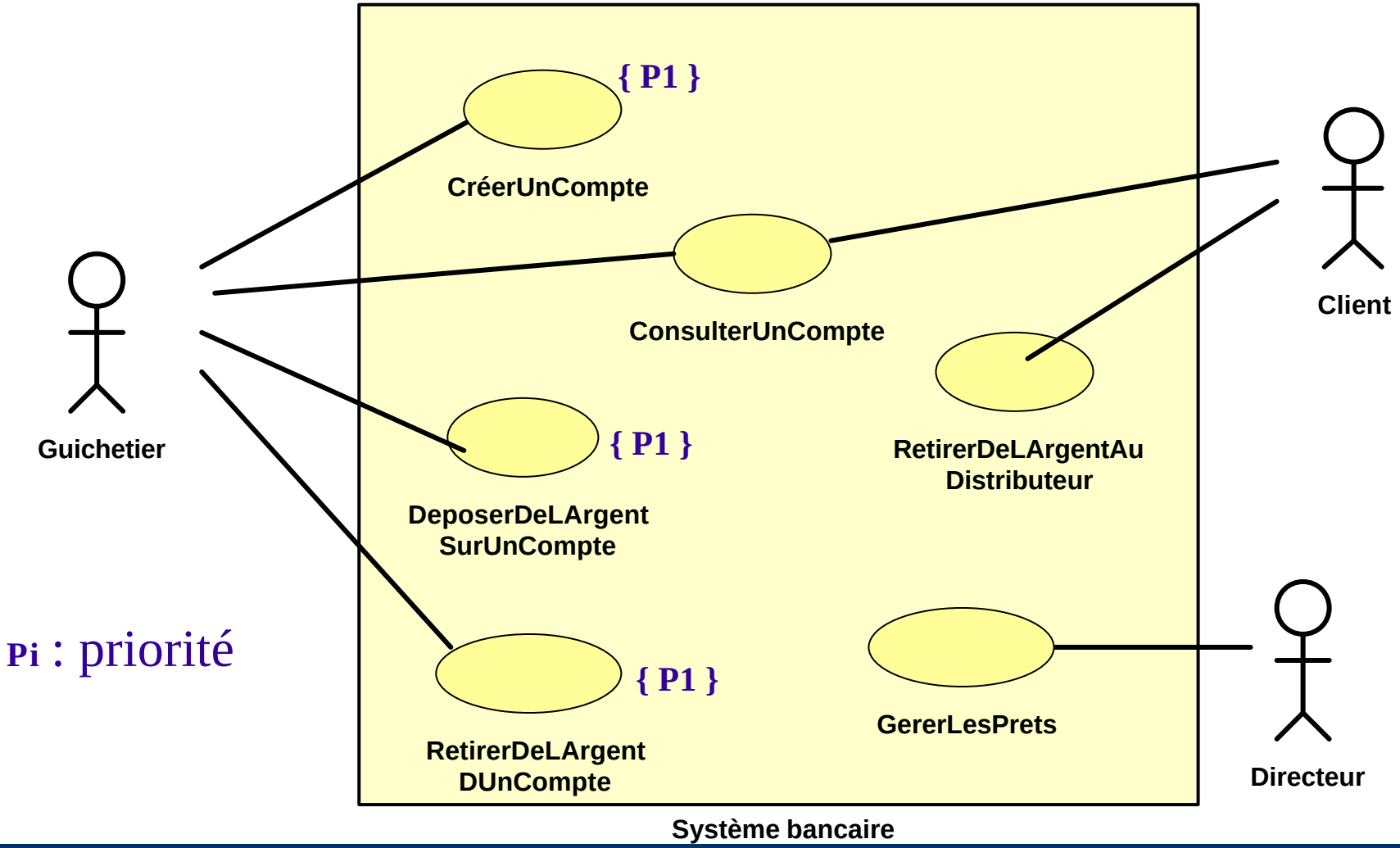


Exemple de diagramme de CU décoré



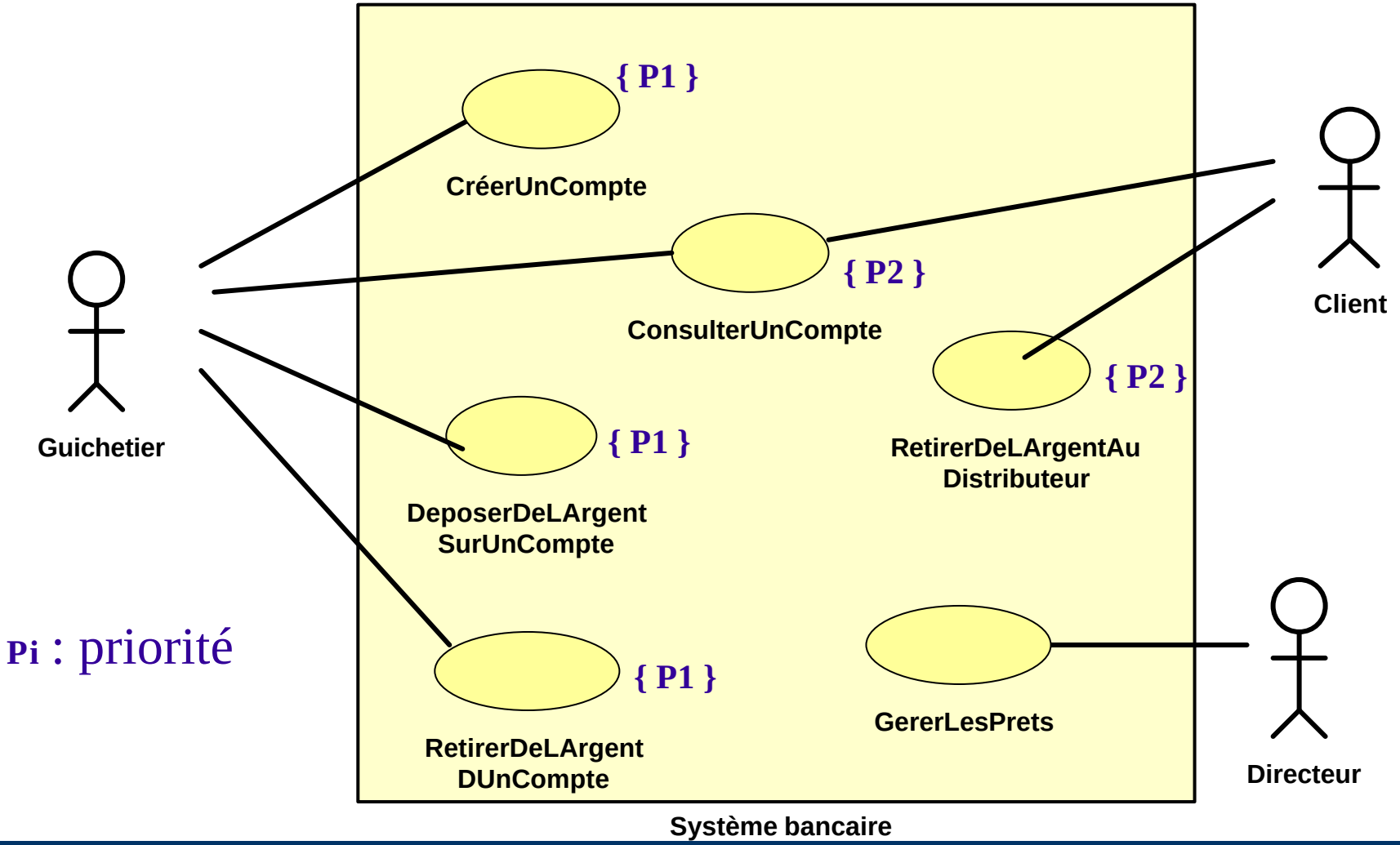
Pi : priorité

Exemple de diagramme de CU décoré



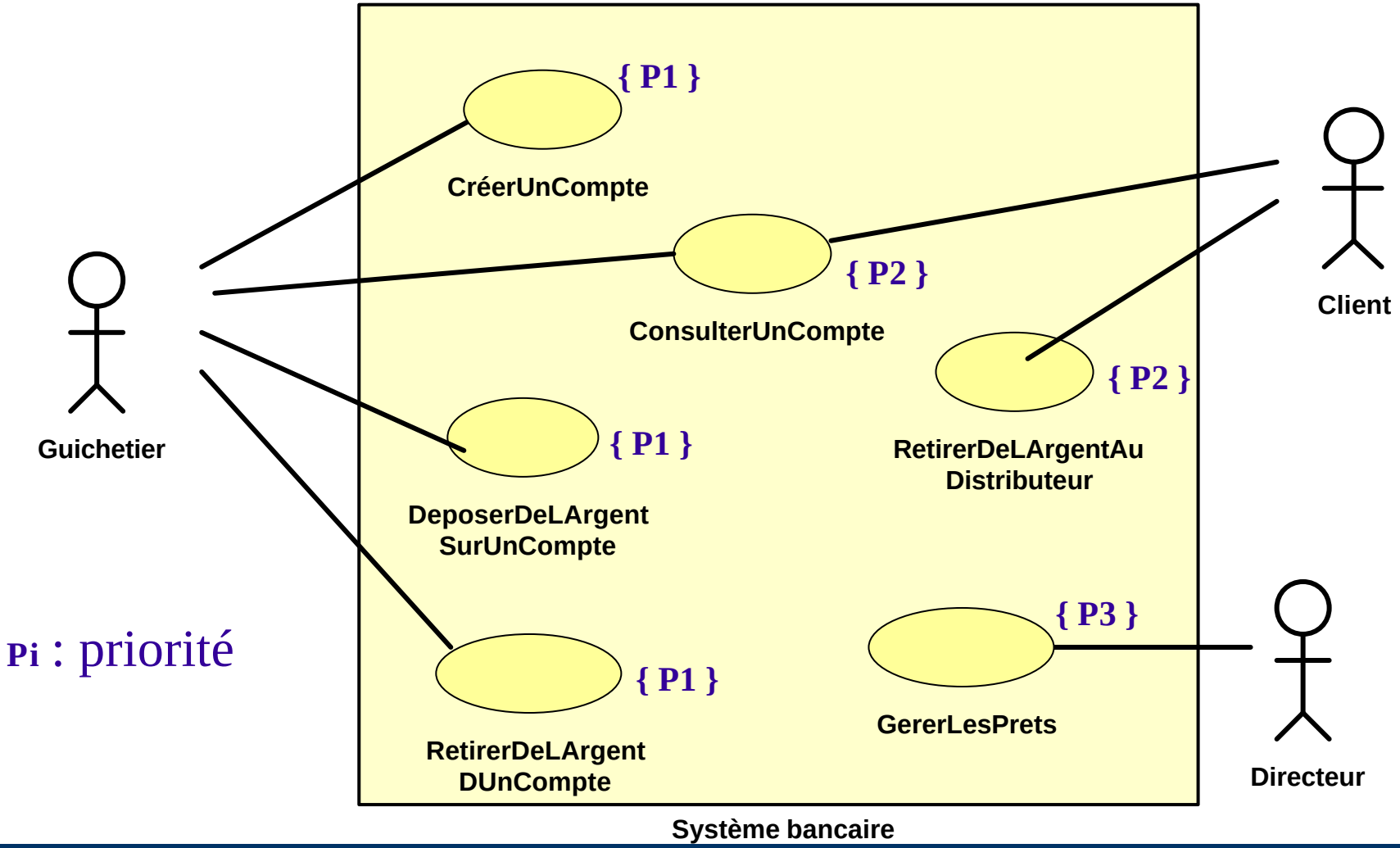
Pi : priorité

Exemple de diagramme de CU décoré



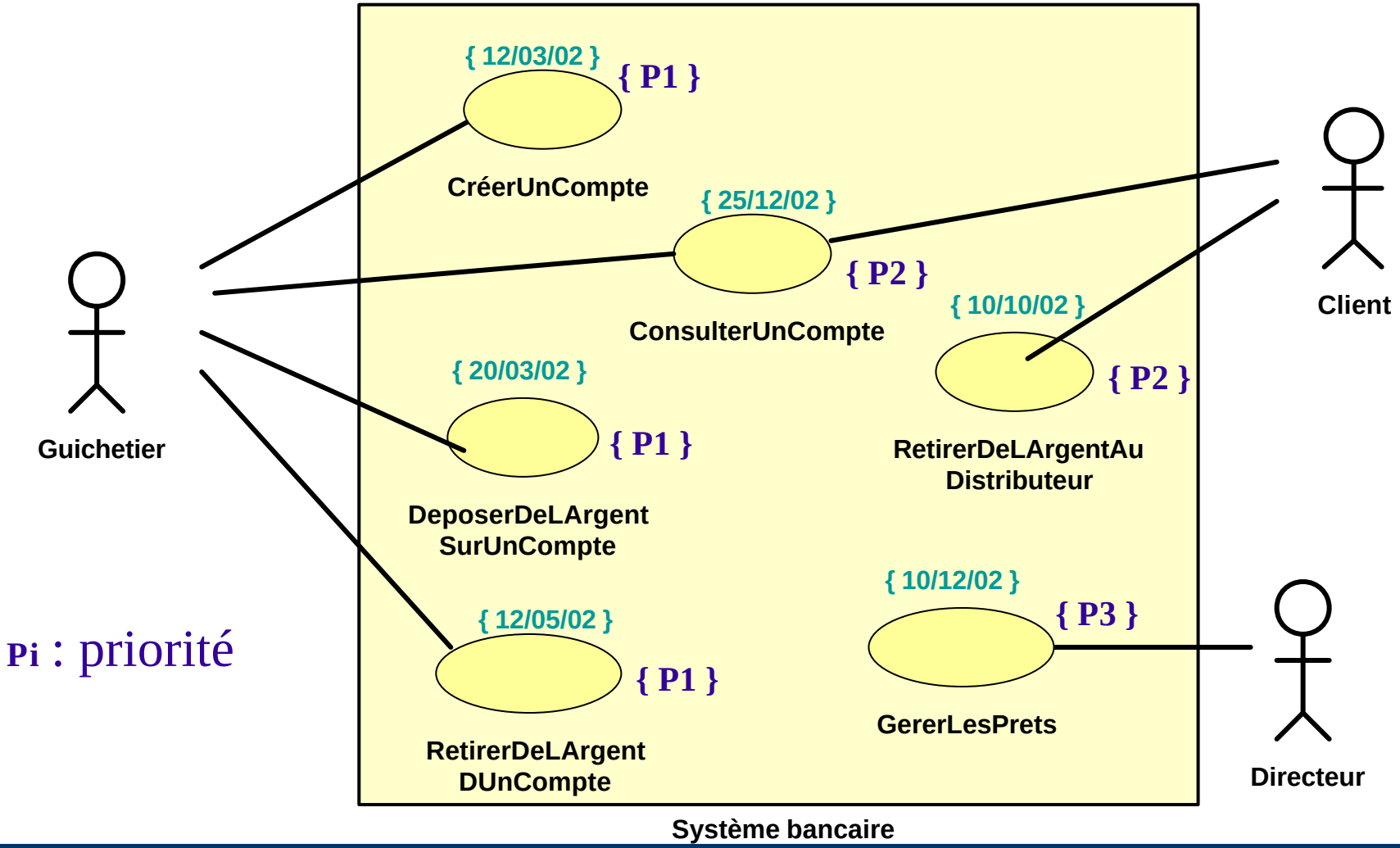
Pi : priorité

Exemple de diagramme de CU décoré



Pi : priorité

Exemple de diagramme de CU décoré



Pi : priorité

Description détaillée des cas d'utilisation

Jean-Marie Favre, Ioannis Parissis ©

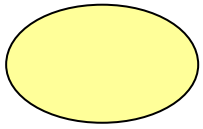
Description détaillée de chaque cas d'utilisation

- Chaque cas d 'utilisation doit être décrit en détail
- Commencer par les CU prioritaires
- Description utile pour la suite du développement
- Description détaillée plus où moins formelle
  - langue naturelle mais structurée, vocabulaire précis
  - diagramme d 'états
  - ...

Informations à décrire

- Quand le CU commence, pré-conditions
- Quand le CU se termine, post-conditions
- Le chemin correspondant au déroulement normal
- Les variantes possibles et les cas d'erreurs
- Les interactions entre le système et les acteurs
- Les informations échangées
- Les éventuels besoins non fonctionnels

Exemple de description détaillée d 'un CU



Retirer DeLArgent AuDistributeur

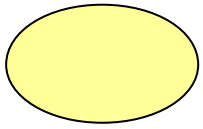
Précondition :
Le distributeur contient des billets, il est en attente d 'une opération, il n'est ni en panne, ni en maintenance

Début : lorsqu 'un client introduit sa carte bancaire dans le distributeur.

Fin : lorsque la carte bancaire et les billets sont sortis.

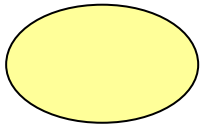
Postcondition :
Si de l 'argent a pu être retiré la somme d'argent sur le compte est égale à la somme d 'argent qu'il y avait avant, moins le montant du retrait. Sinon la somme d 'argent sur le compte est la même qu'avant.

Exemple de description détaillée d 'un CU



Retirer DeLArgent AuDistributeur

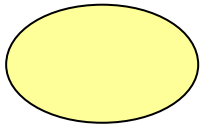
Exemple de description détaillée d 'un CU



Retirer DeLArgent AuDistributeur

Déroulement normal :

Exemple de description détaillée d 'un CU

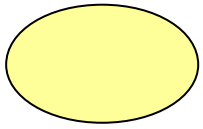


Retirer DeLArgent AuDistributeur

Déroulement normal :

(1) le *client* introduit sa carte bancaire

Exemple de description détaillée d 'un CU



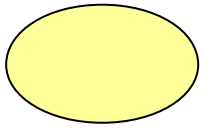
Retirer DeLArgent AuDistributeur

Déroulement normal :

(1) le *client* introduit sa carte bancaire

(2) le *système* lit la carte et vérifie si la carte est valide

Exemple de description détaillée d 'un CU



Retirer DeLArgent AuDistributeur

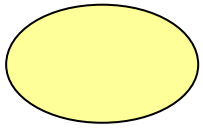
Déroulement normal :

(1) le *client* introduit sa carte bancaire

(2) le *système* lit la carte et vérifie si la carte est valide

(3) le *système* demande au client de taper son code

Exemple de description détaillée d 'un CU

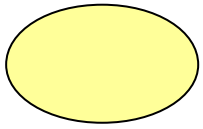


Retirer DeLArgent AuDistributeur

Déroulement normal :

(1) le *client* introduit sa carte bancaire
(2) le *système* lit la carte et vérifie si la carte est valide
(3) le *système* demande au client de taper son code
(4) le *client* tape son code confidentiel

Exemple de description détaillée d 'un CU

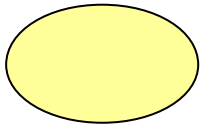


Retirer DeLArgent AuDistributeur

Déroulement normal :

(1) le *client* introduit sa carte bancaire
(2) le *système* lit la carte et vérifie si la carte est valide
(3) le *système* demande au client de taper son code
(4) le *client* tape son code confidentiel
(5) le *système* vérifie que le code correspond à la carte

Exemple de description détaillée d 'un CU

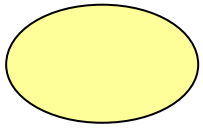


Retirer DeLArgent AuDistributeur

Déroulement normal :

(1) le *client* introduit sa carte bancaire
(2) le *système* lit la carte et vérifie si la carte est valide
(3) le *système* demande au client de taper son code
(4) le *client* tape son code confidentiel
(5) le *système* vérifie que le code correspond à la carte
(6) le *client* choisi une opération de retrait

Exemple de description détaillée d 'un CU

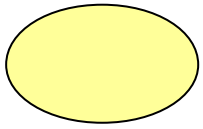


Retirer DeLArgent AuDistributeur

Déroulement normal :

(1) le *client* introduit sa carte bancaire
(2) le *système* lit la carte et vérifie si la carte est valide
(3) le *système* demande au client de taper son code
(4) le *client* tape son code confidentiel
(5) le *système* vérifie que le code correspond à la carte
(6) le *client* choisi une opération de retrait
(7) le *système* demande le montant à retirer

Exemple de description détaillée d 'un CU

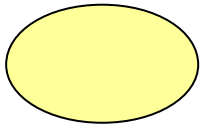


Retirer DeLArgent AuDistributeur

Déroulement normal :

(1) le *client* introduit sa carte bancaire
(2) le *système* lit la carte et vérifie si la carte est valide
(3) le *système* demande au client de taper son code
(4) le *client* tape son code confidentiel
(5) le *système* vérifie que le code correspond à la carte
(6) le *client* choisi une opération de retrait
(7) le *système* demande le montant à retirer
…

Exemple de description détaillée d 'un CU

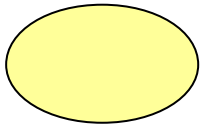


Retirer DeLArgent AuDistributeur

Déroulement normal :
(1) le *client* introduit sa carte bancaire
(2) le *système* lit la carte et vérifie si la carte est valide
(3) le *système* demande au client de taper son code
(4) le *client* tape son code confidentiel
(5) le *système* vérifie que le code correspond à la carte
(6) le *client* choisi une opération de retrait
(7) le *système* demande le montant à retirer
…

Variantes :

Exemple de description détaillée d 'un CU

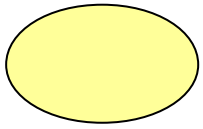


Retirer DeLArgent AuDistributeur

Déroulement normal :
(1) le *client* introduit sa carte bancaire
(2) le *système* lit la carte et vérifie si la carte est valide
(3) le *système* demande au client de taper son code
(4) le *client* tape son code confidentiel
(5) le *système* vérifie que le code correspond à la carte
(6) le *client* choisi une opération de retrait
(7) le *système* demande le montant à retirer
…

Variantes :
(A) *Carte invalide* : au cours de l 'étape (2) si la carte est jugée invalide, le système affiche un message d 'erreur, rejète la carte et le cas d 'utilisation se termine.

Exemple de description détaillée d 'un CU

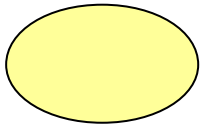


Retirer DeLArgent AuDistributeur

Déroulement normal :
(1) le *client* introduit sa carte bancaire
(2) le *système* lit la carte et vérifie si la carte est valide
(3) le *système* demande au client de taper son code
(4) le *client* tape son code confidentiel
(5) le *système* vérifie que le code correspond à la carte
(6) le *client* choisi une opération de retrait
(7) le *système* demande le montant à retirer
…

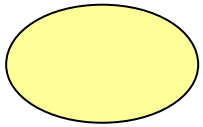
Variantes :
(A) *Carte invalide* : au cours de l 'étape (2) si la carte est jugée invalide, le système affiche un message d 'erreur, rejète la carte et le cas d 'utilisation se termine.
(B) *Code erroné* : au cours de l 'étape (5) ...

Exemple de description détaillée d 'un CU



Retirer DeLArgent AuDistributeur

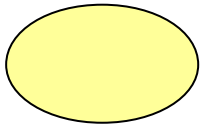
Exemple de description détaillée d 'un CU



Retirer DeLArgent AuDistributeur

Contraintes non fonctionnelles :

Exemple de description détaillée d 'un CU

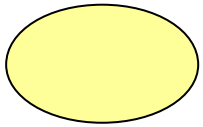


Retirer DeLArgent AuDistributeur

Contraintes non fonctionnelles :

(A) *Performance* : le système doit réagir dans un délai inférieur à 4 secondes, quelque soit l'action de l 'utilisateur.

Exemple de description détaillée d 'un CU



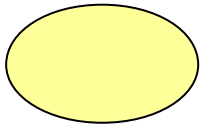
Retirer DeLArgent AuDistributeur

Contraintes non fonctionnelles :

(A) *Performance* : le système doit réagir dans un délai inférieur à 4 secondes, quelque soit l'action de l 'utilisateur.

(B) *Résistance aux pannes* : si une coupure de courant ou une autre défaillance survient au cours du cas d 'utilisation, la transaction sera annulée, l 'argent ne sera pas distribué. Le système doit pouvoir redémarrer automatiquement dans un état cohérent et sans intervention humaine.

Exemple de description détaillée d 'un CU



Retirer DeLArgent AuDistributeur

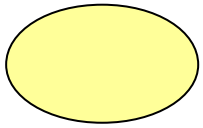
Contraintes non fonctionnelles :

(A) *Performance* : le système doit réagir dans un délai inférieur à 4 secondes, quelque soit l'action de l 'utilisateur.

(B) *Résistance aux pannes* : si une coupure de courant ou une autre défaillance survient au cours du cas d 'utilisation, la transaction sera annulée, l 'argent ne sera pas distribué. Le système doit pouvoir redémarrer automatiquement dans un état cohérent et sans intervention humaine.

(C) *Résistance à la charge* : le système doit pouvoir gérer plus de 1000 retraits d 'argent simultanément

Exemple de description détaillée d 'un CU



Retirer DeLArgent AuDistributeur

Contraintes non fonctionnelles :

(A) *Performance* : le système doit réagir dans un délai inférieur à 4 secondes, quelque soit l'action de l 'utilisateur.

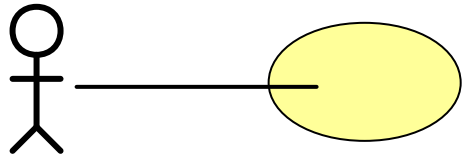
(B) *Résistance aux pannes* : si une coupure de courant ou une autre défaillance survient au cours du cas d 'utilisation, la transaction sera annulée, l 'argent ne sera pas distribué. Le système doit pouvoir redémarrer automatiquement dans un état cohérent et sans intervention humaine.

(C) *Résistance à la charge* : le système doit pouvoir gérer plus de 1000 retraits d 'argent simultanément

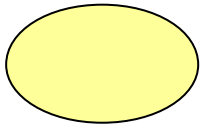
...

