Analyse et Conception de Systèmes Informatiques
Orientés objets en UML

Abdelhak-Djamel SERIAI

http://www.lirmm.fr/~seriai

seriai@lirmm.fr

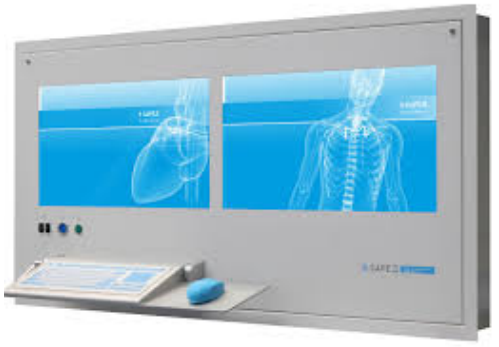
Partie I : Pourquoi UML ?

Le produit logiciel (01)

- **Informatique et logiciels**
  - Informatique de gestion
    - Systèmes d'information d'entreprises
    - Aide à la décision
    - Etc.

  - Informatique industrielle
    - Chaîne de production
    - Systèmes de contrôles : bateau, avion, train, etc.

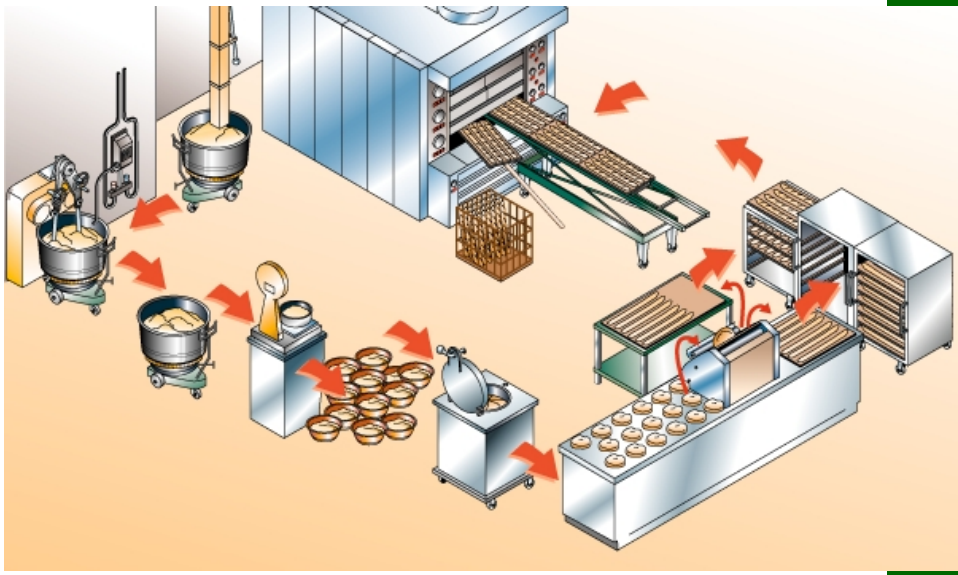
  - Informatique médicale
    - Scanneur, décision, etc.



  - autres

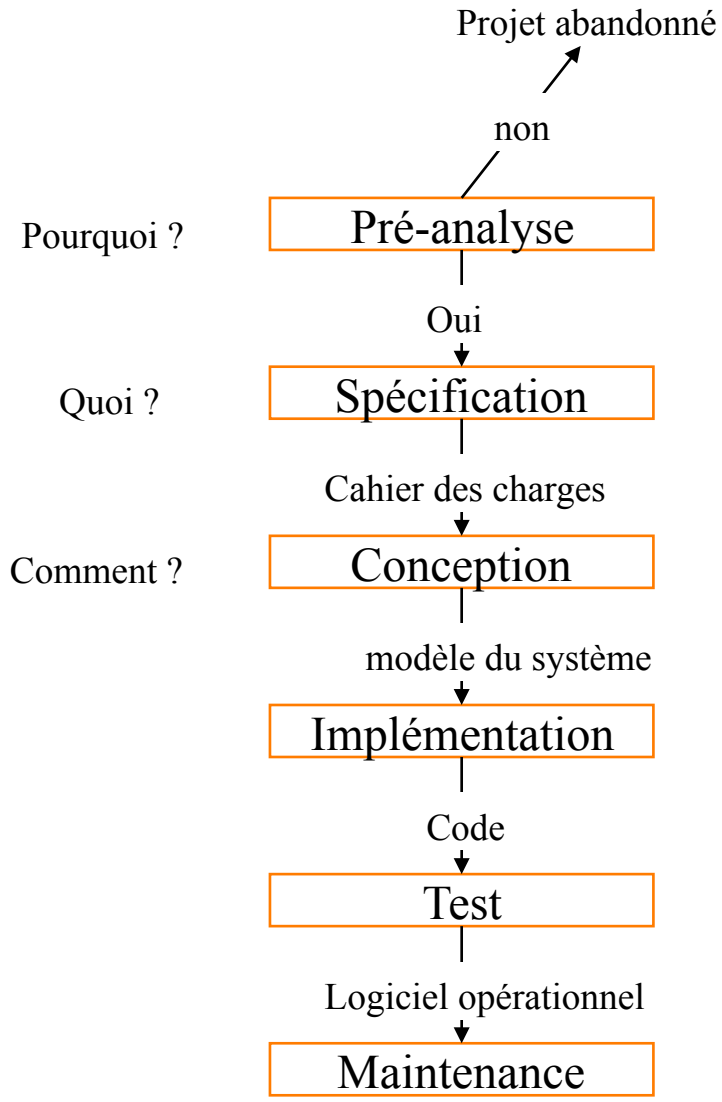
Le produit logiciel (02)

- **Un logiciel est un produit (manufacturé)**
  - Comme une voiture ou une machine à outils
  - Nécessité d'un processus de développement
    - Déterminer les besoins
    - Plan ou architecture du système
    - Conception du système
    - Réalisation du système
    - Etc.



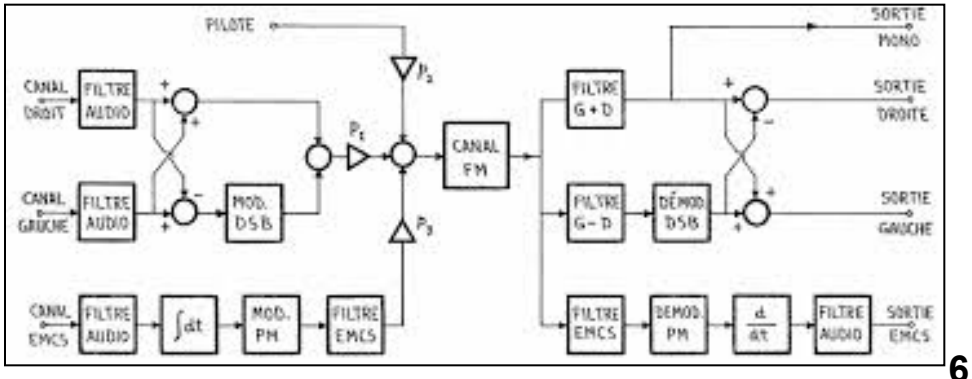
Cycle de vie de logiciel

- **Cycle de vie de logiciel**
  - La phase de pré-analyse
    - Étude d'opportunité
  - La phase de spécification
    - Analyse des besoins
  - La phase de conception
    - La conception architecturale
    - Conception détaillée
  - La phase d'implémentation
    - La programmation
  - La phase de test
    - La validation et vérification
  - La phase de maintenance



Méthodes de développement(1)

- **Développer un modèle d'un système**
  - Un modèle est une représentation abstraite d'une réalité ou une partie d'une réalité
    - Exemple de modèles
      - Modèle d'un événement physique
      - Modèle d'une réaction chimique
      - Modèle d'un système informatique
      - Modèle d'un mécanisme de fonctionnement électronique
    - Exemple de formes de représentation
      - Mathématique
      - Graphique
      - Semi-graphique

