

UML

(Unified Modeling Language)



Objectifs du module

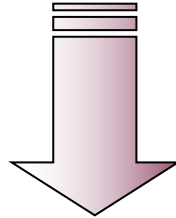
- Comprendre les fondements de base de UML
- Pouvoir utiliser et appliquer UML dans des cas réels
- Apprendre l'Outil: Visual Paradigm for UML

Introduction

- Des logiciels complexes
- Logiciels de grande taille:
 - des millions de lignes de code
 - des équipes nombreuses
 - durée de vie importante
 - des lignes de produits
 - plateformes technologiques complexes
 - évolution continue
- Logiciels critiques
- ...

Introduction

La complexité des logiciels



La crise des logiciels

Les méthodes d'analyse

- Méthodes cartésiennes (Les années 70):
 - Orientées traitement
- Méthodes systémiques (Les années 80) :
 - Orientées données
- Méthodes orientées objets (Les années 90) :
 - On ne sépare pas les données et les traitements ex : Booch, OMT...

Méthodes cartésiennes

- Méthodes d'analyse fonctionnelles
 - décomposition d'une fonction en sous fonctions jusqu'à atteindre un niveau facile à coder
- Exemples: méthodes de programmation structurée, Jackson...

Méthodes systémiques

- Modélisation des données et des traitements
- Séparation entre données et traitements
- Méthodes: Merise,...

Méthodes orientées objets

- Consistent à créer une représentation informatique des éléments du monde réel auxquels on s'intéresse, sans se préoccuper de l'implémentation
- Ils permettent également l'organisation des systèmes d'information en un ensemble d'objets incorporant à la fois la structure des données et le comportement

Historique

Début des années 1990

- les premiers processus de développement **OO** apparaissent

■ Entre 1990 et 1994 : Plus de 50 méthodes objet sont apparues:

- méthode **OOD** de Grady Booch (1991)
- méthode OMT de James Rumbaugh (1991)
- méthode OOSE de Ivar Jacobson (1991)
- méthode OOA/OOD de Coad and Yourdon (1992)
- méthode de Schlaer and Mellor (1992)
- Etc.

Grady Booch et OOD

Description

- **OOD** signifie « Object Oriented Design ».
- Cette méthode a été créée en **1993** par Grady Booch, alors qu'il travaillait chez General Electric pour faciliter la phase de conception orientée objet des gros projets.
- Cette méthode propose des vues logiques et physiques du système.



Ivar Jacobson et OOSE

Description

- **OOSE** signifie « Object Oriented Software Engineering ».
- Cette méthode, créée en **1995** par Ivar Jacobson dans le cadre de ses activités chez Ericsson, introduit la notion de *use-cases* (cas d'utilisation).



John Rumbaugh et OMT

Description

- **OMT** est l'acronyme de « Object Modeling Technique ».
- John Rumbaugh a créé cette méthode en **1996** et a commercialisé un logiciel appelé **Rational Rose** (de la société Rational Rose Software) qui est une référence dans le domaine de la modélisation.
- Cette méthode propose des vues statiques, dynamiques et fonctionnelles d'un système.



Historique

Fin 1994

- J. Rumbaugh rejoint G. Booch chez Rational Software
- OMT + OOD → **Unified Method** (*oct 1995*)

Fin 1995

- I. Jacobson les rejoint chez Rational Software
- Unified Method + OOSE → **UML 0.9** (*juin 1996*)

Début 1997

- Partenaires divers : Microsoft, Oracle, IBM, HP et autres leaders collaborent
- → **UML 1.0** (*jan 1997*)

Fin 1997

- l'OMG (Object Management Group) retient **UML 1.1** comme **norme de modélisation**

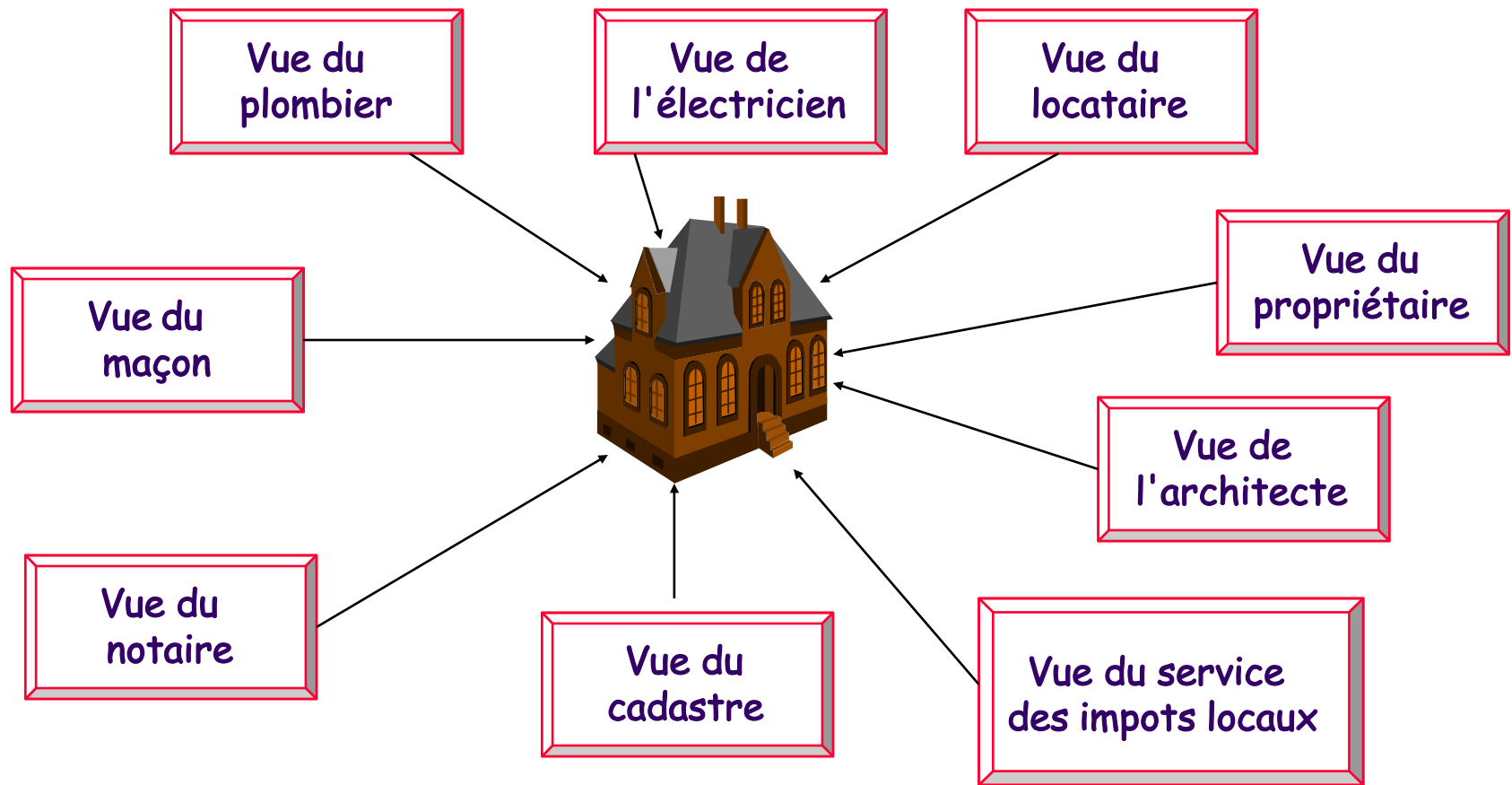
Plusieurs types de notations

- Notations graphiques
 - Notation textuelle
-
- Signification plus ou moins précise
 - Notation standard (mais pas toujours respectée)
 - Notation extensible

Remarque

- Notation pas toujours suffisante

Vues multiples (aspects d'un système logiciel)



L'arrivée d'UML

La normalisation

- **UML** devient une norme de l'OMG en **1997**.
- L'OMG (Object Management Group) est un organisme créé en 1989 afin de promouvoir des standards (comme CORBA par exemple) qui garantissent l'interopérabilité entre des applications orientées objet développées sur des réseaux hétérogènes.
- Cet organisme a été créé et est soutenu par des industriels comme HP, Sun, Unisys, American Airlines, Philips ...

L'arrivée d'UML

Au final, qu'est-ce qu'UML ?

UML : Unified Modeling Language

- Langage de Modélisation Unifié.
- Appliqué à l'analyse et à la conception des logiciels.
- Langage essentiellement graphique.
- Facile à lire et à comprendre.

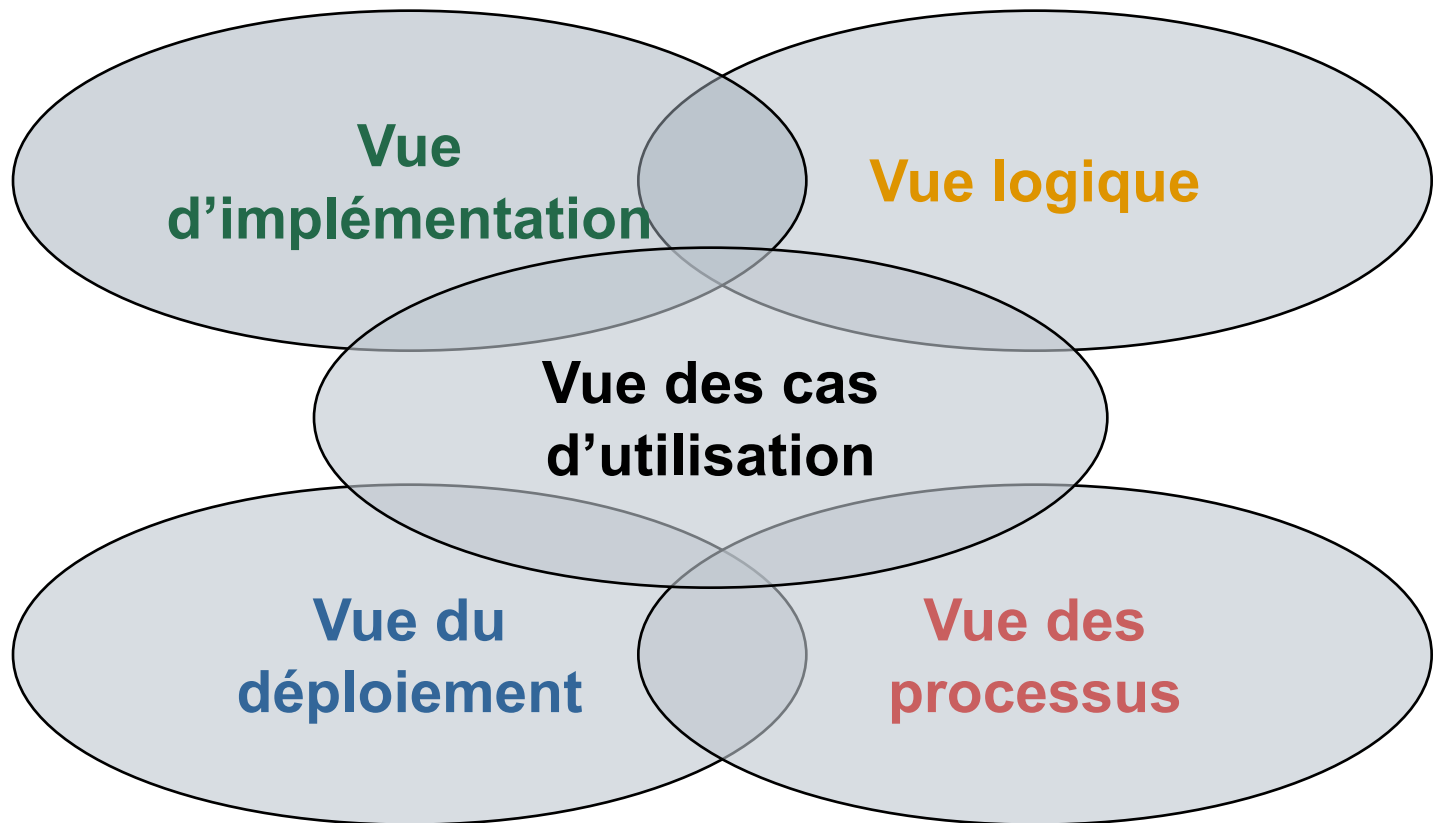
En clair

- UML: norme qui définit les diagrammes et les conventions à utiliser lors de la construction de modèles décrivant la structure et le comportement d'un logiciel.
- Les modèles sont des diagrammes constitués d'éléments graphiques et de texte.
- UML n'est pas une méthode, mais un langage.

L'arrivée d'UML

Les différentes vues

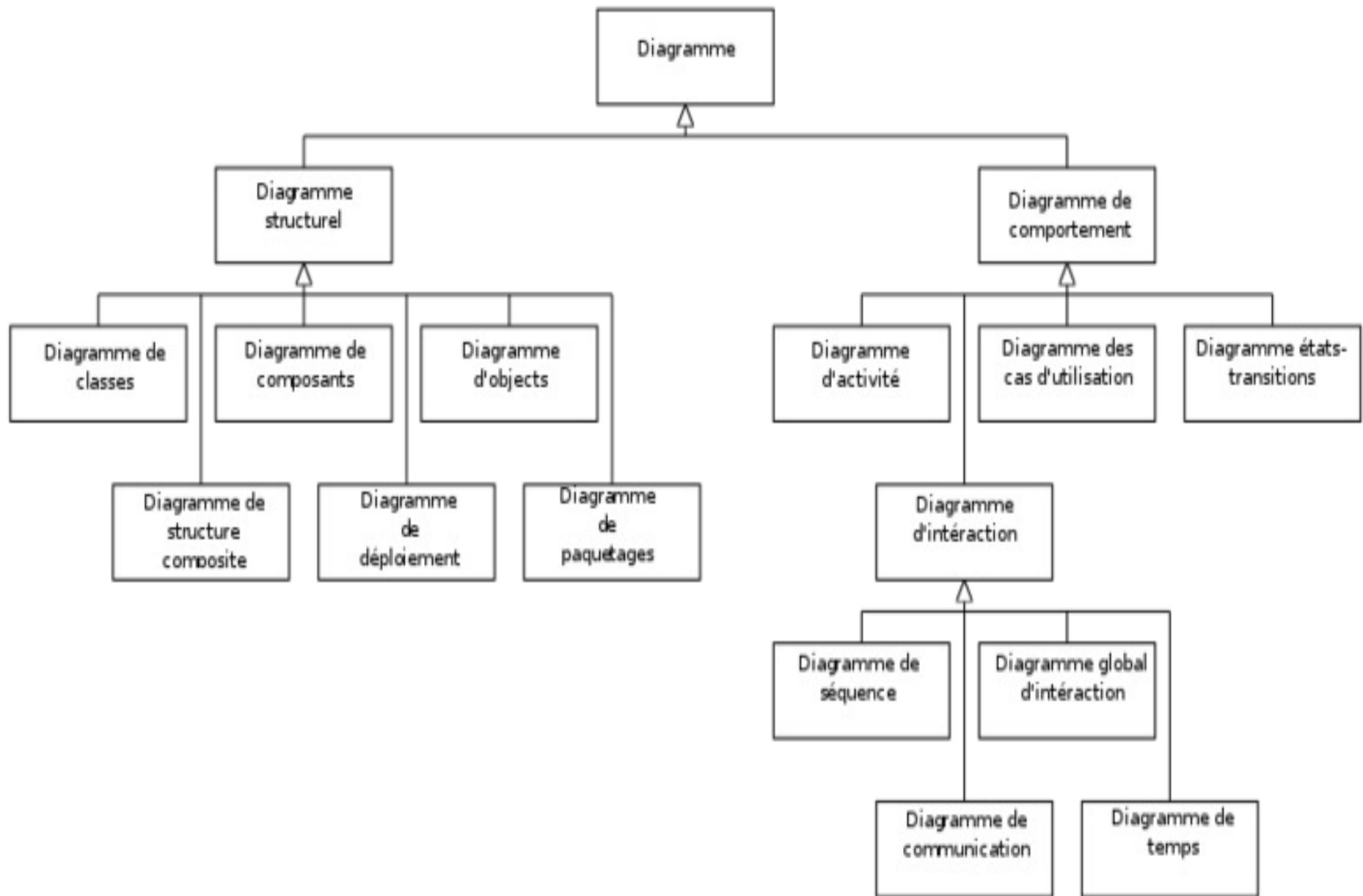
- **UML** propose 5 vues qui se superposent en partie afin de présenter les systèmes sous différents aspects.