Scénario

- Pour décrire ou valider un CU : les **scénarii**
- Un scénario est un exemple :
  - une manière particulière d 'utiliser le système …
  - … par une personne particulière …
  - … dans un contexte particulier.
- cas d 'utilisation = ensemble de scénarios
- scénario = une exécution particulière d'un CU

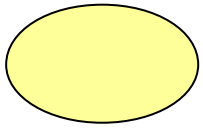


Exemple de scénario



Retirer
DeLArgent
AuDistributeur

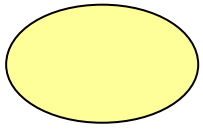
Exemple de scénario



Retirer DeLArgent AuDistributeur

SCENARIO 4

Exemple de scénario

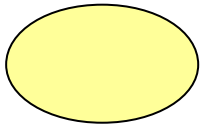


Retirer DeLArgent AuDistributeur

SCENARIO 4

- *Paul* insère sa carte dans le distributeur d2103

Exemple de scénario

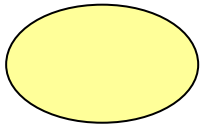


Retirer DeLArgent AuDistributeur

SCENARIO 4

- *Paul* insère sa carte dans le distributeur d2103
- Le *système* accepte la carte et lit le numéro de compte

Exemple de scénario

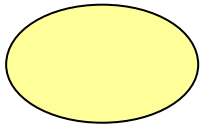


Retirer DeLArgent AuDistributeur

SCENARIO 4

- *Paul* insère sa carte dans le distributeur d2103
- Le *système* accepte la carte et lit le numéro de compte
- Le *système* demande le code

Exemple de scénario

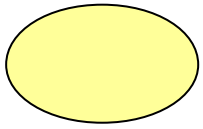


Retirer DeLArgent AuDistributeur

SCENARIO 4

- *Paul* insère sa carte dans le distributeur d2103
- Le *système* accepte la carte et lit le numéro de compte
- Le *système* demande le code
- *Paul* tape ‘ 1234 ’

Exemple de scénario

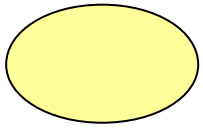


Retirer DeLArgent AuDistributeur

SCENARIO 4

- *Paul* insère sa carte dans le distributeur d2103
- Le *système* accepte la carte et lit le numéro de compte
- Le *système* demande le code
- *Paul* tape ‘ 1234 ’
- Le *système* indique que ce n ’est pas le bon code

Exemple de scénario

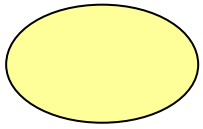


Retirer DeLArgent AuDistributeur

SCENARIO 4

- *Paul* insère sa carte dans le distributeur d2103
- Le *système* accepte la carte et lit le numéro de compte
- Le *système* demande le code
- *Paul* tape ‘ 1234 ’
- Le *système* indique que ce n ’est pas le bon code
- Le *système* affiche un message et propose de recommencer

Exemple de scénario

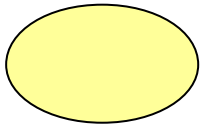


Retirer DeLArgent AuDistributeur

SCENARIO 4

- *Paul* insère sa carte dans le distributeur d2103
- Le *système* accepte la carte et lit le numéro de compte
- Le *système* demande le code
- *Paul* tape ‘ 1234 ’
- Le *système* indique que ce n ’est pas le bon code
- Le *système* affiche un message et propose de recommencer
- *Paul* tape ‘ 6622’

Exemple de scénario

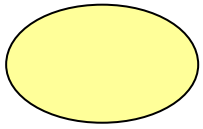


Retirer DeLArgent AuDistributeur

SCENARIO 4

- *Paul* insère sa carte dans le distributeur d2103
- Le *système* accepte la carte et lit le numéro de compte
- Le *système* demande le code
- *Paul* tape ‘ 1234 ’
- Le *système* indique que ce n ’est pas le bon code
- Le *système* affiche un message et propose de recommencer
- *Paul* tape ‘ 6622’
- Le *système* affiche que le code est correct

Exemple de scénario

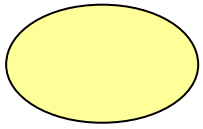


Retirer DeLArgent AuDistributeur

SCENARIO 4

- *Paul* insère sa carte dans le distributeur d2103
- Le *système* accepte la carte et lit le numéro de compte
- Le *système* demande le code
- *Paul* tape ‘ 1234 ’
- Le *système* indique que ce n ’est pas le bon code
- Le *système* affiche un message et propose de recommencer
- *Paul* tape ‘ 6622’
- Le *système* affiche que le code est correct
- Le *système* demande le montant du retrait

Exemple de scénario

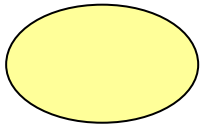


Retirer DeLArgent AuDistributeur

SCENARIO 4

- *Paul* insère sa carte dans le distributeur d2103
- Le *système* accepte la carte et lit le numéro de compte
- Le *système* demande le code
- *Paul* tape ‘ 1234 ’
- Le *système* indique que ce n ’est pas le bon code
- Le *système* affiche un message et propose de recommencer
- *Paul* tape ‘ 6622’
- Le *système* affiche que le code est correct
- Le *système* demande le montant du retrait
- *Paul* tape 50000 fr

Exemple de scénario

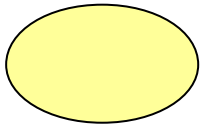


Retirer DeLArgent AuDistributeur

SCENARIO 4

- *Paul* insère sa carte dans le distributeur d2103
- Le *système* accepte la carte et lit le numéro de compte
- Le *système* demande le code
- *Paul* tape ‘ 1234 ’
- Le *système* indique que ce n ’est pas le bon code
- Le *système* affiche un message et propose de recommencer
- *Paul* tape ‘ 6622’
- Le *système* affiche que le code est correct
- Le *système* demande le montant du retrait
- *Paul* tape 50000 fr
- Le *système* vérifie s ’il y a assez d ’argent sur le compte

Exemple de scénario



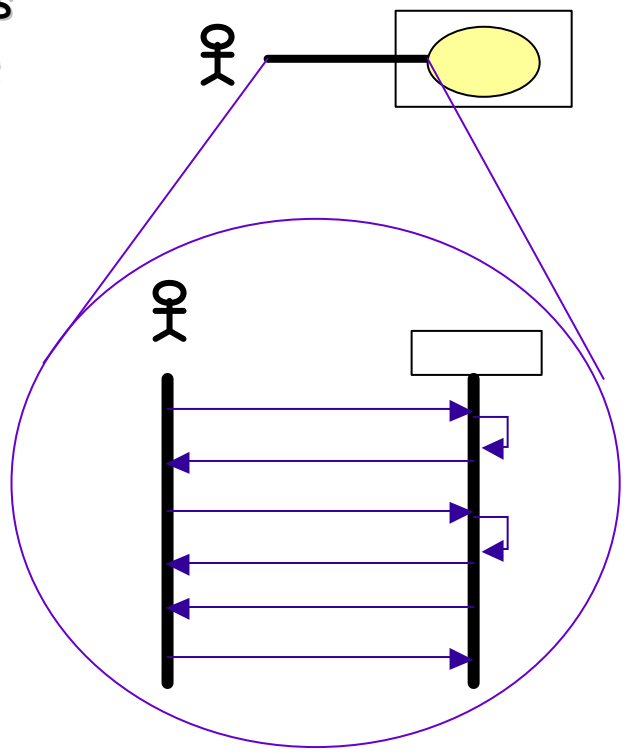
Retirer DeLArgent AuDistributeur

SCENARIO 4

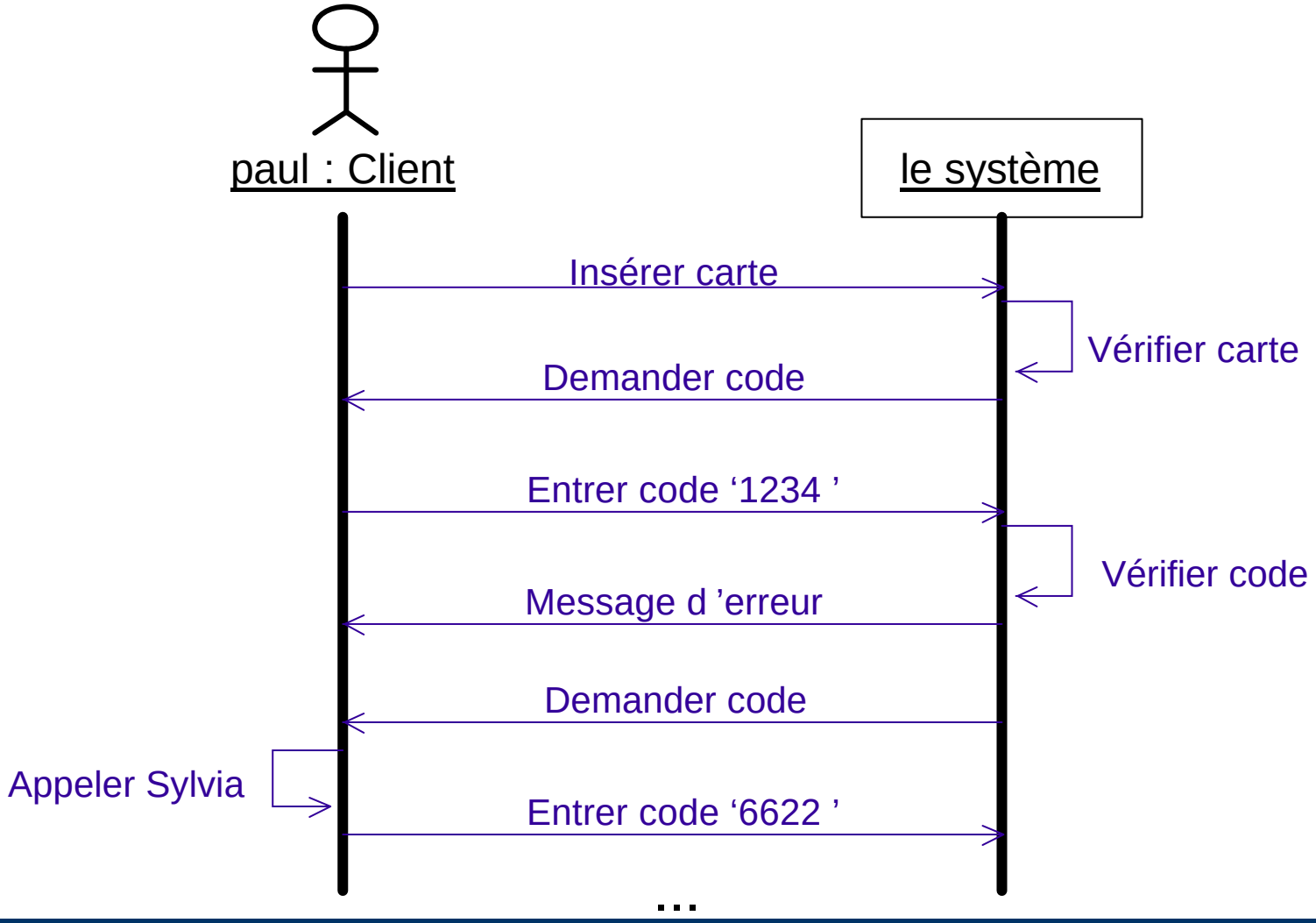
- *Paul* insère sa carte dans le distributeur d2103
- Le *système* accepte la carte et lit le numéro de compte
- Le *système* demande le code
- *Paul* tape ‘ 1234 ’
- Le *système* indique que ce n ’est pas le bon code
- Le *système* affiche un message et propose de recommencer
- *Paul* tape ‘ 6622’
- Le *système* affiche que le code est correct
- Le *système* demande le montant du retrait
- *Paul* tape 50000 fr
- Le *système* vérifie s ’il y a assez d ’argent sur le compte
- ...

Diagrammes de séquences

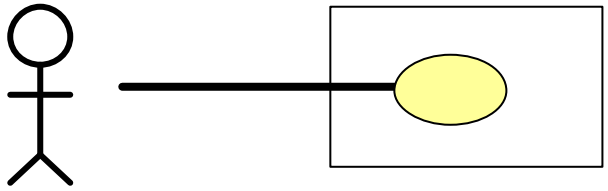
- Pour décrire un scénario : un diagramme de séquence
- Diagramme de séquences :
  - L'une des notations UML, une notation générale
  - Peut être utilisée dans de nombreux contextes
  - Permet de décrire une séquence des messages échangés entre différents objets
  - Différents niveaux de détails
- Pour décrire un scénario simple, deux objets : l'acteur et le système



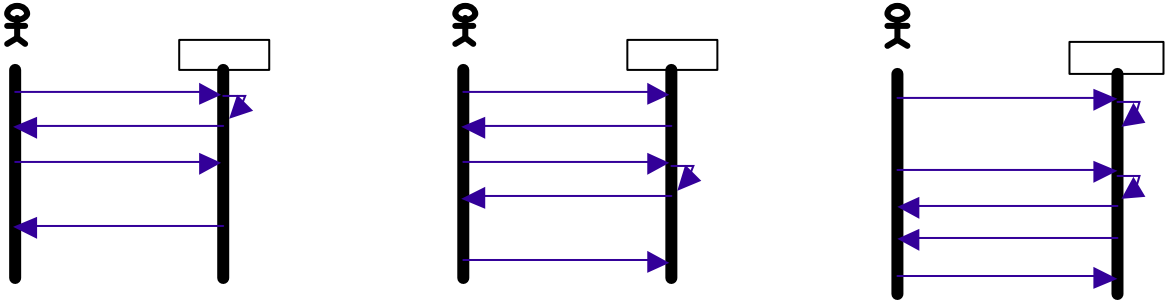
Exemple de scénario



Cas d'utilisation vs. scénarii



Niveau modèle



Niveau instances

Le Processus Unifié

(1) Définir le modèle de cas d'utilisation
- (1.1) Trouver les acteurs
- (1.2) Décrire brièvement chaque acteur
- (1.3) Trouver les cas d 'utilisation
- (1.4) Décrire brièvement chaque cas d 'utilisation
- (1.5) Décrire le modèle comme un tout

(2) Définir des priorités entre CU

(3) Détailler chaque CU (en tenant compte des priorités)

Résumé

- Différents concepts UML
  - Modèle des cas d'utilisation
  - Diagramme des cas d'utilisation
  - Acteur
  - Cas d 'utilisation
  - Scénario
- Processus Unifié : commencer par les acteurs
- Utiliser les schémas mais aussi la langue naturelle!
- Moyen de communication avec le client

UML : Diagrammes de Classes

Concepts de bases

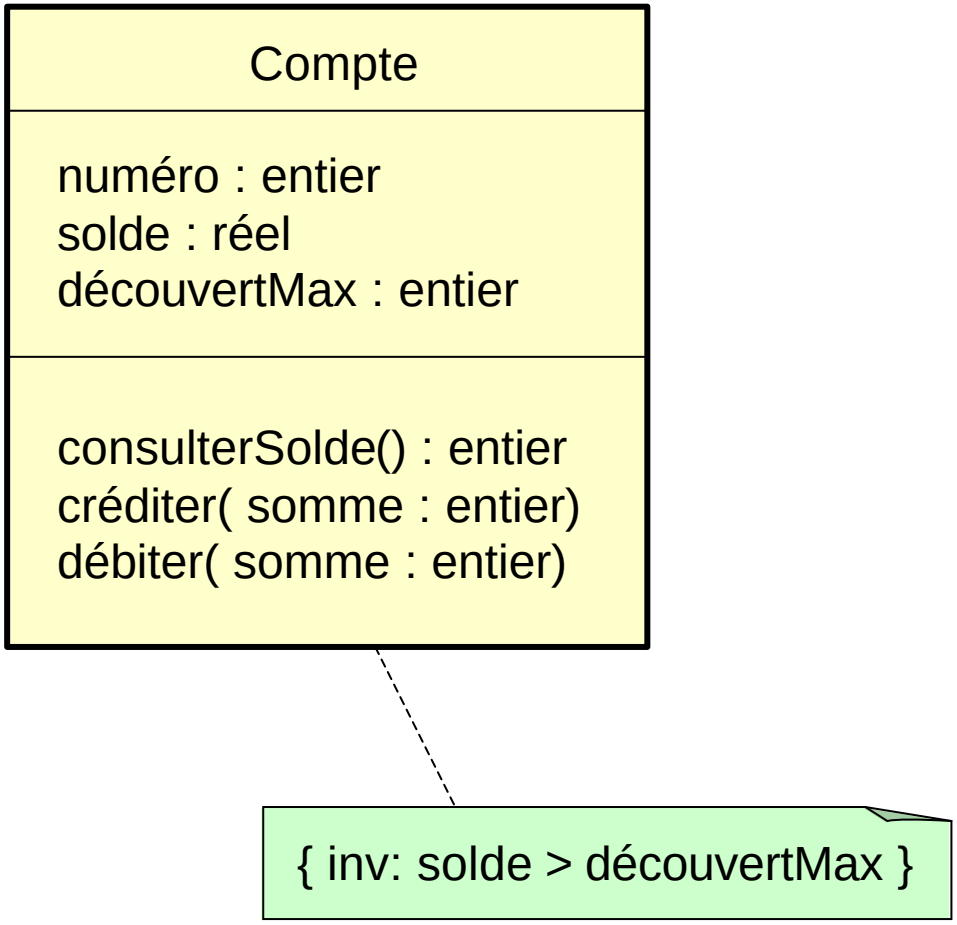
Objets, Classes
Lien, Association, Cardinalité
Généralisation

Jean-Marie Favre, Ioannis Parissis ©

Concepts de base

- UML est basé sur différents concepts de base :
  - Objet, Classe
  - Lien, Association
  - Contrainte
- UML propose des notations et des diagrammes
  - Diagramme de classes (description au niveau modèlisation, cas général)
  - Diagramme d'objets (description au niveau instance, exemples)

Notation pour les classes



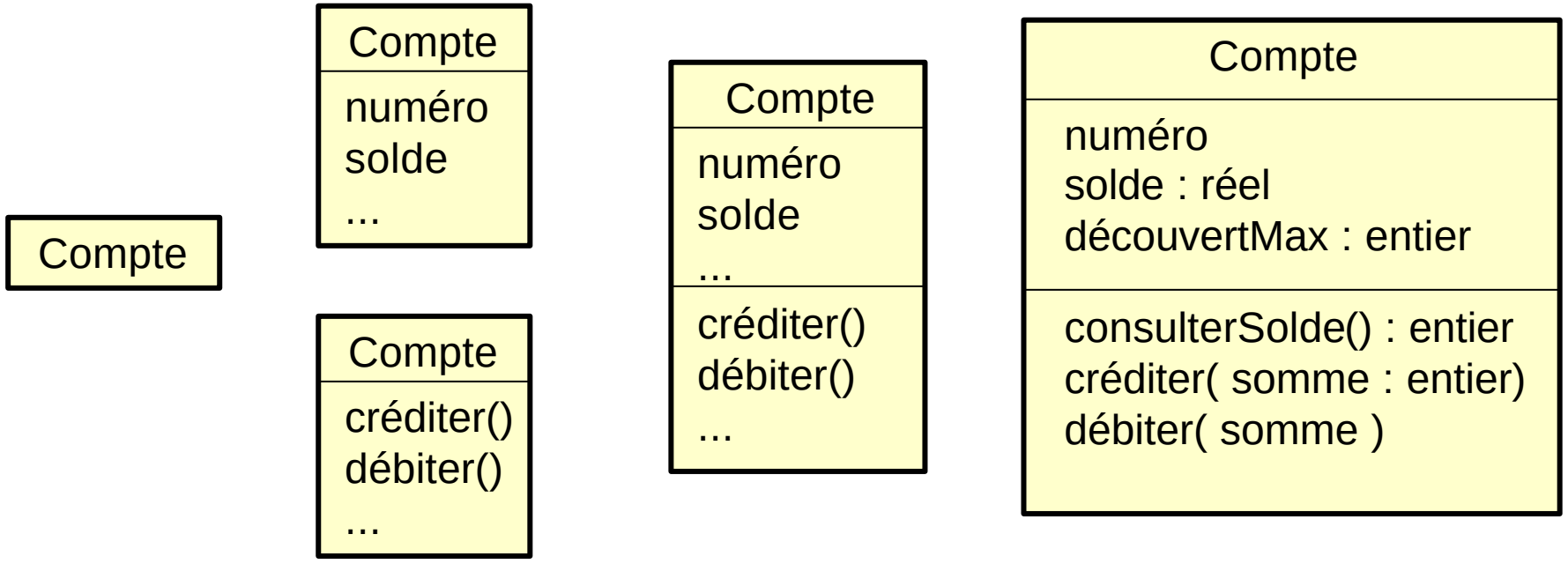
Nom de la classe

Attributs
- nom
- type

Méthodes
- nom
- paramètre
- type du résultat

Contraintes

Notations simplifiées pour les classes



Conventions :

- les noms de classes commencent par une majuscule
- les noms d 'attributs et de méthodes commencent par une minuscule

Notations pour les objets

leCompteDePaul

: Compte

leCompteDePaul : Compte

leCompteDePaul : Compte

numéro = 6688
solde = 5000
découvertMax = -100

Convention :

- les noms d 'objets commencent par une minuscule et sont soulignés

Classe vs. Objets

Une **classe** spécifie la structure et le comportement d'un ensemble d'objets de même nature

- La structure d'une classe est constante

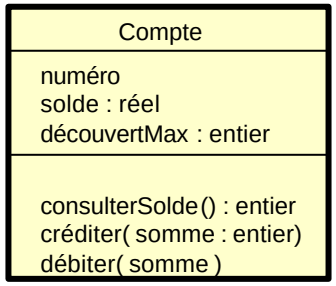


Diagramme de classes (modèlisation)

- Des **objets** peuvent être ajoutés ou détruits pendant l'exécution
- La valeur des attributs des objets peut changer

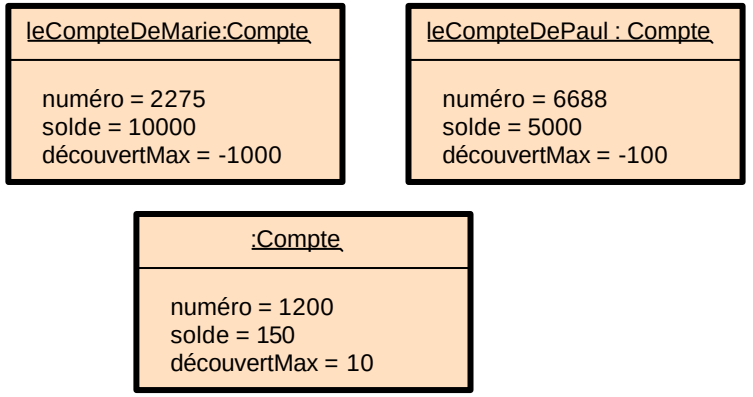
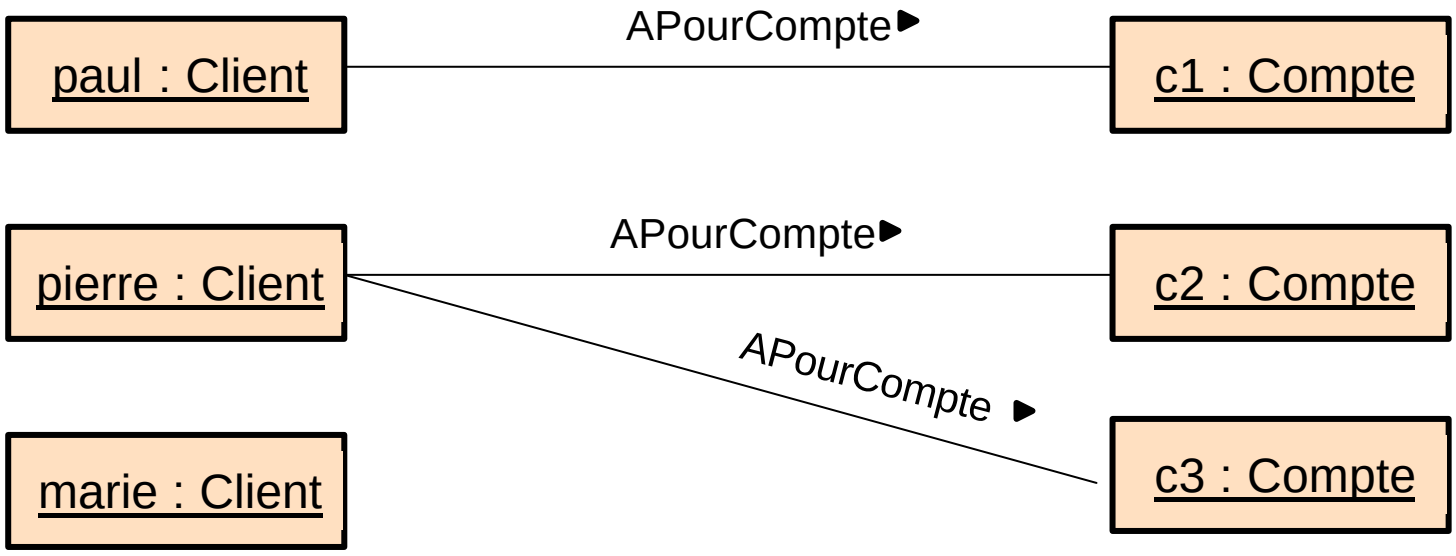


Diagramme d'objets (exemplaires)

Liens (entre objets)

Un **lien** indique une connexion entre deux objets

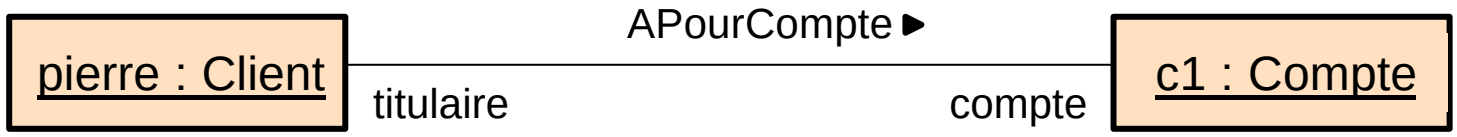


Conventions :

- les noms des liens sont des formes verbales et commencent par une majuscule
- ▸ indique le sens de la lecture (ex: « paul APourCompte c1 »)

Rôles

Chacun des deux objets joue un **rôle** différent dans le lien



pierre assume le rôle de titulaire pour le compte c1
c1 assume le rôle de compte pour pierre

Conventions :

- choisir un groupe nominal pour désigner un rôle
- si un nom de rôle est omis, le nom de la classe fait office de nom

Associations (entre classes)

Une **association** décrit un ensemble de liens de même "sémantique"

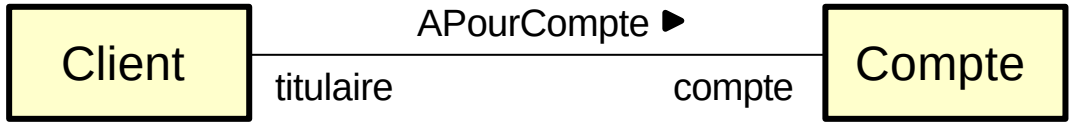


Diagramme de classes (modèlisation)

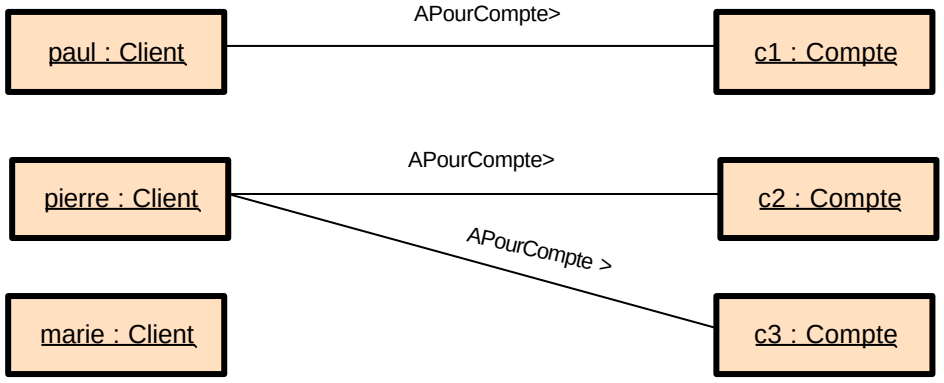
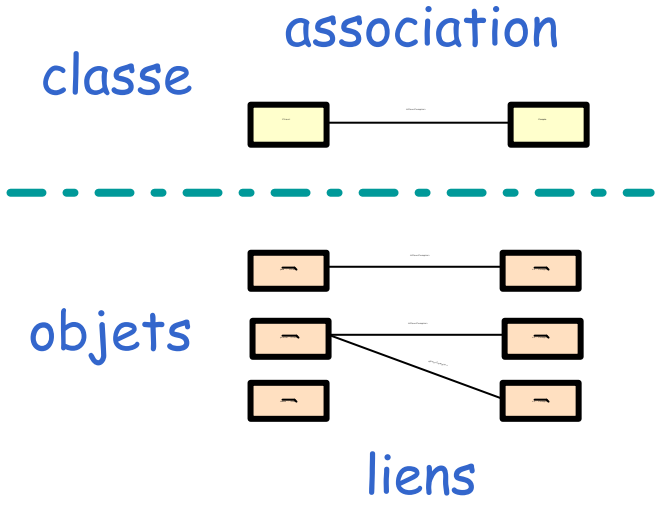


Diagramme d 'objets (exemplaires)

Association vs. Liens



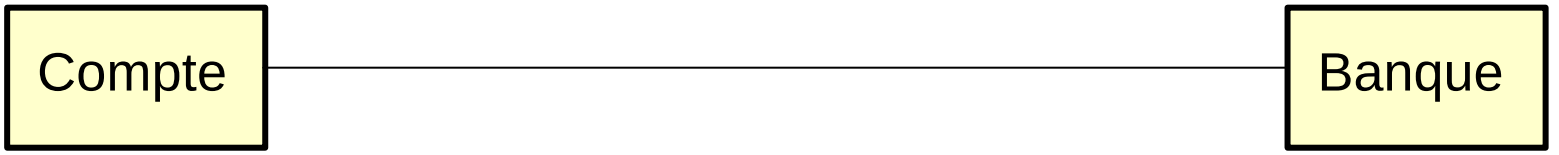
- Un **lien** lie deux **objets**
- Une **association** lie deux **classes**

- Un **lien** est une instance d'**association**
- Une **association** décrit un ensemble de **liens**

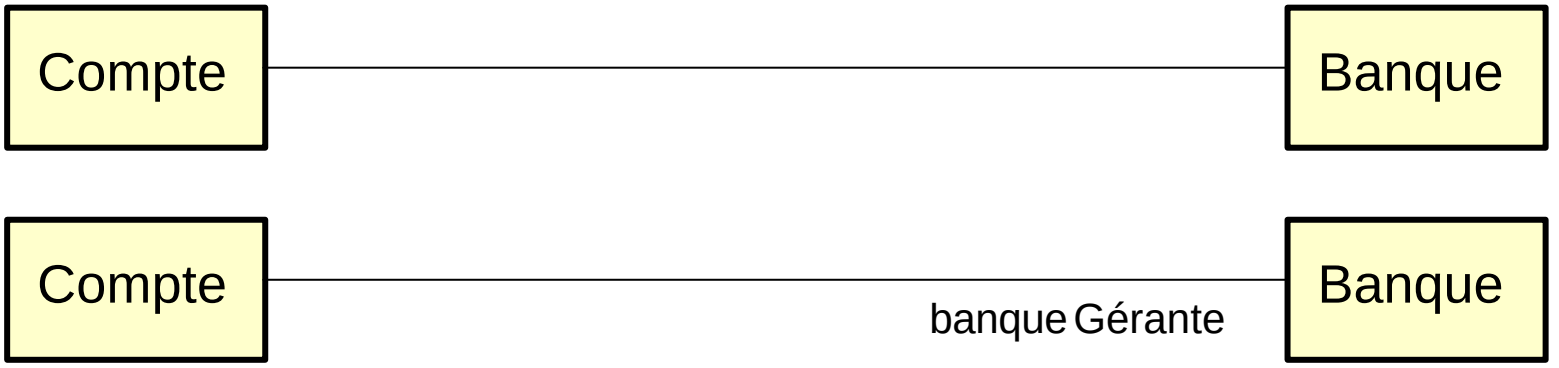
- Des **liens** peuvent être ajoutés ou détruits pendant l 'exécution, (ce n 'est pas le cas des associations)