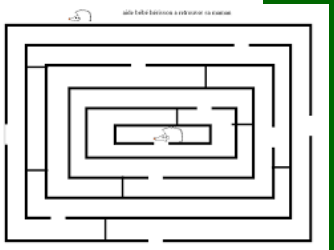


Méthodes de développement(2)

- **Pourquoi développer des modèles**

  - **Problèmes des spécifications des besoins**

  - **Taille et complexité des systèmes importantes et croissantes**



  - **Gestion des équipes**

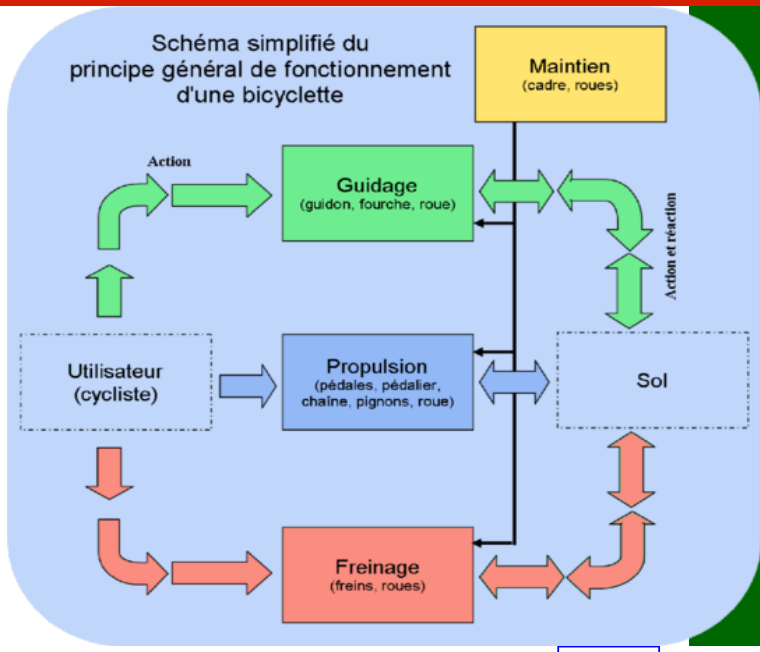
Méthodes de développement(3)

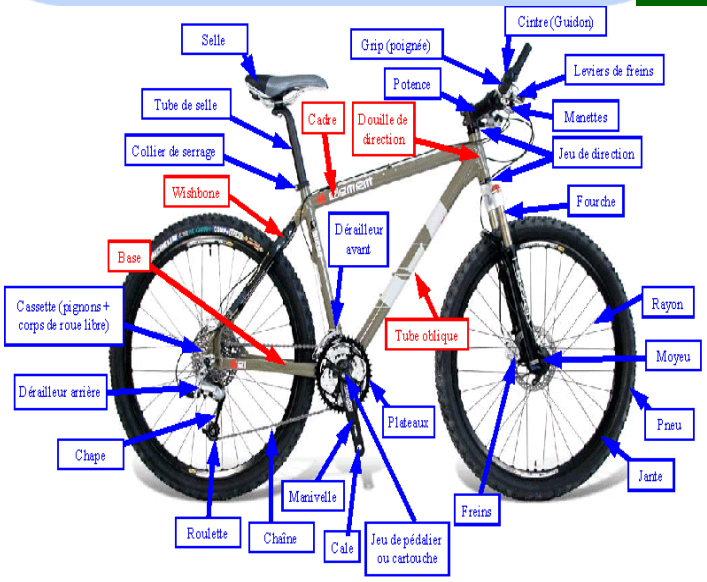
- **Les méthodes d'analyse et de conception**
  - Proposent une démarche qui distingue les étapes du développement dans le cycle de vie du logiciel
  - S'appuient sur un formalisme de représentation qui facilite la communication, l'organisation et la vérification : Le langage de modélisation
  - Produisent des documents (modèles) qui facilitent les retours sur conception et l'évolution des applications

Méthodes de développement(4)

Classification des méthodes de développement

- La distinction fonctionnel /orientée objet.
  - Les stratégies fonctionnelles (dirigée par le traitement) :
    - Un système est vu comme un ensemble d'unités en interaction, ayant chacune une fonction clairement définie.
    - Exemple : SA, SADT, SA/RT, RAPID/ USE, JSD, MASCOT, Automates; AEFC, RDP
  - Les stratégies orientées objet
    - Considèrent qu'un système est un ensemble d'objets interagissant.
    - Exemples: OMT, UML

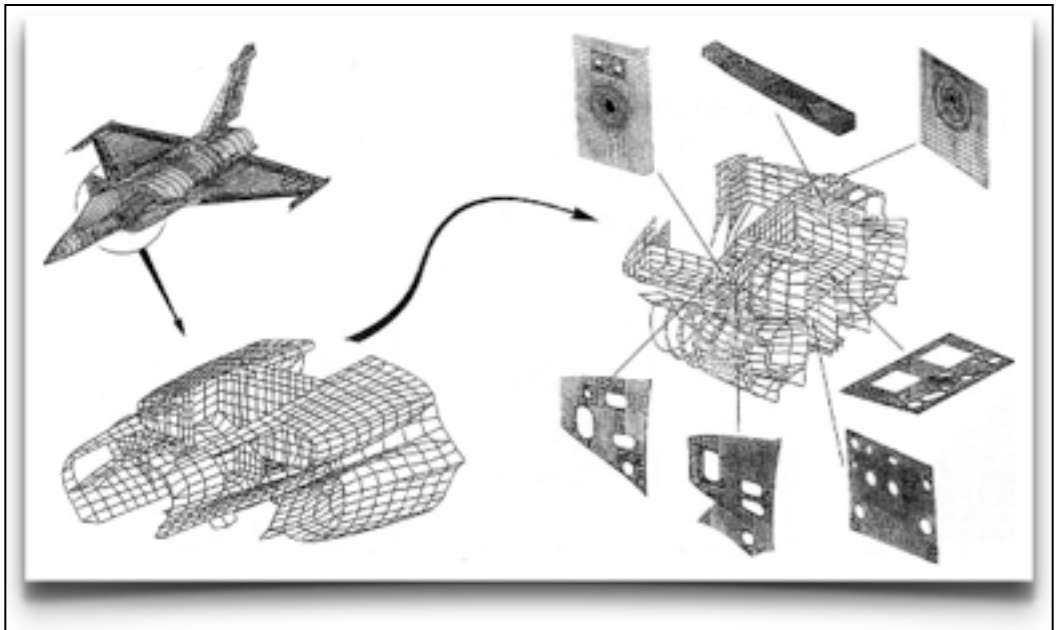




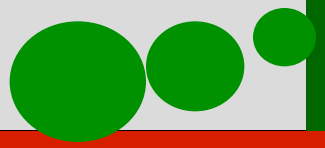
Méthodes de développement(5)

Classification des méthodes de développement

- La distinction composition/décomposition :
  - Les méthodes ascendantes consistent à construire un logiciel par composition à partir de modules existants
  - Les méthodes descendantes décomposent récursivement un système jusqu'à arriver à des modules programmables « simplement »



Méthodes de développement(6)