L'arrivée d'UML

Les différents diagrammes

- **UML** propose 13 types de diagrammes.
- Ces diagrammes sont présentés dans la norme sous forme d'un diagramme de classes afin de mettre en évidence les deux types de diagrammes :
 - les diagrammes de structure pour modéliser l'aspect **statique** d'un système ;
 - les diagrammes de comportement pour modéliser l'aspect plutôt **dynamique** d'un système.



Les diagrammes d'UML

- **Diagramme de cas d'utilisation:** représentation des fonctions du système du point de vue de l'utilisateur.
- **Diagramme de classe:** description graphique des différentes classes que le système utilise ainsi que leurs liens
- **Diagramme d'objets** permet d'éclairer un diagramme de classes en l'illustrant par des exemples.

Les diagrammes d'UML

- ***Le diagramme de séquence***: représentation temporelle des interactions entre objets
- ***Le diagramme de communication***: représentation spatiale des objets, des liens et des interactions
- ***Diagramme d'état transition***: représentation de la façon dont évoluent les objets appartenant à une même classe.
- ***Les diagramme d'activités*** : représentation du comportement d'une opération en terme d'action

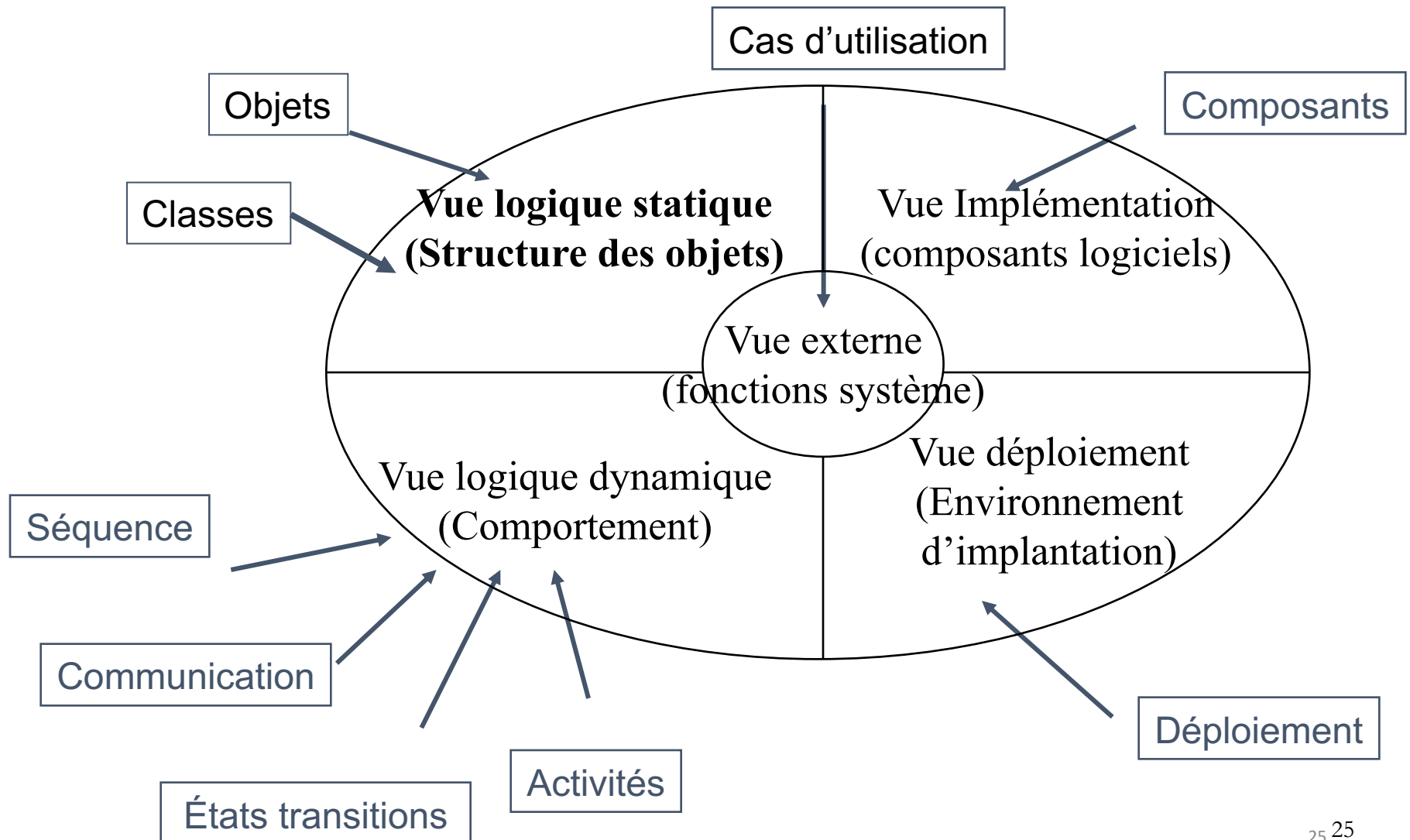
Les diagrammes d'UML

- ***Diagramme de composants*** : composants logiciels réalisant l'application (code source, bibliothèques, dépendances, etc.) ;
- ***diagramme de déploiement*** : répartition des composants logiciels sur des matériels.
- ***Diagramme des paquetages*** : représentation des dépendances entre paquetages;
- ***Diagramme de structure composite*** : depuis UML 2.x, permet de décrire sous forme de boîte blanche les relations entre composants d'une classe.

Les diagrammes d'UML

- **Diagramme de communication** : depuis UML 2.x, représentation simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets.
- **Diagramme de temps** : depuis UML 2.x, permet de décrire les variations d'une donnée au cours du temps.

Diagrammes d'UML



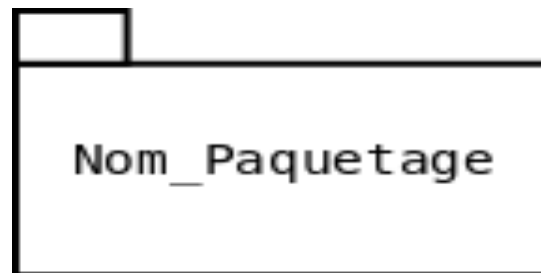
Logiciels de modélisation UML

- Il existe de nombreux outils logiciels de modélisation UML.
- Aucun d'entre eux ne respecte strictement aucune des versions de UML, particulièrement UML2
- Logiciels open-source: ArgoUML, Papyrus UML, StarUML, BOUML...
- Logiciels payants: Rational Rose ,EDGE Diagrammer, Visual Paradigm
...

Les éléments de la modélisation UML

Les paquets

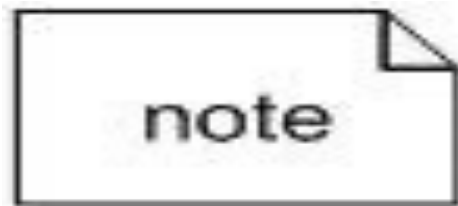
- Un paquetage est un regroupement logique de différents éléments de la modélisation
- Un paquetage peut contenir d'autres sous paquets
- Chaque paquetage doit avoir un nom



Les éléments de la modélisation UML

Les notes

- Commentaire attaché à un ou plusieurs éléments de modélisation



Les éléments de la modélisation UML

Les stéréotypes

- Les stéréotypes permettent d'étendre la sémantique des éléments de modélisation UML
- Représentation:

`<<nomStéréotype>>`

- UML propose de nombreux stéréotypes standards: `<<include>>`
`<<extend>>` `<<utility>>`...

Différentes catégories d'élément dans un modèle UML

- Stéréotype :
 - s'applique sur un élément de modèle pour caractériser des variétés d'un même concept.
 - Représentation : chaîne de caractères entre (« ») dans, ou à proximité du symbole de l'élément de modèle de base.
- Espace de noms : ex: paquetages, classeurs, etc.
 - Un élément nommé est défini par son nom qualifié, qui est constitué de la série des noms des paquetages ou des autres espaces de noms (séparé par deux doubles points (::)) depuis la racine jusqu'à l'élément en question.
 - Ex. A::B::X

Diagramme de Packages

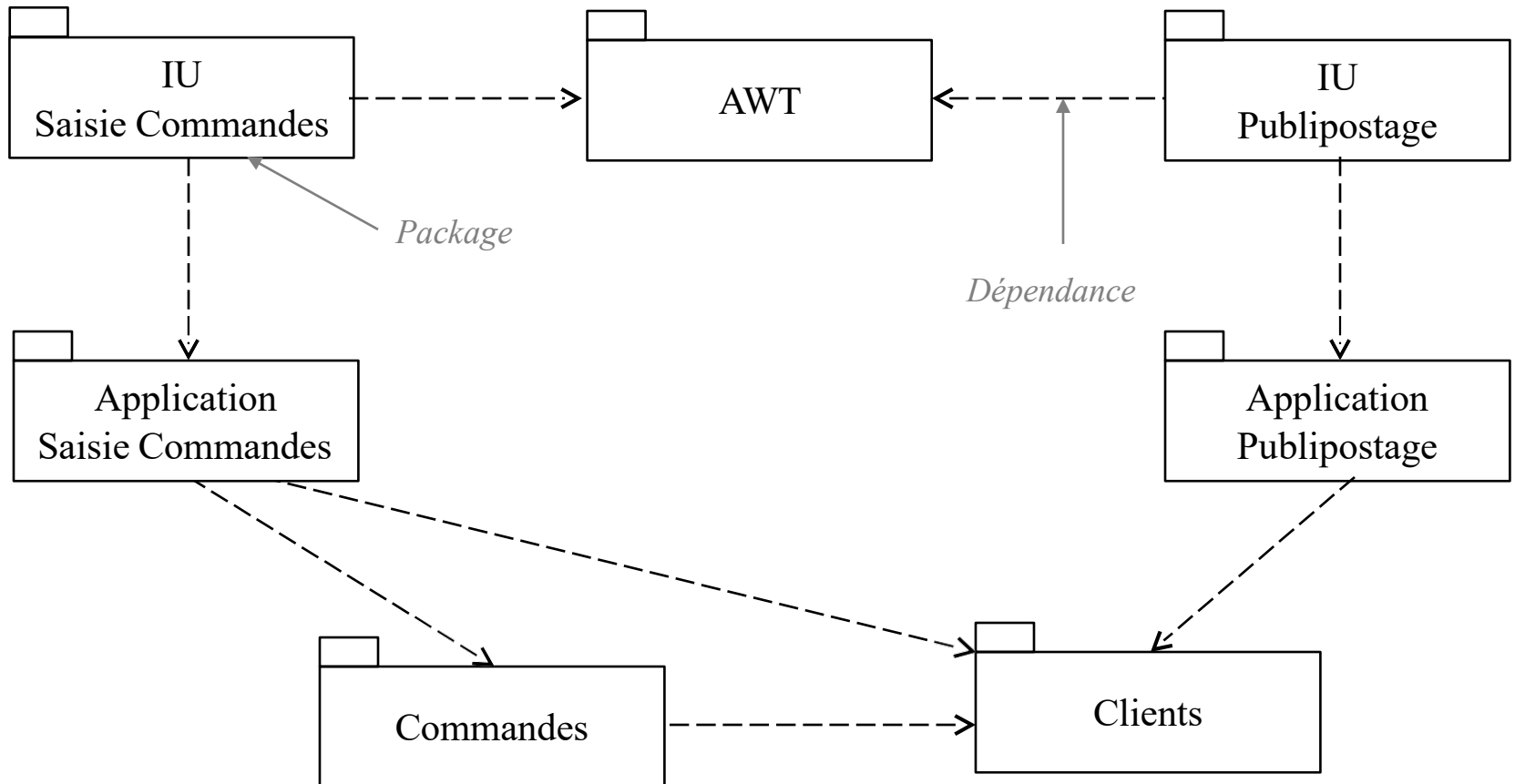
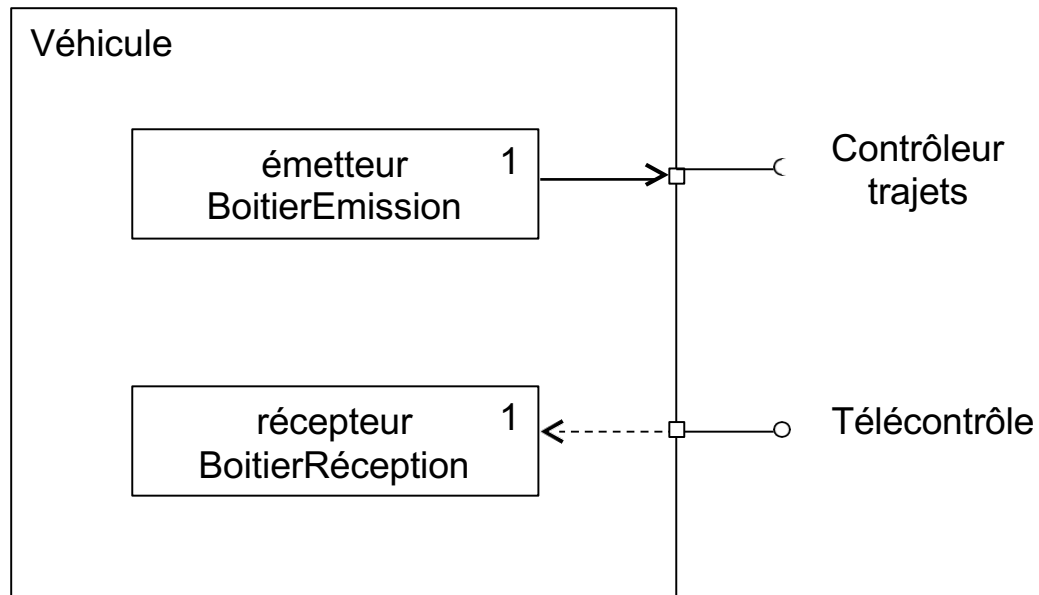


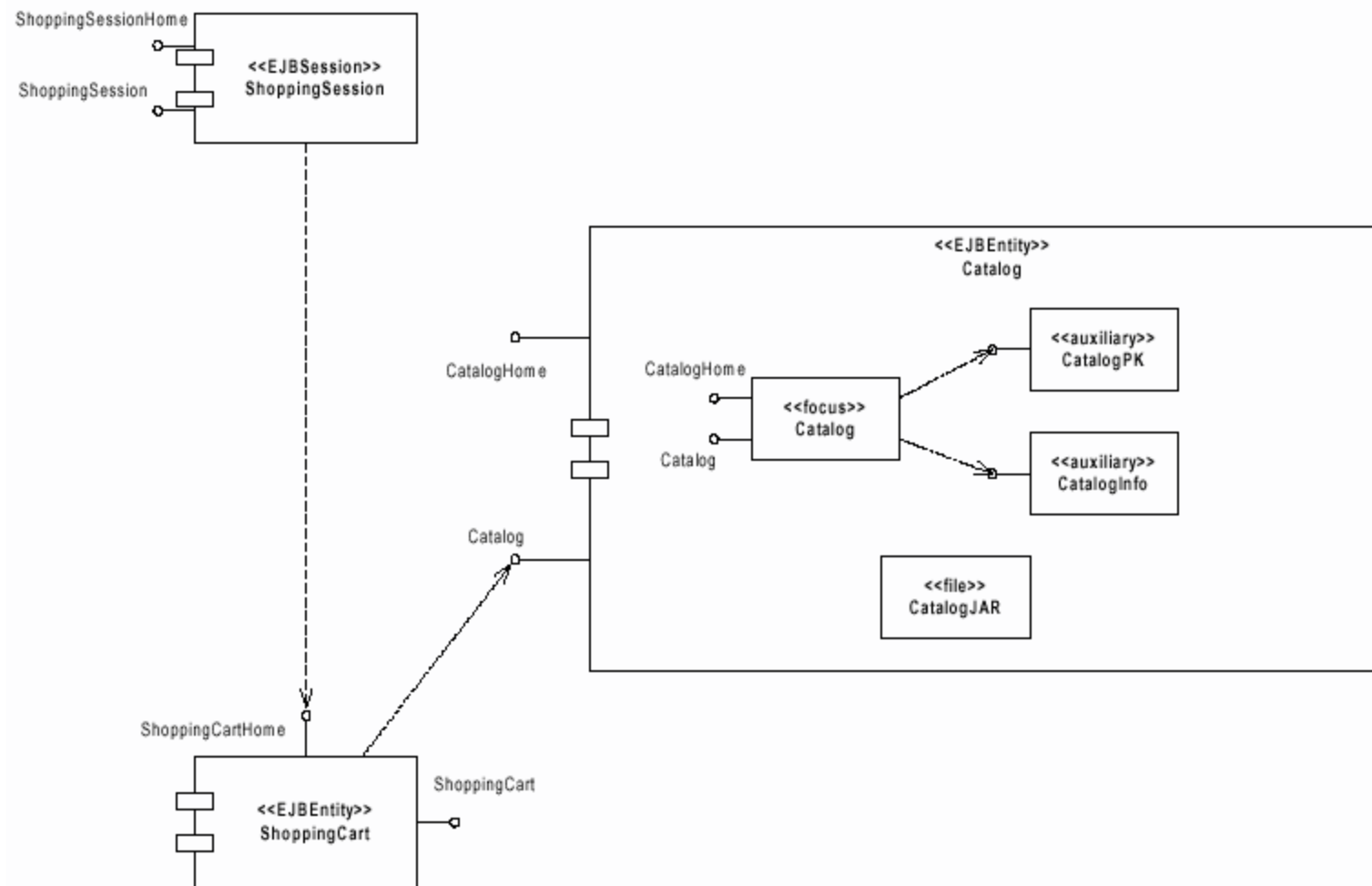
Diagramme de structure composite



Éléments :