Le terme "relation" ne pas partie du vocabulaire UML

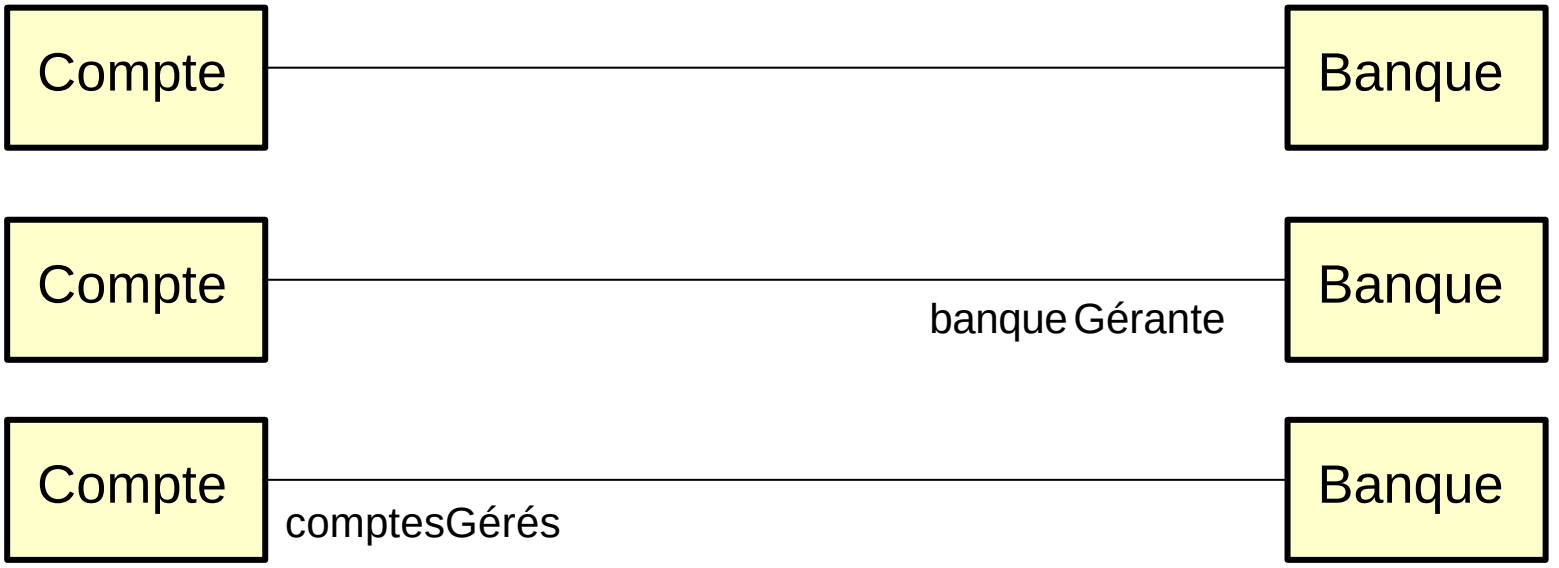
Nommer les associations



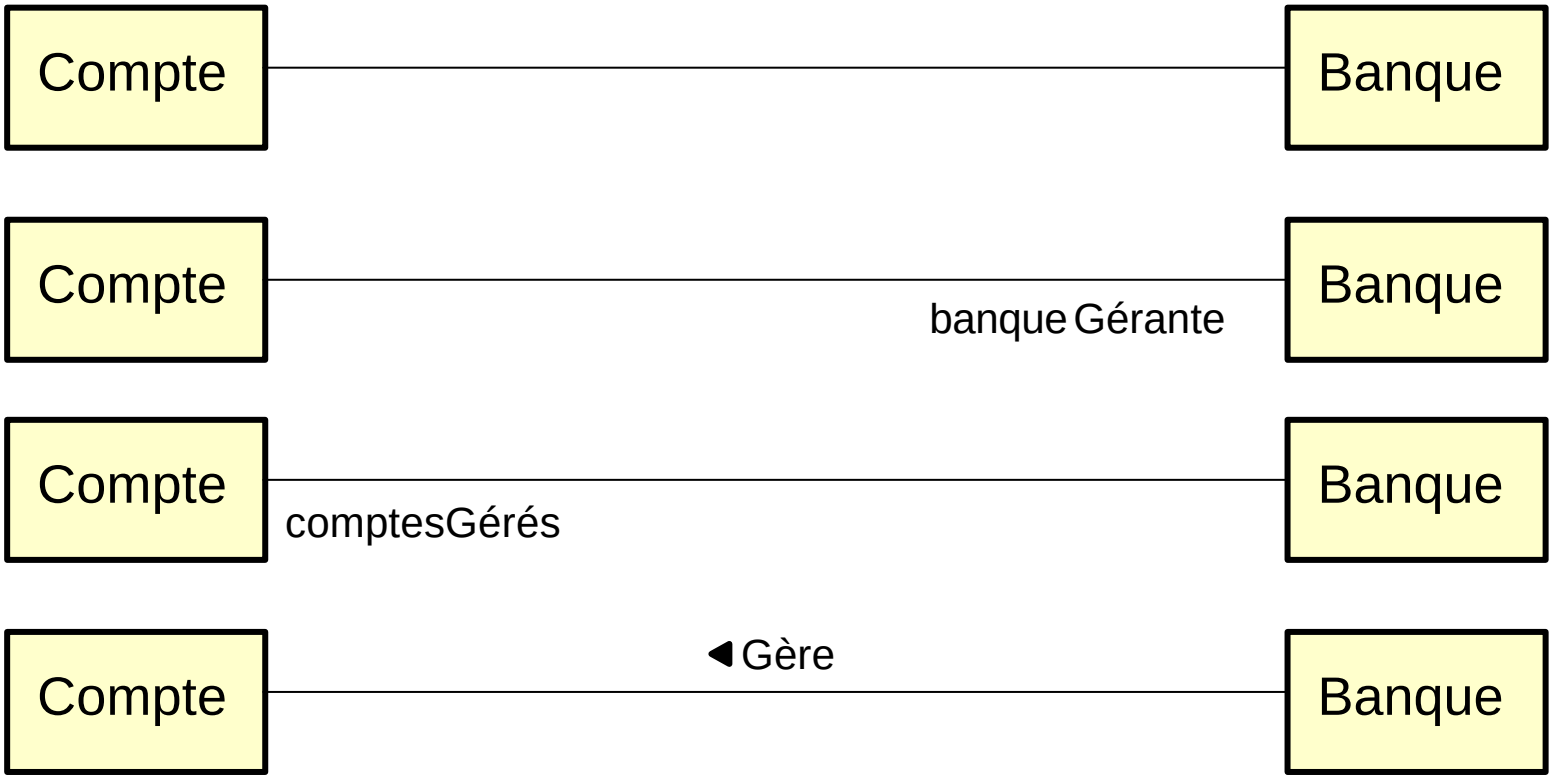
Nommer les associations



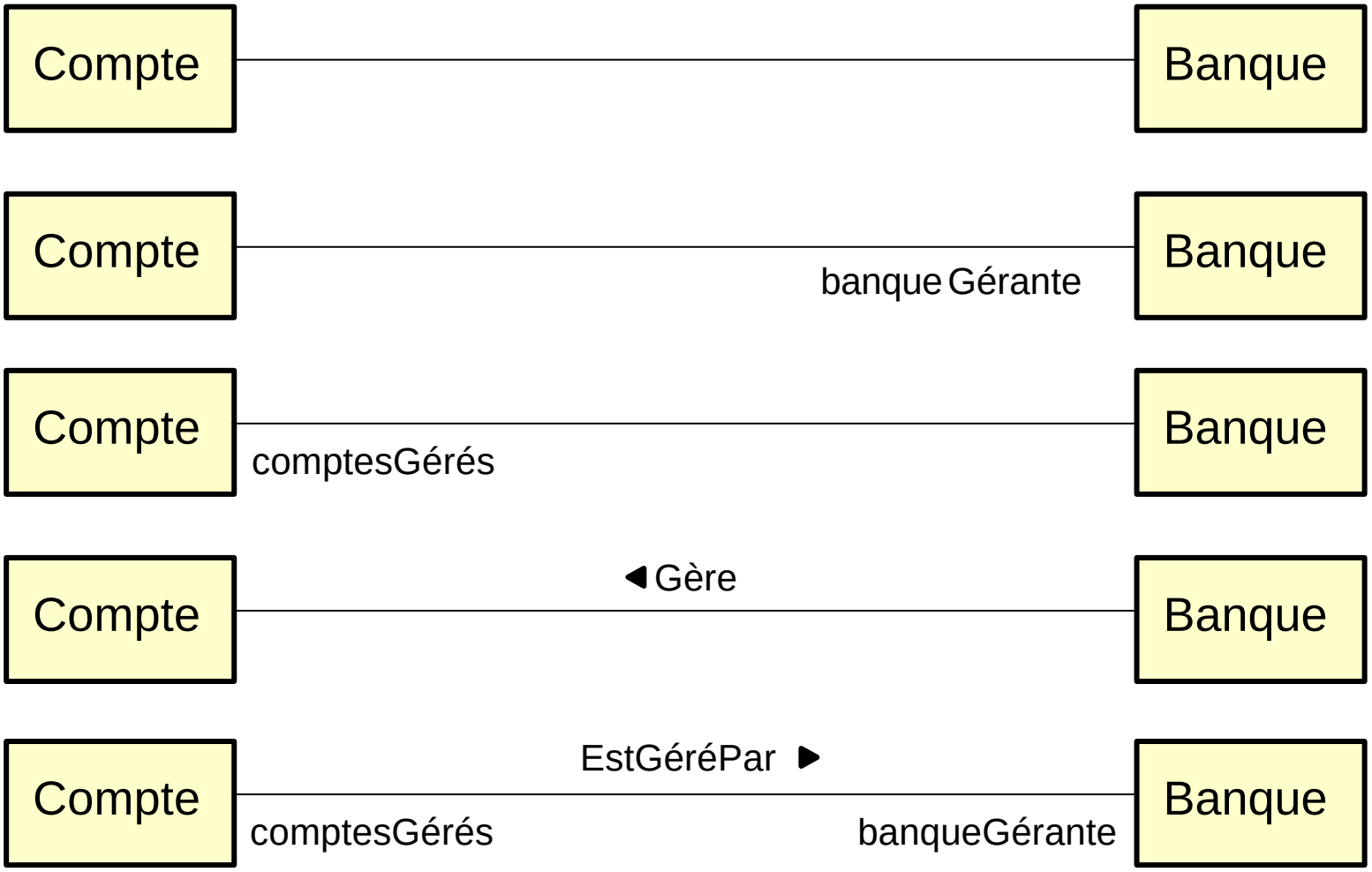
Nommer les associations



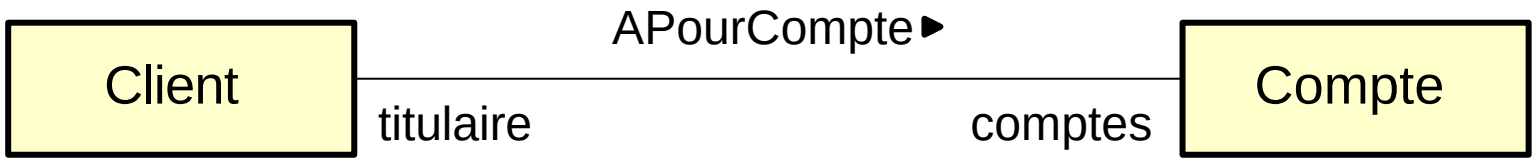
Nommer les associations



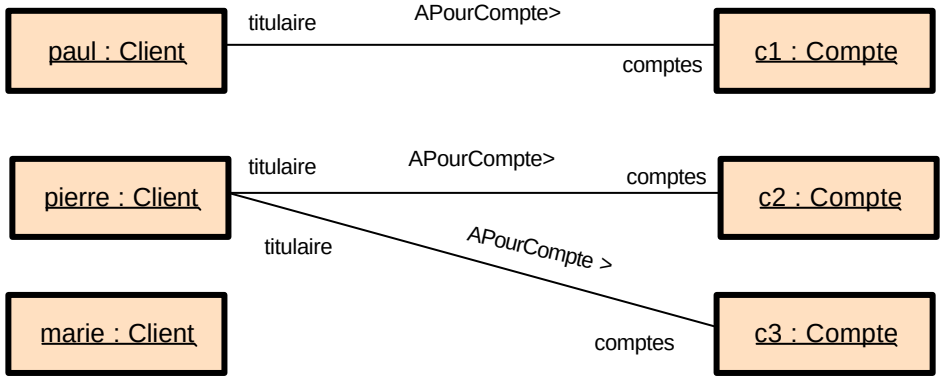
Nommer les associations



Utiliser les rôles pour «naviguer»



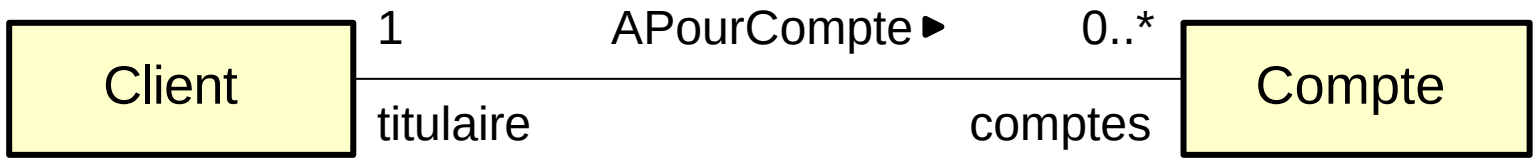
paul.comptes = {c1}
pierre.comptes = {c2,c3}
marie.comptes = { }
c1.titulaire = paul
c2.titulaire = pierre
c3.titulaire = pierre



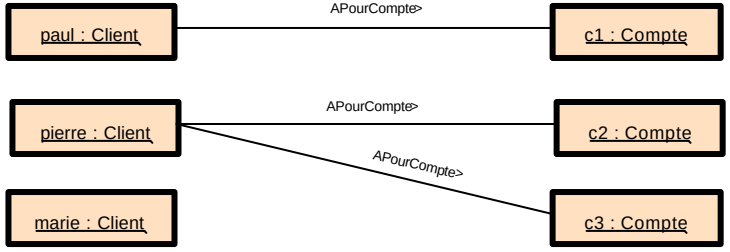
Nommer en priorité les rôles

Cardinalités d'une association

- Précise combien d'objets peuvent être liés à un seul objet source
- Cardinalité minimale et cardinalité maximale ($C_{min}..C_{max}$)

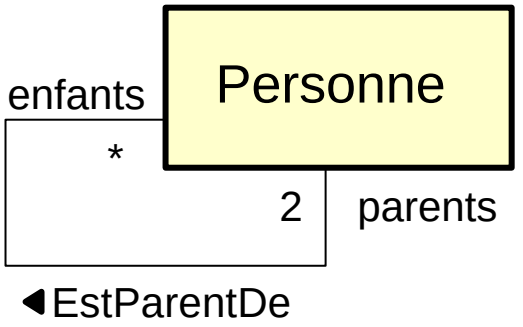


« Un client a 0 ou plusieurs comptes »
« Un compte a toujours 1 et 1 seul titulaire »

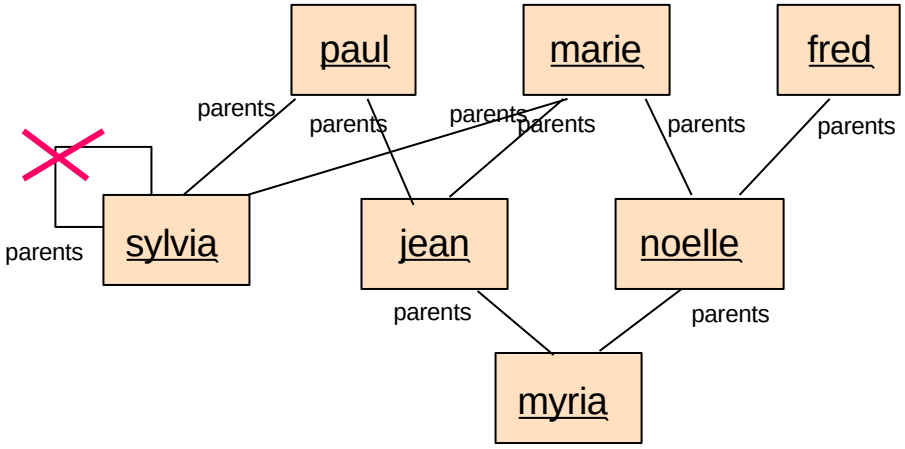


Associations "reflexives"

- Rien n'empêche une association de relier des "objets" d'une même classe

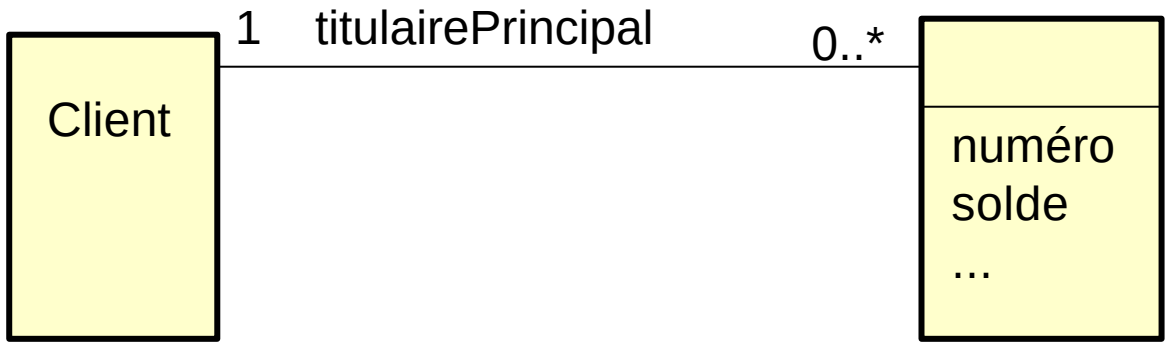


- N'implique pas forcément des liens "reflexifs"

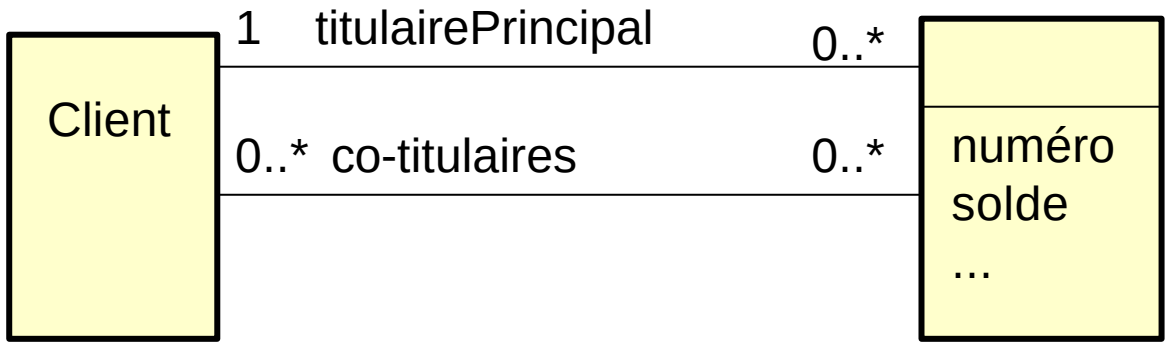


attention: ce diagramme d'objets n'est pas conforme au diagramme de classe

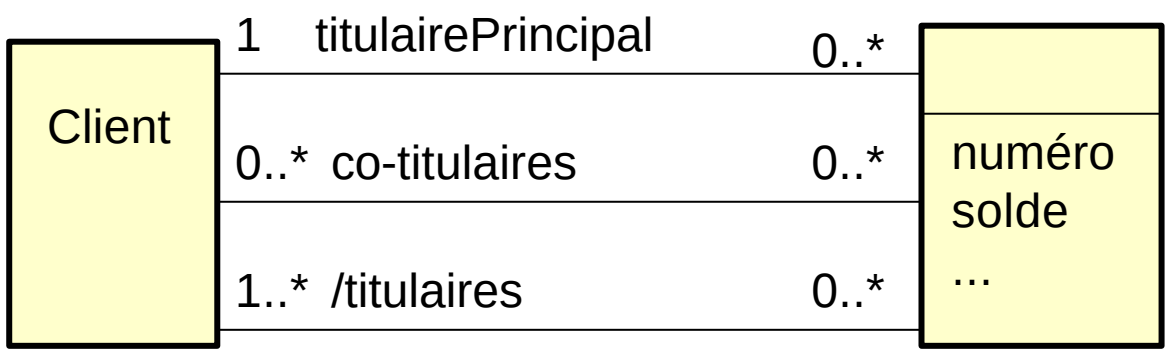
Contraintes entre associations



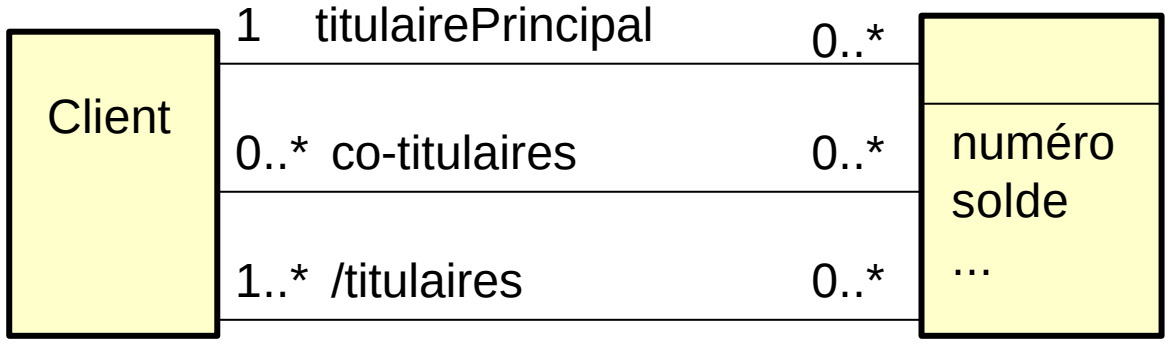
Contraintes entre associations



Contraintes entre associations

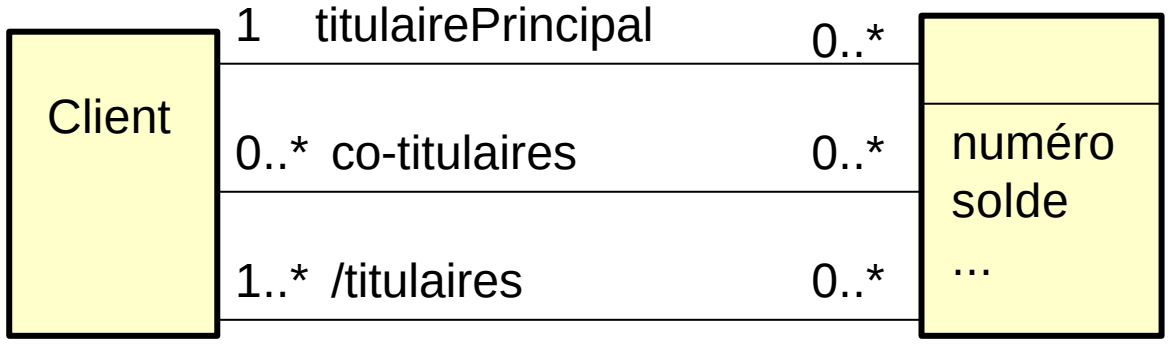


Contraintes entre associations



Les cardinalités sont loins d'être suffisantes pour exprimer toute les contraintes...

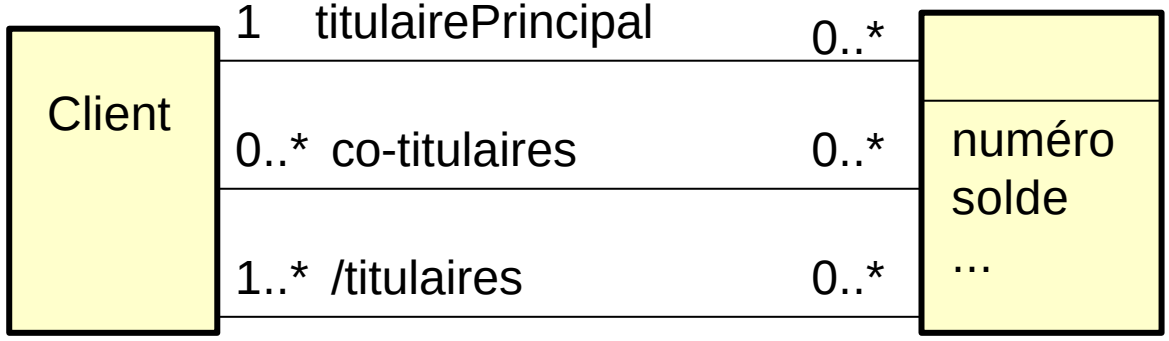
Contraintes entre associations



Les cardinalités sont loins d'être suffisantes pour exprimer toute les contraintes...

... décrire les contraintes en langue naturelle (ou en OCL)

Contraintes entre associations



Les cardinalités sont loins d'être suffisantes pour exprimer toute les contraintes...

... décrire les contraintes en langue naturelle (ou en OCL)

(1) Un client ne peut pas être à la fois titulaire principal et co-titulaire d 'un même compte.
(2) Les titulaires d'un compte sont le titulaire principal et les co-titulaires le cas échéant