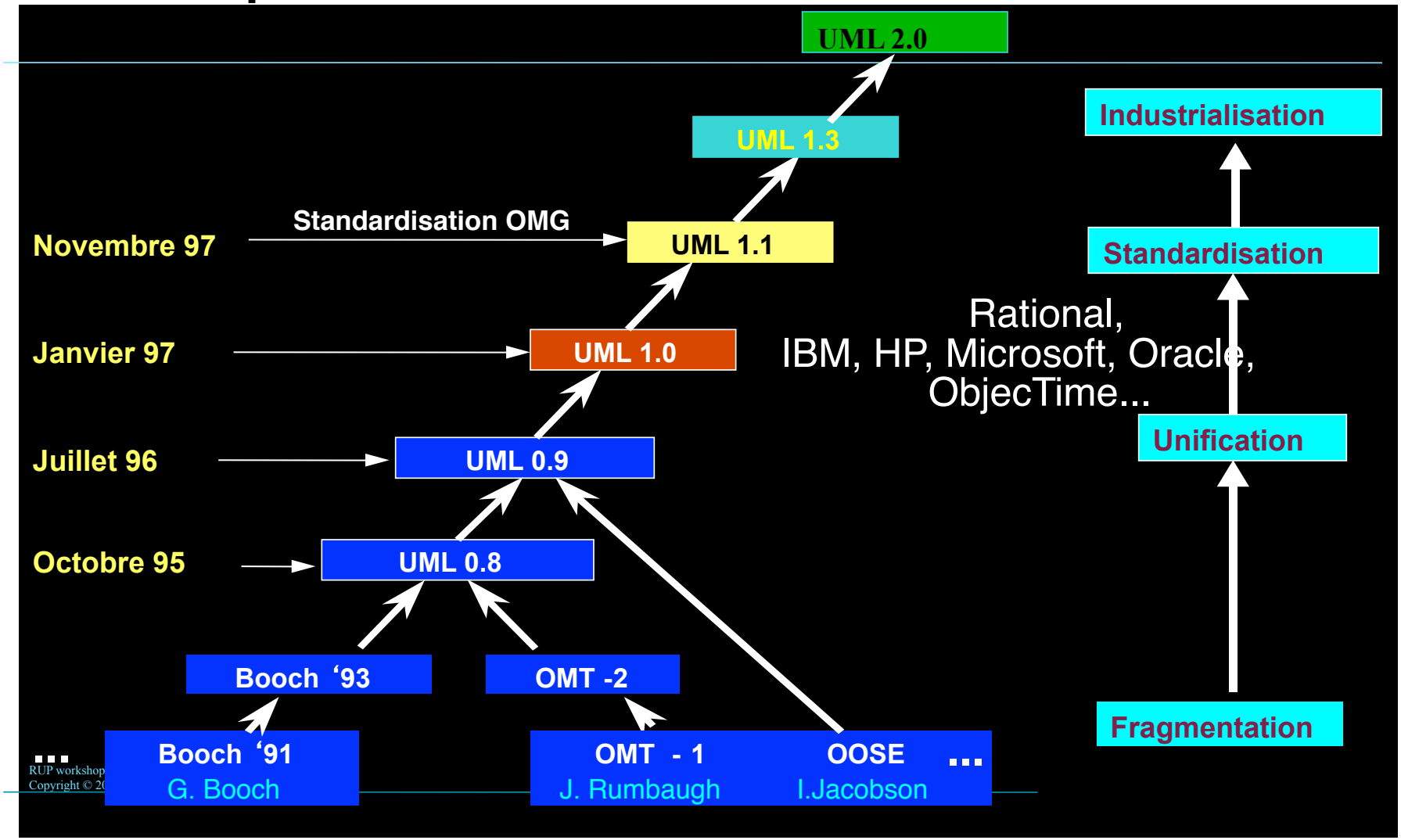


Classification des méthodes de développement

- La distinction formelle/non formelle
  - Approche non formelle ou semi-formelle
    - Permettent de spécifier certaines parties de l'analyse ou/et de la conception à l'aide de langages formels tel que le langage OCL pour UML
  - Approches formelles
    - Approches mathématiques de spécification et de modélisation
    - Exemples : spécification en langage B et Z

Introduction UML(1)

- Historique UML



Introduction UML(2)

Présentation de la méthode UML

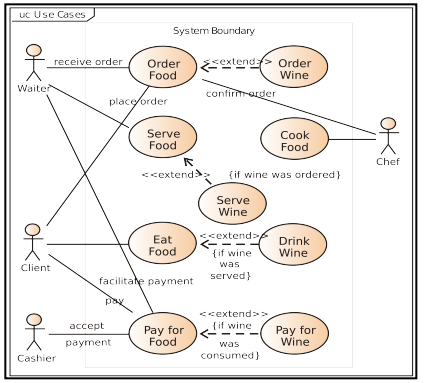
- UML est la forme contractée de *Unified Modeling Language* qui peut se traduire en français par *langage unifié pour la modélisation*
- C'est un langage de modélisation objet
- UML se base sur une notation graphique expressive.
- Il permet de modéliser de manière claire et précise la structure et le comportement d'un système indépendamment de toute méthode ou de tout langage de programmation
- La notation UML repose sur plusieurs diagrammes adaptés au développement logiciel pour les phases de spécification, conception et implémentation du cycle de vie.

Introduction UML(3)

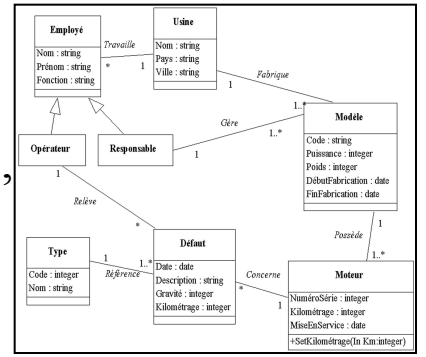
Présentation du langage UML

- Différents types de diagrammes

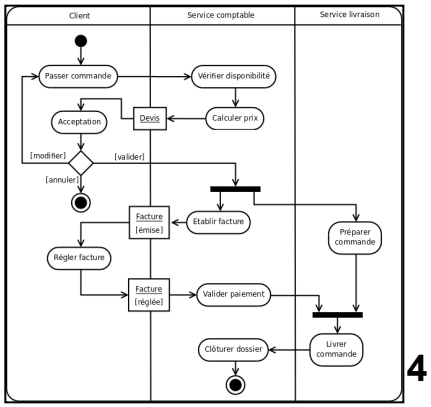
  - Expression des besoins
    - Use cases et diagrammes d'activité



  - Identification des entités (de la structure) du système étudié
    - Diagrammes de classes : Classes, Associations, Attributs, Opérations; Assertions, invariants, pré-conditions, postconditions; Généralisations, Classes associations

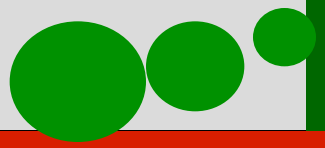


  - Description du comportement attendu
    - Diagrammes d'interaction : Descriptions des interactions entre « groupes » d'objets: messages échangés entre objets.



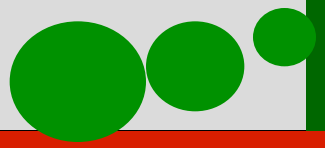
    - Diagrammes d'états :Description des états des objets

Introduction UML(4)



- **Comment modéliser avec UML ?**
  - Définition
    - UML est un langage qui permet de représenter des modèles
      - UML permet de définir et de visualiser un modèle à l'aide de diagrammes
      - Un diagramme UML est une représentation graphique, qui s'intéresse à un aspect précis du modèle ;
        - Combinés, les différents types de diagrammes UML offrent une vue complète des aspects statiques et dynamiques d'un système.
      - UML offre des représentations semi-formelles
        - Graphique
        - Contraintes mathématiques (formelles) dans le langage OCL (Object constraint language)
    - UML ne définit pas le processus d'élaboration des modèles