- Les parties
- Les ports
- Les connecteurs

Diagrammes de composants



Diagrammes de communication

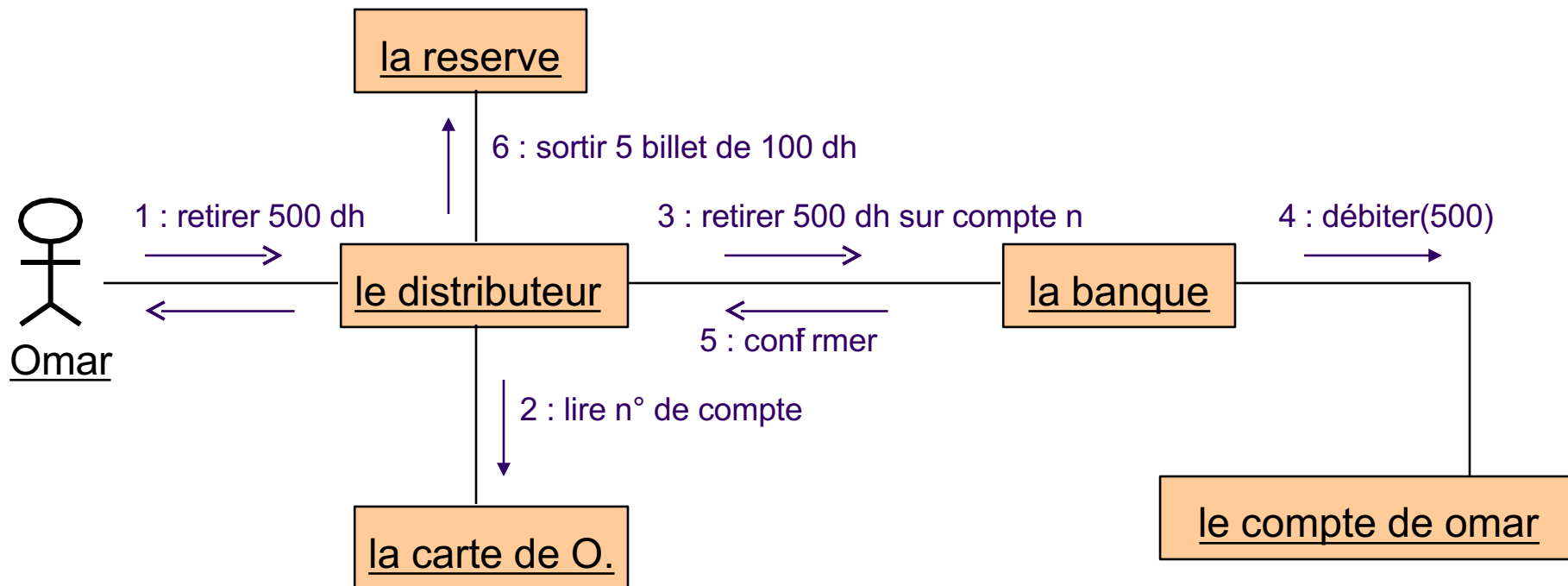
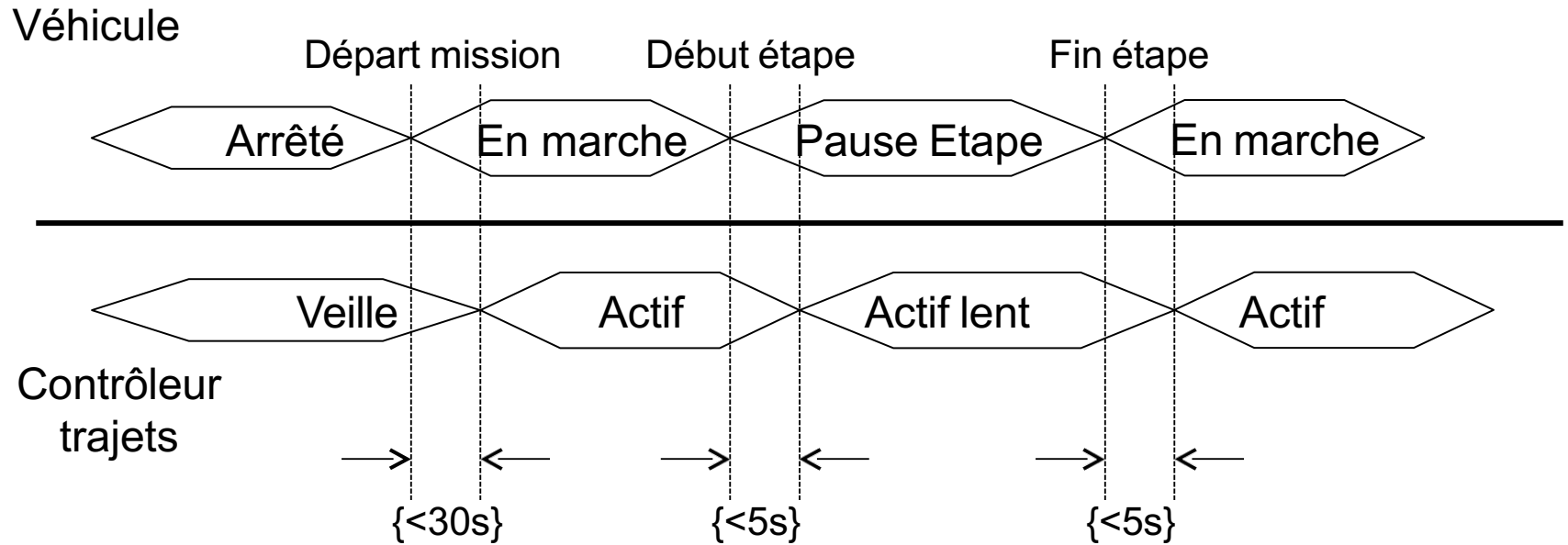
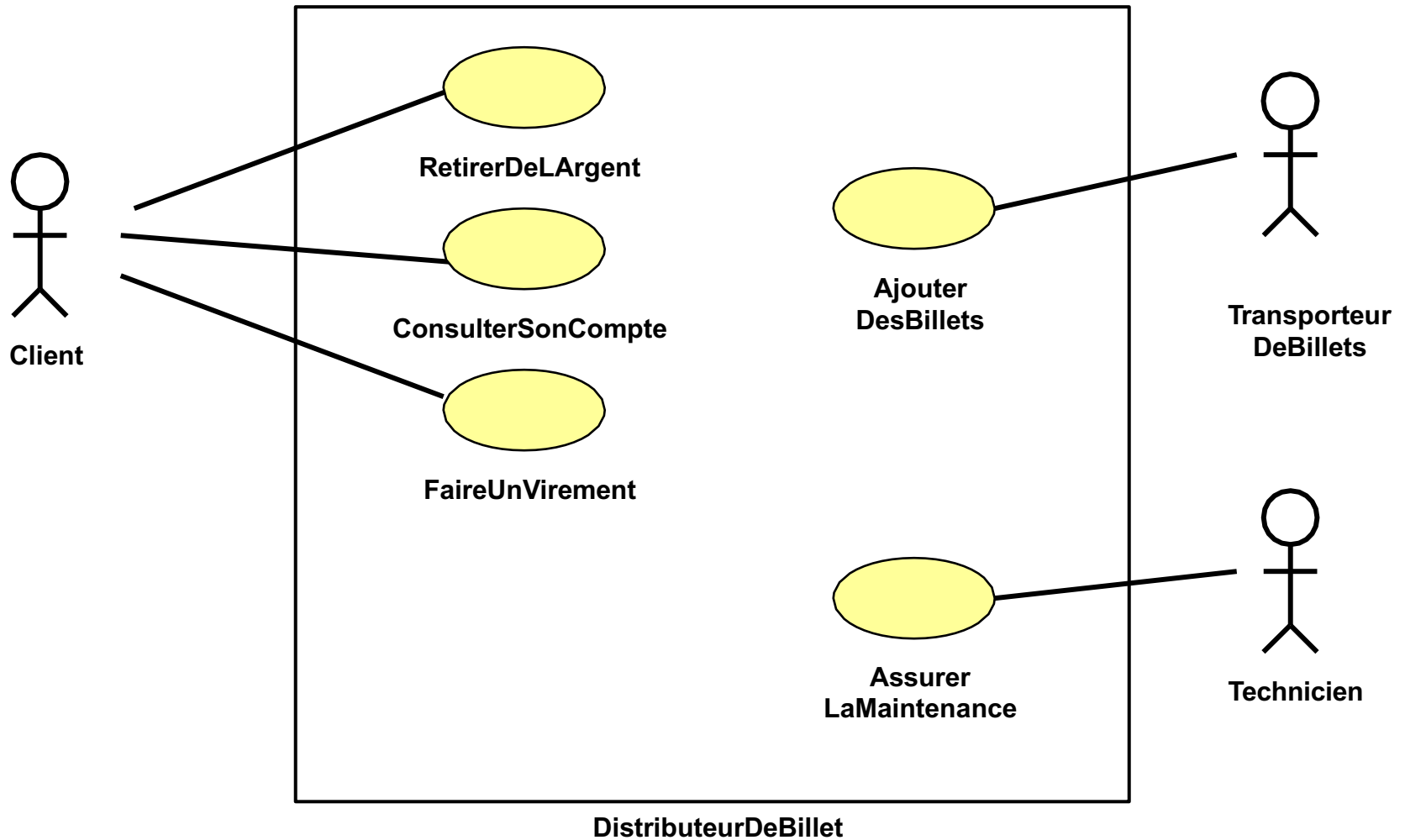