Diagramme de classes

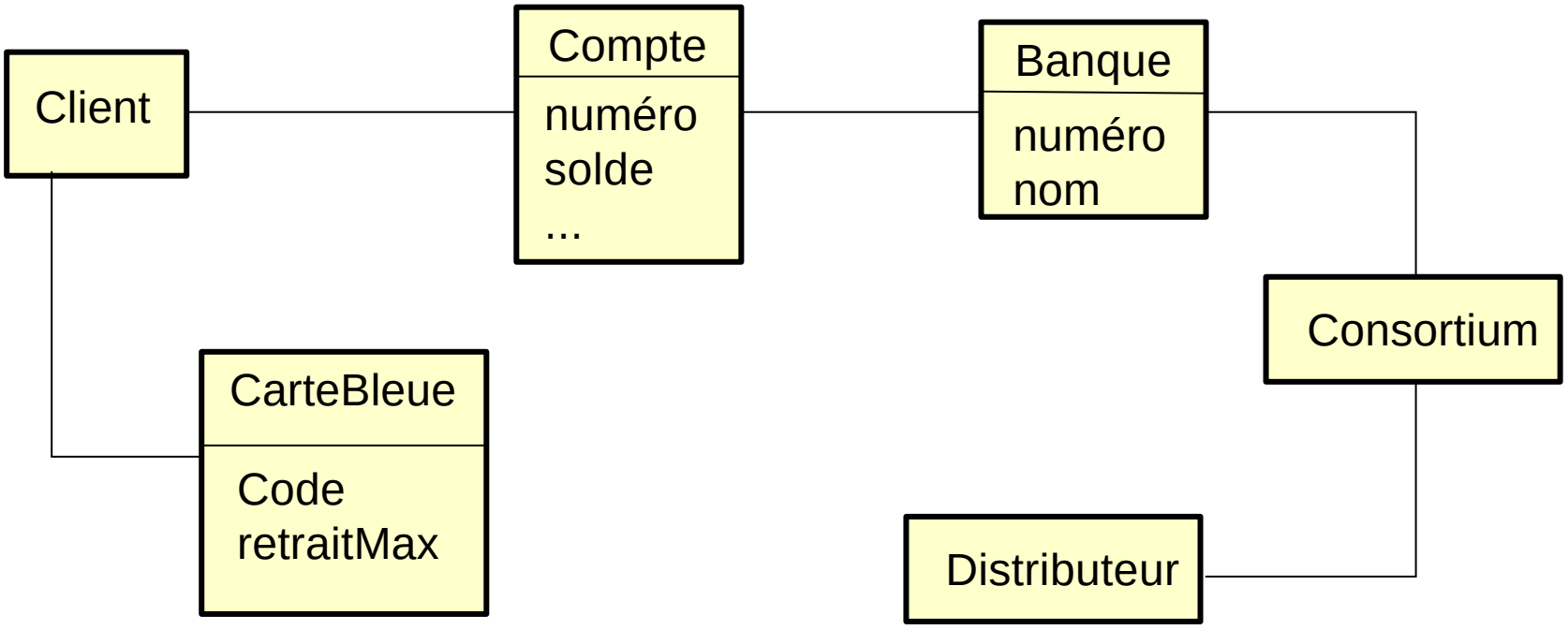


Diagramme de classes

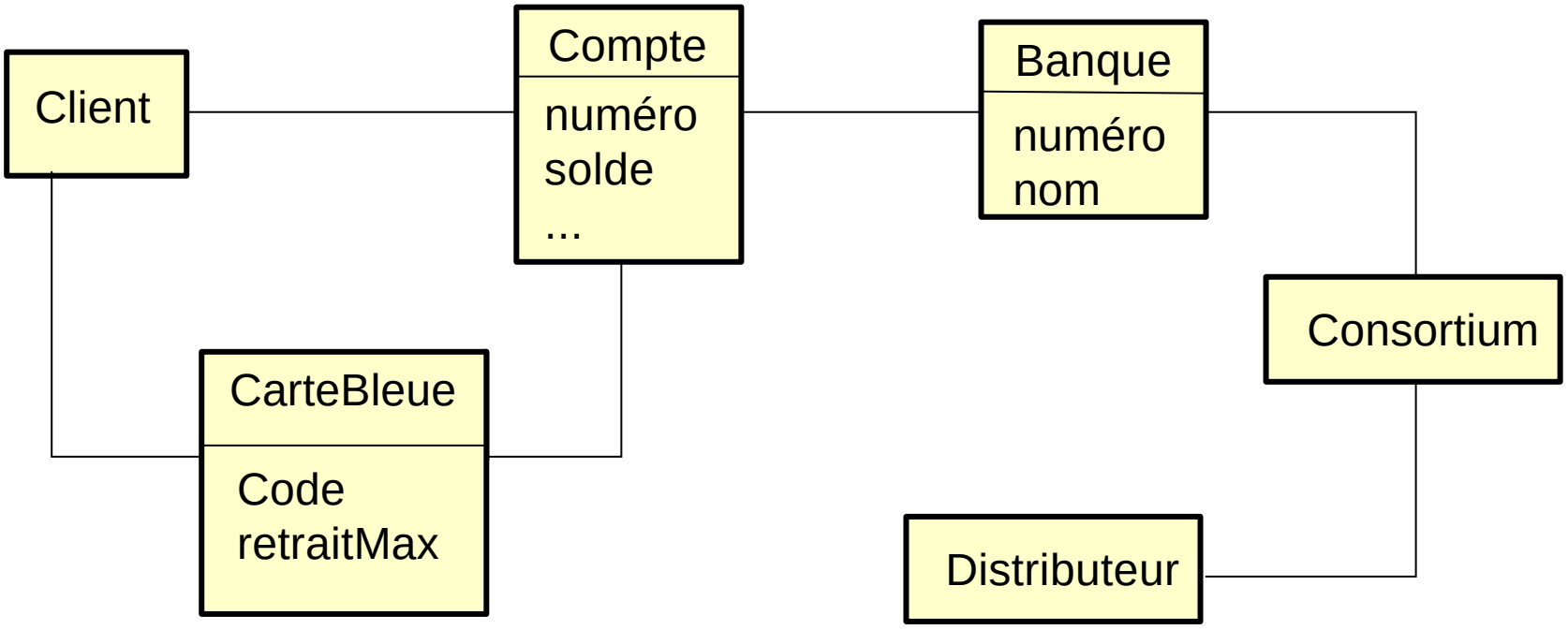
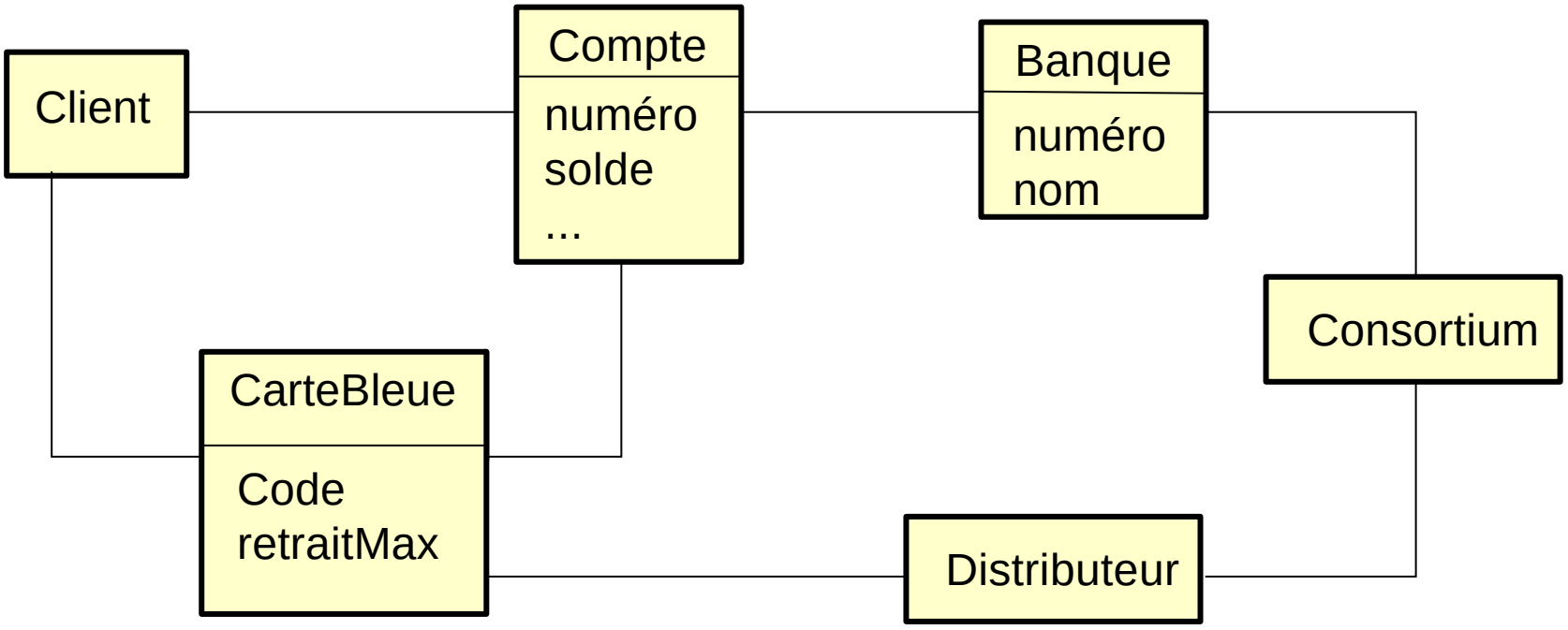
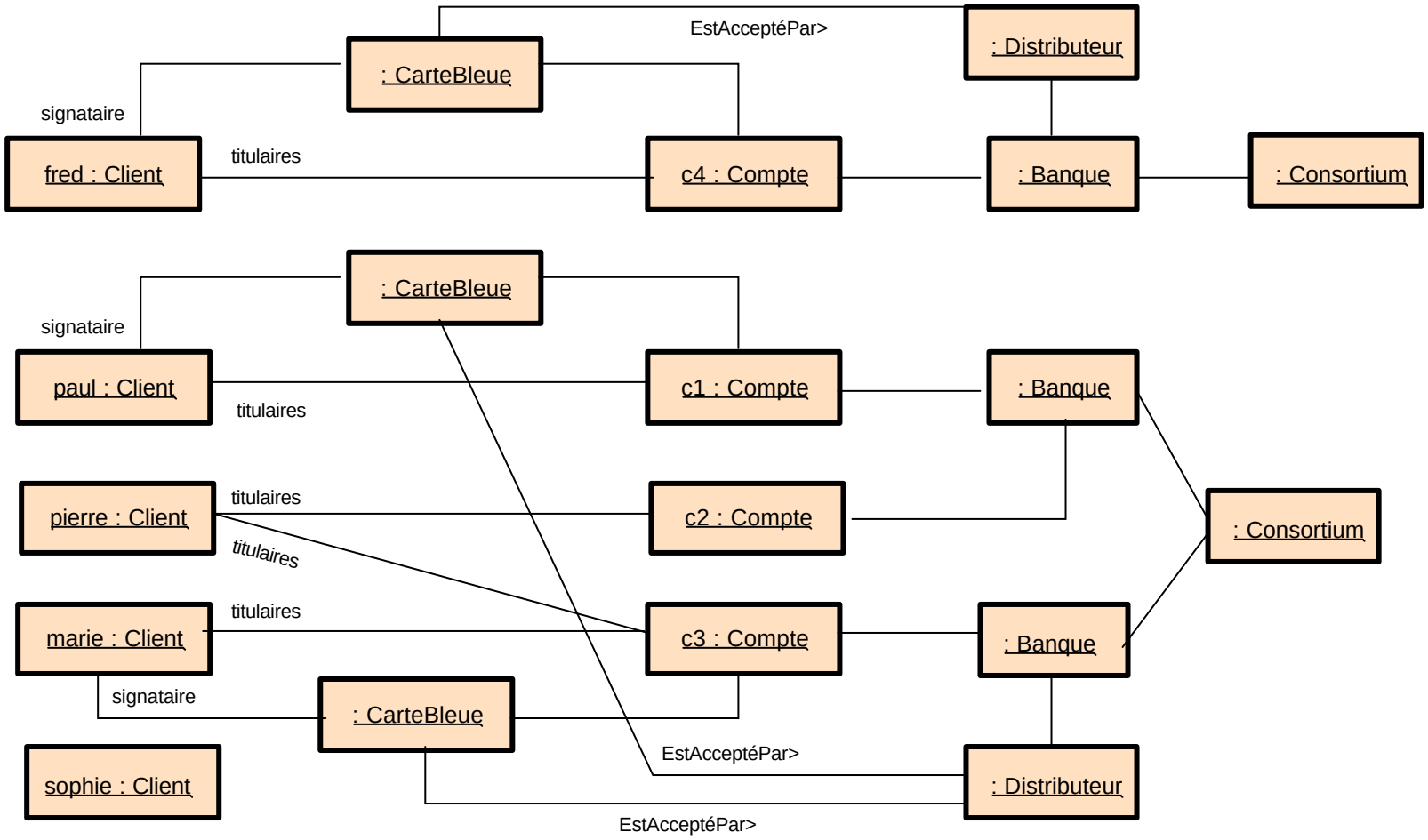


Diagramme de classes

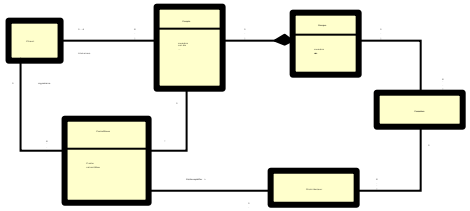


Diagrammes d'objets

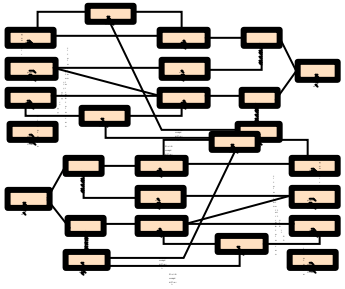


Diagrammes de classes vs. d'objets

- Un diagramme de classes
  - défini l'ensemble de tous les états possibles
  - les contraintes doivent toujours être vérifiées

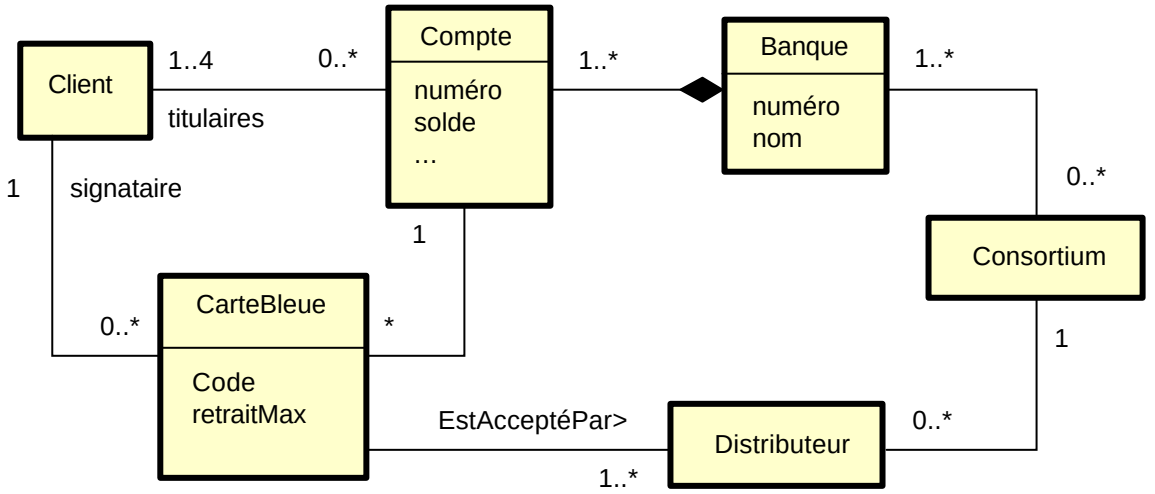


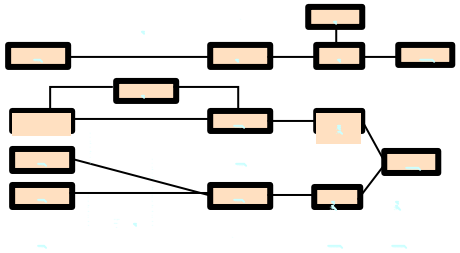
- Un diagramme d'objets
  - décrit un état possible à un instant t, un cas particulier
  - doit être conforme au modèle de classes

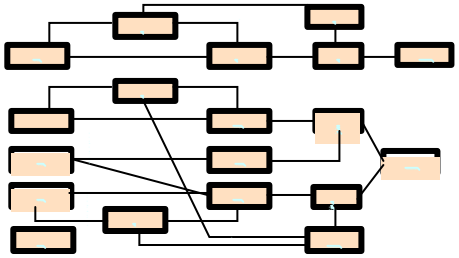


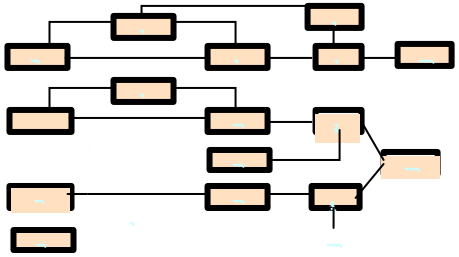
- Les diagrammes d'objets peuvent être utilisés pour
  - expliquer un diagramme de classe (donner un exemple)
  - valider un diagramme de classe (le "tester")

Diagrammes de classes vs. d'objets







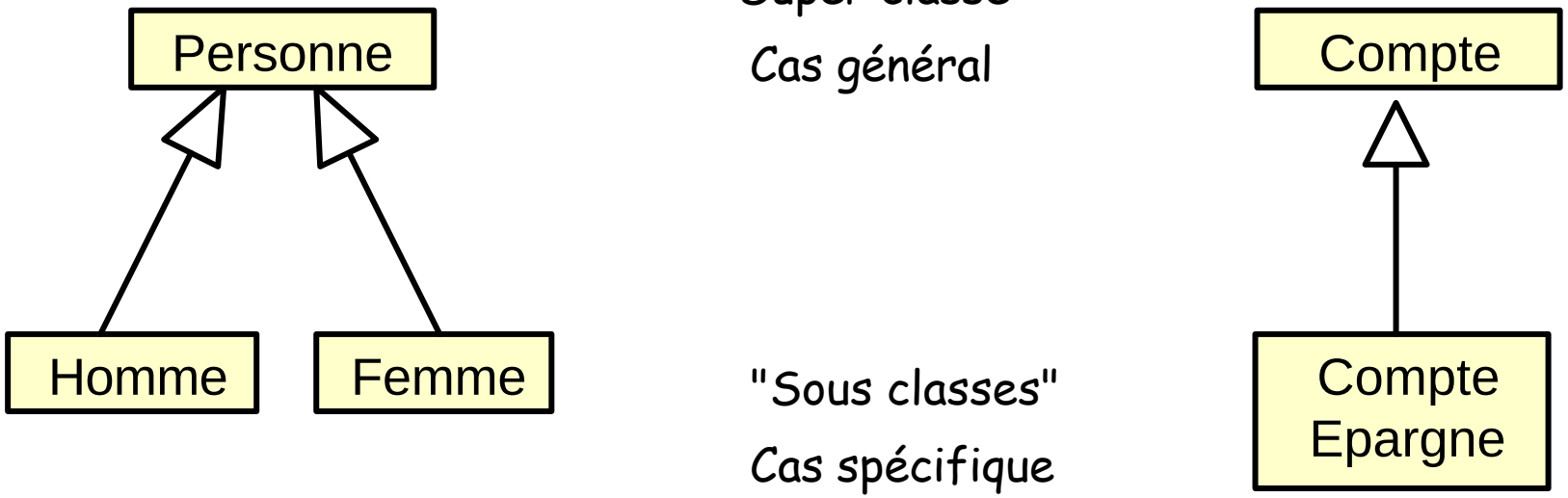


• • •

t_1 t_2 t_3

Généralisation / Spécialisation

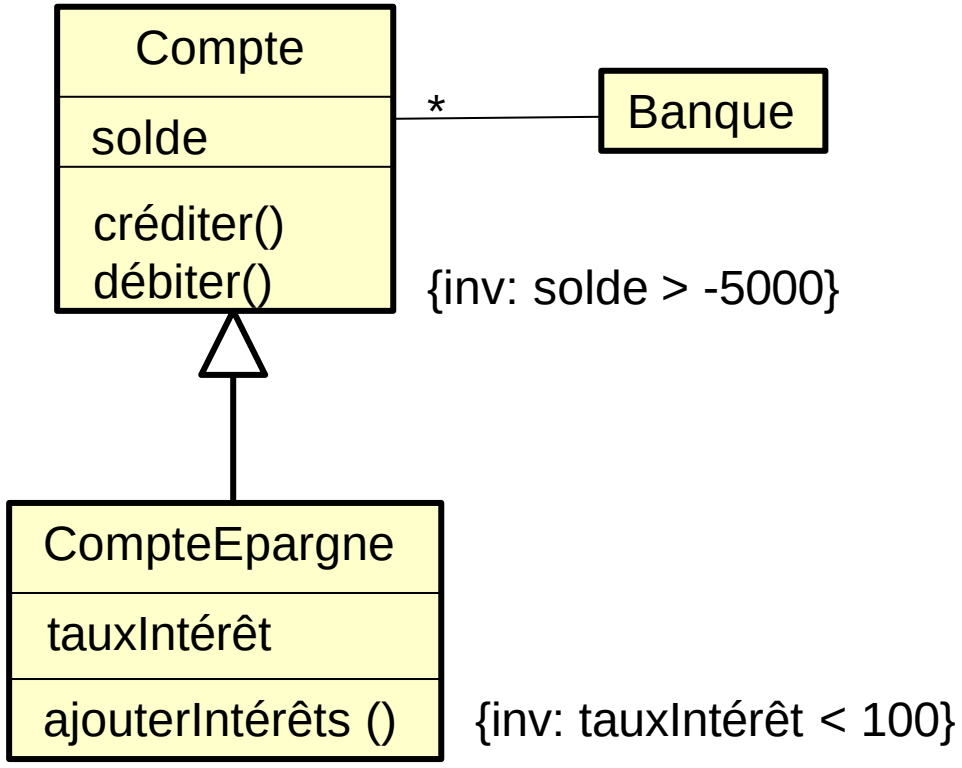
Une classe peut être la généralisation d'une ou plusieurs autres classes.
Ces classes sont alors des spécialisations de cette classe.



Deux points de vue lié (en UML) :
- relation d'héritage
- relation de sous-typage

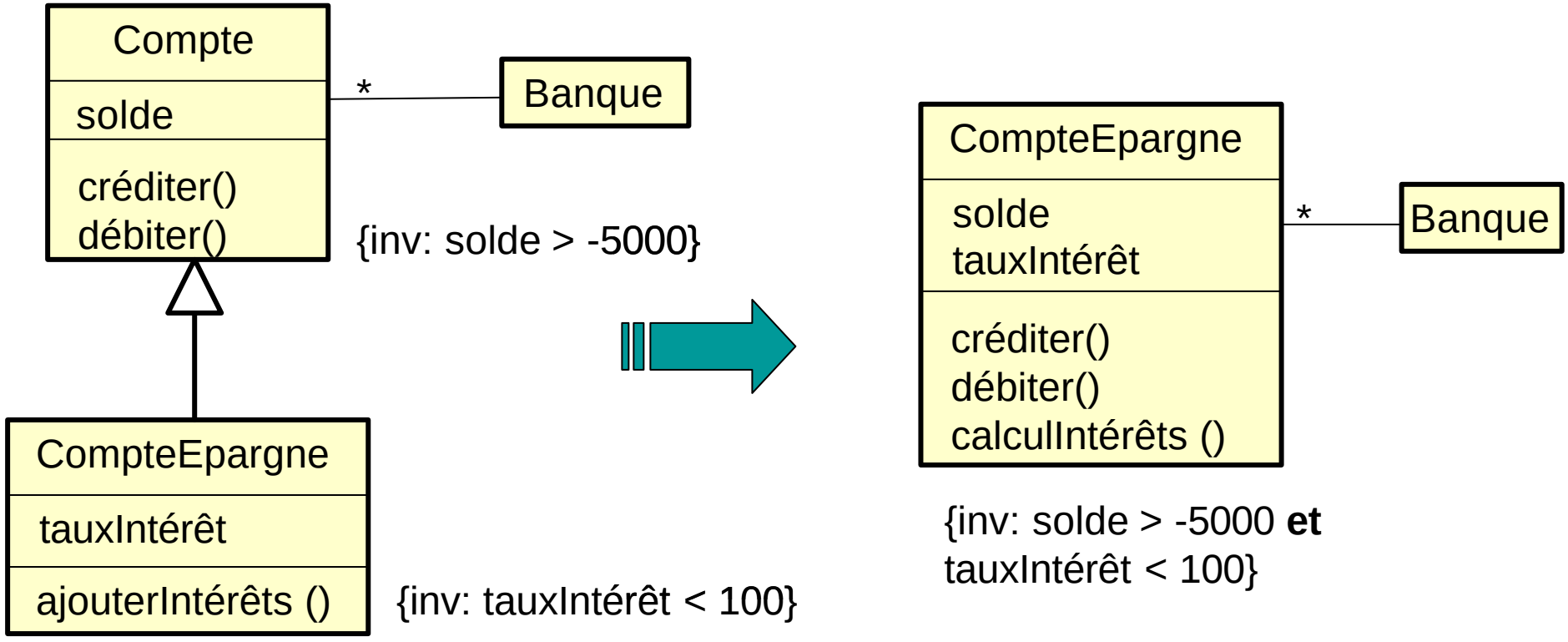
Héritage

Les sous-classes « héritent » des propriétés des super-classes (attributs, méthodes, associations, contraintes)



Héritage

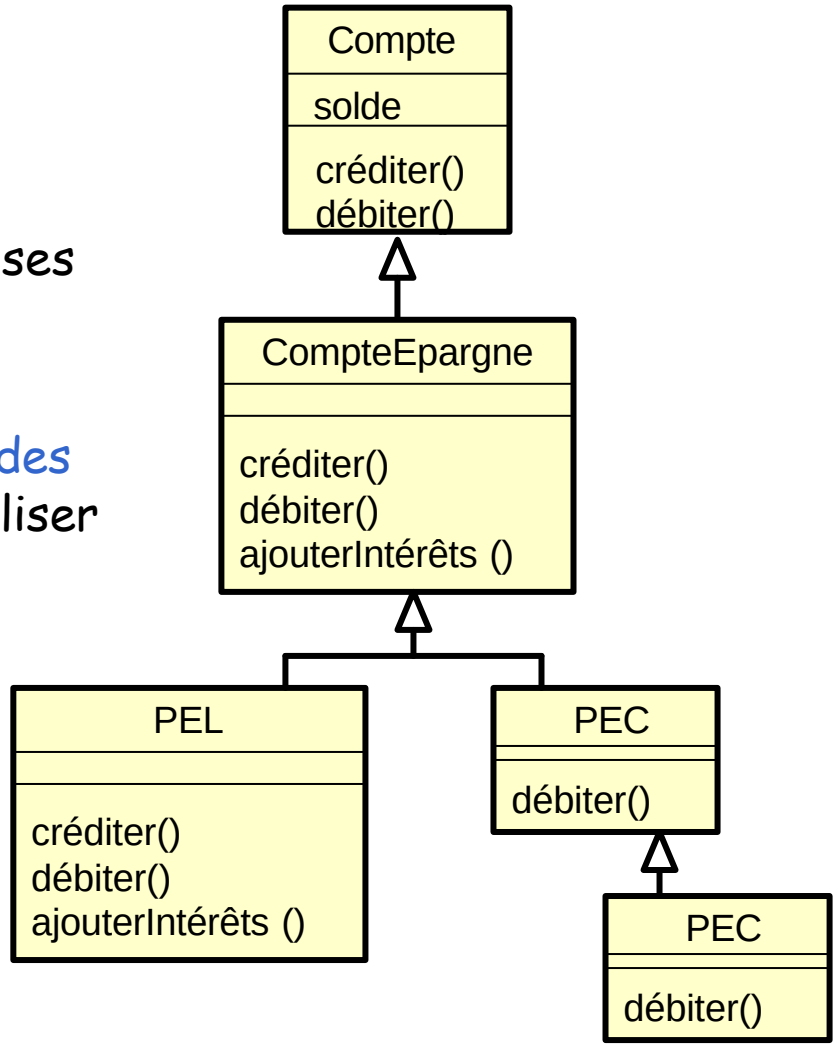
Les sous-classes « héritent » des propriétés des super-classes (attributs, méthodes, associations, contraintes)



Redéfinition d'opérations

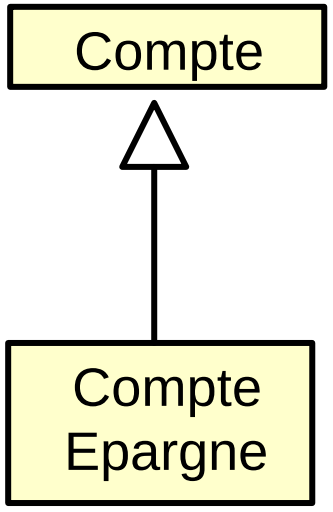
Une opération peut être "redéfinie" dans les sous-classes

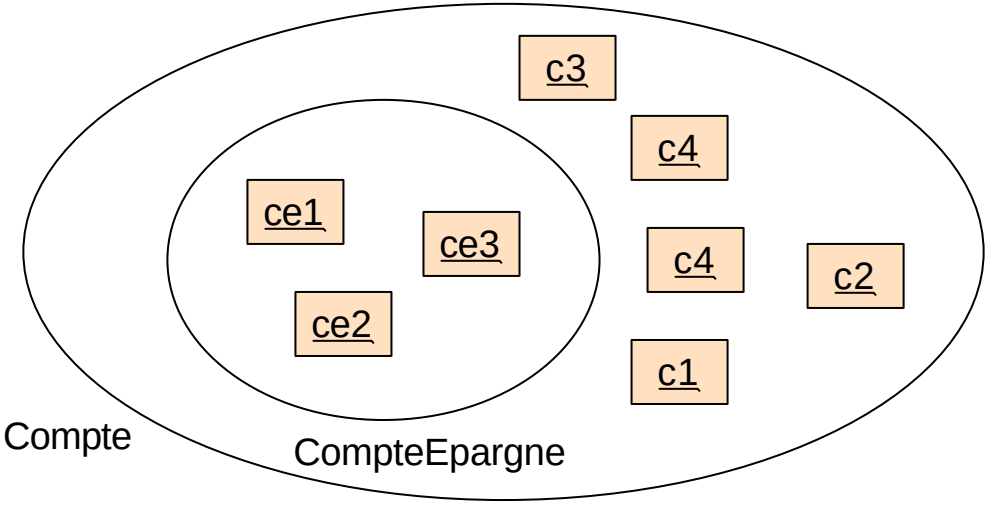
Permet d'associer des méthodes spécifiques à chaque pour réaliser une même opération



Vision ensembliste

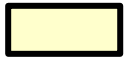
tout objet d'une sous-classe appartient également à la super-classe
=> inclusion des ensembles d'objets



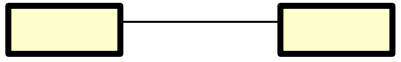


Synthèse des concepts de base

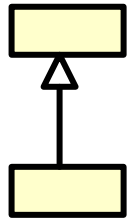
- Classe
  - attribut
  - méthode

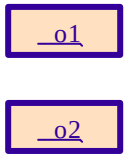


- Association
  - rôle
  - cardinalité

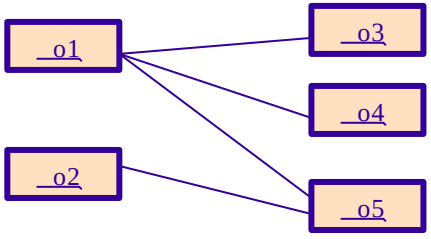


- Héritage

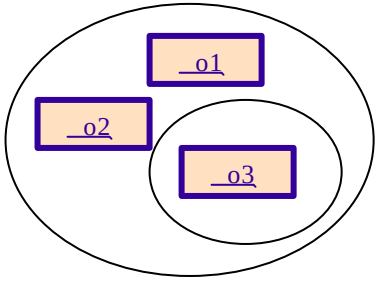




- Objet



- Lien



- Inclusion ensembliste

UML : Diagrammes de Classes

Concepts avancés

Visibilité, Déclaration d'attributs et d'opérations
Navigation, Composition, Aggrégation, Classe associative, Association qualifiée…
Classe abstraite, Héritage multiple, Classification multiple, dynamique

Concepts à utiliser à bon escient, lorsque nécessaire
et uniquement lorsque nécessaire…

Jean-Marie Favre, Ioannis Parissis ©

Rappels

- UML est un ensemble de notations

- De nombreuses utilisations possibles
- Notation simple vs. complexe
- Notation plus ou moins formelle
- Notation extensible

Savoir s'adapter

S'adapter

- au niveau d'abstraction
- au domaine d'application
- à ses collègues
- aux outils utilisés
  - pour écrire des diagrammes UML,
  - pour générer du code, des schémas de bases de données, etc.

  ...

Savoir lire vs. savoir écrire

Ingénierie vs. Retro-ingénierie

+ # - Visibilité des éléments

- Restreindre l'accés aux éléments d'un modèle
- Contrôler et éviter les dépendances entre classes et paquetages

+	public	visible
#	protégé	visible dans la classe et ses sous-classes
-	privé	visible dans la classe uniquement
~	package	visible dans la package uniquement

- Utile lors de la conception et de l'implémentation, pas avant !
- N'a pas de sens dans un modèle conceptuel (abstrait)
- N'utiliser que lorsque nécessaire
- La sémantique exacte dépend du langage de programmation !