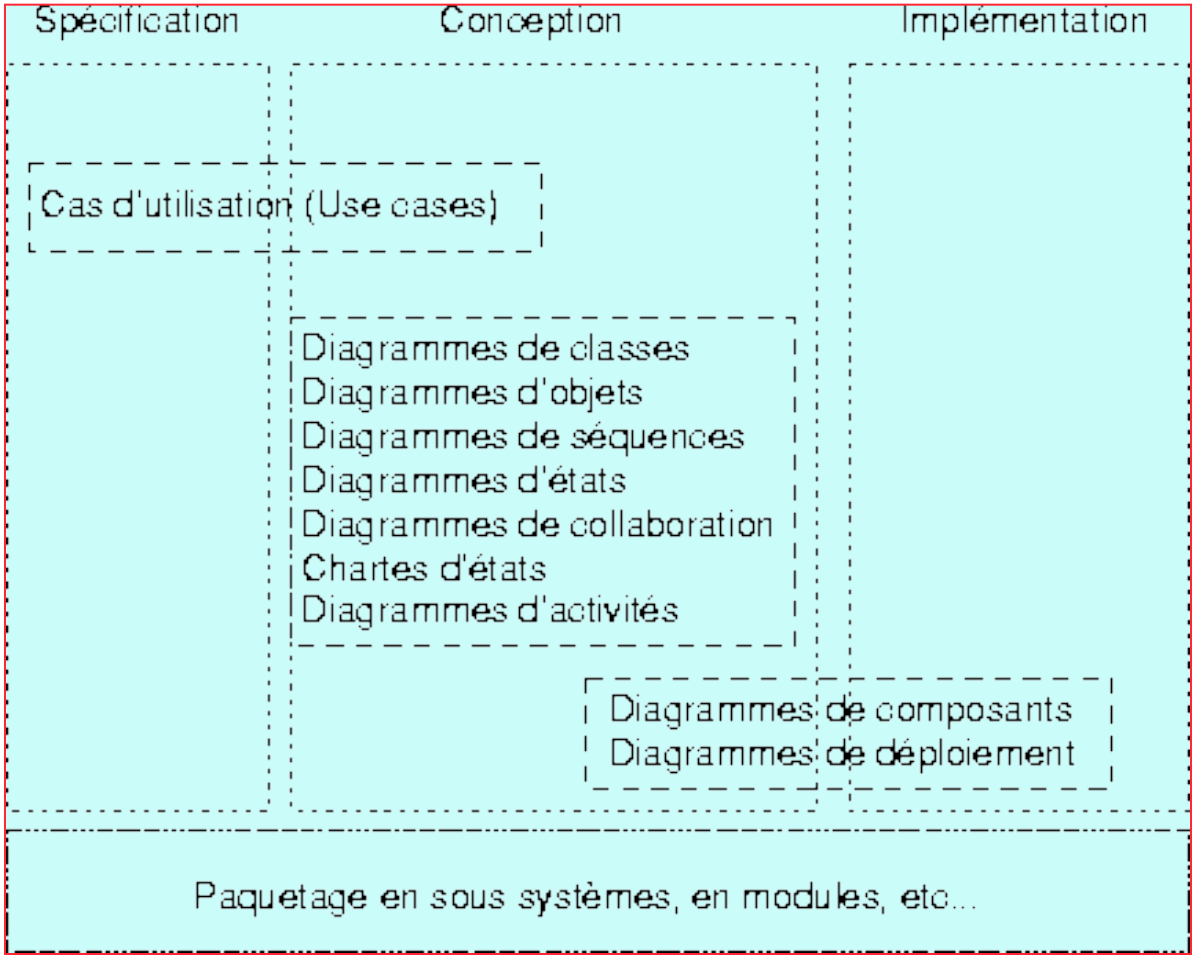
Introduction UML(5)



- **Comment modéliser avec UML ?**
  - Diagrammes UML
    - Vues statiques ou structurelles du système
      - diagrammes de cas d'utilisation
      - diagrammes d'objets
      - diagrammes de classes
      - diagrammes de composants
      - diagrammes de déploiement
    - Vues dynamiques ou comportementales du système
      - diagrammes de collaboration
      - diagrammes de séquence
      - diagrammes d'états-transitions
      - diagrammes d'activités

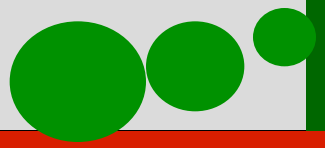
Introduction UML(6)

- **Comment modéliser avec UML ?**
  - Diagrammes UML



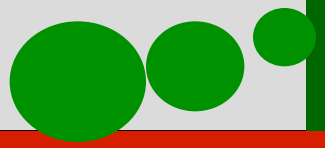
Les concepts objet à travers UML

Les concepts objet à travers UML (01)



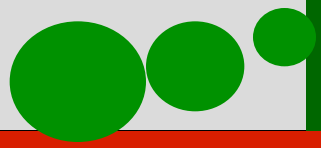
- **Un peu d'histoire**
  - 1967 : Simula, 1er langage de programmation à implémenter le concept de type abstrait à l'aide de classes
  - 1976 : Smalltalk implémente les concepts fondateurs de l'approche objet (encapsulation, agrégation, héritage) à l'aide de :
    - classes
    - associations entre classes
    - hiérarchies de classes
    - messages entre objets
  - 1980 : le 1er compilateur C++. C++ est normalisé par l'ANSI et de nombreux langages orientés objets académiques ont étayés les concepts objets : Eiffel, Objective C, Loops, java, …
  - Standardisation d'UML 1.1 en 1997

Les concepts objet à travers UML (02)



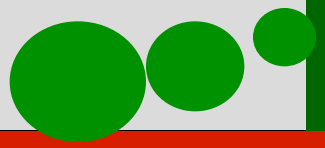
- **Un peu d'histoire**
  - L'approche fonctionnelle n'est pas adaptée au développement d'applications qui évoluent sans cesse et dont la complexité croit continuellement
    - Les fonctions ne sont pas les éléments les plus stables
    - On peut pas décrire toujours un système par une fonction principale
    - Difficulté de la réutilisation

Les concepts objet à travers UML (03)



- **Système à objets ou orienté objet**
  - Les systèmes non objet
    - Procéduraux : sous-programmes
    - Fonctionnels : fonctions
    - Déductifs : règles
  - Les systèmes objets
    - Le logiciel est une collection d'objets dissociés définis par des propriétés
    - L'approche objet facilite le développement et l'évolution d'applications complexes.

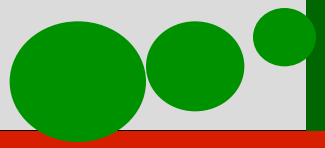
Les concepts objet à travers UML (04)



Le paradigme Objet

- Programmation orientée objet
  - Méthode d'implantation dans laquelle le programme est organisé en collection d'objets coopératifs. Chaque objet est une instance de classe. Toutes les classes sont membres d'une hiérarchie de classes liées pas des relations d'héritage
- Conception orientée objet
  - Méthode de conception qui mène à une décomposition orientée objet et utilise une notation pour représenter les différents aspects du système en cours de conception
- Analyse orientée objet
  - Méthode d'analyse qui examine les besoins en termes de classes et d'objets trouvé dans l'espace du problème.

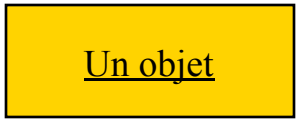
Les concepts objet à travers UML (05)