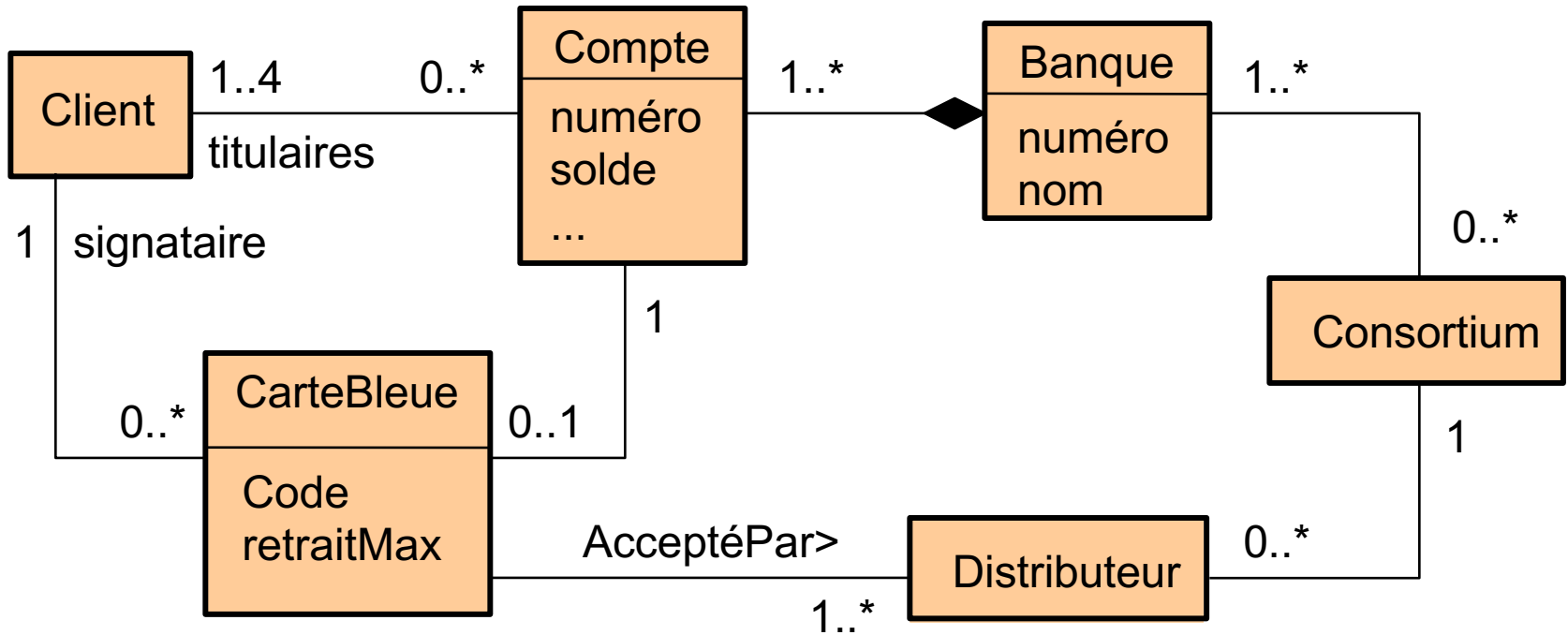
Diagramme de temps



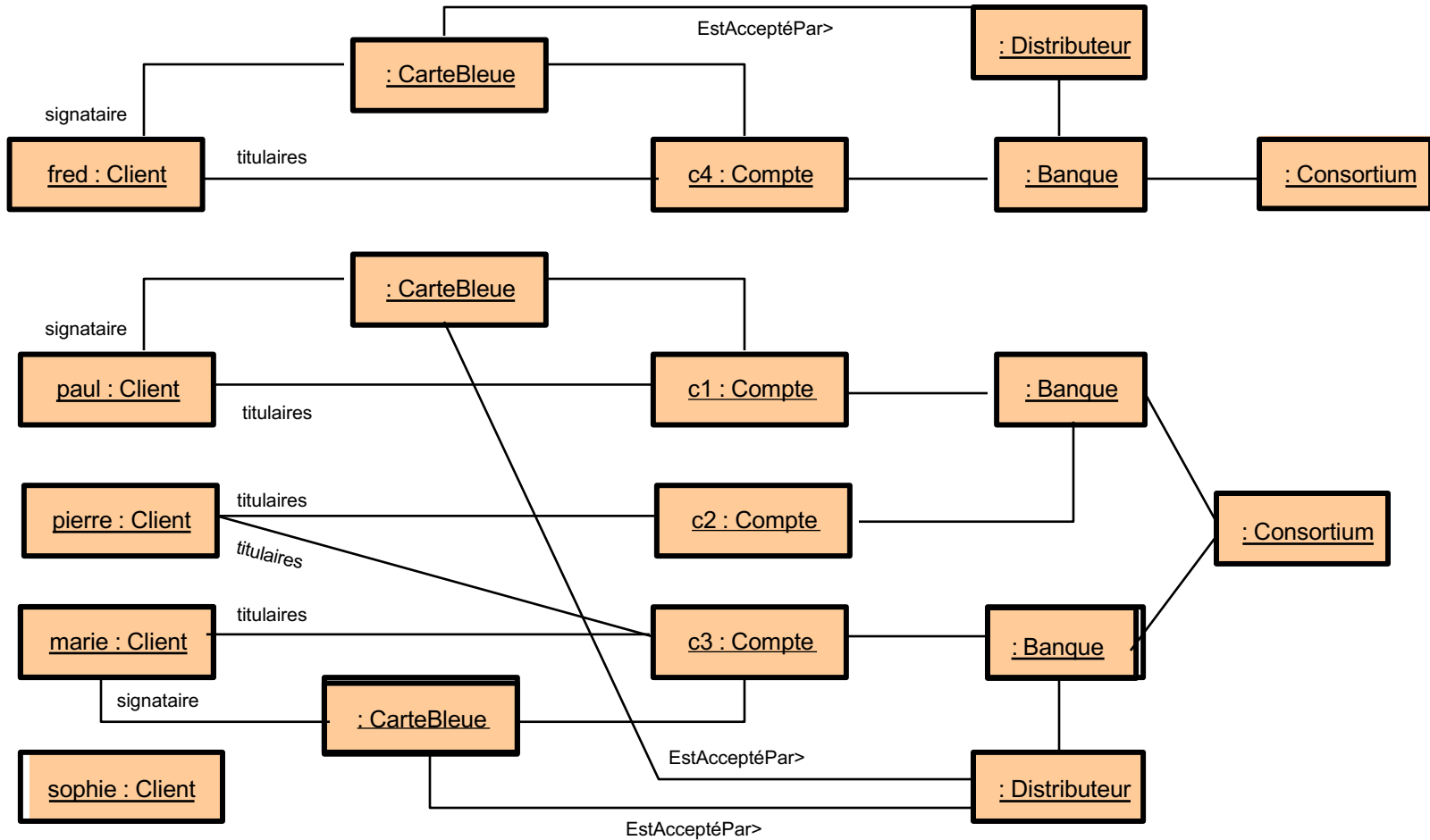
Diagrammes des cas d'utilisation



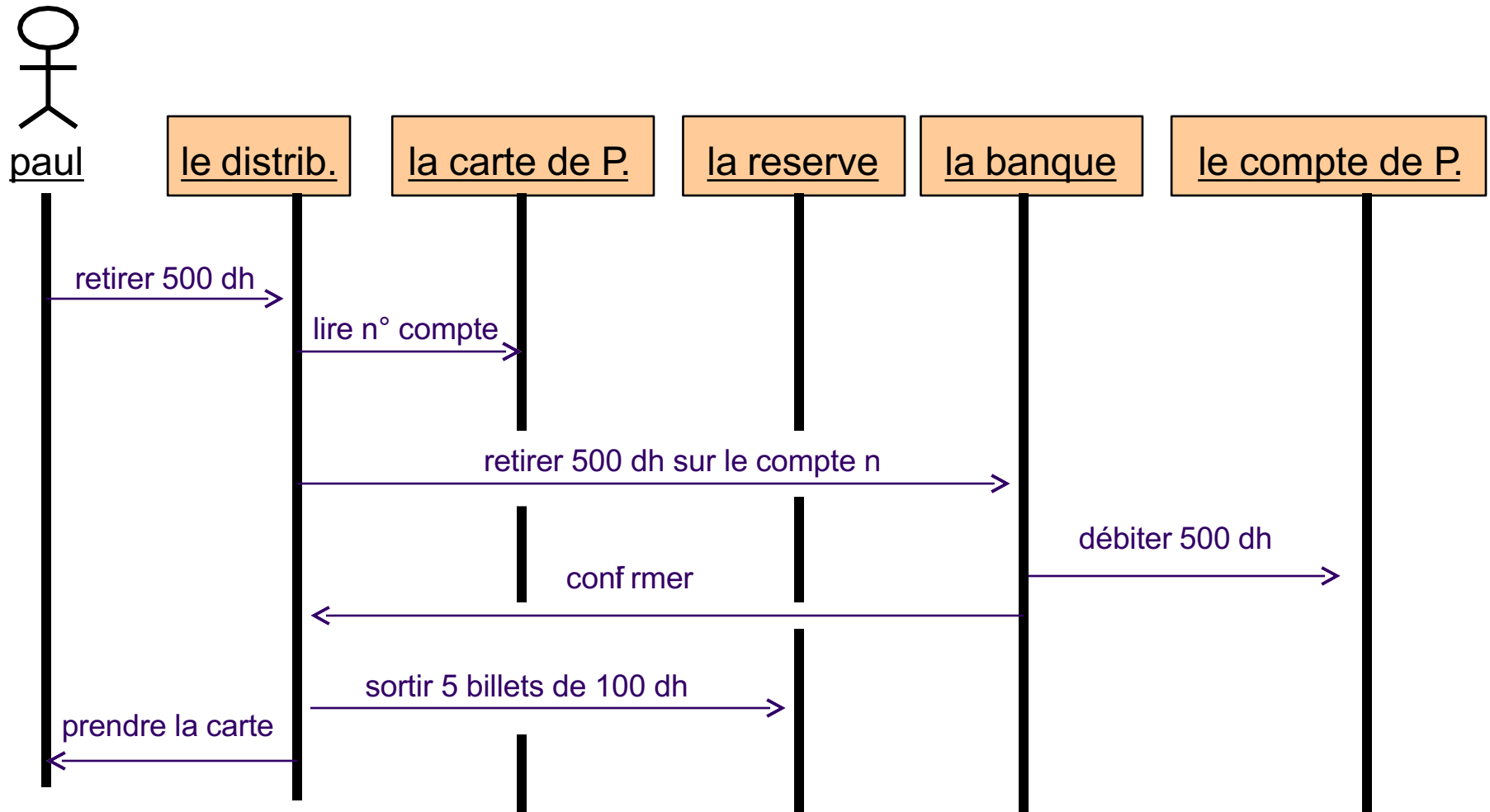
Diagrammes de classes



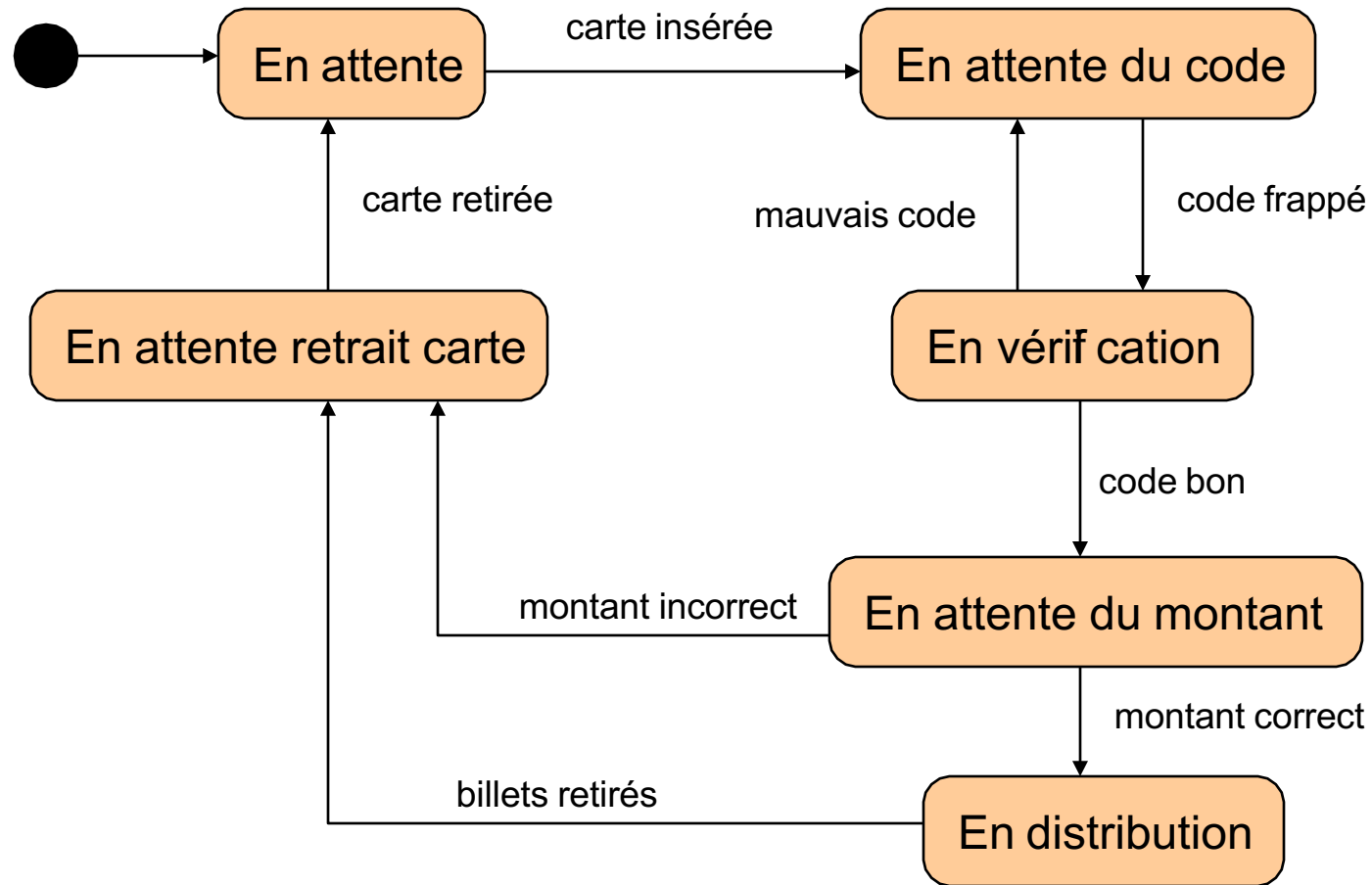
Diagrammes d'objets



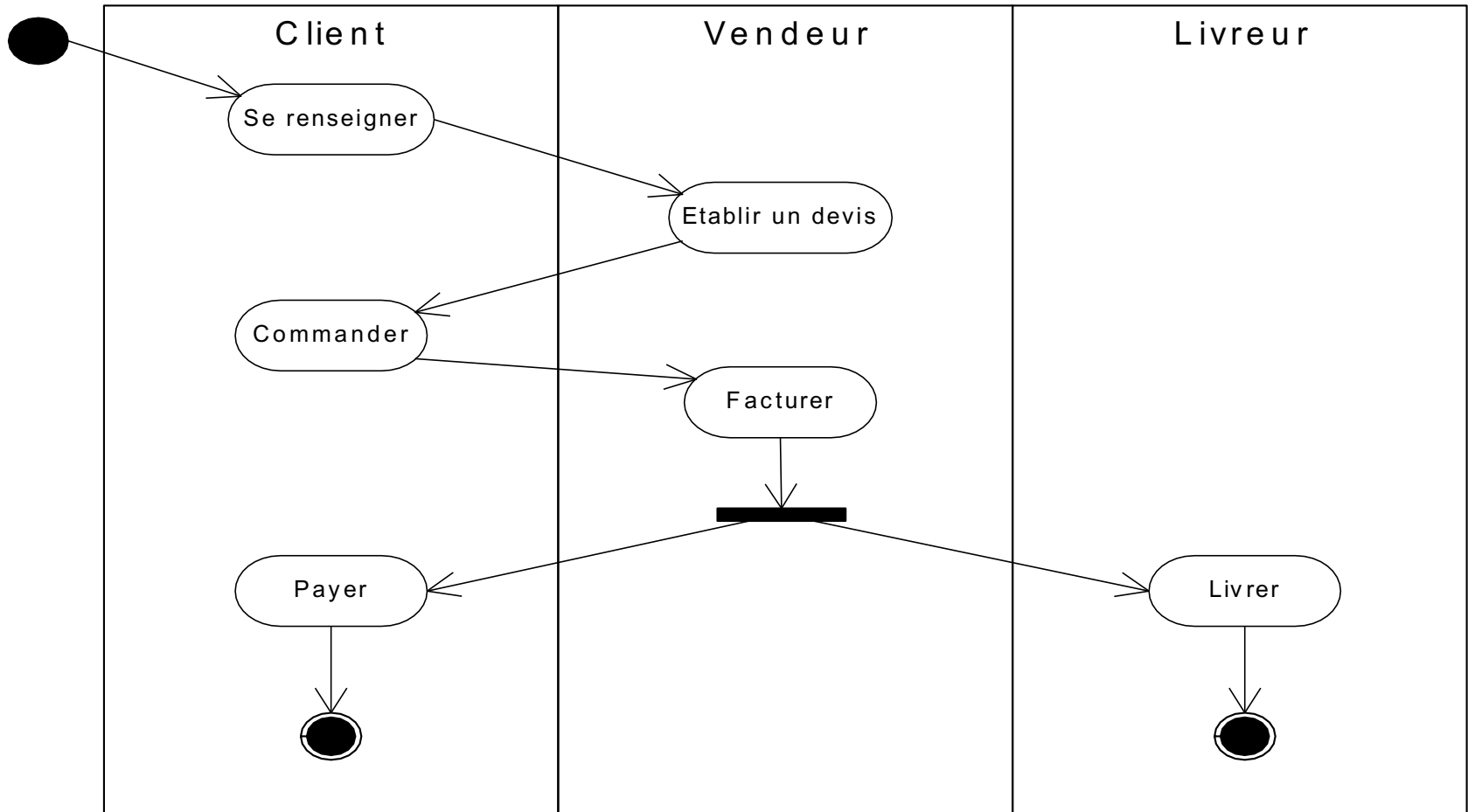
Diagrammes de séquence



Diagrammes d'états



Diagrammes d'activités



Diagrammes de déploiement

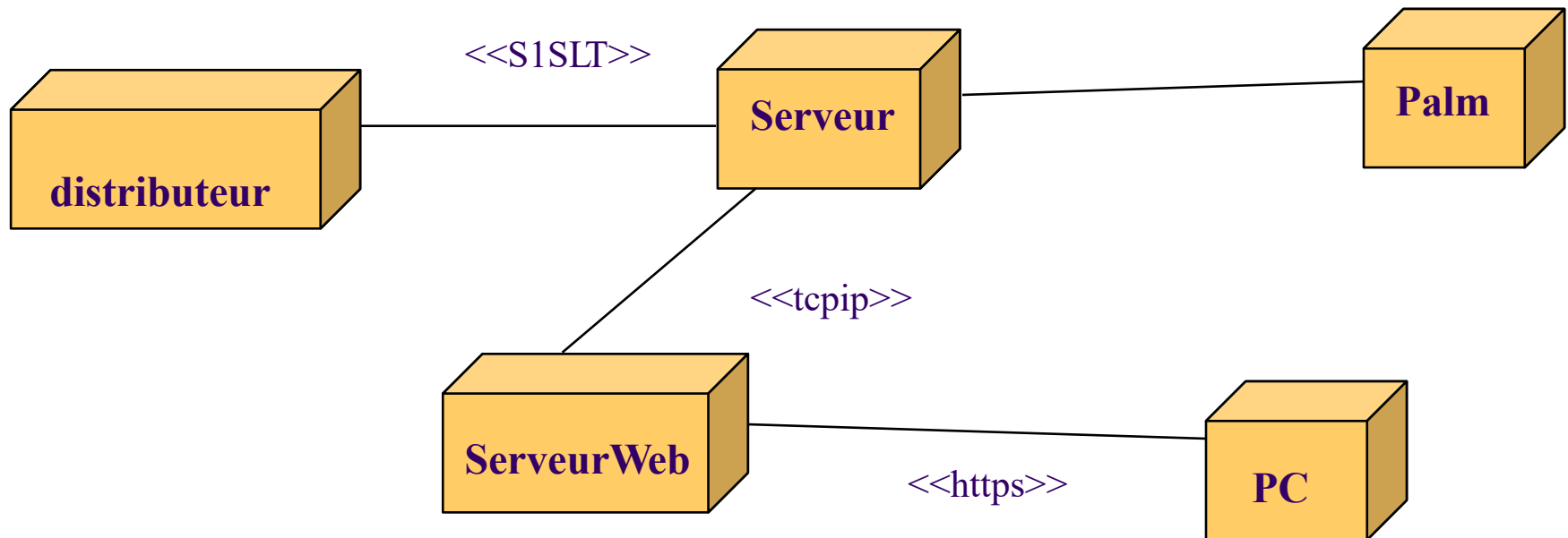


Diagramme de cas d'utilisation (Use Case Diagram)

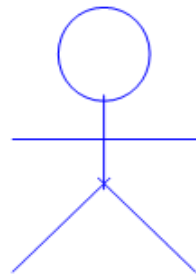
Diagramme de cas d'utilisation (Use Case Diagram)

- Le diagramme de cas d'utilisation permet de recueillir, d'analyser et d'organiser les besoins, et de recenser les grandes fonctionnalités d'un système.