Déclaration d'attributs

[visibilité] nom [: type] [card ordre] [= valeur-initiale] [{ props... }]

exemples:

Déclaration d'attributs

[visibilité] nom [: type] [card ordre] [= valeur-initiale] [{ props... }]

exemples:

age

Déclaration d'attributs

[visibilité] nom [: type] [card ordre] [= valeur-initiale] [{ props... }]

exemples:

age
+age

Déclaration d'attributs

[visibilité] nom [: type] [card ordre] [= valeur-initiale] [{ props... }]

exemples:

age
+age
- solde : Integer = 0

Déclaration d'attributs

[visibilité] nom [: type] [card ordre] [= valeur-initiale] [{ props... }]

exemples:

age
+age
- solde : Integer = 0
age [0..1] : Integer

Déclaration d'attributs

[visibilité] nom [: type] [card ordre] [= valeur-initiale] [{ props... }]

exemples:

age
+age
- solde : Integer = 0
age [0..1] : Integer
numsecu : Integer {frozen}

Déclaration d'attributs

[visibilité] nom [: type] [card ordre] [= valeur-initiale] [{ props... }]

exemples:

age
+age
- solde : Integer = 0
age [0..1] : Integer
numsecu : Integer {frozen}
motsClés : String [*] {addOnly}

Déclaration d'attributs

[visibilité] nom [: type] [card ordre] [= valeur-initiale] [{ props... }]

exemples:

```
age
+age
- solde : Integer = 0
# age [0..1] : Integer
# numsecu : Integer {frozen}
# motsClés  : String [*] {addOnly}
```

- Adapter le niveau de détail au niveau d'abstraction

Déclaration d'opérations

[visibilité] nom [(params)] [: type] [{ props... }]
params := [in | out| inout] nom [: type] [=defaut] [{ props... }]

Déclaration d'opérations

[visibilité] nom [(params)] [: type] [{ props... }]
params := [in | out| inout] nom [: type] [=defaut] [{ props... }]

getAge()

Déclaration d'opérations

[visibilité] nom [(params)] [: type] [{ props... }]
params := [in | out| inout] nom [: type] [=defaut] [{ props... }]

getAge()
+ getAge() : Integer

Déclaration d'opérations

[visibilité] nom [(params)] [: type] [{ props... }]
params := [in | out| inout] nom [: type] [=defaut] [{ props... }]

getAge()
+ getAge() : Integer
- updateAge(in date : Date) : Boolean

Déclaration d'opérations

[visibilité] nom [(params)] [: type] [{ props... }]
params := [in | out| inout] nom [: type] [=defaut] [{ props... }]

getAge()
+ getAge() : Integer
- updateAge(in date : Date) : Boolean
getName() : String [0..1]

Déclaration d'opérations

[visibilité] nom [(params)] [: type] [{ props... }]
params := [in | out| inout] nom [: type] [=defaut] [{ props... }]

getAge()
+ getAge() : Integer
- updateAge(in date : Date) : Boolean
getName() : String [0..1]
+ getAge() : Integer {isQuery}

Déclaration d'opérations

[visibilité] nom [(params)] [: type] [{ props... }]
params := [in | out| inout] nom [: type] [=defaut] [{ props... }]

getAge()
+ getAge() : Integer
- updateAge(in date : Date) : Boolean
getName() : String [0..1]
+ getAge() : Integer {isQuery}
+addProject() : { concurrency = sequential }

Déclaration d'opérations

[visibilité] nom [(params)] [: type] [{ props... }]
params := [in | out| inout] nom [: type] [=defaut] [{ props... }]

getAge()
+ getAge() : Integer
- updateAge(in date : Date) : Boolean
getName() : String [0..1]
+ getAge() : Integer {isQuery}
+addProject() : { concurrency = sequential }
+addProject() : { concurrency = concurrent }

Déclaration d'opérations

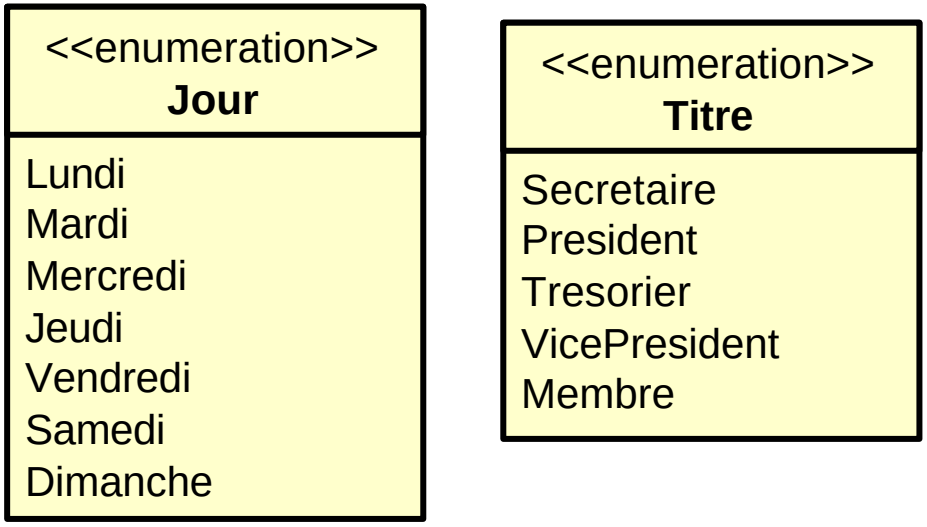
[visibilité] nom [(params)] [: type] [{ props... }]
params := [in | out| inout] nom [: type] [=defaut] [{ props... }]

getAge()
+ getAge() : Integer
- updateAge(in date : Date) : Boolean
getName() : String [0..1]
+ getAge() : Integer {isQuery}
+addProject() : { concurrency = sequential }
+addProject() : { concurrency = concurrent }

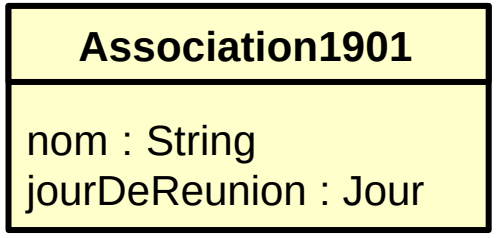
- Adapter le niveau de détail au niveau d'abstraction

Enumération

- Définition

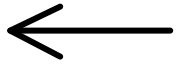


- Utilisation



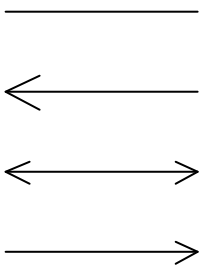
Raffinement du concept d'association

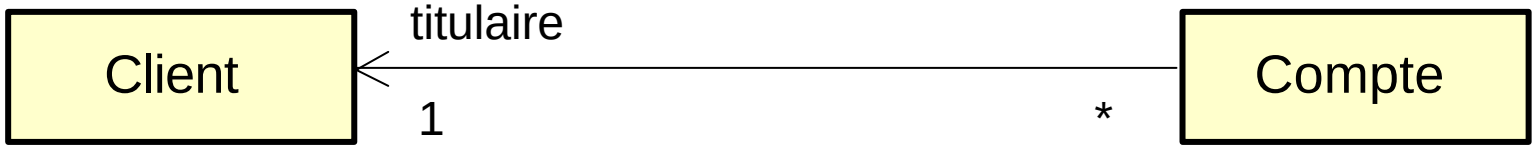
- Association uni/bi directionnelle
- Composition
- Aggrégation
- Classes associatives
- Associations qualifiées
- {frozen}, {addonly}, {ordered}



Navigation

Association unidirectionnelle
On ne peut naviguer que dans un sens

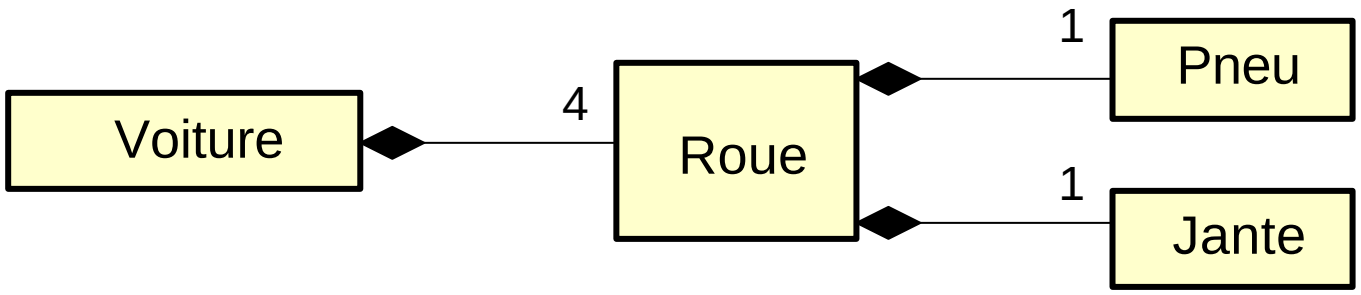




A priori utile que lors de la conception et de l'implémentation
En cas de doute, ne pas mettre de flêche !!!

Composition

Notion intuitive de "composants" et de "composites"

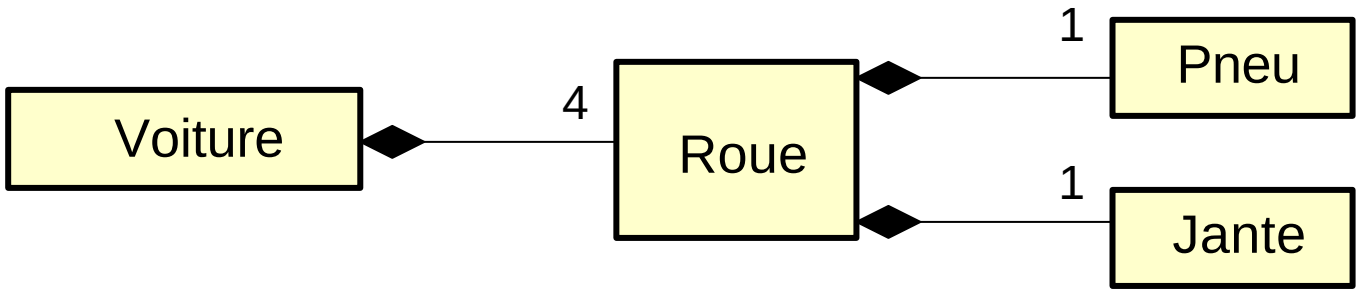


composition =
cas particulier d'association
+ contraintes décrivant la notion de "composant"...

Composition

Contraintes liées à la composition :

1. Un objet composant ne peut être que dans 1 seul objet composite
2. Un objet composant n'existe pas sans son objet composite
3. Si un objet composite est détruit, ses composants aussi

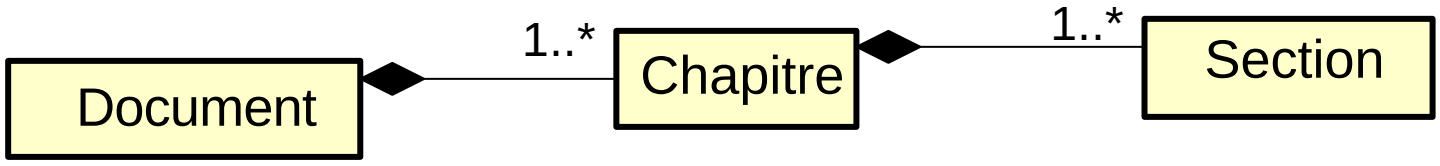


Dépend de la situation modélisée !
(Ex: vente de voitures vs. casse)

Composition

Contraintes liées à la composition :

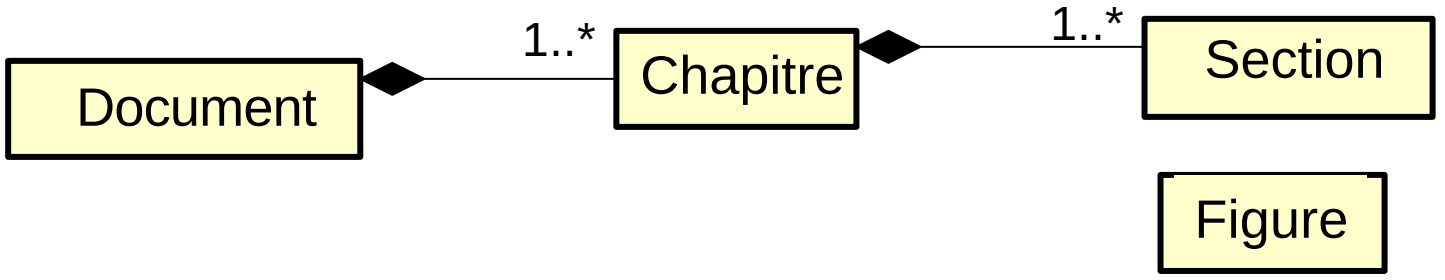
1. Un objet composant ne peut être que dans 1 seul objet composite
2. Un objet composant n'existe pas sans son objet composite
3. Si un objet composite est détruit, ses composants aussi



Composition

Contraintes liées à la composition :

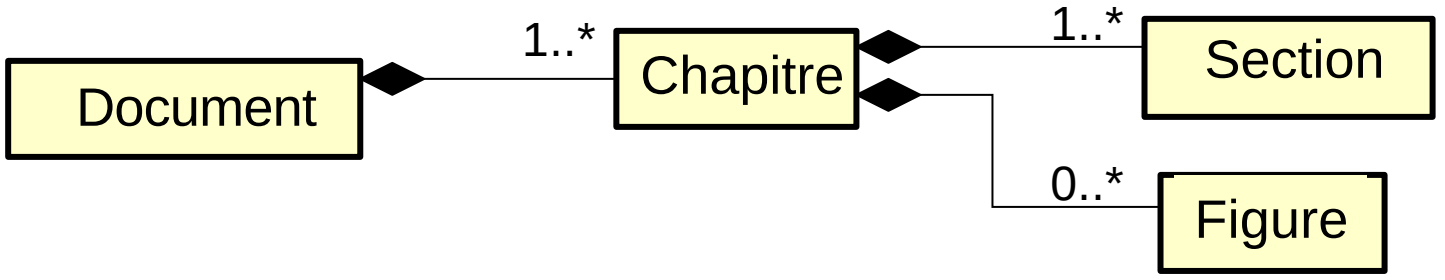
1. Un objet composant ne peut être que dans 1 seul objet composite
2. Un objet composant n'existe pas sans son objet composite
3. Si un objet composite est détruit, ses composants aussi



Composition

Contraintes liées à la composition :

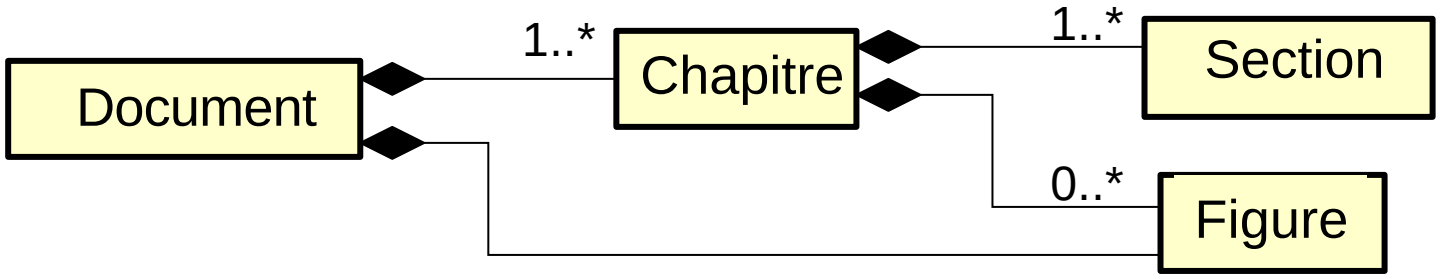
1. Un objet composant ne peut être que dans 1 seul objet composite
2. Un objet composant n'existe pas sans son objet composite
3. Si un objet composite est détruit, ses composants aussi



Composition

Contraintes liées à la composition :

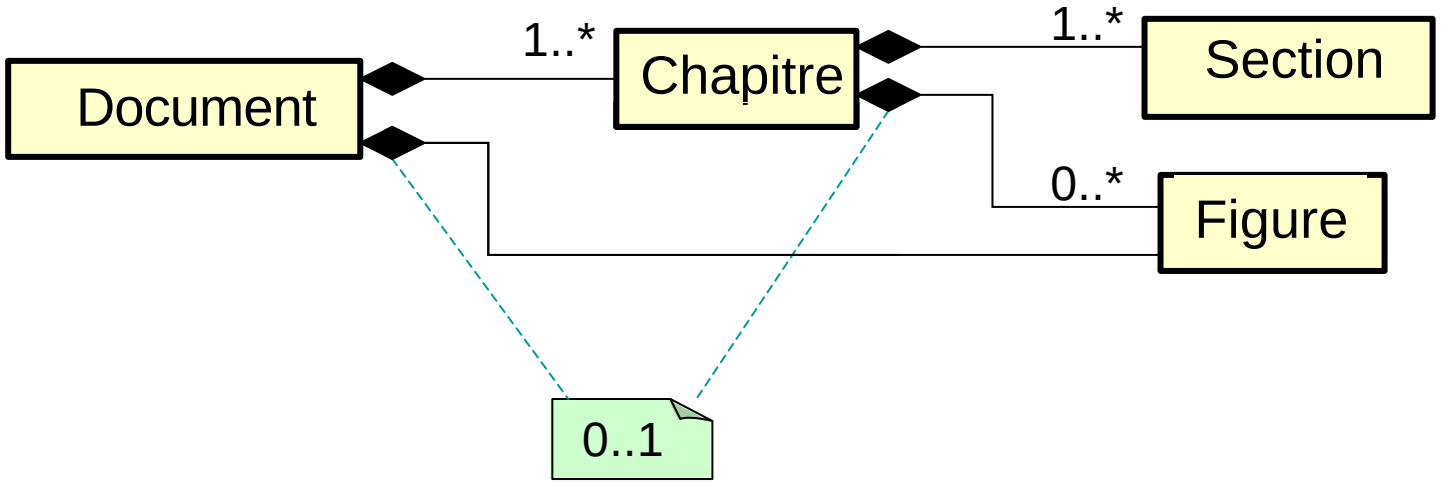
1. Un objet composant ne peut être que dans 1 seul objet composite
2. Un objet composant n'existe pas sans son objet composite
3. Si un objet composite est détruit, ses composants aussi



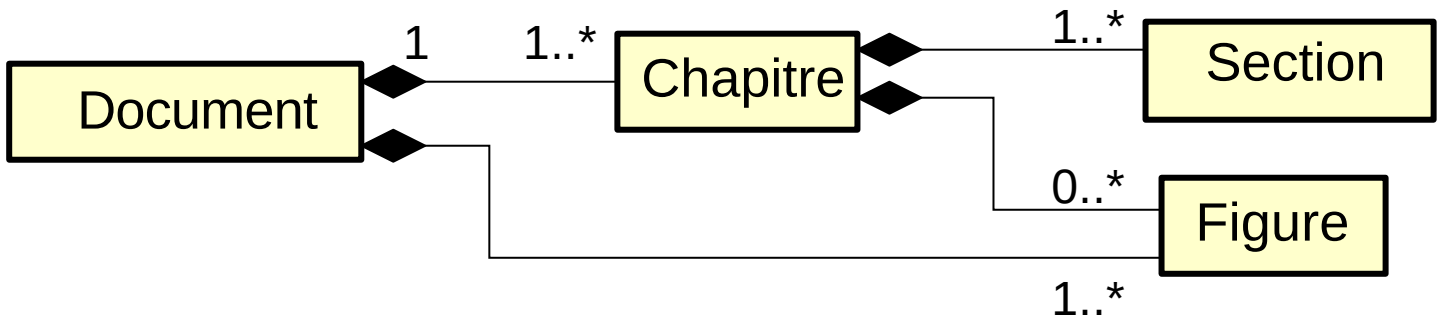
Composition

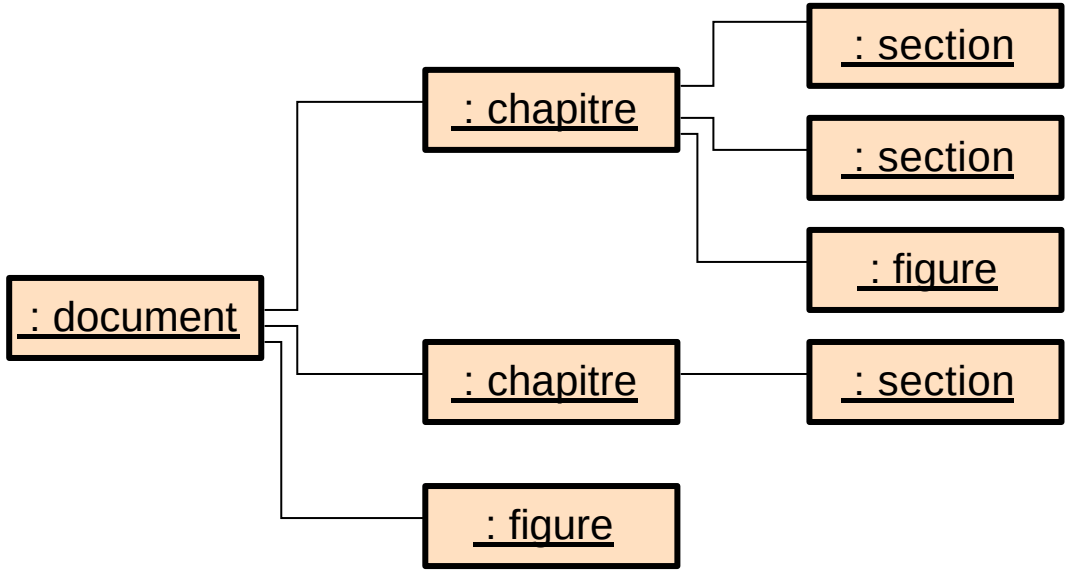
Contraintes liées à la composition :

1. Un objet composant ne peut être que dans 1 seul objet composite
2. Un objet composant n'existe pas sans son objet composite
3. Si un objet composite est détruit, ses composants aussi