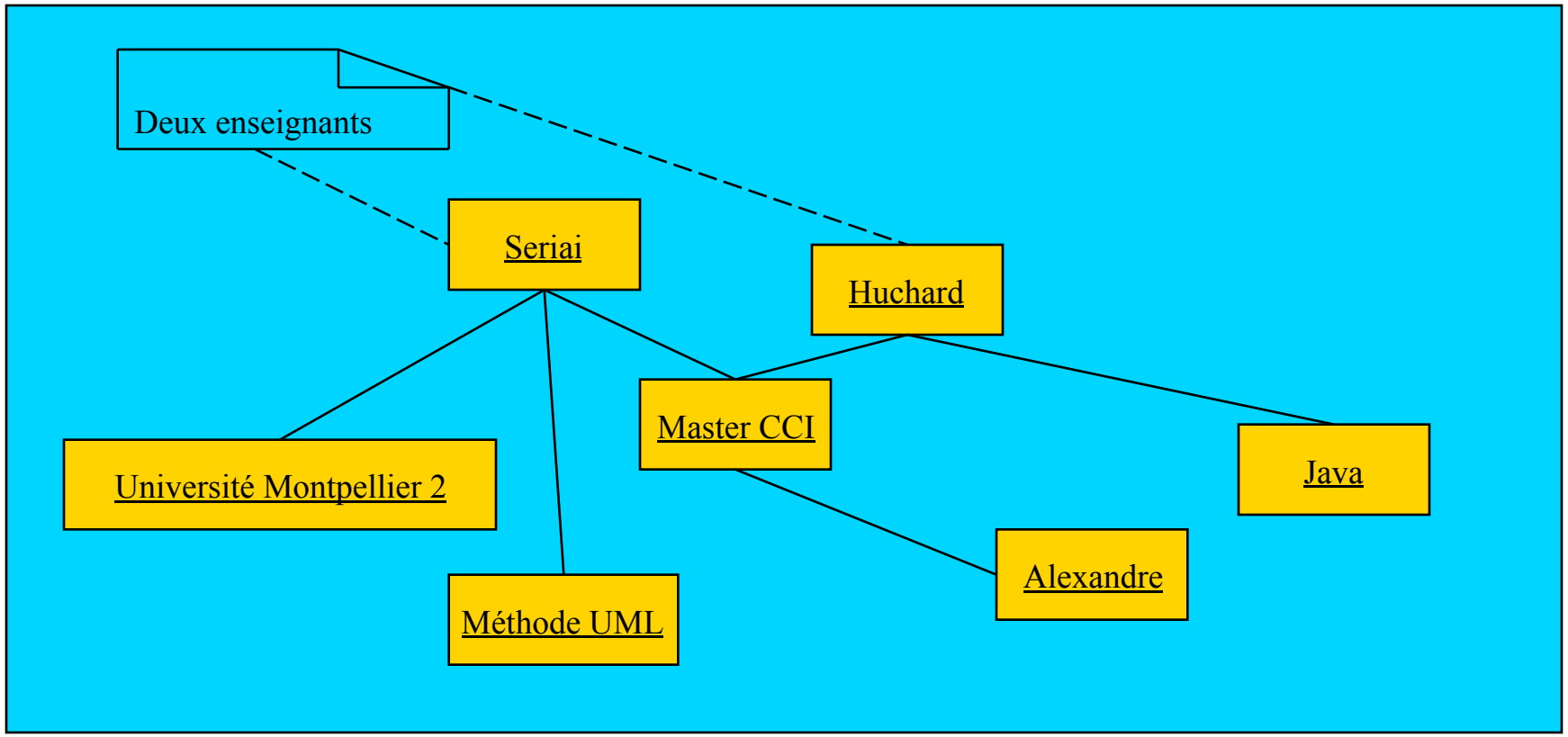


- **Objet**
  - Définition
    - Objet du monde réel :
      - Perception fondée sur le concept de masse
        - grains de sable, les étoiles
    - Objet informatique :
      - Est une unité atomique formée de l'union d'un état et d'un comportement
      - Définit une représentation abstraite d'une entité du monde réel ou virtuel, dans le but de la piloter ou de la simuler
        - Grain de sable, étoile, Compte en banque, police d'assurance, équation mathématiques, etc.
    - Les objets du monde réel et du monde informatique naissent, vivent et meurent

Les concepts objet à travers UML (06)

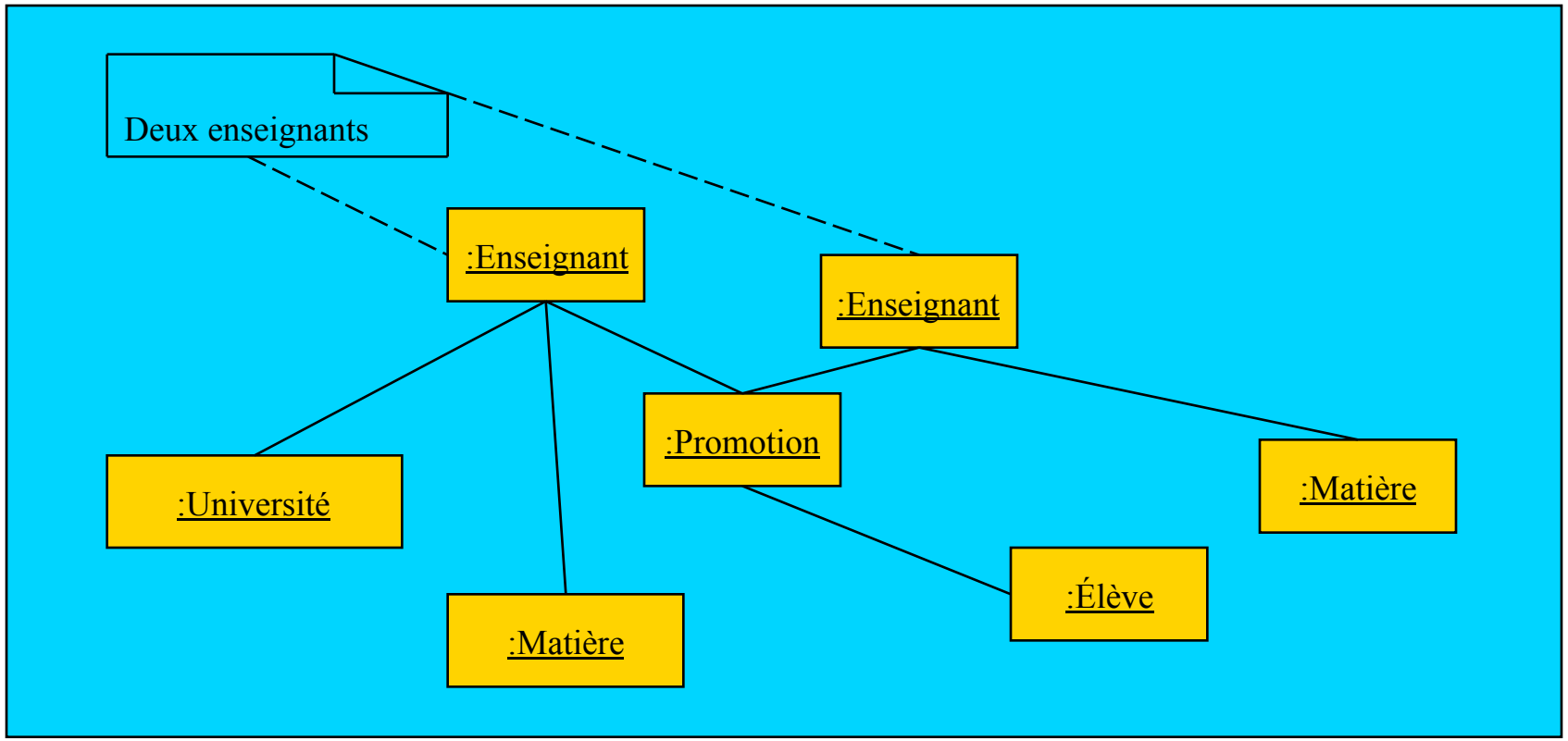
- **Objet**
  - Représentation d'objets en UML





Les concepts objet à travers UML (07)

- **Objet**
  - Représentation d'objets en UML

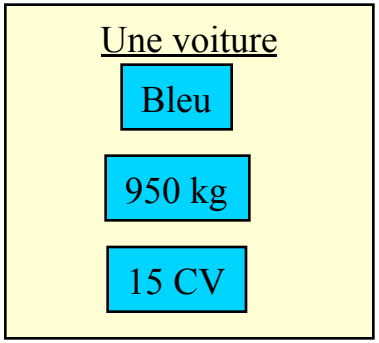


Les concepts objet à travers UML (08)

- **Objet**
  - Caractéristique fondamentales d'un objet (informatique)

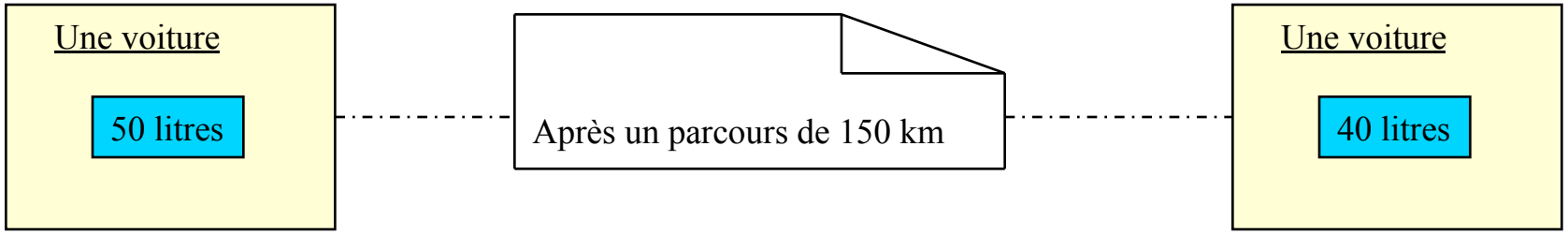
Objet = État + Comportement + Identité

    - État
      - Regroupe les valeurs instantanées de tous les attributs d'un objet :
        - Un attribut est une information qui qualifie l'objet qui le contient
        - Chaque attribut peut prendre une valeur dans un domaine de définition donné
      - Exemple : Un objet voiture regroupe les valeurs des attributs couleur, masse et puissance fiscale