Éléments des diagrammes de cas d'utilisation

Acteur

- Un acteur: est un utilisateur externe du système qui communique et interagit directement avec le système étudié
- Un acteur peut être une personne ou un autre système.



Nom acteur

Éléments des diagrammes de cas d'utilisation

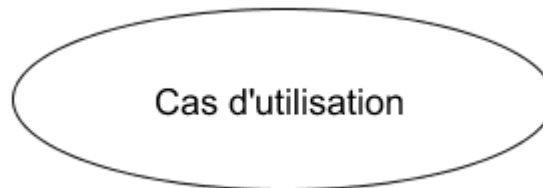
Acteur

- Types d'acteurs :
 - **les acteurs principaux (primaires)** : Les utilisateurs du système
 - **les acteurs secondaires** : Les administrateurs du système
 - **les autres systèmes** : les systèmes avec lesquels le système doit interagir.

Éléments des diagrammes de cas d'utilisation

Cas d'utilisation

- Le cas d'utilisation (ou use case) correspond à une fonctionnalité du système, utilisée par un ou plusieurs acteurs.
- Un cas d'utilisation se représente par une ellipse contenant le nom du cas (un verbe à l'infinitif)



Éléments des diagrammes de cas d'utilisation

La relation entre cas d'utilisation

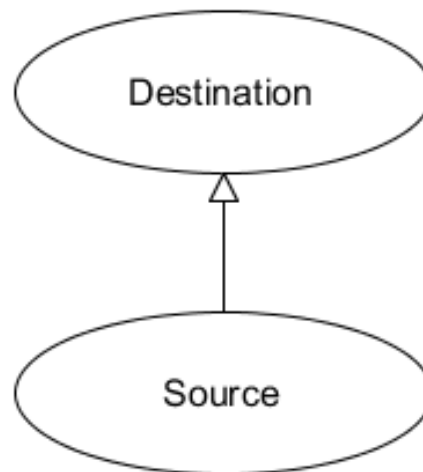
- Il existe 3 types de relations entre cas d'utilisation :
 - La relation de généralisation
 - La relation d'extension
 - la relation d'inclusion

Éléments des diagrammes de cas d'utilisation

La relation entre cas d'utilisation

- ***La relation de généralisation:***

Le cas d'utilisation source hérite le comportement du cas d'utilisation destination

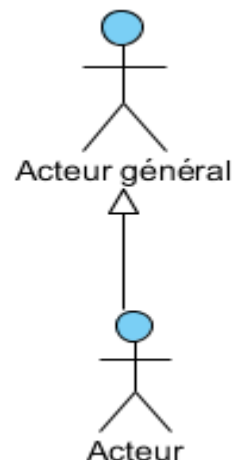


Éléments des diagrammes de cas d'utilisation

La relation entre cas d'utilisation

- ***La relation de généralisation:***

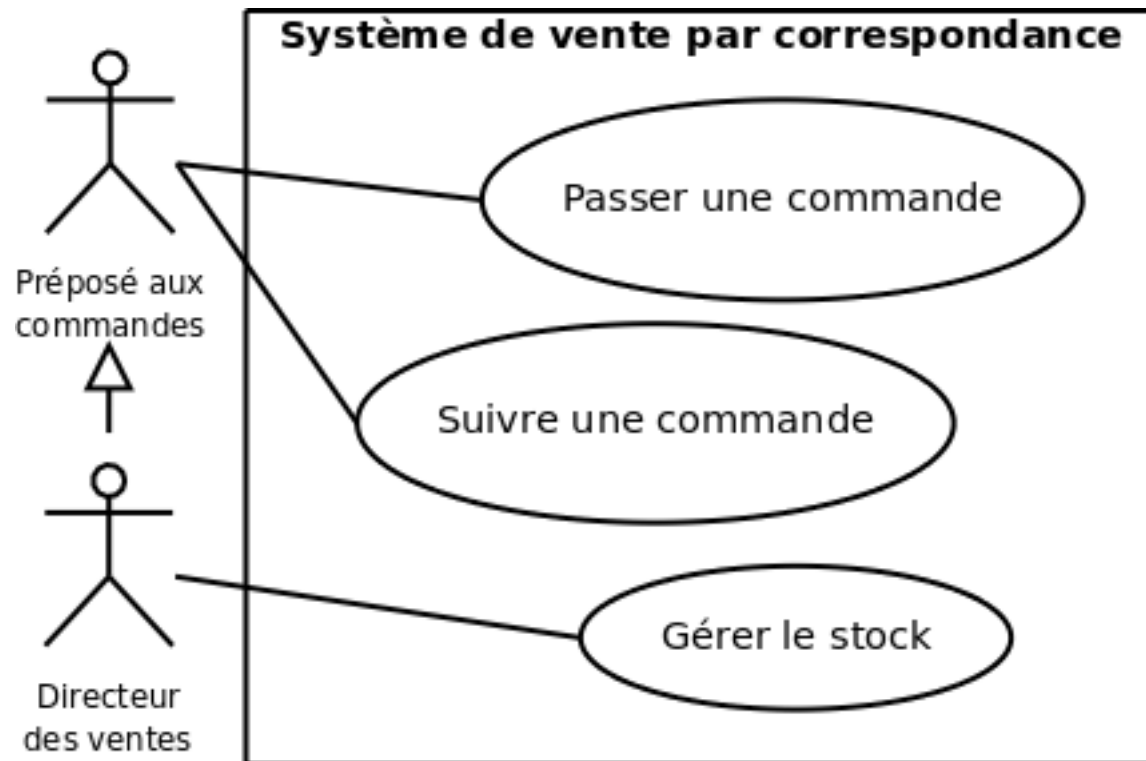
La seule relation possible entre deux acteurs est la généralisation : un acteur A est une généralisation d'un acteur B si l'acteur A peut être substitué par l'acteur B. Dans ce cas, tous les cas d'utilisation accessibles à A le sont aussi à B, mais l'inverse n'est pas vrai.



Éléments des diagrammes de cas d'utilisation

La relation entre cas d'utilisation

- *La relation de généralisation:*

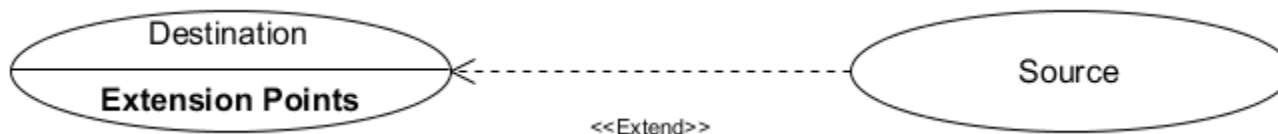


Éléments des diagrammes de cas d'utilisation

La relation entre cas d'utilisation

- ***La relation d'extension :***

Le cas d'utilisation source étend cad ajoute son comportement (optionnellement) au comportement du cas d'utilisation destination

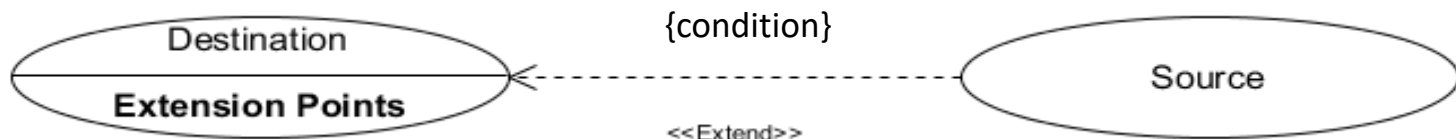


Éléments des diagrammes de cas d'utilisation

La relation entre cas d'utilisation

- ***La relation d'extension :***

- Le point où se passe l'extension peut être précisé
- L'extension peut être soumise à une condition

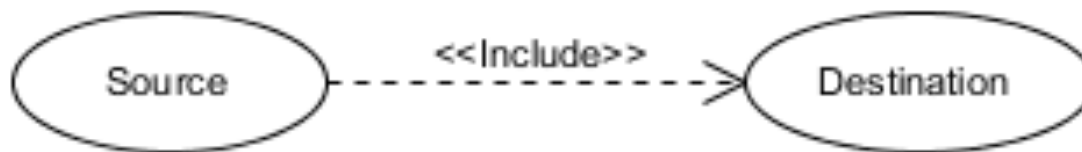


Éléments des diagrammes de cas d'utilisation

La relation entre cas d'utilisation

- ***La relation d'inclusion*** :

Le cas d'utilisation source inclut le cas d'utilisation destination **obligatoirement** le comportement du cas d'utilisation destination



Les diagrammes de cas d'utilisation

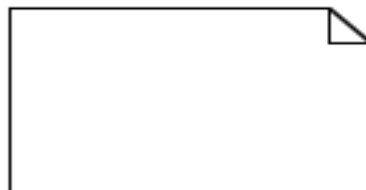
Démarche

1. Identifier et décrire les acteurs
2. Identifier et décrire les cas d'utilisation
3. Structurer les cas d'utilisation en package

Cas d'utilisation

Description textuelle

- Une description textuelle d'un cas d'utilisation comprend:
 - Les acteurs
 - Les pré-conditions: L'ensemble des conditions qui doivent être satisfaites avant de déclencher le cas d'utilisation
 - Les post-conditions: L'état du système après le déroulement du cas d'utilisation



Cas d'utilisation

Description textuelle

Les scénarios

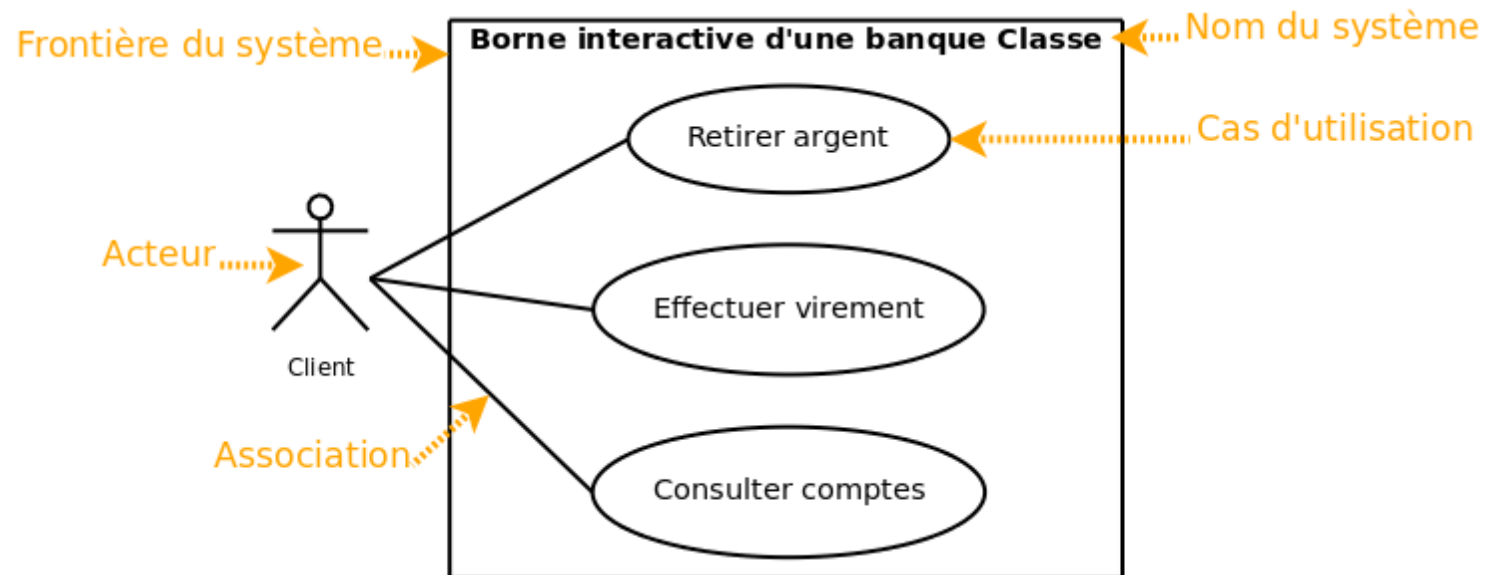
- Définition:
un **scénario** est une séquence d'actions exécutées par un système qui produit un résultat observable par un acteur particulier
- Un cas d'utilisation est décrit par un scénario nominal et plusieurs scénarios alternatifs

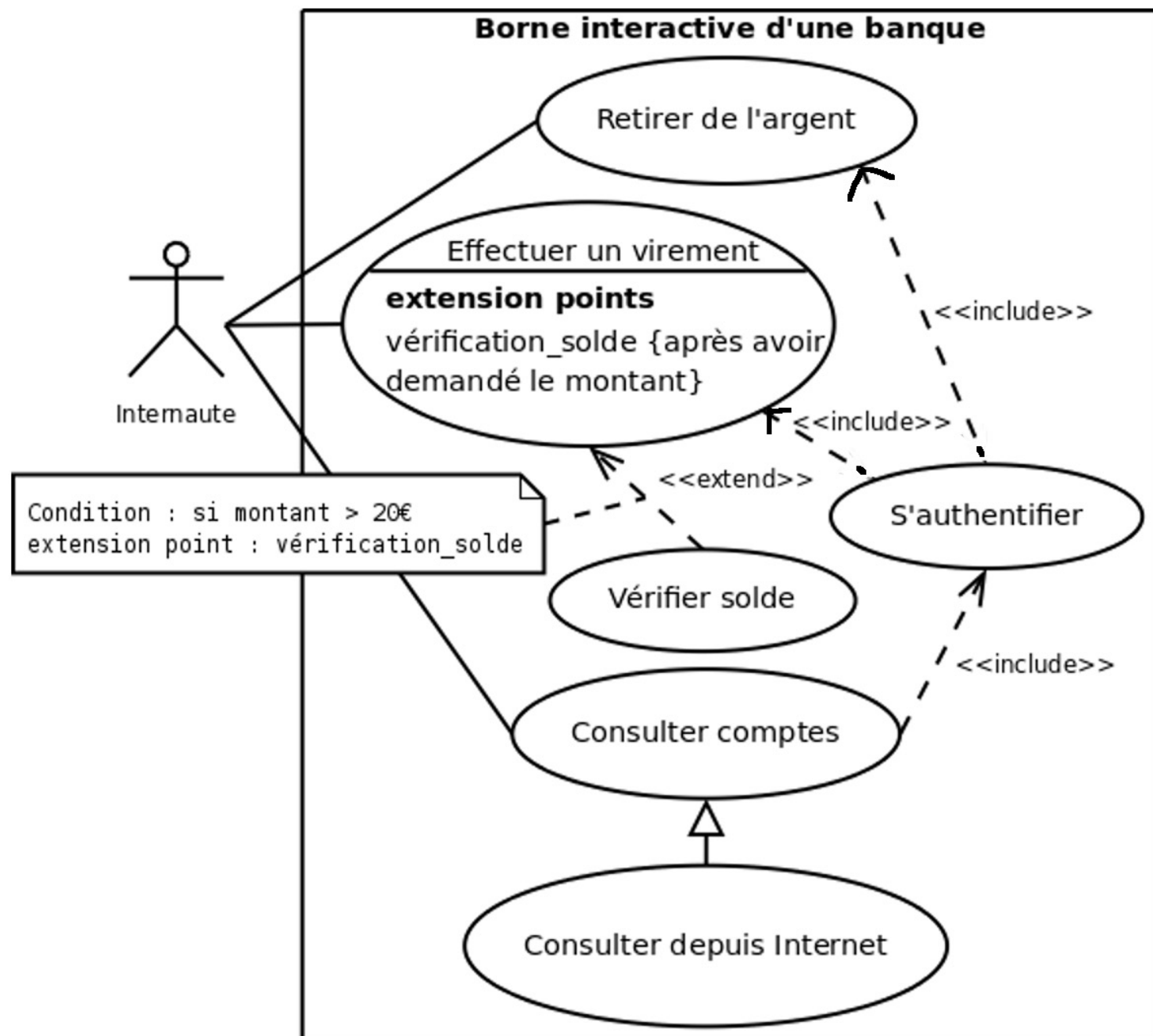
Les scénarios

- Un scénario peut être présenté dans un tableau de la forme suivante:

Actions des acteurs	Actions du système
1. L'acteur déclanche...	2. Le système répond...
3. l'acteur choisi...	4. Le système répond...
....