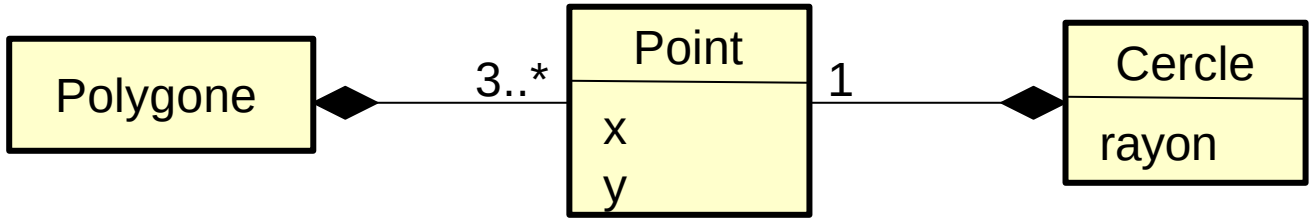
Composition

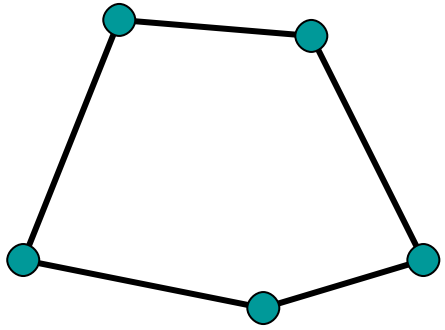


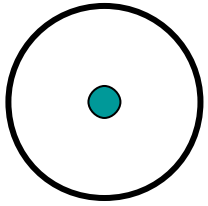


Contrainte :
les composants forment un arbre

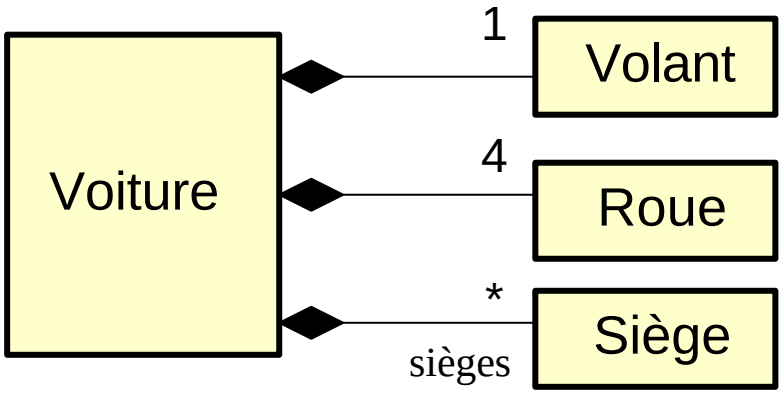
Composition



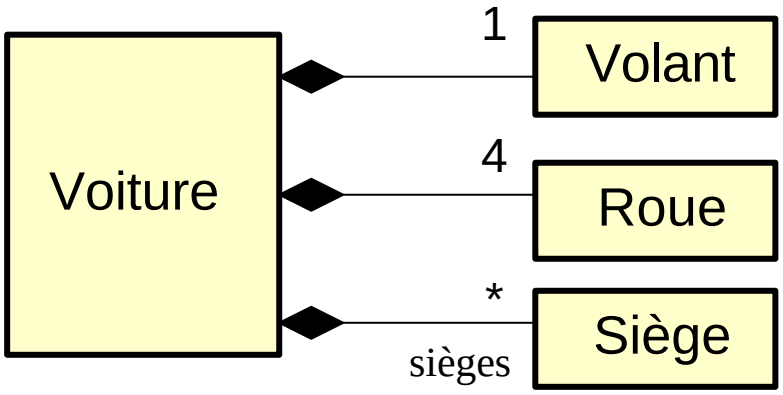


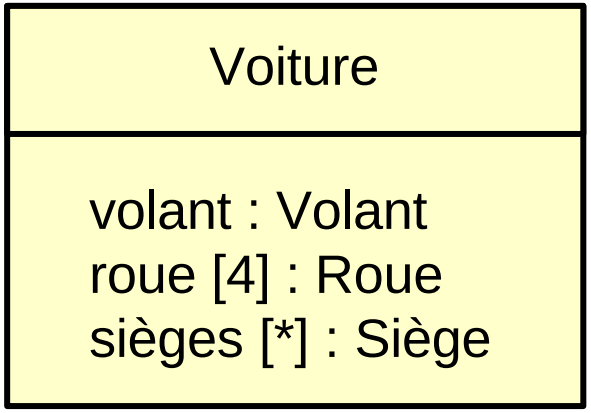


Autres notations pour la Composition

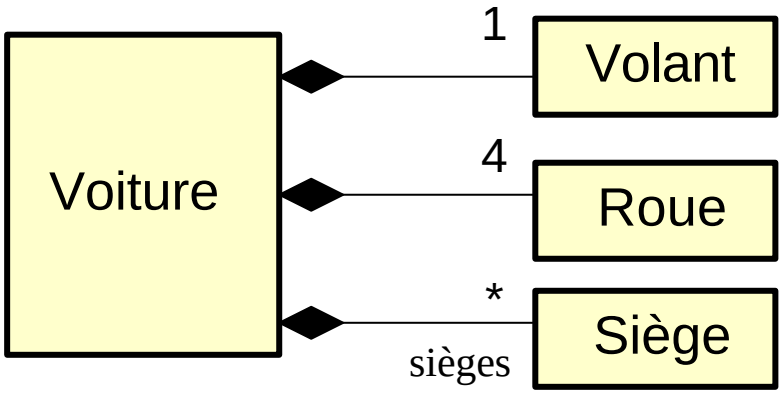


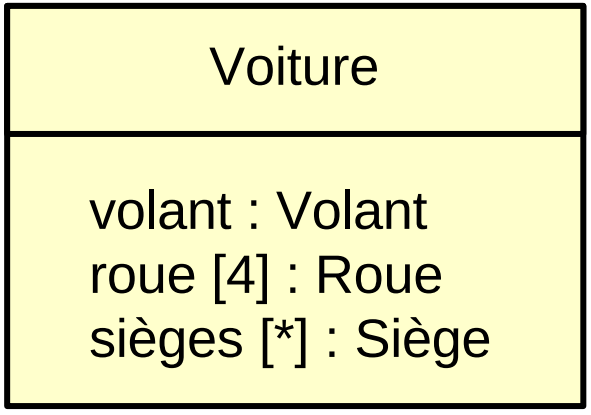
Autres notations pour la Composition

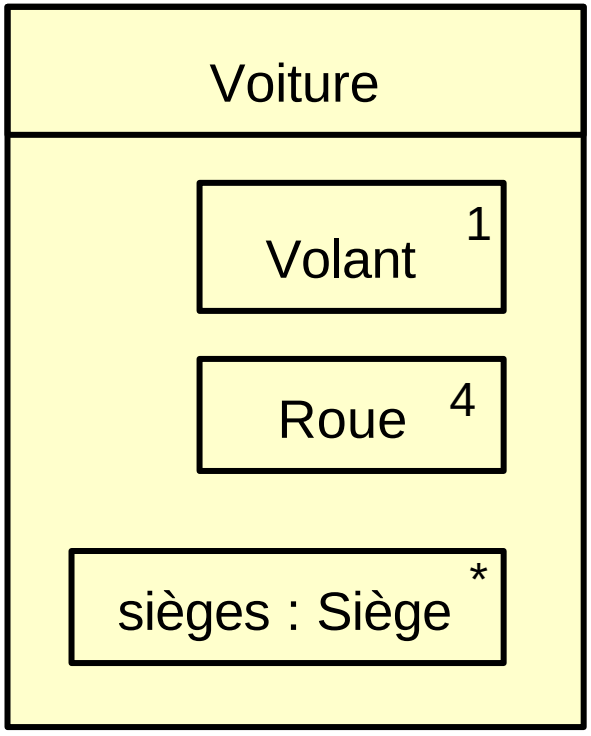




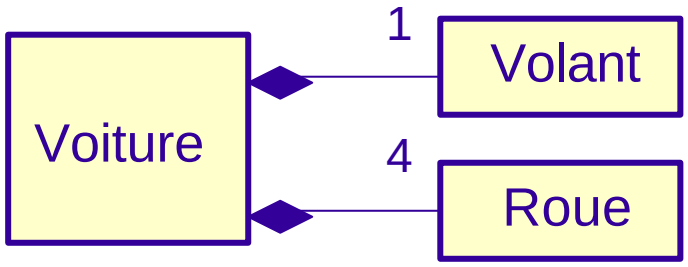
Autres notations pour la Composition



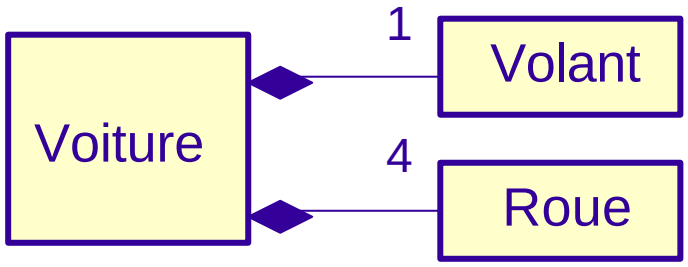


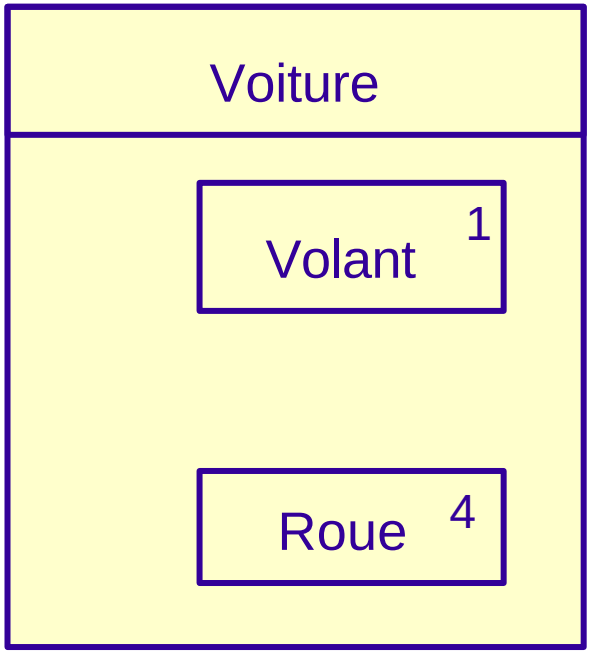


Associations intra vs. inter composites

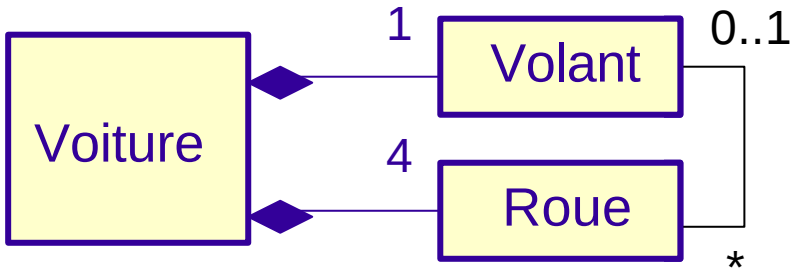


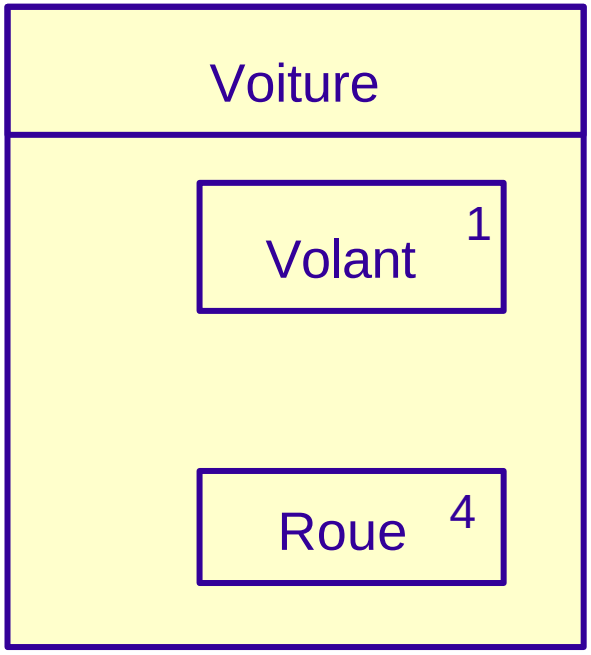
Associations intra vs. inter composites



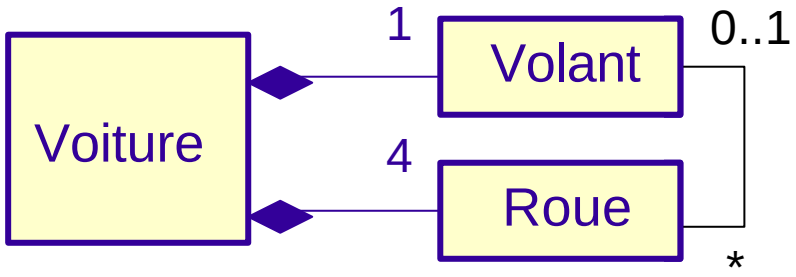


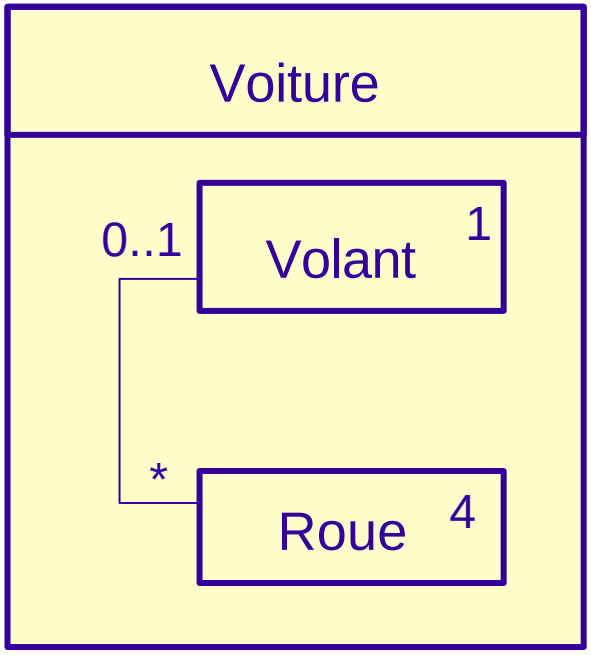
Associations intra vs. inter composites



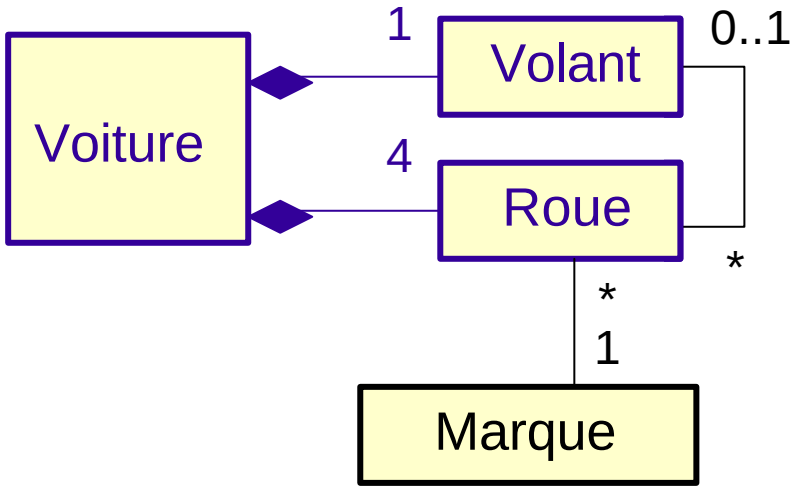


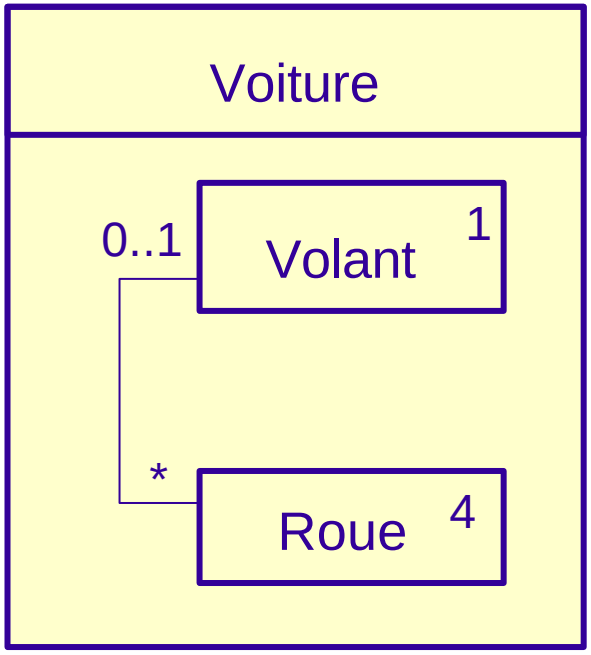
Associations intra vs. inter composites



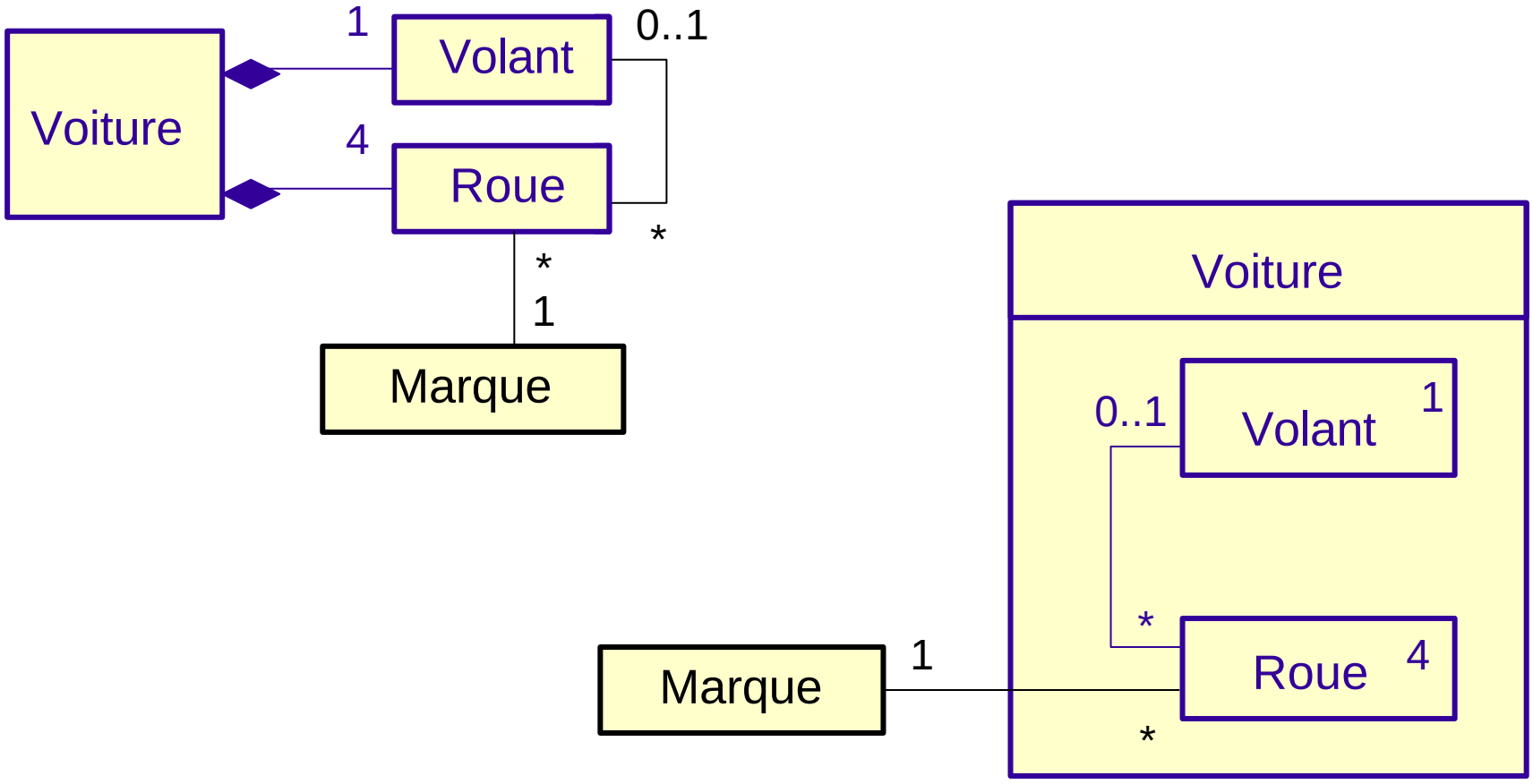


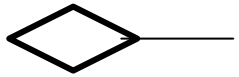
Associations intra vs. inter composites





Associations intra vs. inter composites



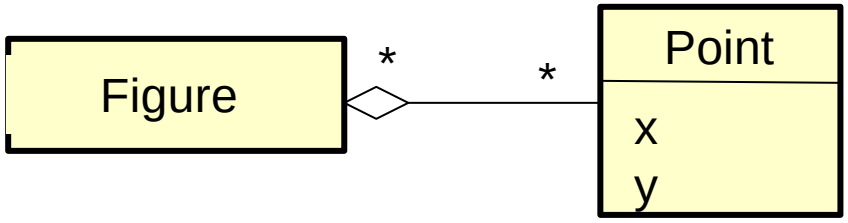


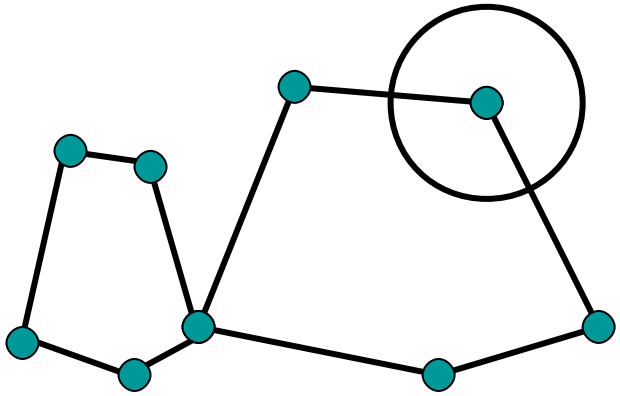
Aggregation

aggregation =

cas particulier d'association

+ contraintes décrivant la notion d'appartenance... ?

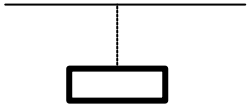




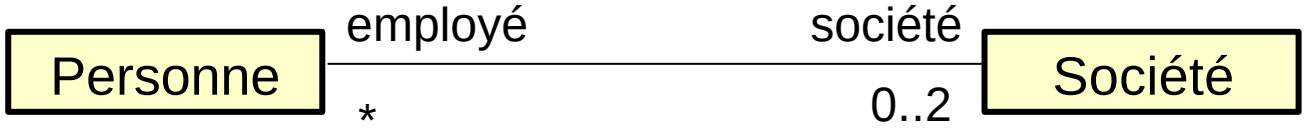
Partage possible

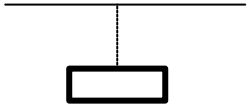
Pas de concensus sur la signification exacte de l'aggrégation

Utiliser avec précautions (ou ne pas utiliser...)



Classes associatives

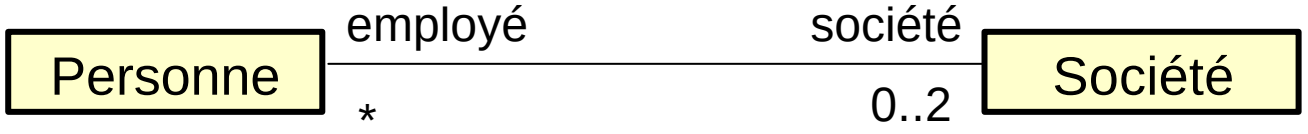


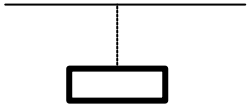


Classes associatives

Pour associer des attributs et/ou des méthodes aux associations

=> **classes associatives**

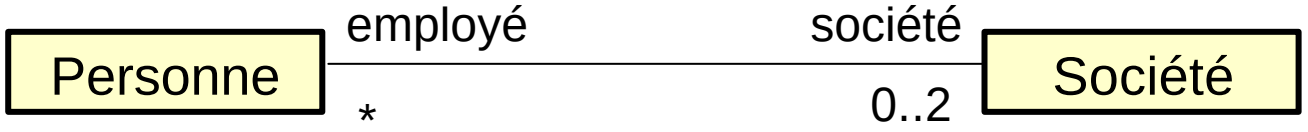




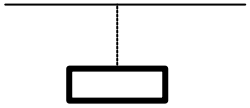
Classes associatives

Pour associer des attributs et/ou des méthodes aux associations

=> **classes associatives**



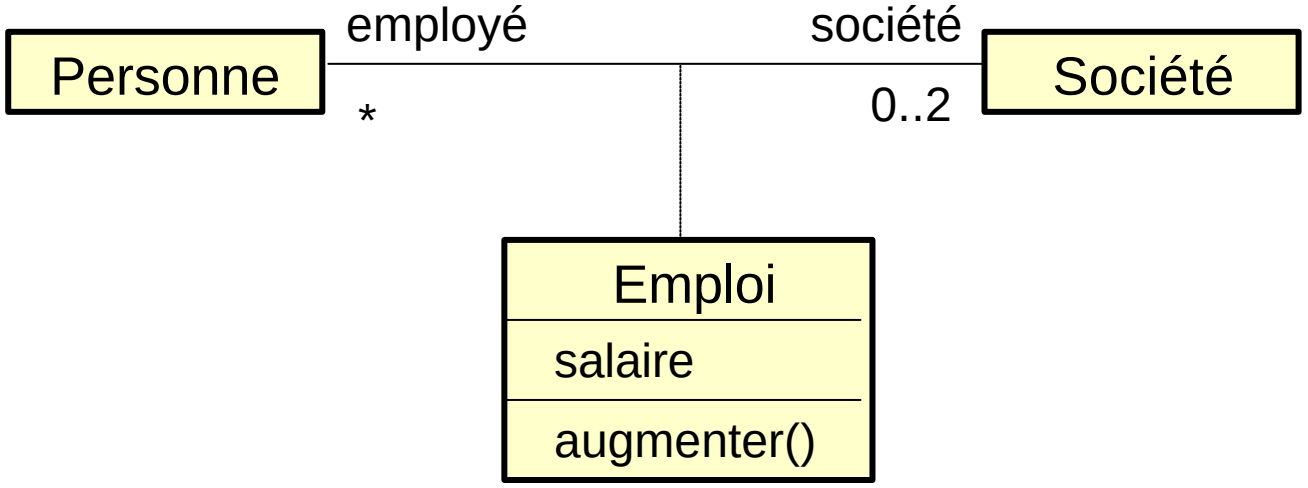
Le nom de la classe correspond au nom de l'association



Classes associatives

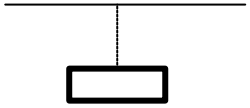
Pour associer des attributs et/ou des méthodes aux associations

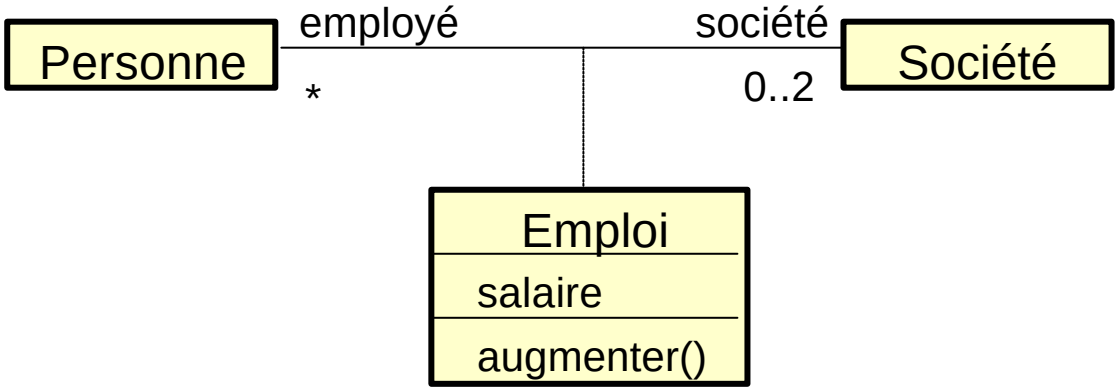
=> **classes associatives**

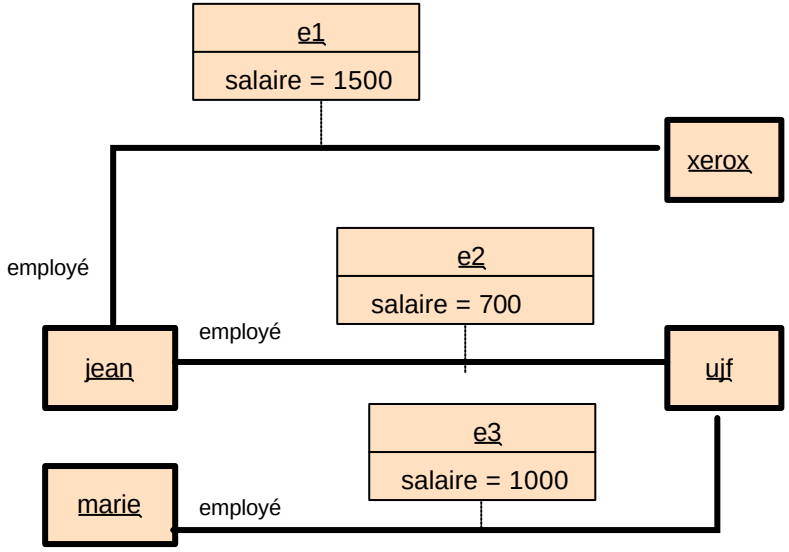


Le nom de la classe correspond au nom de l'association

Classes associatives

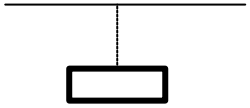






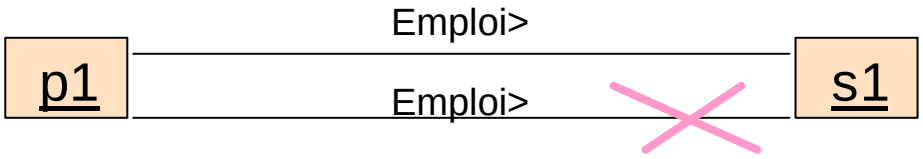
Le salaire est une information correspondant
- ni à une personne,
- ni à une société,

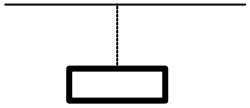
mais à un emploi (un couple personne-société).



Classes associatives

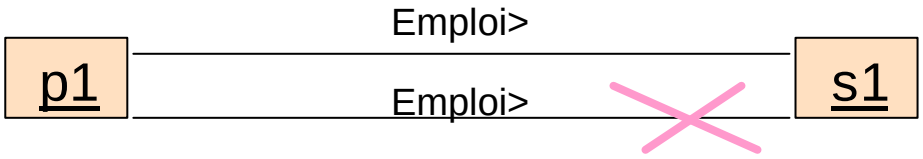
Pour une association donnée, deux objets ne peuvent être connectés que par un seul lien correspondant à cette association.



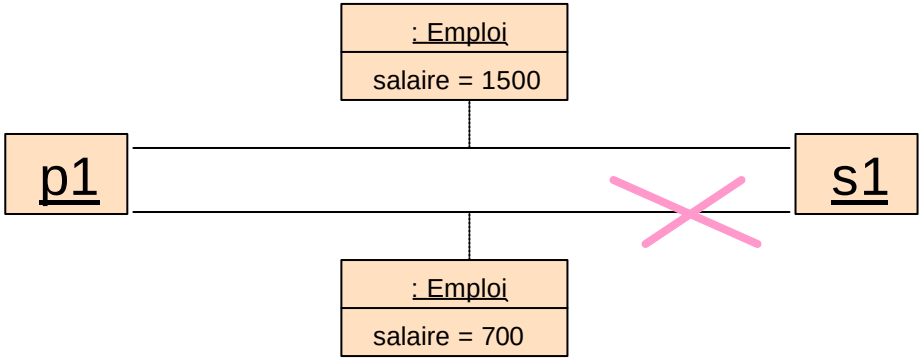


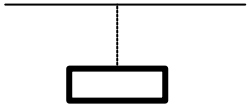
Classes associatives

Pour une association donnée, deux objets ne peuvent être connectés que par un seul lien correspondant à cette association.

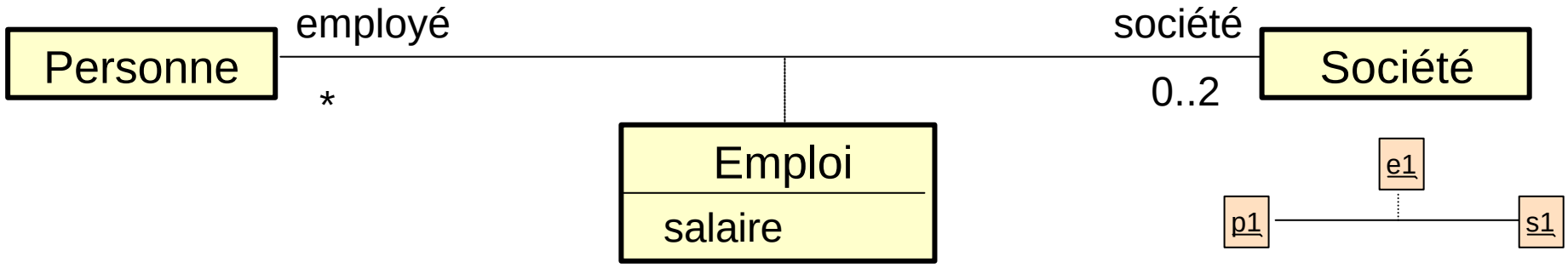


Cette contrainte reste vraie dans le cas où l'association est décrite à partir d 'une classe associative.

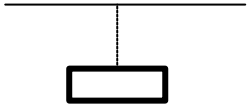




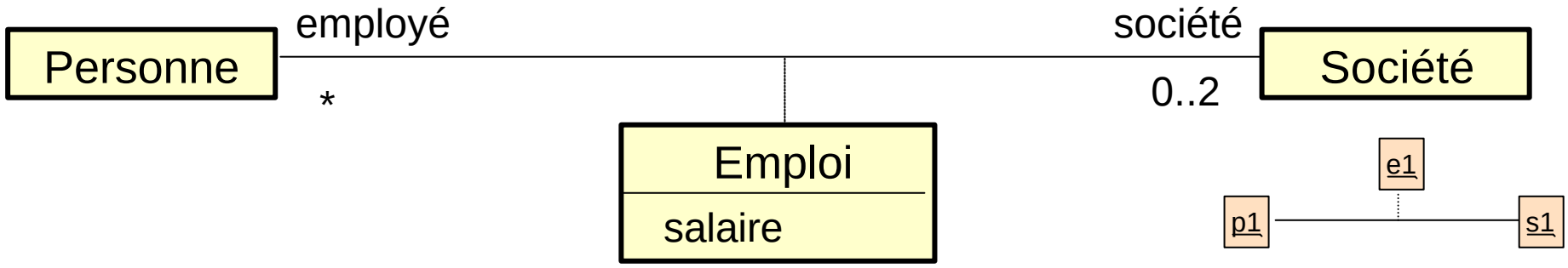
Classes associatives



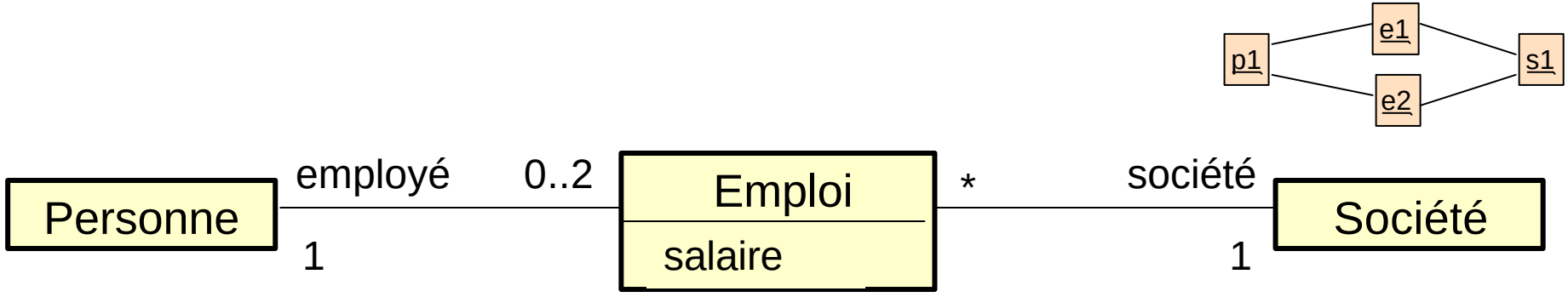
Ci-dessus, une personne peut avoir deux emplois, mais pas dans la même société



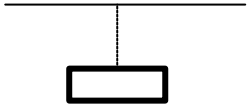
Classes associatives



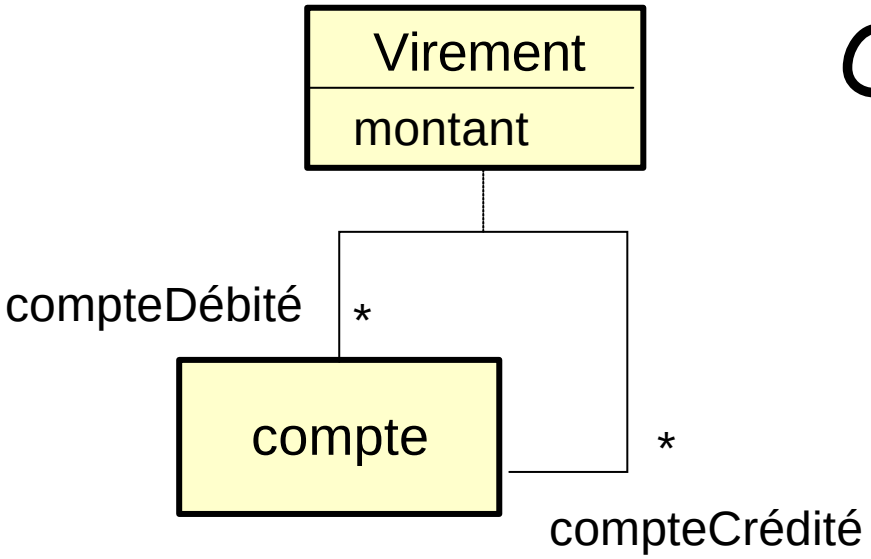
Ci-dessus, une personne peut avoir deux emplois, mais pas dans la même société



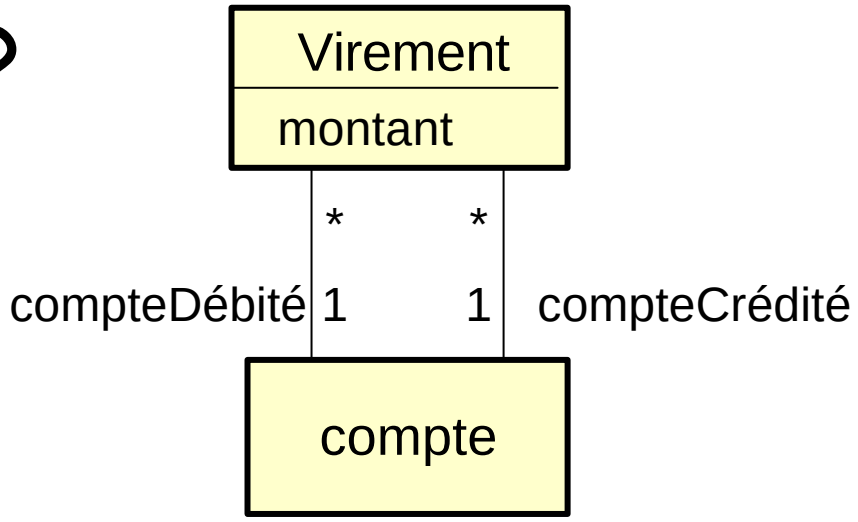
Ci-dessus, une personne peut avoir deux emplois dans la même société

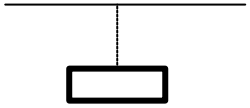


Classes associatives



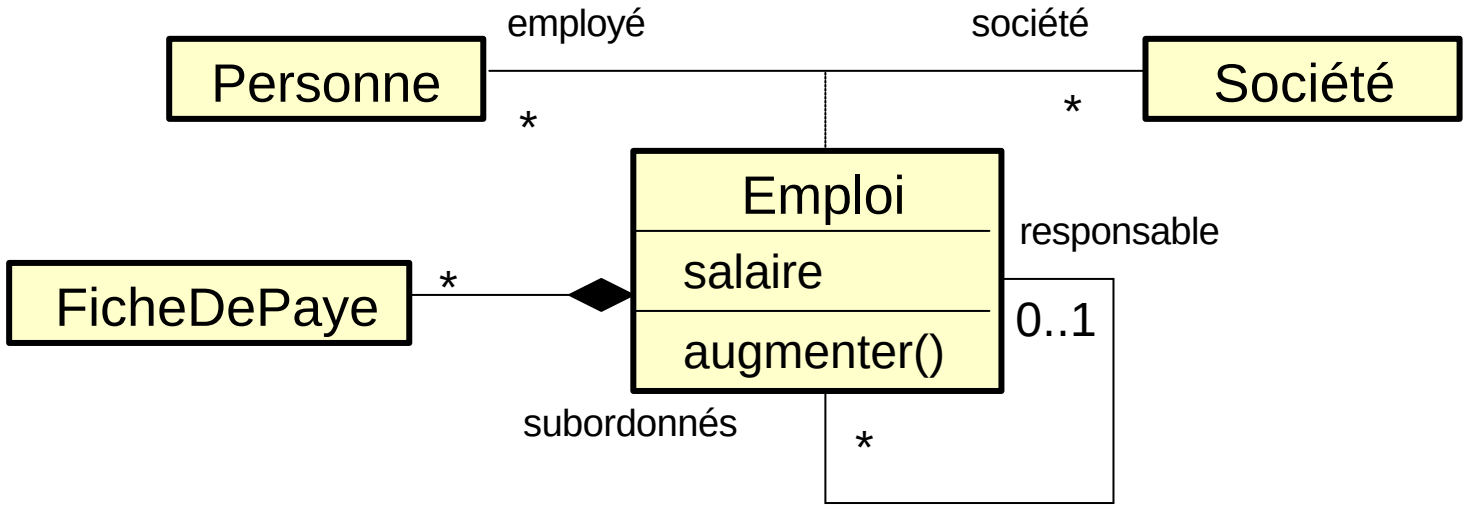
Ou ?