Les concepts objet à travers UML (09)

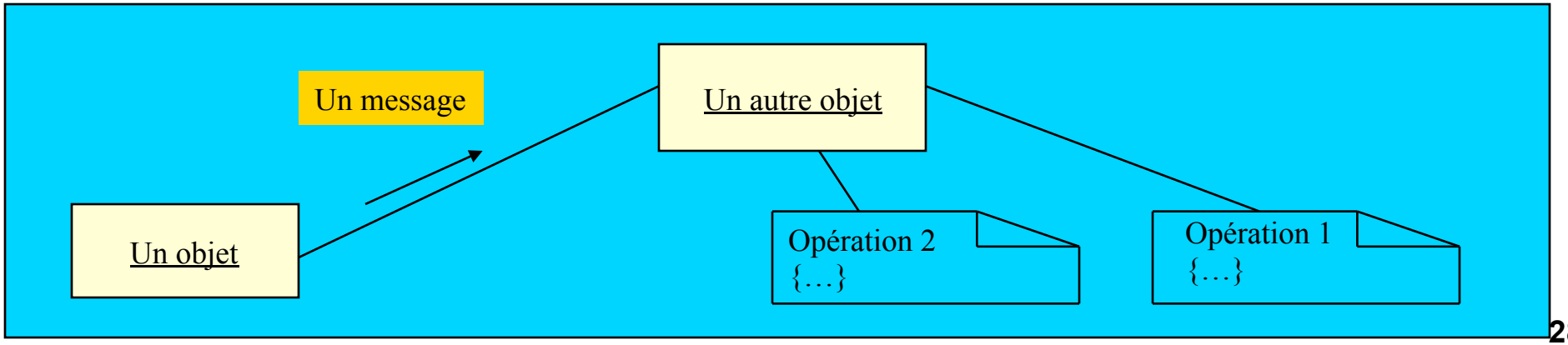
- **Objet**
  - Caractéristique fondamentales d'un objet (informatique)
    - État
      - L'état d'un objet à un instant donné, correspond à une sélection de valeurs, parmi toutes les valeurs possibles des différent attributs
      - L'état évolue au cours du temps, il est la conséquence de ses comportements passés
        - Une voiture roule, la quantité de carburant diminue, les pneus s'usent, etc.



      - Certaines composantes de l'état peuvent être constantes
        - La marque de la voiture, pays de la construction de la voiture

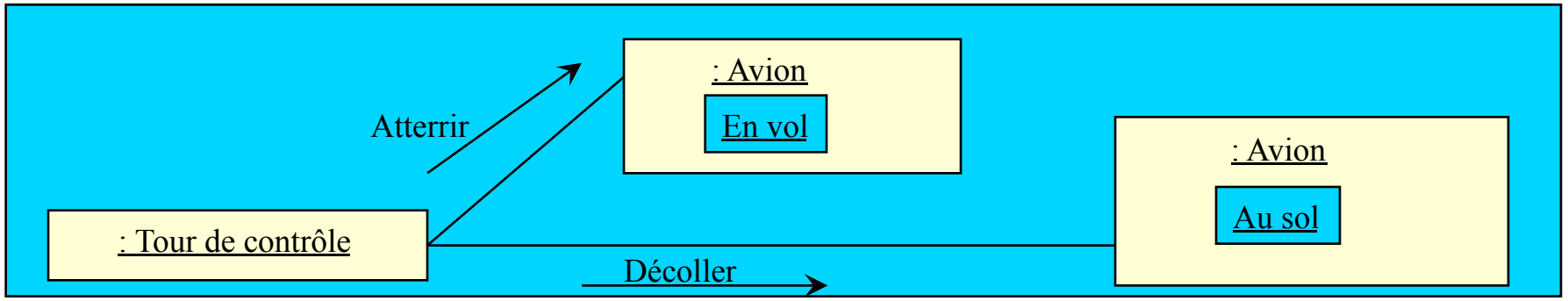
Les concepts objet à travers UML (10)

- **Objet**
  - Caractéristique fondamentales d'un objet (informatique)
    - Le comportement
      - Regroupe toutes les compétences d'un objet et décrit les actions et les réactions de cet objet
      - Chaque atome (partie) de comportement est appelé opération
        - Les opérations d'un objet sont déclenchées suite à une stimulation externe, représentée sous la forme d'un message envoyé par un autre objet
        - L'état et le comportement sont liés



Les concepts objet à travers UML (11)

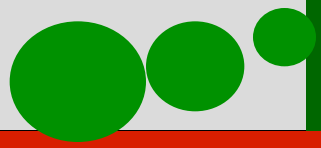
- **Objet**
  - Caractéristique fondamentales d'un objet (informatique)
    - Le comportement
      - Le comportement à un instant donné dépend de l'état courant et l'état peut être modifié par le comportement
        - Exemple : il n'est pas possible de faire atterrir un avion que s'il est en train de voler: le comportement *Atterrir* n'est valide que si l'information *En vol* est valide
        - Après l'atterrissage, l'information *En vol* devient invalide et l'opération *Atterrir* n' plus de sens



Les concepts objet à travers UML (12)

- **Objet**
  - Caractéristique fondamentales d'un objet
    - L'identité
      - Chaque objet possède une identité qui caractérise son existence propre
      - L'identité permet de distinguer tout objet de façon non ambiguë, indépendamment de son état
      - Permet de distinguer deux objets dont toutes les valeurs d'attributs sont identiques
        - deux pommes de la même couleur, du même poids et de la même taille sont deux objets distincts.
        - Deux véhicules de la même marque, de la même série et ayant exactement les mêmes options sont aussi deux objets distincts.

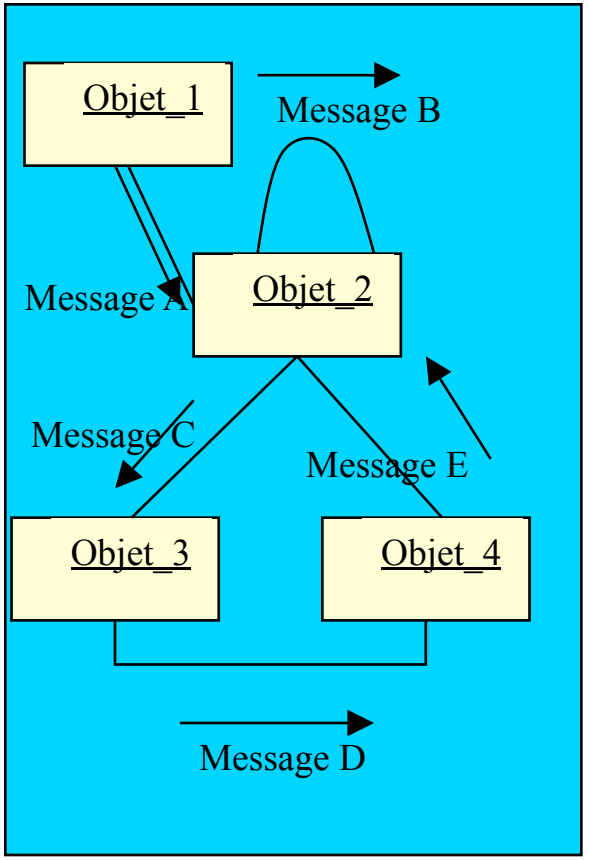
Les concepts objet à travers UML (13)