Exemple :



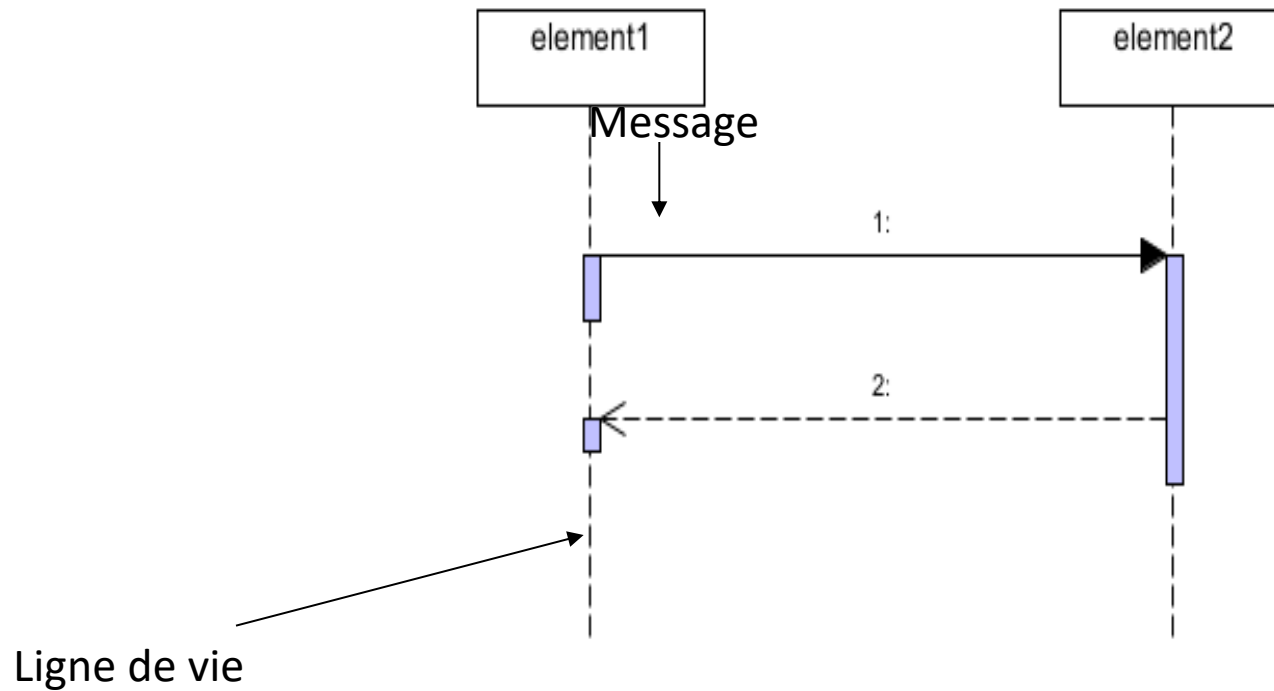


Le diagramme de séquence

Le diagramme de séquence

- Un diagramme de séquence permet d'illustrer les interactions des acteurs avec le système
- Les acteurs et le système communiquent entre-eux par envoi de [messages](#).

Le diagramme de séquence



Le diagramme de séquence

Types de messages

1. Message synchrone

- Dans un message synchrone, l'émetteur reste bloqué le temps que le récepteur traite le message envoyé ;
- Un message synchrone se représente par une flèche en traits pleins et à l'extrémité pleine
- Ce message peut être suivi d'une réponse qui se représente par une flèche en pointillé.



Le diagramme de séquence

Types de messages

2. Message asynchrone

- Dans un message asynchrone : l'émetteur n'est pas bloqué lorsque le récepteur traite le message envoyé.
- Un message asynchrone se représente par une flèche en traits pleins et à l'extrémité ouverte

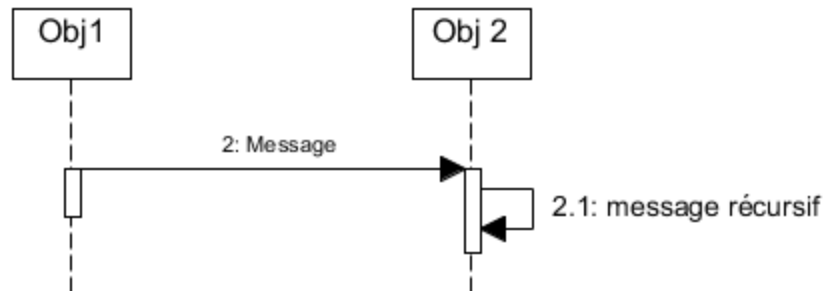


Le diagramme de séquence

Types de messages

3. Message récursif

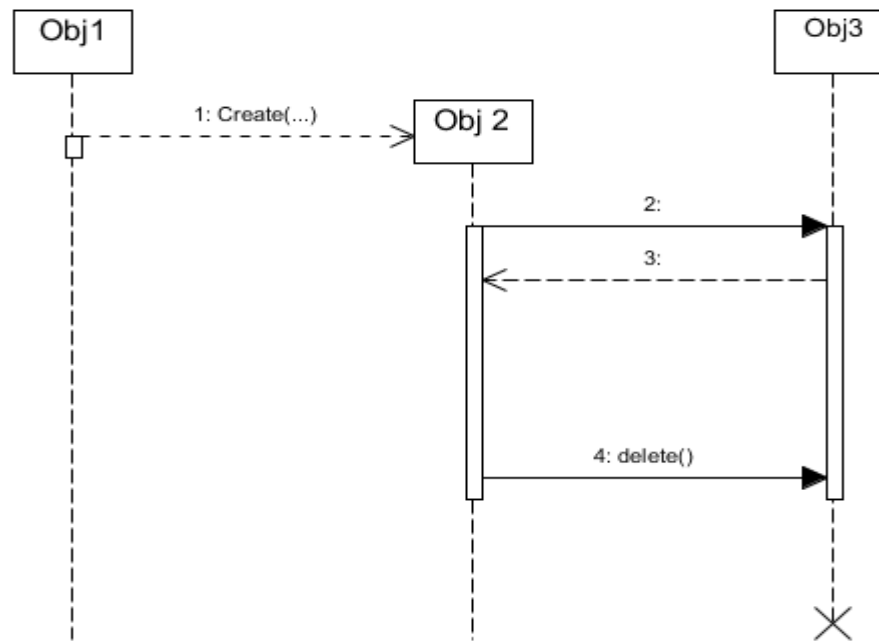
- Un message récursif est un message qu'un objet s'envoie à lui-même.



Le diagramme de séquence

Types de messages

4. Message création/destruction d'un objet



Le diagramme de séquence

Types de messages



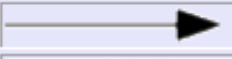
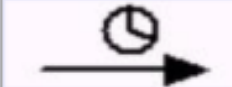

4. Message création/destruction d'un objet

- La création d'un objet est matérialisée par une flèche qui pointe sur le sommet d'une ligne de vie.
- La destruction d'un objet est matérialisée par une croix qui marque la fin de la ligne de vie de l'objet.

Le diagramme de séquence

Types de messages

5. Autres messages

Symbol	Meaning
	message simple qui peut être synchrone ou asynchrone
	Retour simple (optionnel)
	message synchrone
	message minuté [<i>TimeOut</i>]
	message asynchrone

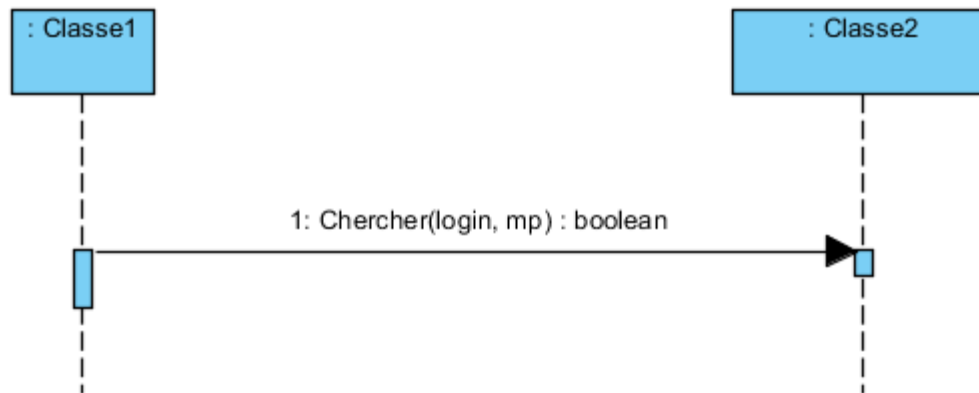
Le diagramme de séquence

Message sous forme de méthode

- Dans la plupart des cas, la réception d'un message est suivie de l'exécution d'une méthode d'une classe.
- Cette méthode peut recevoir des arguments et la syntaxe des messages permet de transmettre ces arguments.

Le diagramme de séquence

Message sous forme de méthode



Les diagrammes de séquence

Structures de contrôle

Le diagramme de séquences peut inclure un certain nombre de structures

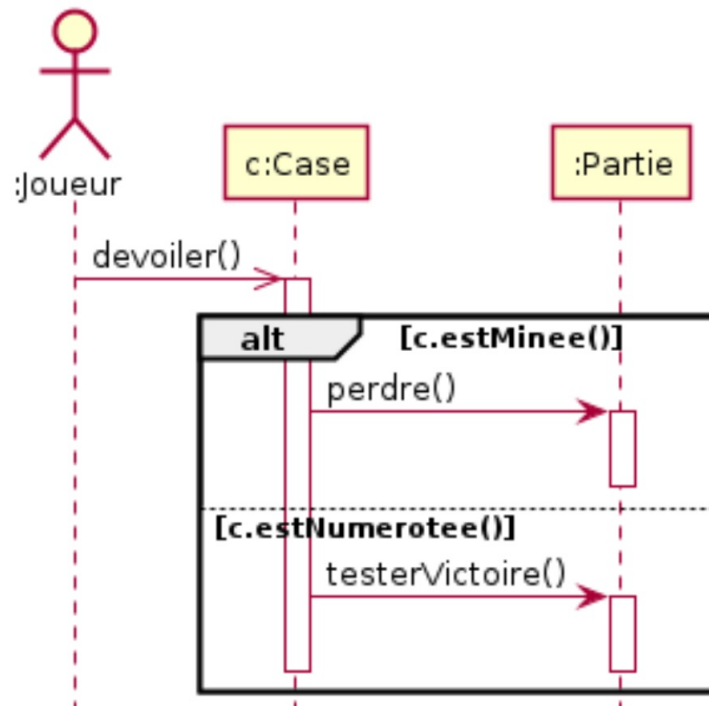
- Les tests
- Répétitions (itérations, boucles)

Le diagramme de séquence

Fragments

Les différentes alternatives sont spécifiées dans des zones délimitées par des pointillés.

- Les conditions sont spécifiées entre crochets dans chaque zone.
- On peut utiliser une clause [else]



Le diagramme de séquence

Fragments

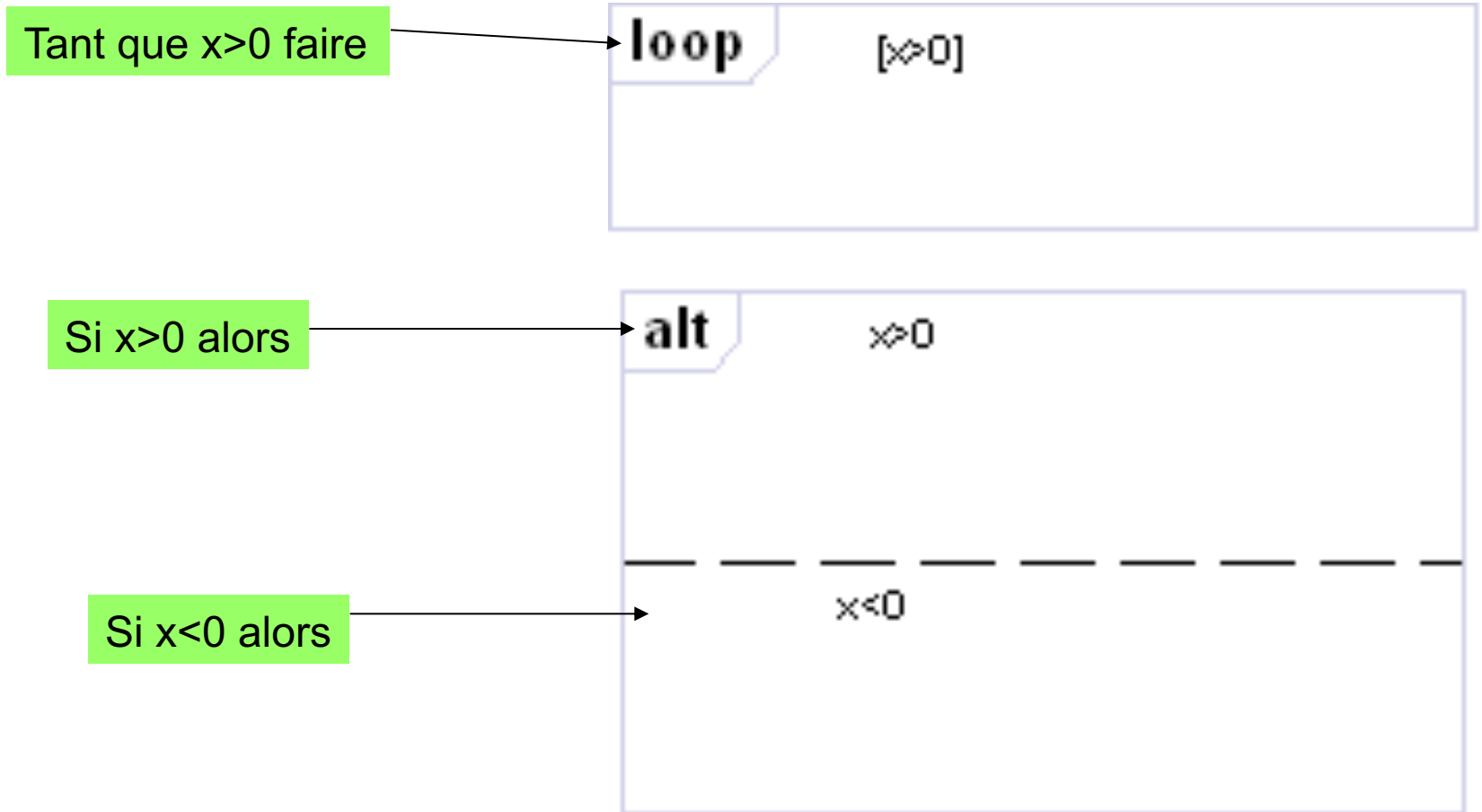


Diagramme de Classes

Définitions

- Diagramme de Classe : Représentation abstraite des objets du système qui vont interagir ensemble pour la réalisation des cas d'utilisation.
 - Vue statique : facteur temps pas pris en compte dans le comportement du système.
 - Modélisation des concepts du domaine d'application.
 - => modélisation des classes du système et leurs relations indépendamment d'un langage de programmation particulier.
 - Les principaux éléments des **classes** et leurs relations :
association, **généralisation** et d'autres types de **dépendances**.