Classes associatives

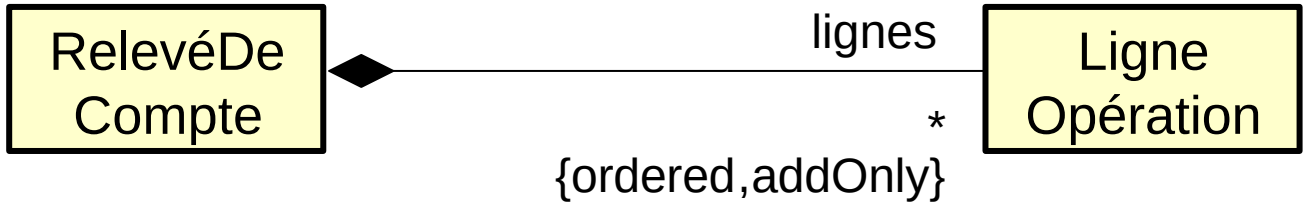
Les classes associatives sont des associations mais aussi des classes. Elles ont donc les mêmes propriétés et peuvent par exemple être liées par des associations.



Contraintes prédéfinies sur les associations

Contraintes prédéfinies sur les associations.
Par exemple :

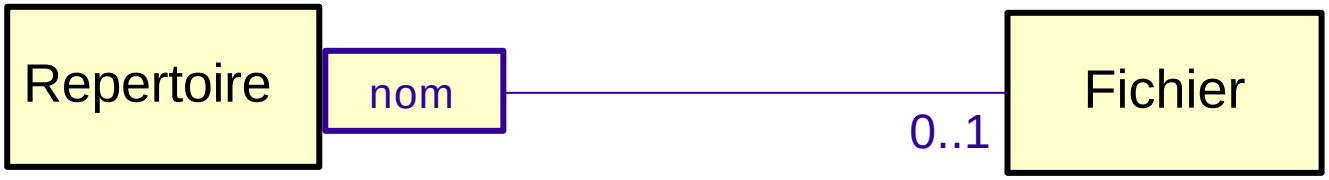
- { frozen } : fixé lors de la création de l 'objet, ne peut pas changer
- { ordered } : les éléments de la collection sont ordonnés
- { addOnly } : impossible de supprimer un élément



Il est possible de définir de nouvelles contraintes

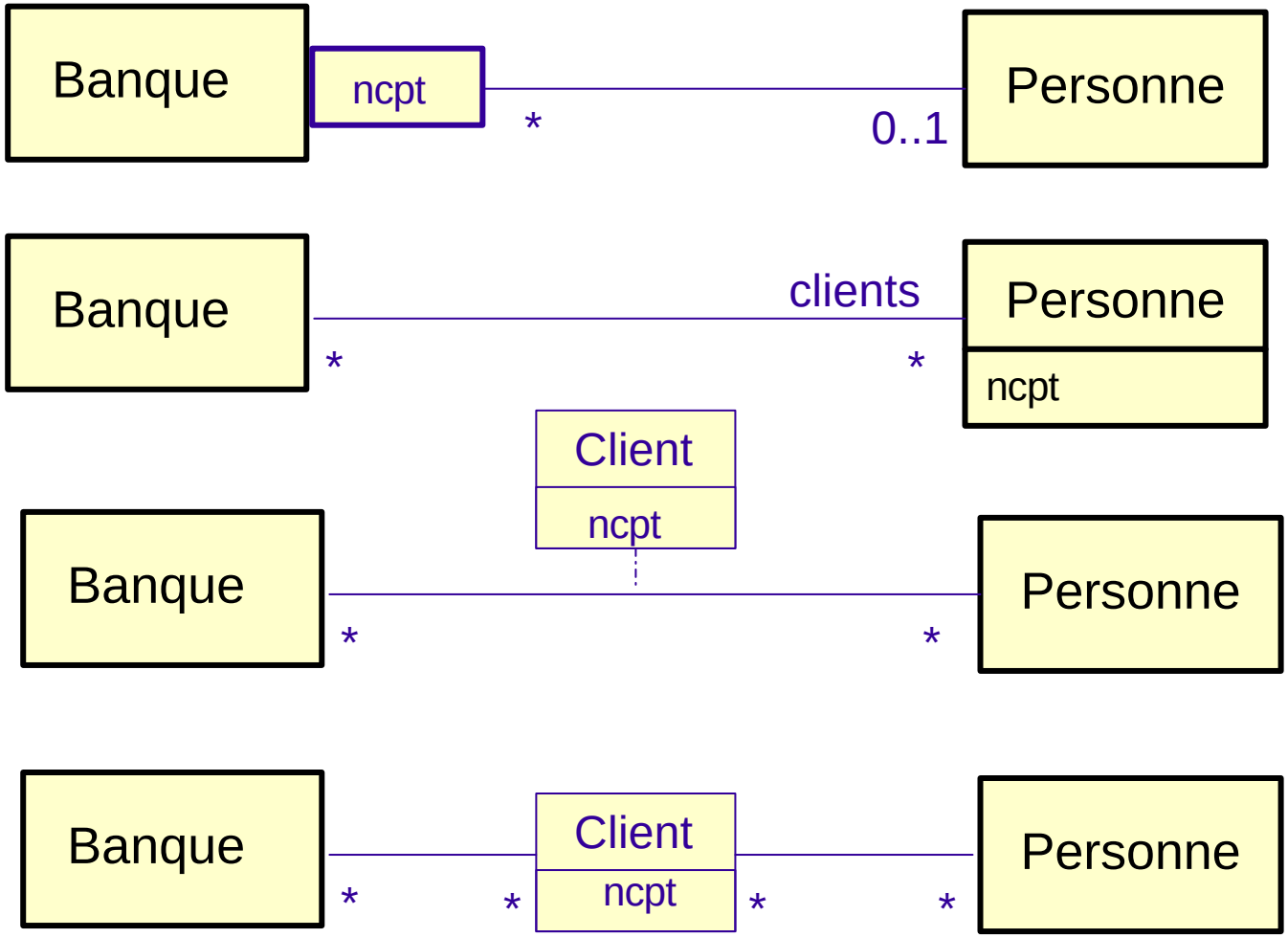
Associations qualifiées

Un **qualifieur** est un attribut ou un ensemble d'attributs dont la valeur sert à déterminer l'ensemble des instances associées à une instance via une association.

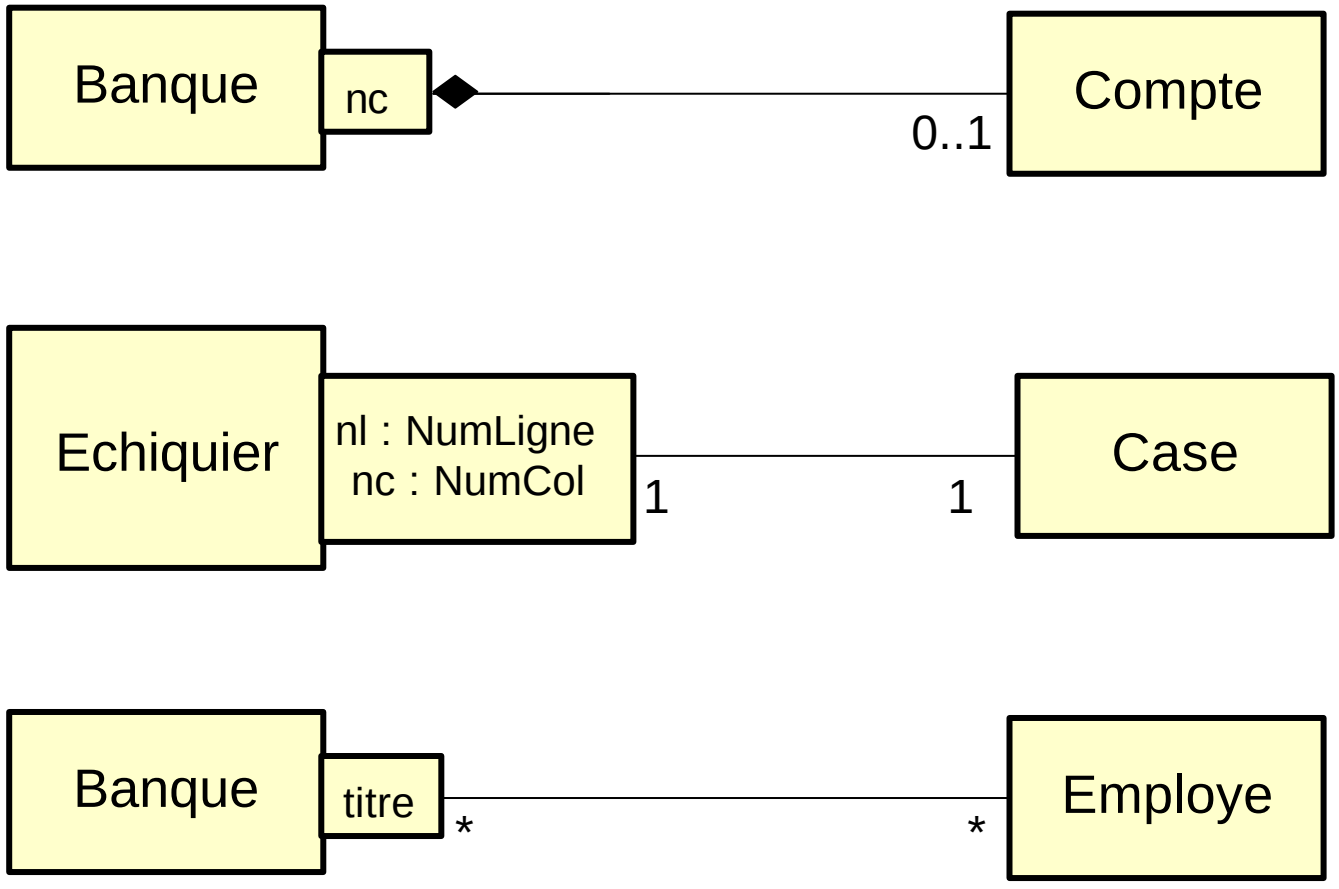


Les attributs du qualifieur sont des attributs de l'association.

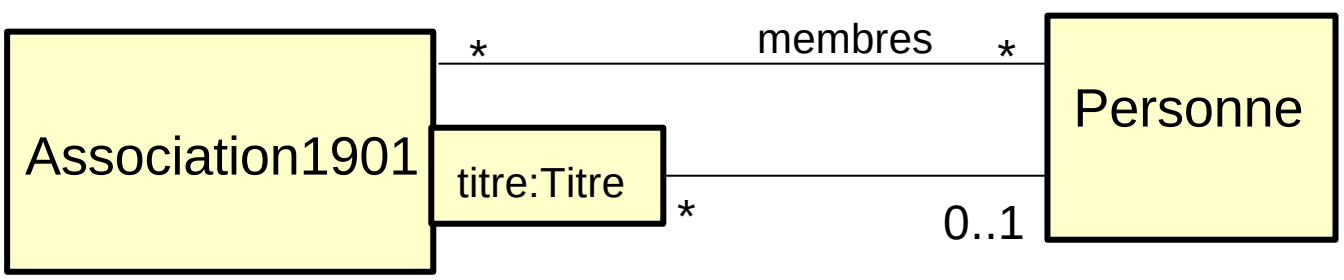
Associations vs. Associations qualifiées

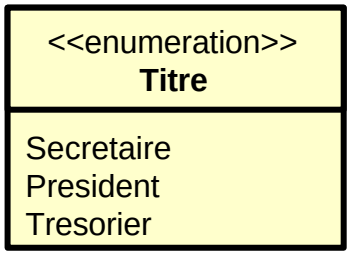


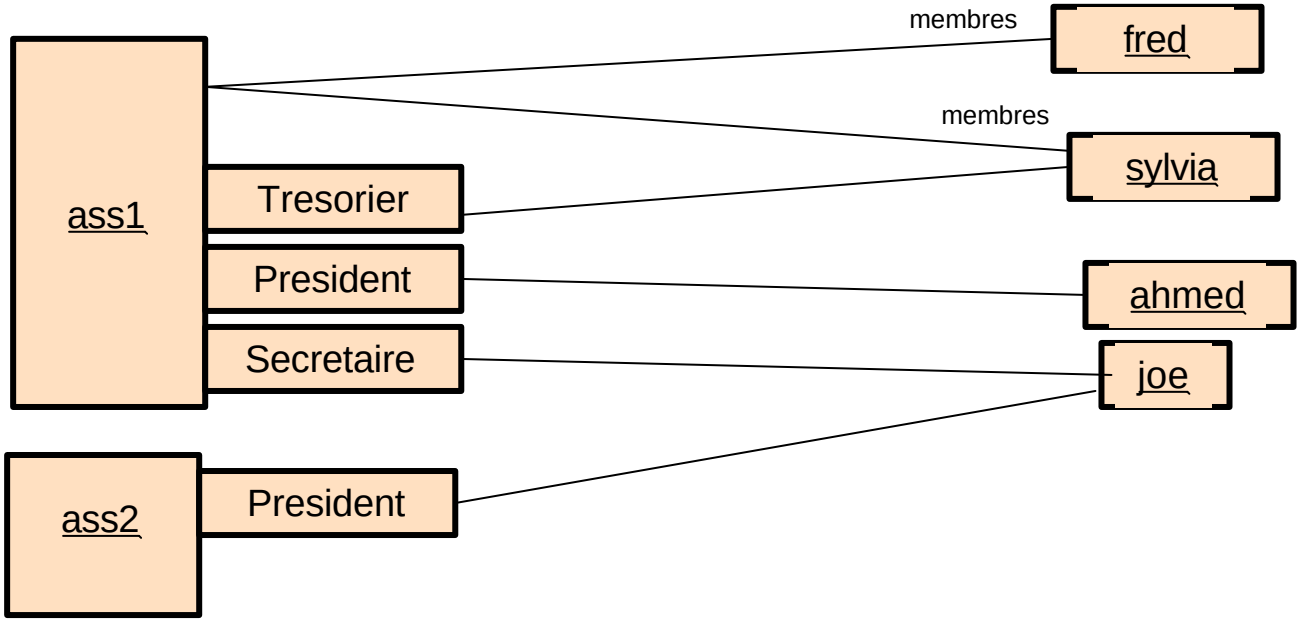
Cardinalité des Associations Qualifiées



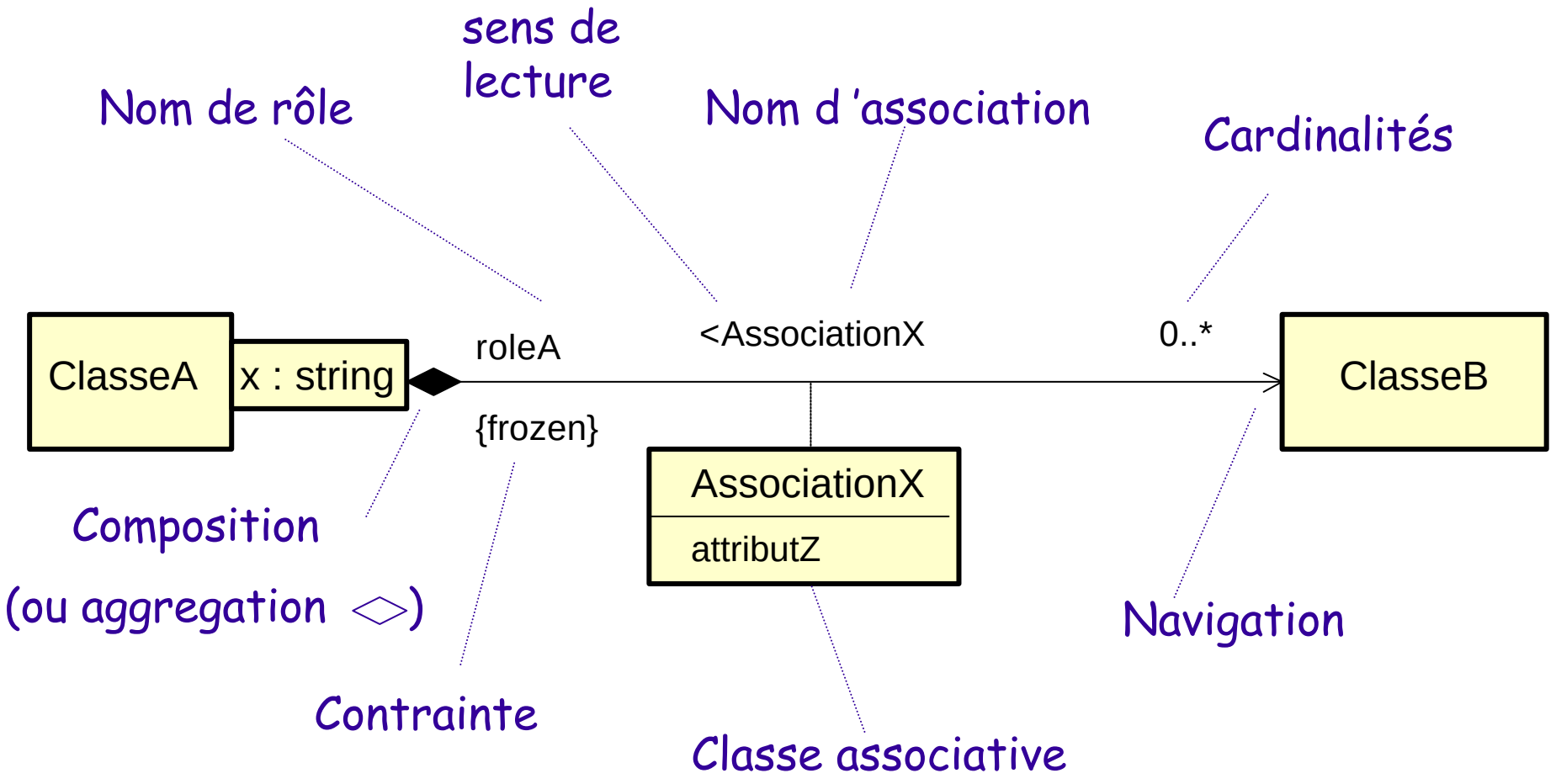
Exemple


Association1901
titre:Titre
*
membres
*
*
0..1
Personne


<<enumeration>>
Titre
Secretaire
President
Tresorier


ass1
Tresorier
President
Secretaire
ass2
President
membres
membres
fred
sylvia
ahmed
joe

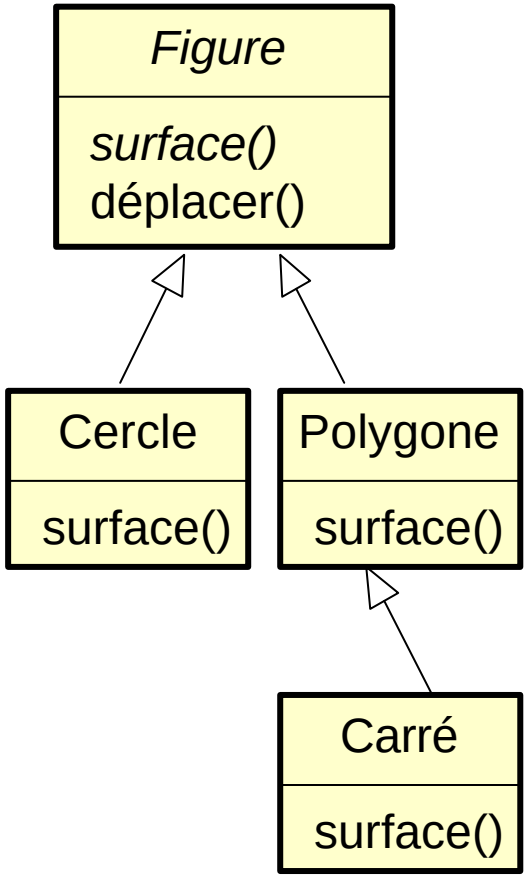
Synthèse sur les associations



Raffinement du concept de généralisation

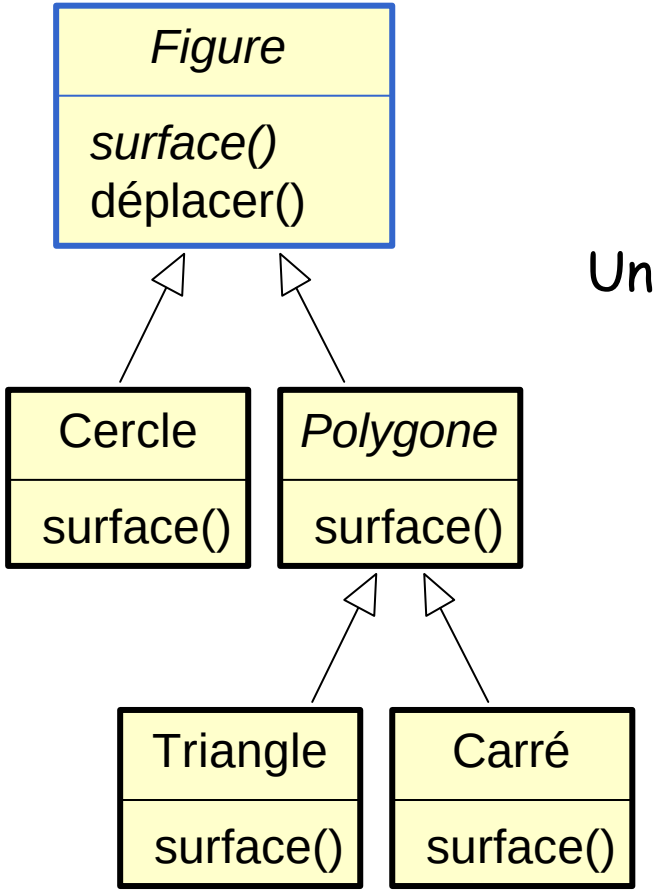
- Ré-définition
- Classes abstraites
- Méthodes abstaites
- Héritage simple vs. multiple
- Classification simple vs. multiple
- Classification statique vs. dynamique

Héritage et redéfinition



Une sous classe peut redéfinir une méthode,
à condition toutefois de rester compatible avec la définition originale

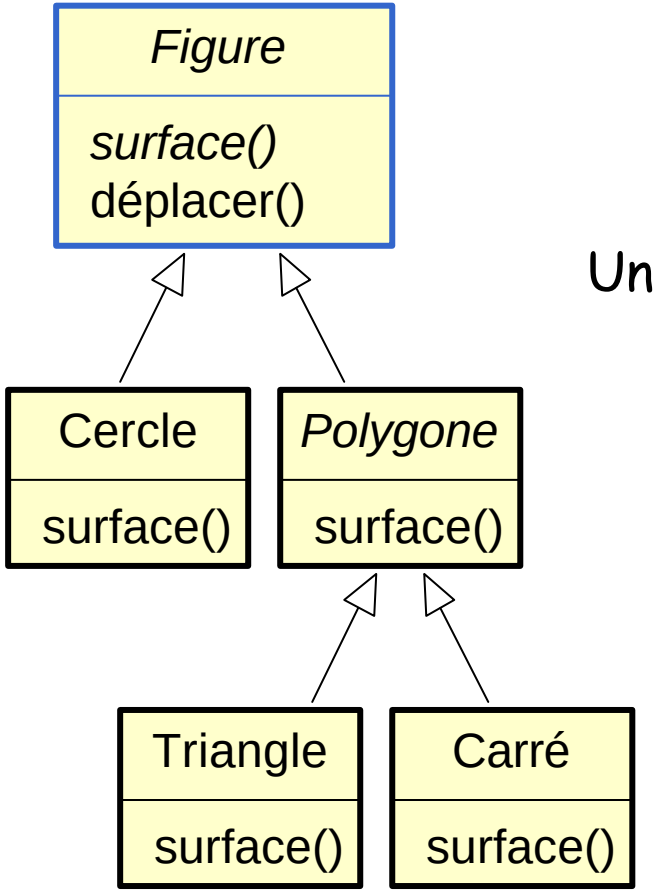
Classes et méthodes abstraites



Une classe abstraite
- ne peut pas être instanciée
- utile pour définir un comportement abstrait
- peut contenir des méthodes abstraites

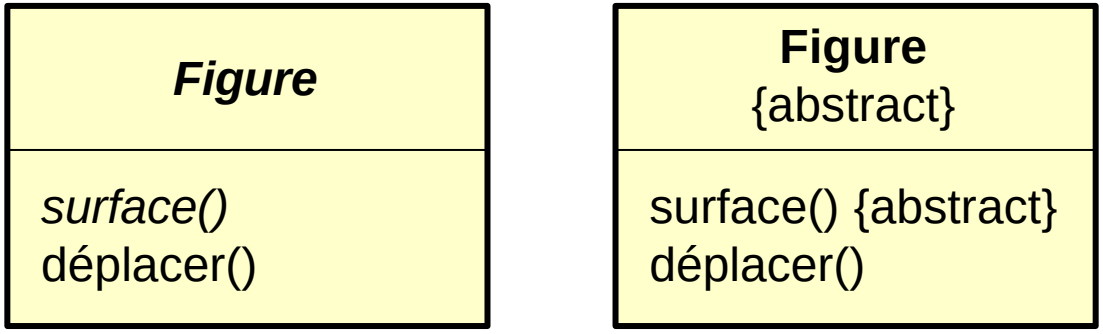
Un méthode abstraite
- doit être définie dans une sous classe
- est dans un classe abstraite

Classes et méthodes abstraites



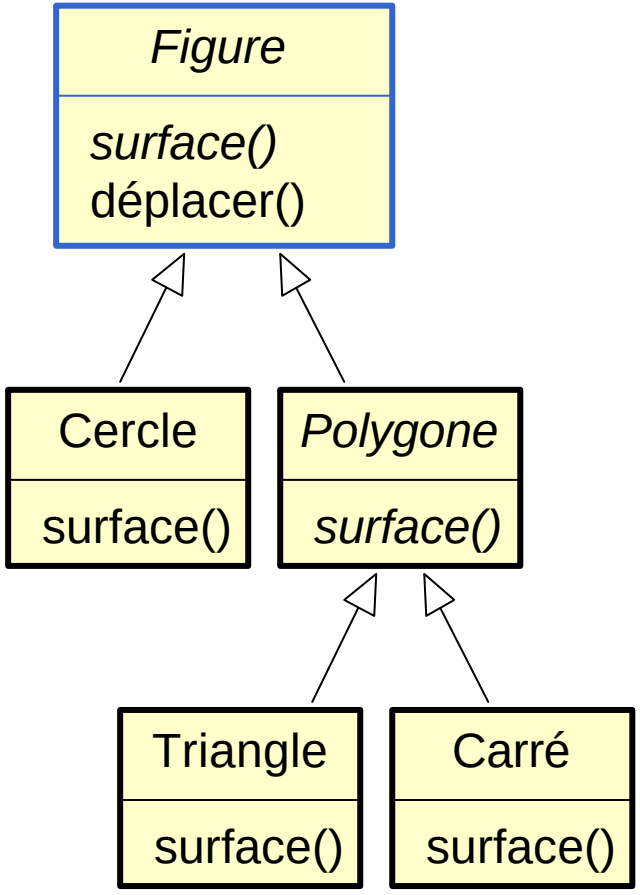
Une classe abstraite
- ne peut pas être instanciée
- utile pour définir un comportement abstrait
- peut contenir des méthodes abstraites

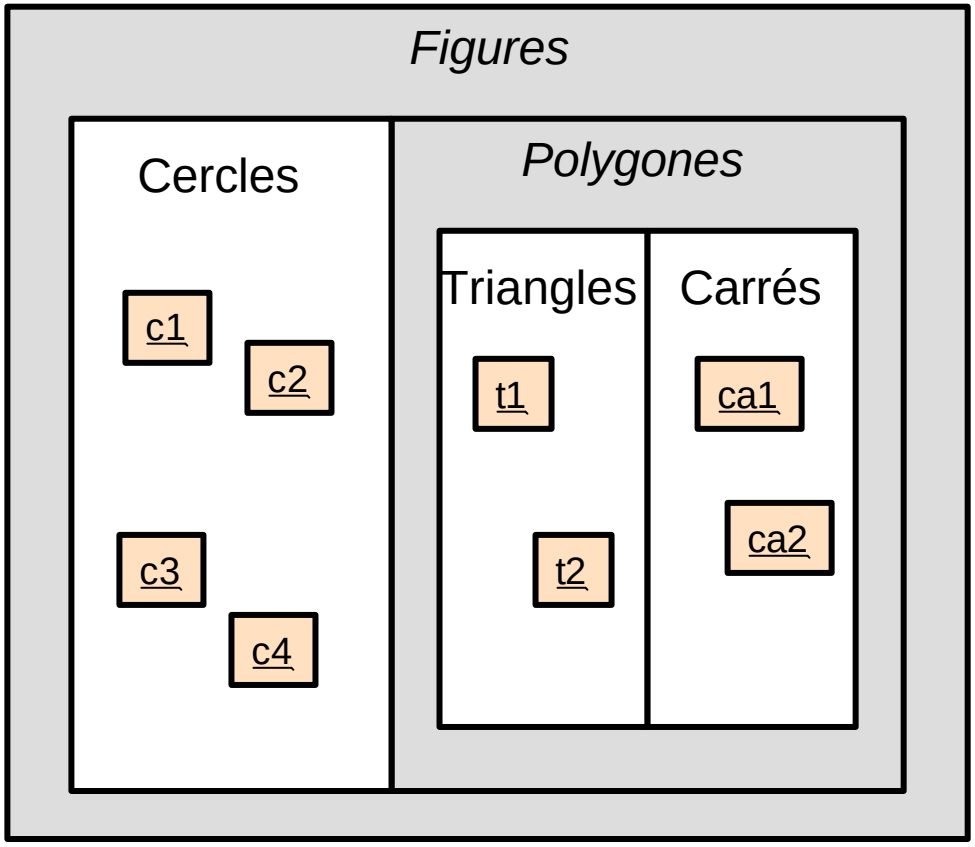
Un méthode abstraite
- doit être définie dans une sous classe
- est dans un classe abstraite