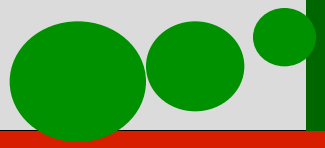
- **Objet**
  - Caractéristique fondamentales d'un objet
    - L'identité
      - L'identité est un concept, elle ne se représente pas de manière spécifique en modélisation. : chaque objet possède une identité de manière implicite
      - En phase de réalisation, l'identité est souvent construite à partir d'un identifiant issu naturellement du domaine du problème.
        - Nos voitures possèdent toutes un numéro d'immatriculation, nos téléphones un numéro d'appel et nous-mêmes sommes identifiés par nos numéros de sécurité social

Les concepts objet à travers UML (14)

- **Objet**
  - Communication entre objets : le concept de message
    - Les systèmes informatiques à objets peuvent être vus comme des sociétés d'objets qui travaillent en synergie afin de réaliser les fonctions de l'application
    - Le comportement global d'une application repose sur la communication entre les objets qui la composent
    - L'unité de communication entre objets s'appelle message

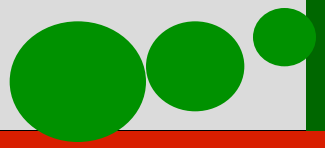


Les concepts objet à travers UML (15)



- **Classe**
  - Définition
    - Une classe décrit une abstraction d'objets ayant
      - Des propriétés similaires
      - Un comportement commun
      - Des relations identiques avec les autres objets
      - Une sémantique commune
    - Par exemple Fichier (resp. Paragraphe, resp. Véhicule) est la classe de tous les fichiers (resp. paragraphes, resp. véhicules)
    - Une classe a trois fonctions:
      - Sert de ``patron'' (*template*) à objets : elle définit la structure générale des objets qu'elle crée en indiquant quelles sont les *variables d'instance* ;
      - Sert de ``conteneur'' d'objets : contient et donc donne l'accès à l'ensemble des objets qu'elle a créés;
      - Sert de ``réceptacle'' des *méthodes* que ses objets peuvent utiliser puisque tous les objets d'une classe utilisent les mêmes *méthodes* , il serait inutile de les dupliquer dans ces objets eux-mêmes.

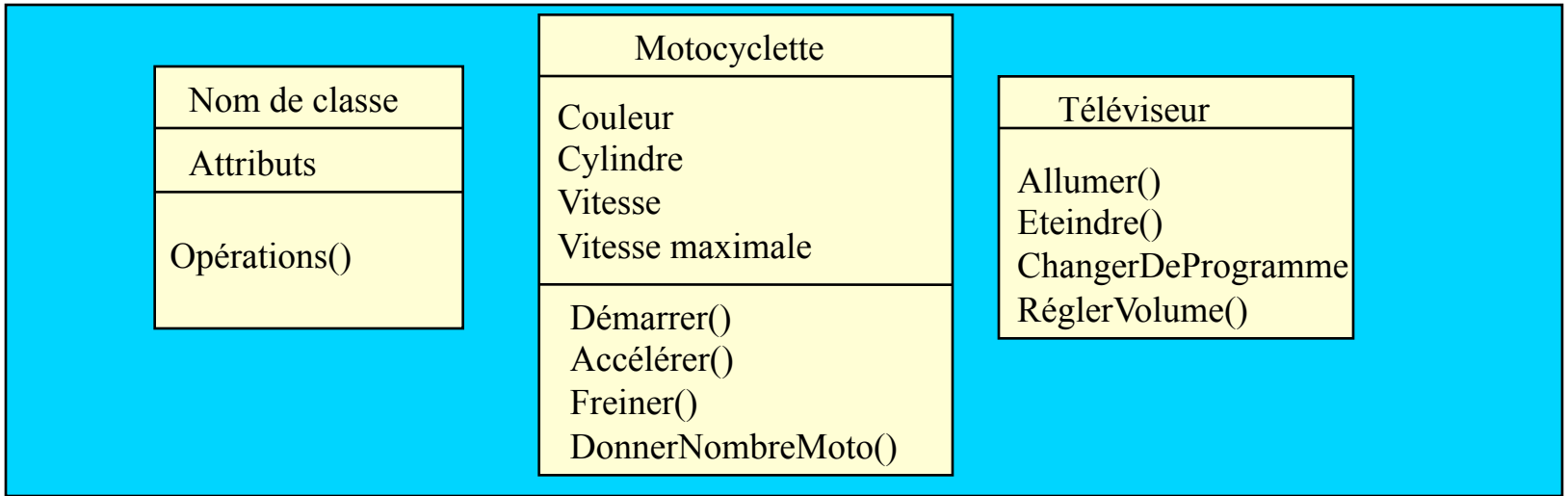
Les concepts objet à travers UML (16)



- **Classe**
  - Caractéristiques d'une classe
    - Un objet créé par (on dit également appartenant à) une classe sera appelé une *instance de cette classe* ce qui justifie le terme ``*variables d'instances*''
    - les valeurs des *variables d'instances* sont propres à chacune de ces instances et les caractérisent
    - Les généralités sont contenues dans la classe et les particularité sont contenues dans les objets
    - Les objets sont construits à partir de la classe, par un processus appelé instanciation : tout objet est une instance de classe
    - Nous distinguons deux types de classes
      - Classe concrète : peut être instaciée
      - Classe abstraite : est une classe qui ne donne pas directement des objets.

Les concepts objet à travers UML (17)

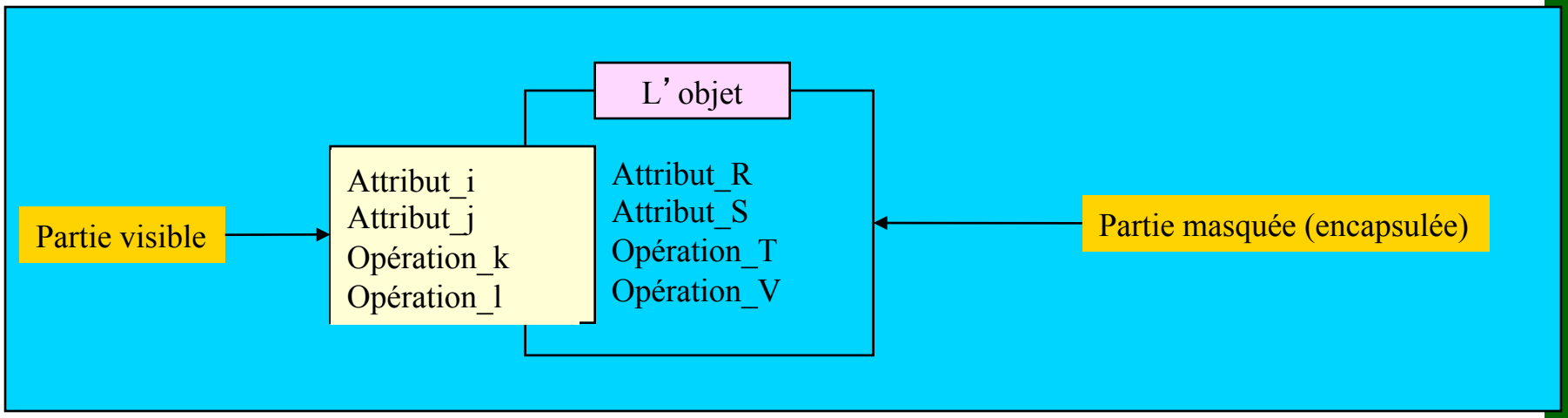
- **Classe**
  - Représentation graphique d'une classe en UML
    - Chaque classe est représentée sous la forme d'un rectangle divisé en trois compartiments
    - Les compartiments peuvent être supprimés pour alléger les diagrammes
    - Représentation des classes abstraites : le non d'une classe abstraite est en italique



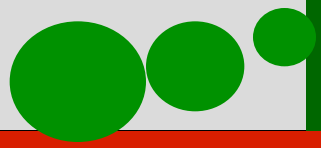
Les concepts objet à travers UML (18)

Encapsulation

- Consiste à masquer les détails d'implémentation d'un objet, en définissant une interface.
  - Est la séparation entre les propriétés externes, visibles des autres objets, et les aspects internes, propres aux choix d'implantation d'un objet.
- L'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet.