La classe

- Une classe est un concept abstrait qui représente divers éléments du système :
 - éléments concrets comme des étudiants,
 - éléments abstraits comme des inscriptions,
 - partie d'une application comme des boîtes de dialogue,
 - structures informatiques comme des tables de hachage,
 - éléments comportementaux comme des traitements.
- Une classe est associée à d'autres classes

Instance d'une classe

- Une instance est une concrétisation d'un concept abstrait.
Par exemple :
 - le Maroc est une instance du concept abstrait Pays ;
 - la FSR est une instance du concept abstrait Etablissement Supérieur
 - L'étudiant Ali est une instance du concept abstrait Etudiant
 - L'inscription de Ali à la FSR est une instance du concept abstrait Inscription
- Un objet est une instance d'une classe.
 - C'est une entité discrète dotée d'une identité, d'un état et d'un comportement.
 - Les objets sont des éléments individuels d'un système en cours d'exécution
 - Les objets sont reliés entre eux (par des associations).
- Exemple :
 - Si : Etudiant est un concept abstrait, on peut dire que l'étudiant Ali est une instance de Etudiant.
 - Si Etudiant était une classe, Ali en serait une instance : un objet.

Etat et comportement d'un objet

- **État d'un objet :**
 - Ce sont les attributs et les terminaisons d'associations qui décrivent l'état d'un objet.
 - Les propriétés décrites par les attributs prennent des valeurs lorsque la classe est instanciée.
 - L'instance d'une association est appelée un lien.
- **Comportement d'un objet :**
 - Les opérations décrivent les éléments individuels d'un comportement que l'on peut invoquer.
 - Ce sont des fonctions qui peuvent prendre des valeurs en entrée et modifier les attributs ou produire des résultats.
 - Une opération est la spécification (i.e. déclaration) d'une méthode..
- Les attributs, les terminaisons d'association et les méthodes constituent donc les propriétés d'une classe (et de ses instances).

Représentation graphique

Personne

- nom : Chaîne
- annéeNaiss : Entier
- téléphone [0..2] : Chaîne
- nbreEnfants : Entier
- marié : Booléen = vrai
- moyenneNbreEnfants : Réel

- + age(annéeCour : Entier) : Entier
- + ajouterEnfant()
- + nomPers() : Chaîne
- + moyenneNbreEnfants() : Réel

Classe avec attributs et opérations

nom de la classe (*commence par une majuscule*)

attribut

visibilité

nom (*commence par une minuscule*)

multiplicité (*1 par défaut*)

type de la valeur

valeur par défaut

attribut de classe (*souligné*)

opération

visibilité

nom (*commence par une minuscule*)

paramètres typés

type du résultat

opération de classe (*souligné*)

La **valeur par défaut** est affectée à l'attribut à la création des instances de la classe à moins qu'une autre valeur ne soit spécifiée.

La **multiplicité** indique le nombre de valeurs possibles pour l'attribut

+ public
protégé
- privé

Notation visibilité

Attribut d'une classe

- Un attribut définit des caractéristiques qu'une classe ou d'un objet. Il est défini par :
 - Un nom, un type de données, une visibilité et peut être initialisé.
 - Le nom de l'attribut doit être unique dans la classe.
- La syntaxe de la déclaration d'un attribut est la suivante :

```
<visibilité> <nom_attribut>:  
<Type> [ '[' <multiplicité> ']' [ '{' <contrainte> '}' ] ]  
      [ = <valeur_par_défaut> ]
```

- <Type> : nom de classe, nom d'interface ou type de donné
- <multiplicité> : nombre de valeurs que l'attribut peut contenir. (si > 1 possibilité d'ajout d'une contrainte
- <contrainte> : précision sur les valeurs ({ordered}, {list} ...).

Attribut de classe

- Un attribut qui s'applique à une classe et non a une instance de cette classe
 - Dans un DC les attributs de classe sont soulignés

Client
<u>nb_clients</u>
nom
adresse
solvabilité() : Chaîne

Méthode d'une classe

- **Une** opération (même nom et même types de paramètres) doit être unique.
- Surcharge \Leftrightarrow même nom et des paramètres différents



La valeur retournée ne distingue pas 2 opérations.

- Syntaxe :
`<visibilité> <nom_méthode> ([<paramètre> [, <paramètre> [, <paramètre> ...]]]) :
[<valeur_renvoyé>] [{ <propriétés> }]`
 - La syntaxe de définition d'un paramètre (<paramètre>) :
`[<direction>] <nom_paramètre>:<Type> ['<multiplicité>'] [=<valeur_par_défaut>]`
 - La direction :
in : valeur. (par défaut)
out : sortie
inout : entrée/sortie
 - <Type> classe, interface ou type de donné
 - <propriétés> \Leftrightarrow contraintes ou informations (exceptions, les préconditions, les postconditions ou indication qu'une méthode est abstraite (mot-clef abstract), ...)

Méthode de classe

- Un méthode qui s'applique à une classe et non a une instance de cette classe
 - Dans un DC les méthodes de classe sont soulignés

Client
<u>nb_clients</u>
nom
adresse
<u>Nb_cli_ville(ville) : Integer</u>

Méthode abstraite

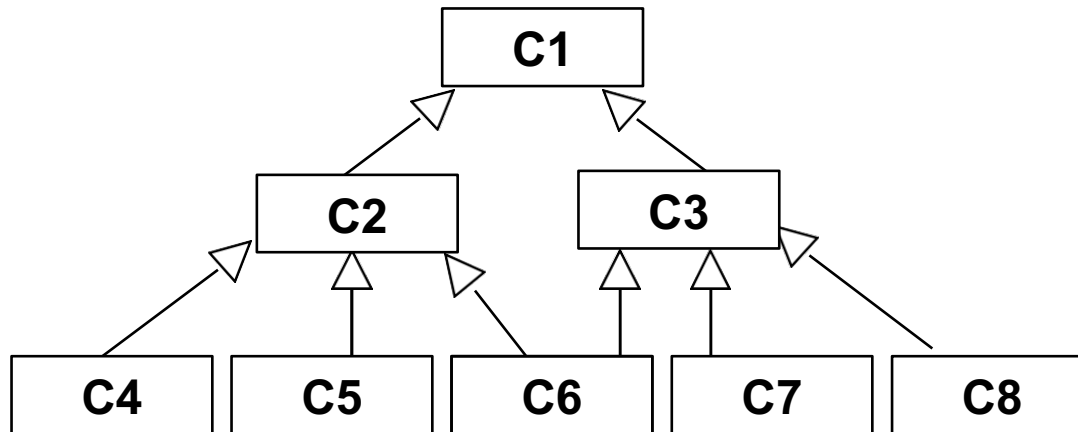
- Méthode dont on connaît l'entête mais pas la manière dont elle peut être réalisée
- Classe abstraite
 - définit au moins une méthode abstraite ou une classe parent contient une méthode abstraite non encore réalisée.
 - non instanciable
 - peut contenir des méthodes concrètes.
- Interface
 - classe abstraite pure qui ne comporte que des méthodes abstraites.

Relation entre classes

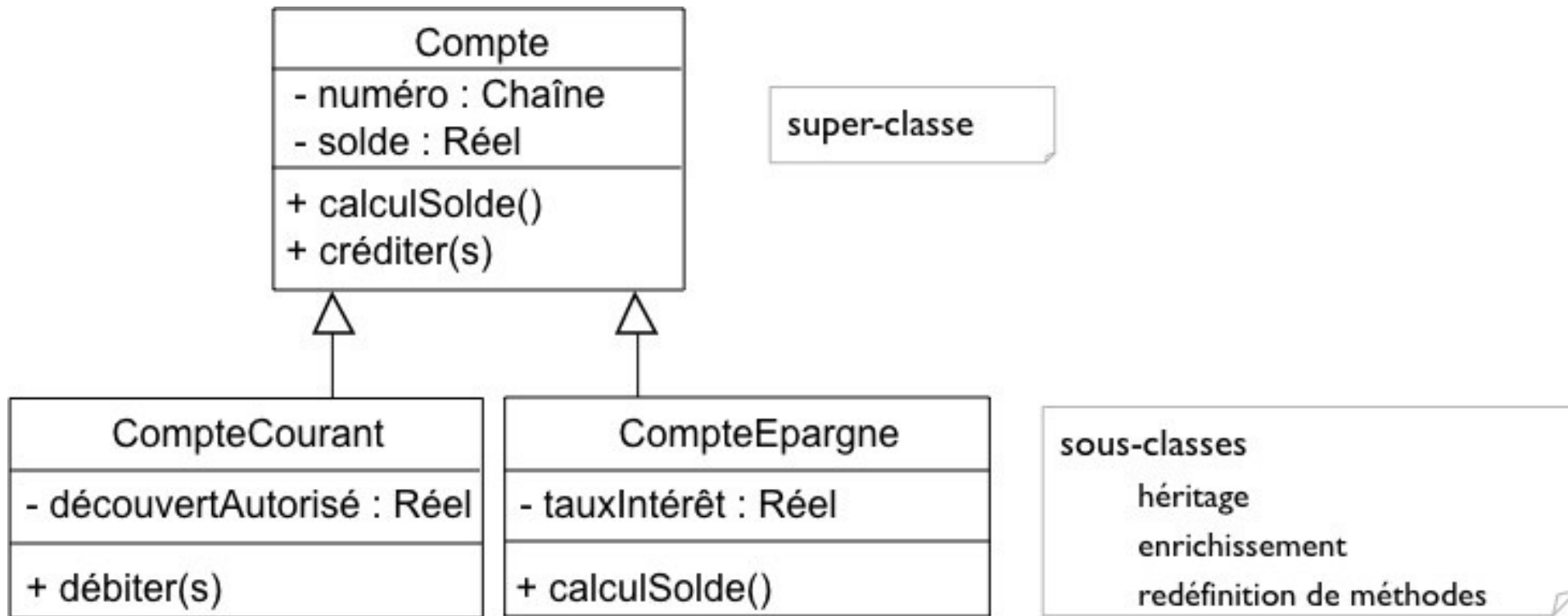
- La généralisation et héritage
- L'association
- Multiplicité ou cardinalité
- Navigabilité
- Qualification
- Classe-association
- Agrégation et composition
- Dépendance

Généralisation et héritage

- Dans le langage UML, la relation de généralisation se traduit par le concept d'héritage.
 - Nommée également relation d'héritage.
- Elle permet la classification des objets.



Généralisation et héritage

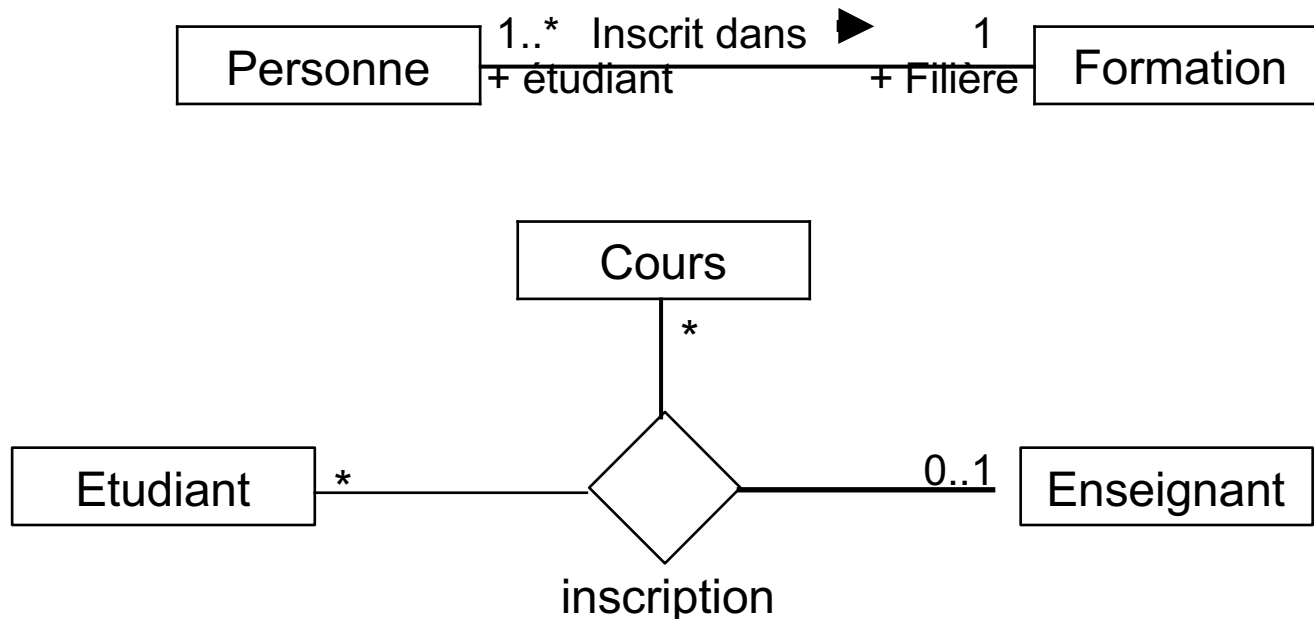


Relation entre classes

- La généralisation et héritage
- **L'association**
- Multiplicité ou cardinalité
- Navigabilité
- Qualification
- Classe-association
- Agrégation et composition
- Dépendance

Association

- Relation entre deux ou plusieurs classes décrivant les liens structurels entre leurs instances.
- Les terminaisons d'associations comme les attributs sont des propriétés structurelles de la classe.



Association

- Propriété structurelle est paramétrée par :
 - **nom** : appelé nom du rôle, il est situé à proximité de la terminaison et est facultatif.
=> autant de noms de rôle que de terminaisons
 - **visibilité** : mentionnée à proximité de la terminaison, (éventuellement devant le nom de rôle)
 - **multiplicité** : mentionnée à proximité de la terminaison et facultative. Par défaut une terminaison d'association est non spécifiée.
 - **navigabilité** : peut être précisée

Relation entre classes

- La généralisation et héritage
- L'association
- **Multiplicité ou cardinalité**
- Navigabilité
- Qualification
- Classe-association
- Agrégation et composition
- Dépendance

Multiplicité ou cardinalité

- La multiplicité \Leftrightarrow nombre d'objets pouvant être associés.
 - un et un seul : 1 ou 1..1
 - plusieurs : * ou 0.. *
 - au moins un : 1..*
 - de un à n : 1..n

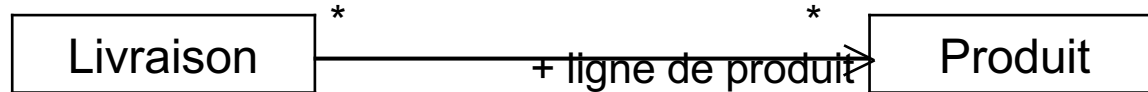


Relation entre classes

- La généralisation et héritage
- L'association
- Multiplicité ou cardinalité
- **Navigabilité**
- Qualification
- Classe-association
- Agrégation et composition
- Dépendance

La navigabilité

- Sens de l'association. Par défaut, une association est navigable dans les deux sens.