Notations équivalentes

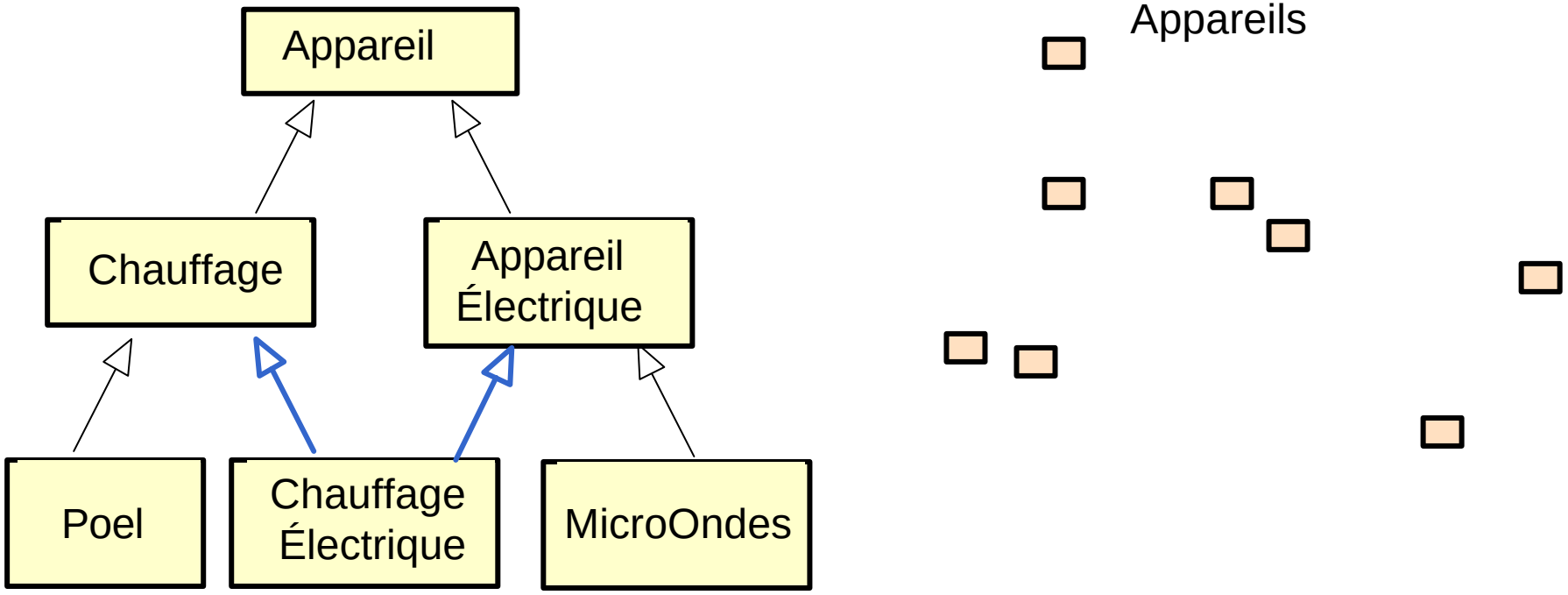
Classes abstraites du point de vue ensembliste





Héritage multiple

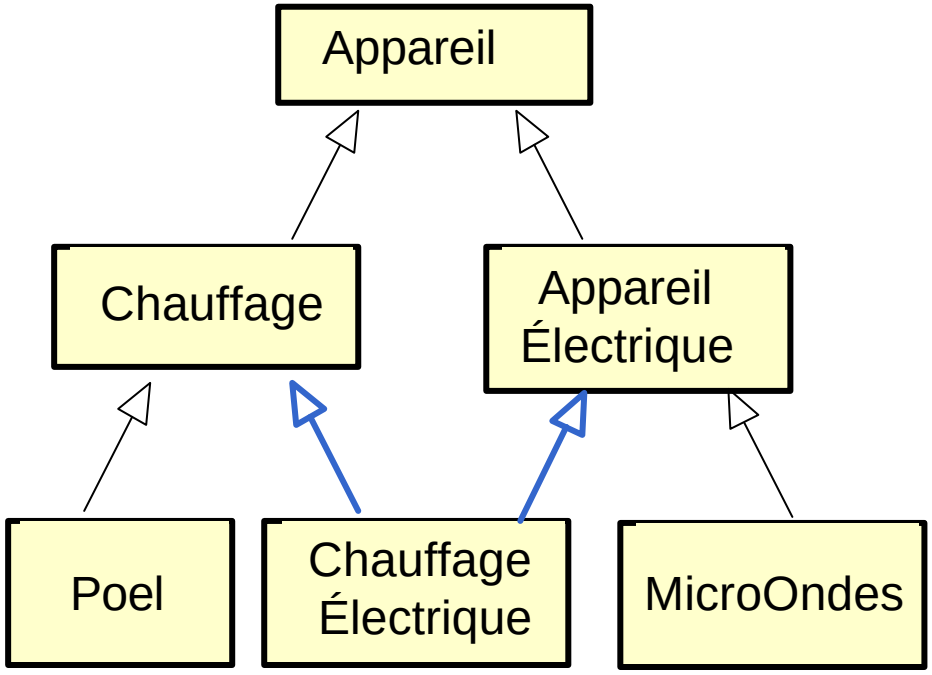
Une classe peut hériter de plusieurs super-classes

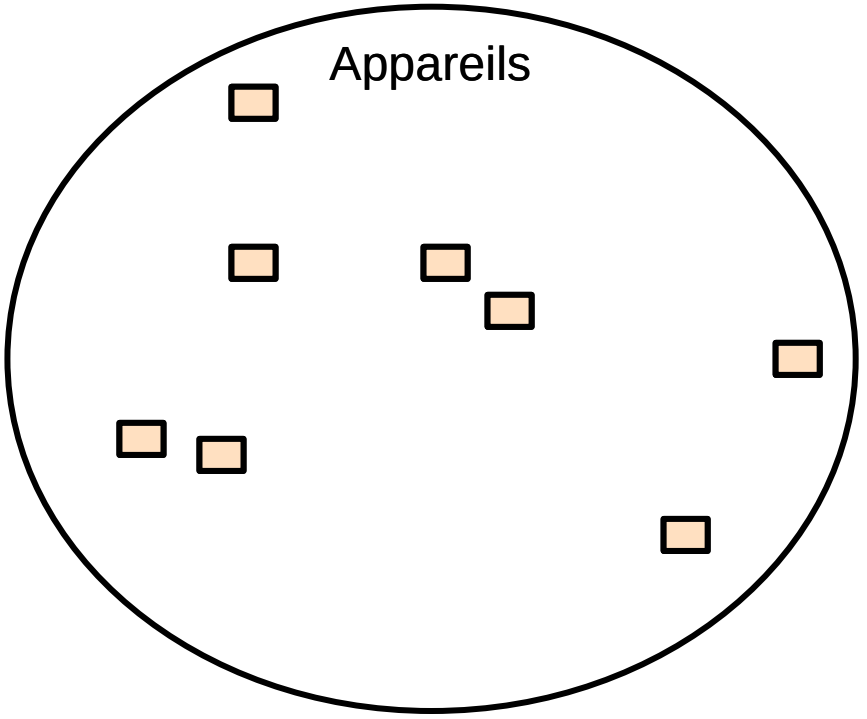


Interdit dans certains langages de programmation (p.e. Java et C#)

Héritage multiple

Une classe peut hériter de plusieurs super-classes

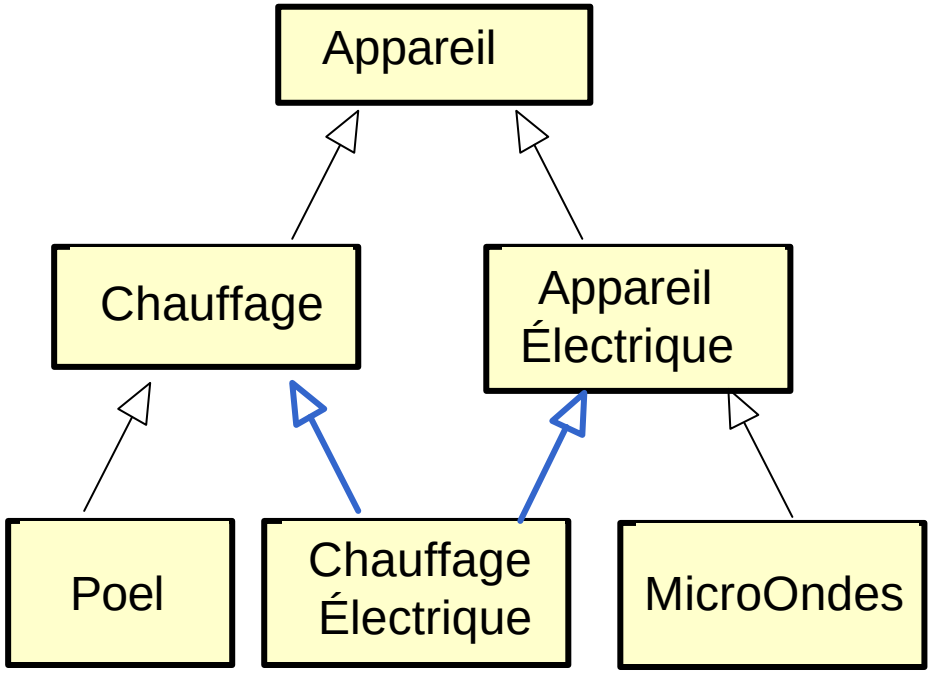


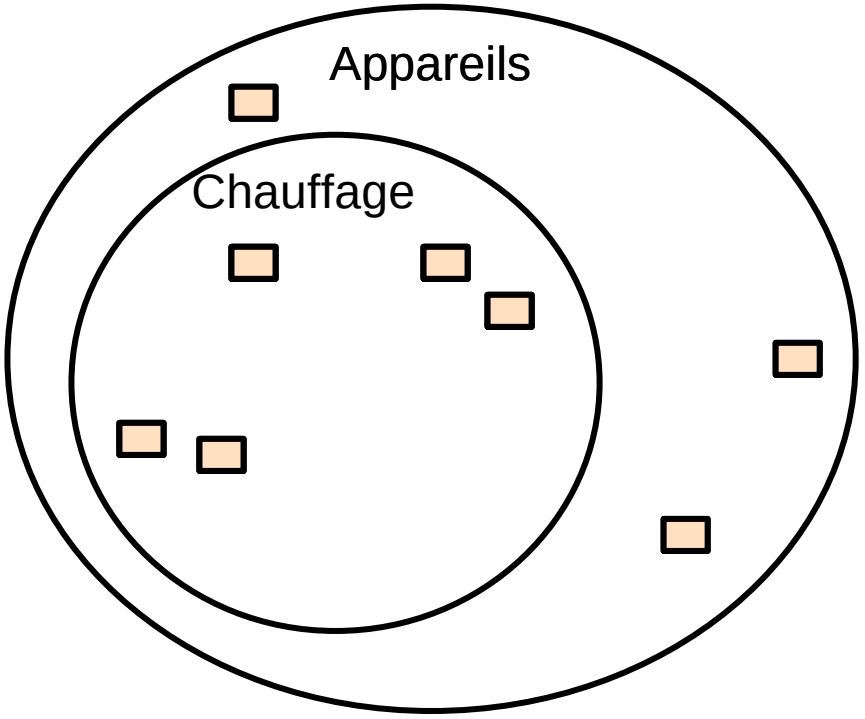


Interdit dans certains langages de programmation (p.e. Java et C#)

Héritage multiple

Une classe peut hériter de plusieurs super-classes

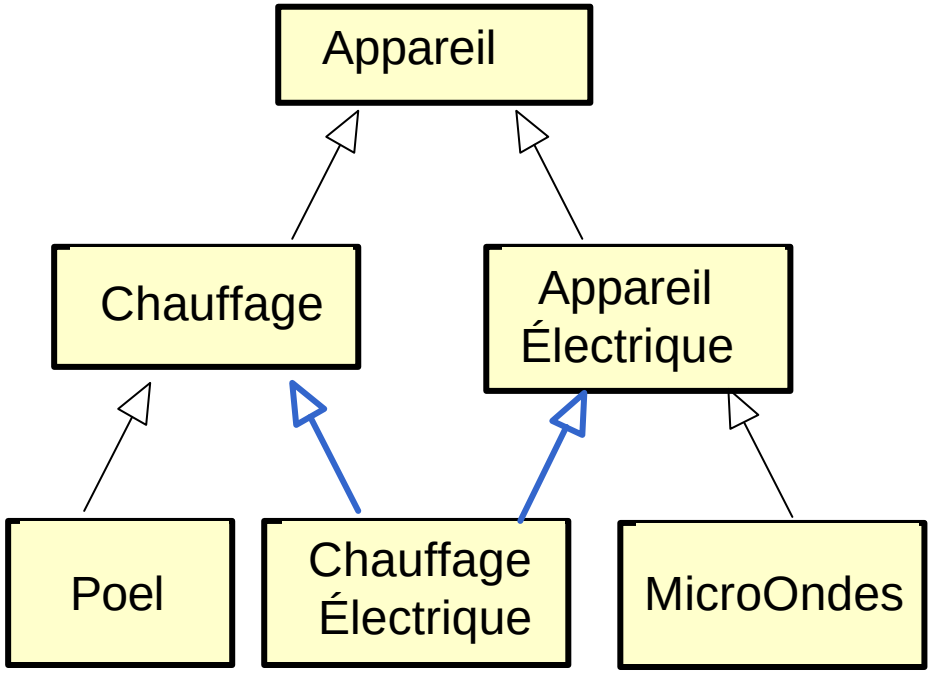


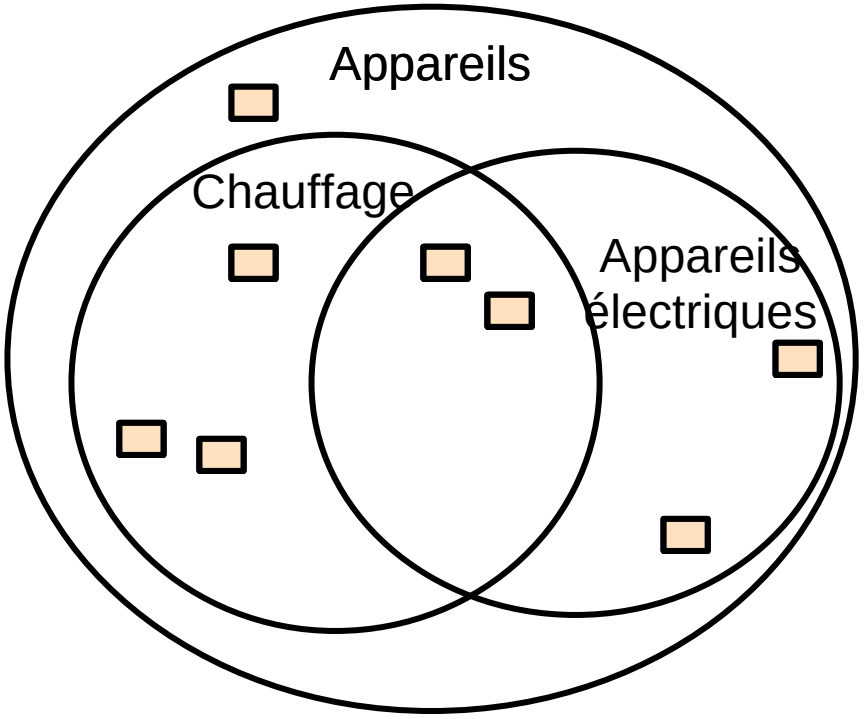


Interdit dans certains langages de programmation (p.e. Java et C#)

Héritage multiple

Une classe peut hériter de plusieurs super-classes

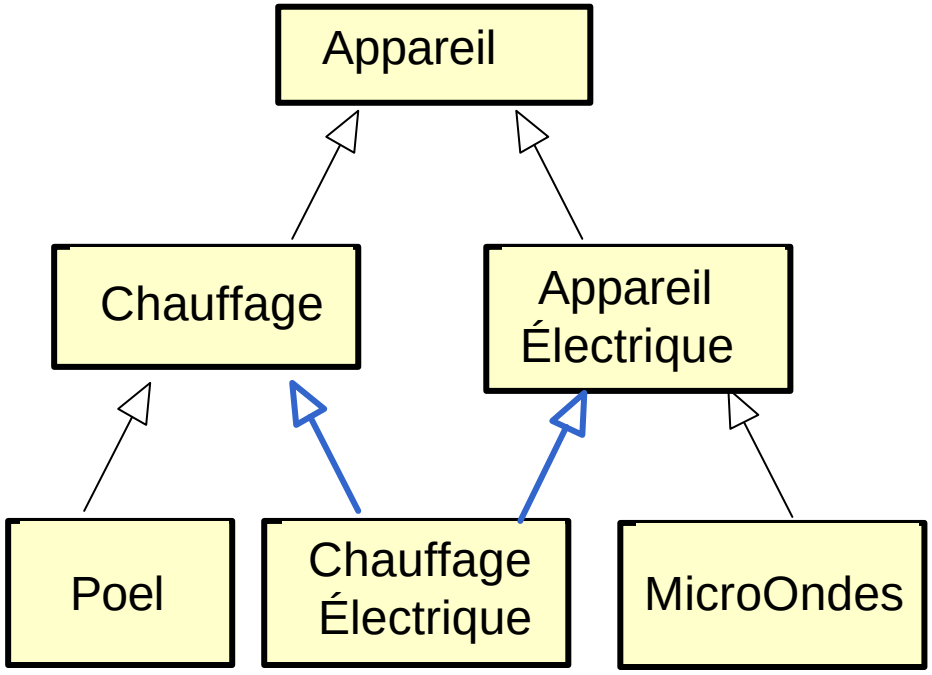


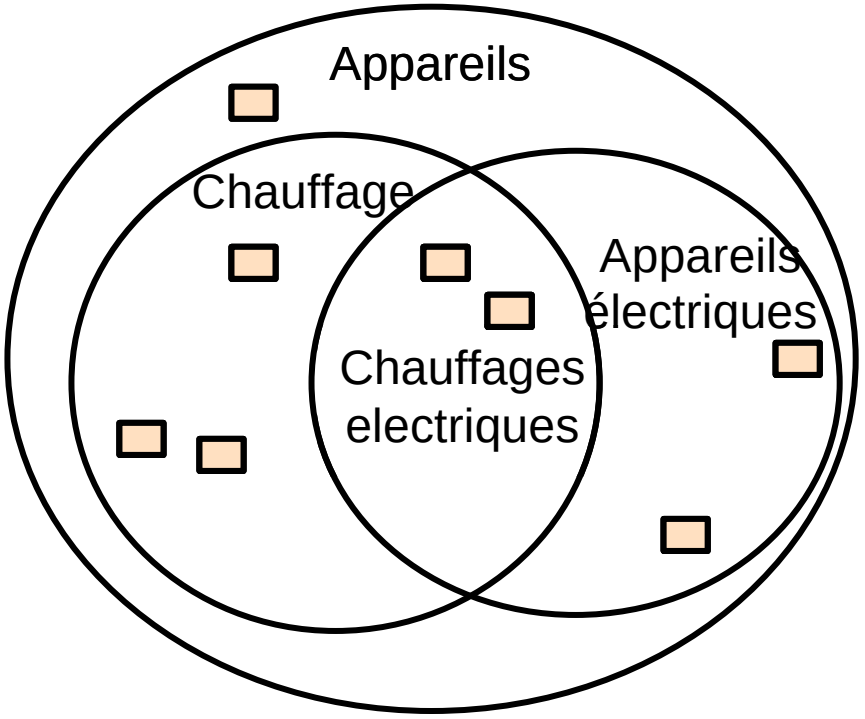


Interdit dans certains langages de programmation (p.e. Java et C#)

Héritage multiple

Une classe peut hériter de plusieurs super-classes

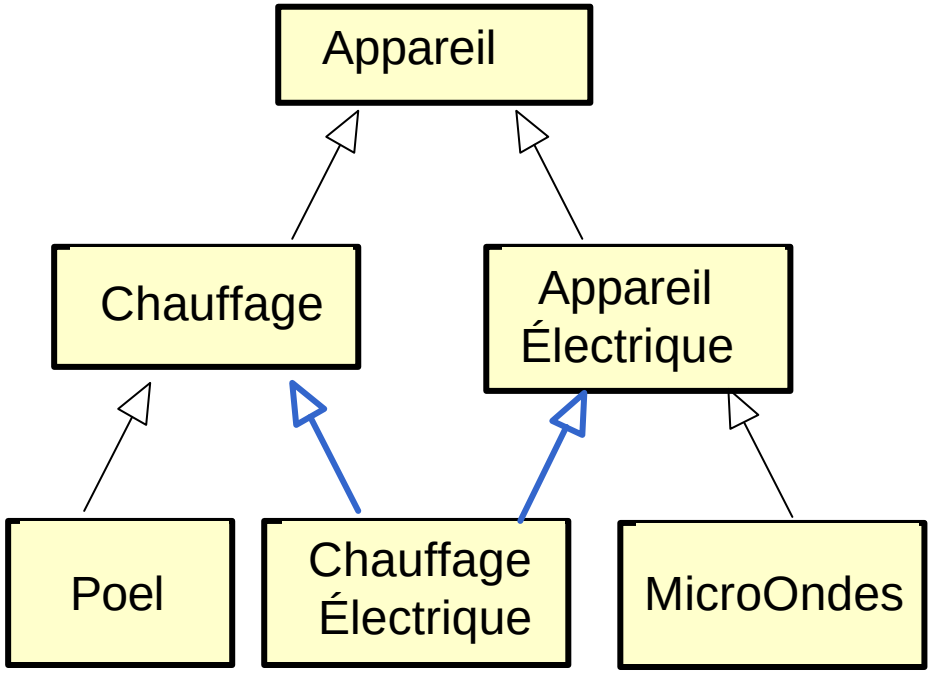


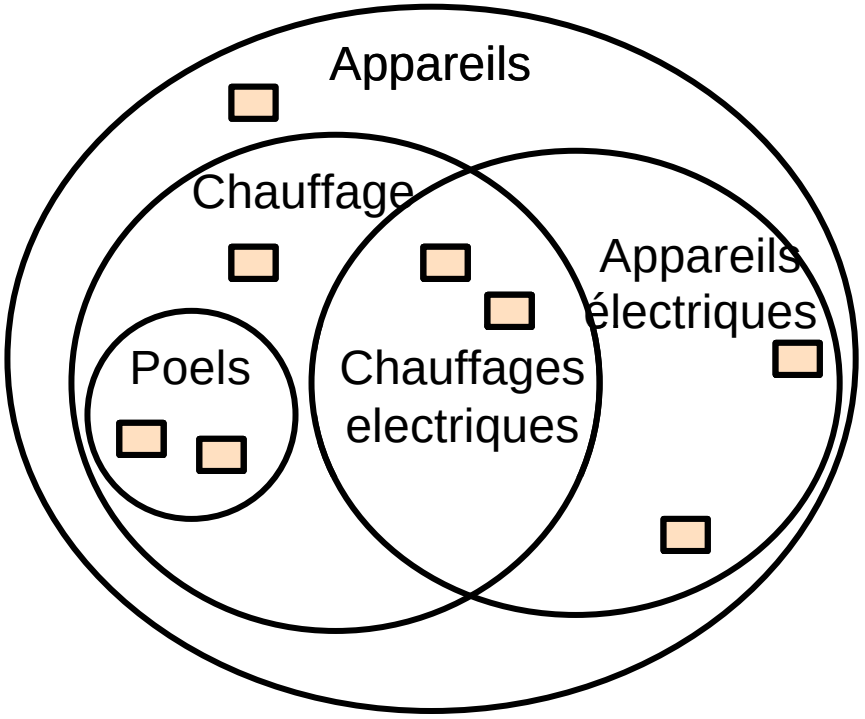


Interdit dans certains langages de programmation (p.e. Java et C#)

Héritage multiple

Une classe peut hériter de plusieurs super-classes

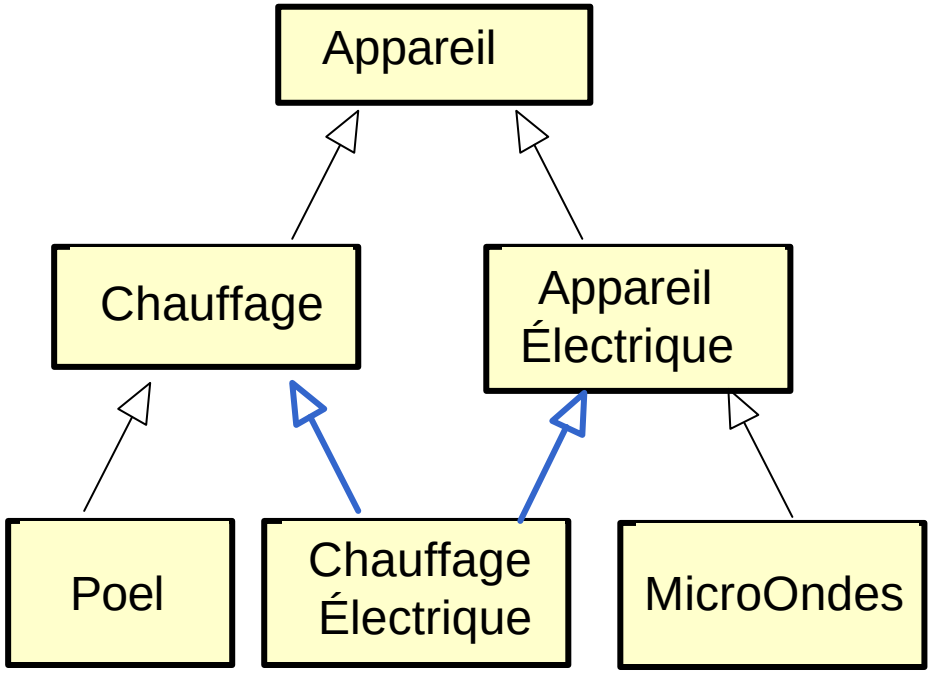


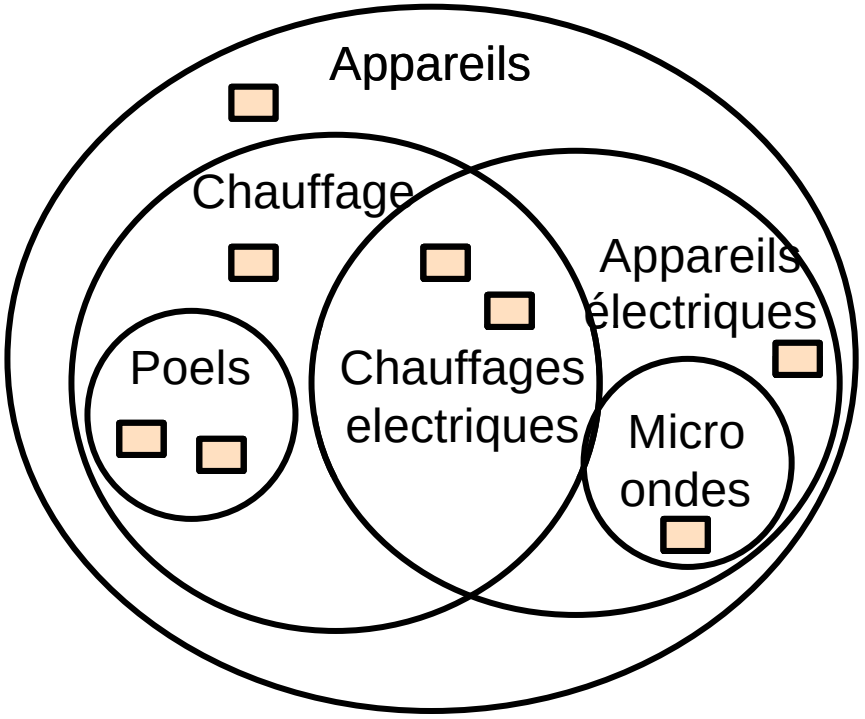


Interdit dans certains langages de programmation (p.e. Java et C#)

Héritage multiple

Une classe peut hériter de plusieurs super-classes





Interdit dans certains langages de programmation (p.e. Java et C#)

Points liés à l'héritage et à la classification

Les modèles orientés-objets
ne font pas tous les mêmes hypothèses

Héritage simple vs. héritage multiple
Une classe peut elle hériter de plusieurs classes ?

Classification simple vs. classification multiple
Un objet peut-il être simultanément
instance de plusieurs classes?

Classification statique vs. classification dynamique
Un objet peut-il changer de classe pendant l'exécution ?

Hypothèses UML par défault

Sauf si le contraire est indiqué explicitement,
en UML les hypothèses par défaut sont :

Héritage multiple
: une classe peut hériter de plusieurs classes

Classification simple
: un objet est instance d'une seule classe

Classification statique
: un objet est créé à partir d'une classe donnée
et n'en change pas