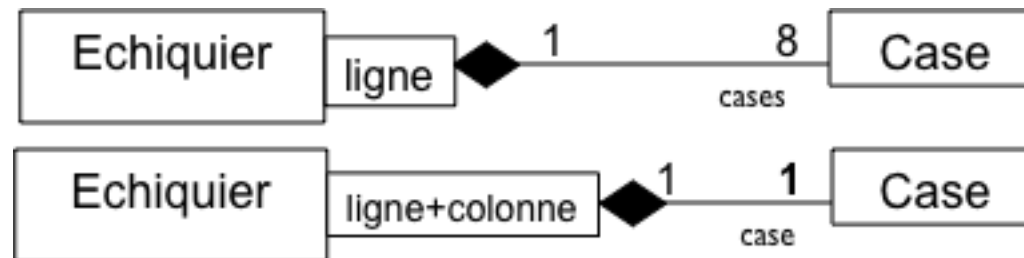


Relation entre classes

- La généralisation et héritage
- L'association
- Multiplicité ou cardinalité
- Navigabilité
- **Qualification**
- Classe-association
- Agrégation et composition
- Dépendance

Associations qualifiées

- Equivalent UML de : tableau associatif, "map" ou dictionnaire
- restreindre la portée de l'association à quelques éléments ciblés (qualificatif)

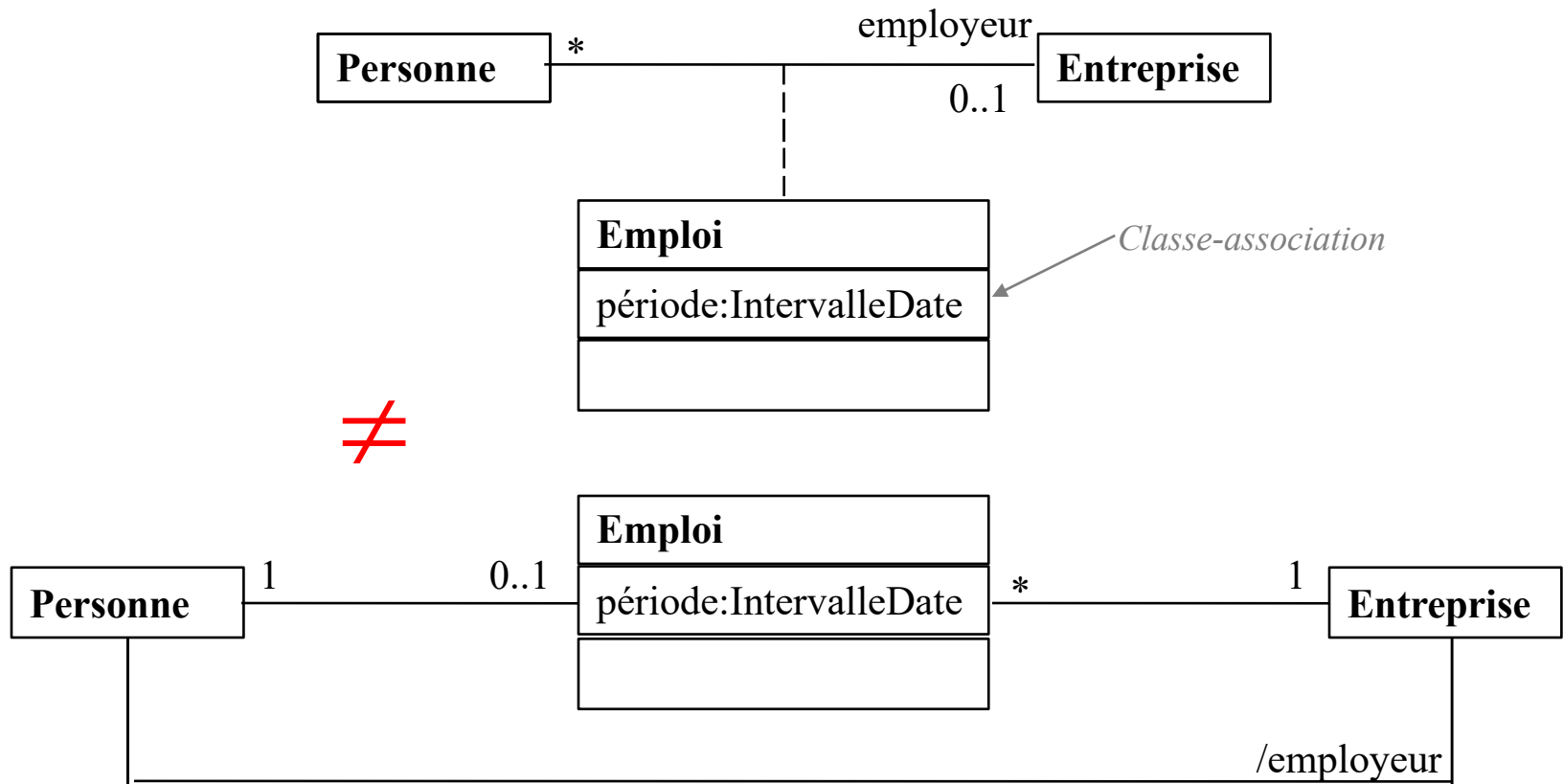


Relation entre classes

- La généralisation et héritage
- L'association
- Multiplicité ou cardinalité
- Navigabilité
- Qualification
- **Classe-association**
- Agrégation et composition
- Dépendance

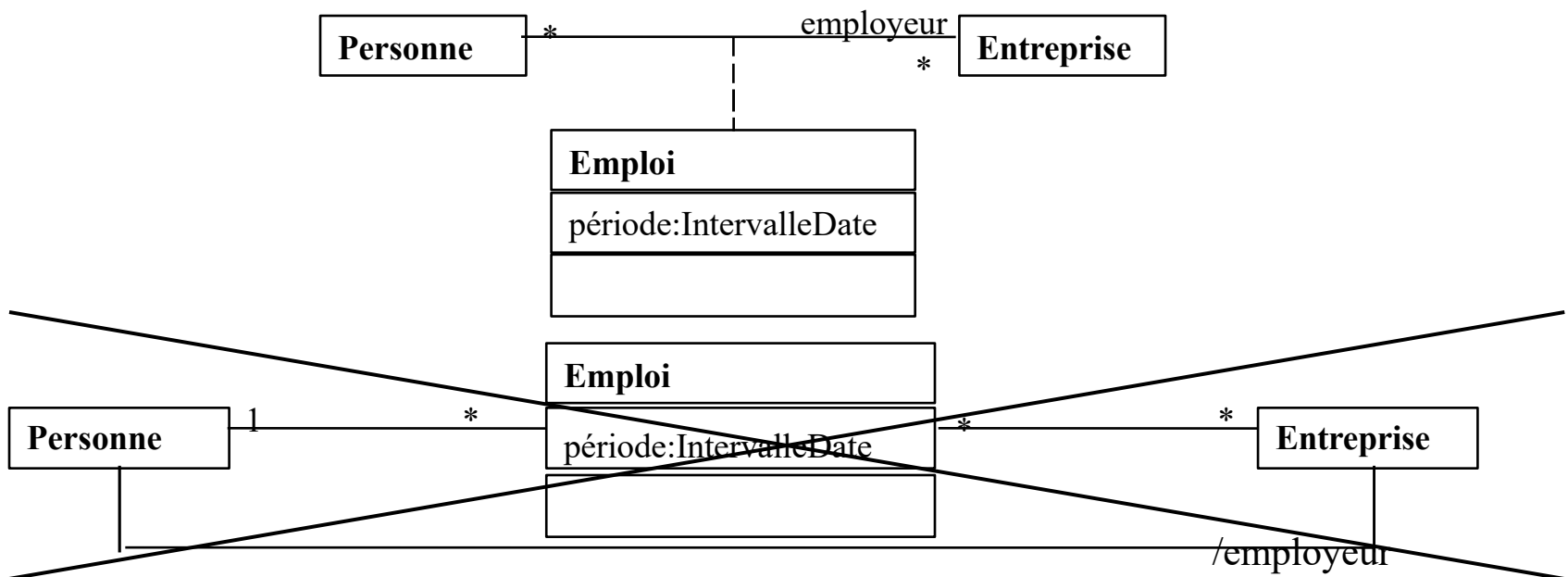
Classe-association

- Une personne ne peut travailler que dans une entreprise



Classe-association

- **Avantage** : il ne peut y avoir qu'une seule instance pour un lien entre deux objets quelconques
 - Une personne peut travailler dans plusieurs entreprises en même temps mais elle ne peut pas avoir plus d'un emploi dans la même entreprise



Relation entre classes

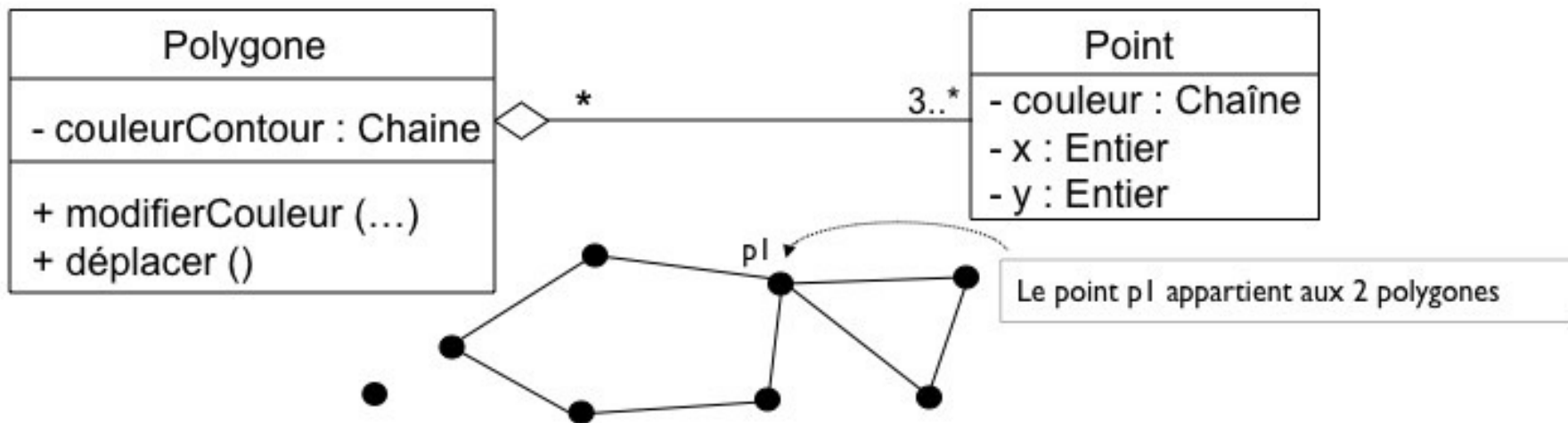
- La généralisation et héritage
- L'association
- Multiplicité ou cardinalité
- Navigabilité
- Qualification
- Classe-association
- **Agrégation et composition**
- Dépendance

Agrégation

Forme particulière d'association non symétrique dans laquelle l'une des extrémités joue un rôle prédominant par rapport à l'autre.

Les 2 objets liés par une agrégation sont distingués : l'un fait " partie de" l'autre, considéré comme l'objet global (l'agrégat).

Un objet "partie" peut être partagé (pas d'exclusivité)



Composition

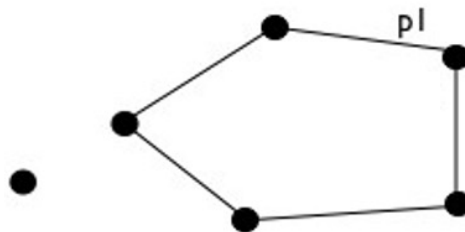
Forme particulière d'agrégation avec une dépendance forte.

Un objet composant n'appartient qu'à un seul composé (pas de partage).

le composant est détruit lors de la destruction du composé.

il peut éventuellement être créé hors de son composé.

le composant n'intervient pas dans d'autres associations de composition ni d'agrégation.



Relation entre classes

- La généralisation et héritage
- L'association
- Multiplicité ou cardinalité
- Navigabilité
- Qualification
- Classe-association
- Agrégation et composition
- Dépendance

Dépendante

- Relation unidirectionnelle qui exprime une dépendance sémantique entre les éléments du modèle.
 - \Leftrightarrow modification de la cible \Rightarrow une modification de la source
- Représentée par un trait discontinu orienté



Diagrammes de classes

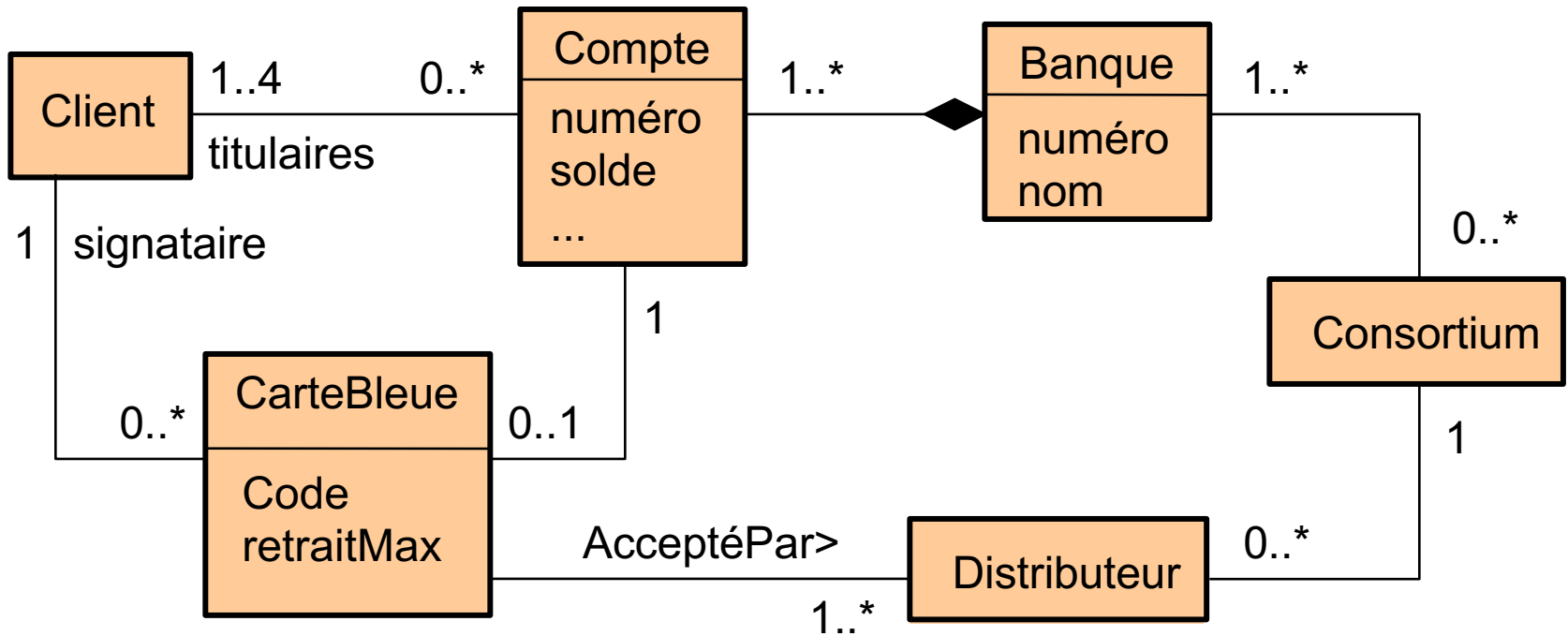


Diagramme d'objets

Le diagramme d'objets

Présentation

- Un diagramme d'objets représente des objets et leurs liens pour donner une vue de l'état du système à un instant donné.
- Un diagramme d'objets permet d'illustrer le diagramme de classes.
- **Faciliter la validation** d'un diagramme de classes complexe en présentant une ou plusieurs instanciation de celui-ci
- Le diagramme d'objets est **très peu utilisé**

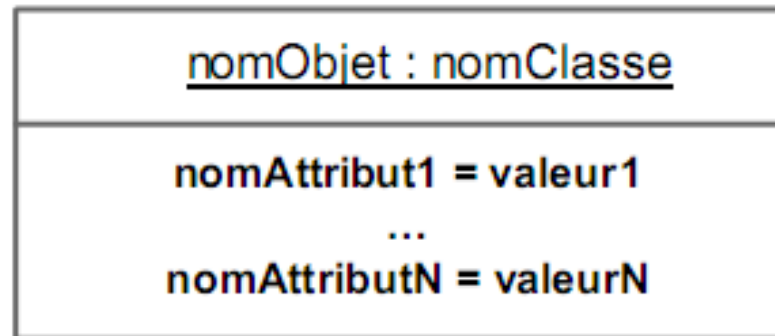
Le diagramme d'objets

Représentation des objets

- Graphiquement, un objet se représente comme une classe.
- En revanche:
 - La section des opérations n'est pas utile
 - Les noms des objets sont soulignés
 - le nom de la classe est précédé d'un « : »
 - les attributs reçoivent des valeurs.

Le diagramme d'objets

Représentation des objets



Le diagramme d'objets

Nom de l'objet

- Le nom de l'objet peut être présenté selon trois formats plus ou moins précis
- Il peut contenir :
 - Un nom optionnel identifiant l'objet de manière unique
 - Le nom optionnel de la classe à laquelle appartient l'objet

nomObjet

Ou

nomObjet : nomClasse

Ou

: nomClasse

Le diagramme d'objets

Les liens

- Un lien est une instance d'une association.
- Un lien se représente comme une association

Diagrammes d'objets