Les concepts objet à travers UML (19)



- **Encapsulation**
  - Présente un double avantage
    - facilite l'évolution d'une application car elle stabilise l'utilisation des objets : on peut modifier l'implémentation des attributs d'un objet sans modifier son interface
      - Les utilisateurs d'une abstraction ne dépendent pas de la réalisation de l'abstraction mais seulement de sa spécification : ce qui réduit le couplage dans les modèles
    - garantit l'intégrité des données, car elle permet d'interdire l'accès direct aux attributs des objets (utilisation d'accesseurs)
      - Les données encapsulées dans les objets sont protégées des accès intempestifs

Les concepts objet à travers UML (20)

Encapsulation

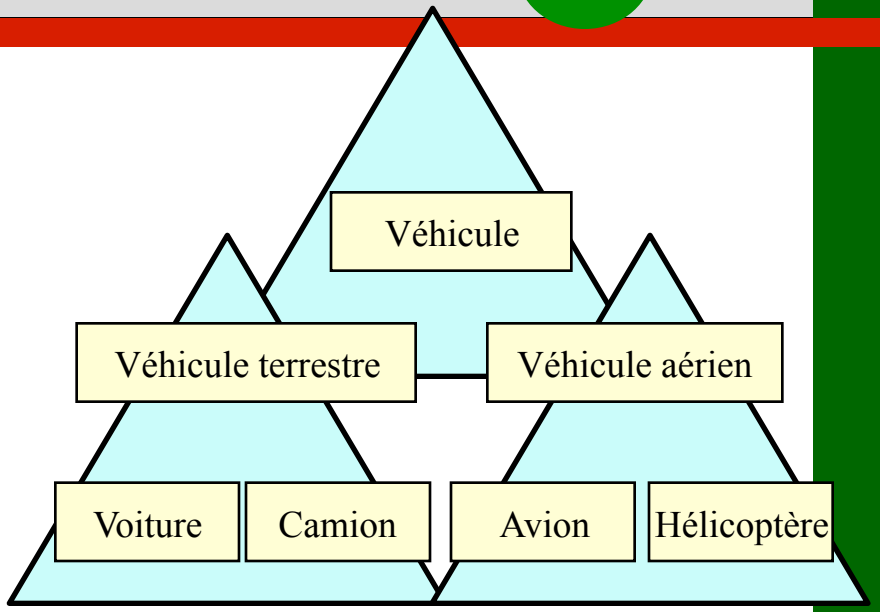
- Il est possible d'assouplir le degré d'encapsulation au profit de certaines classes utilisatrices bien particulières
  - En définissant des niveaux de visibilité
- Les trois niveaux distincts d'encapsulation couramment retenus sont:
  - Niveau privé : c'est le niveau le plus fort; la partie privée de la classe est totalement opaque
  - Niveau protégé : c'est le niveau intermédiaire ; les attributs placés dans la partie protégée sont visibles par les classes dérivées de la classe fournisseur. Pour toutes les autres classes, les attributs restent invisibles
  - Niveau publique : ceci revient à se passer de la notion d'encapsulation et de rendre visibles les attributs pour toutes les classes

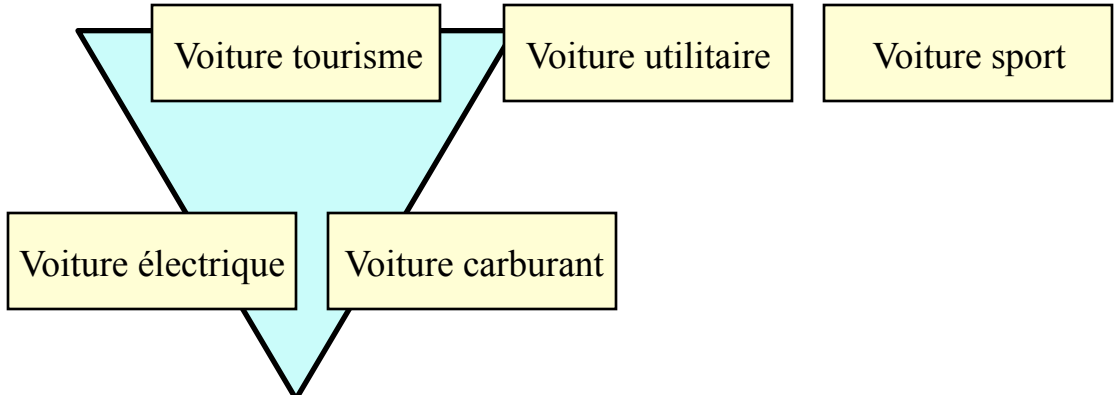
Règle de visibilité
+ Attribut public # Attribut protégé - Attribut privé
+ Opération publique() # Opération protégée() - Opértion privée()

Salarié
+ nom # age - salaire
+ donnerSalaire() # changerSalaire() - calculerPrime()

Les concepts objet à travers UML (21)

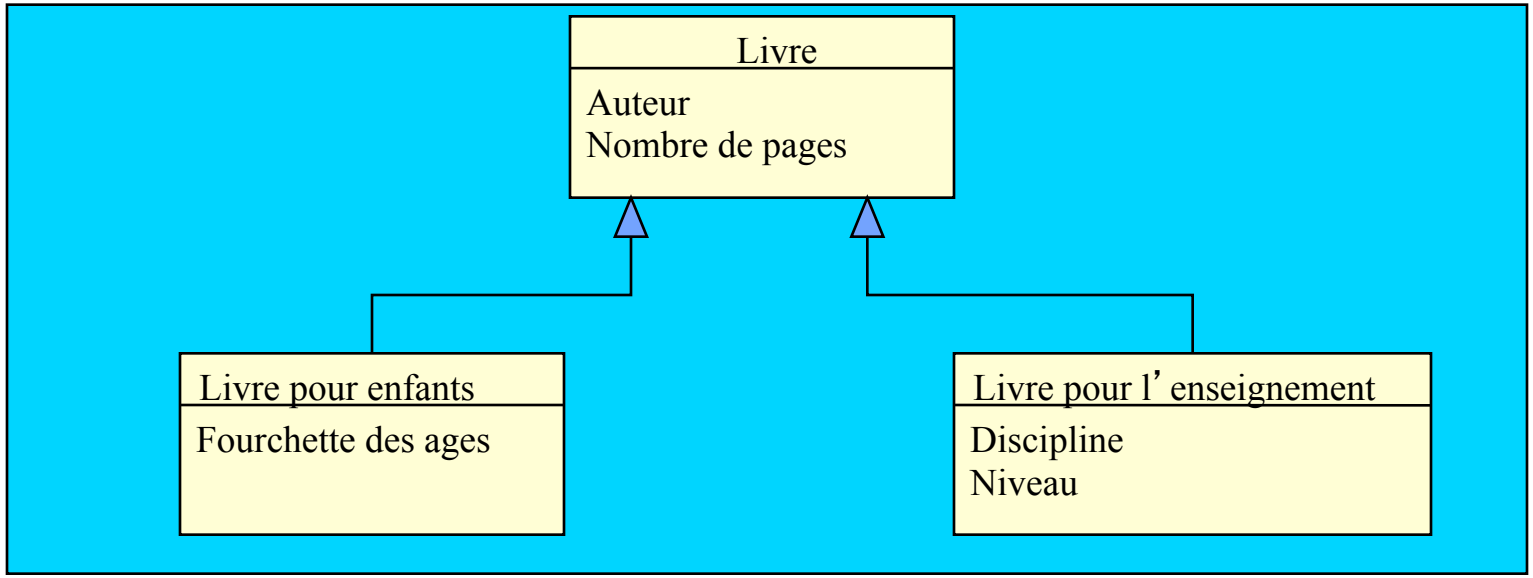
- **Les hiérarchies de classes**
  - Généralisation et spécialisation
    - La généralisation consiste à factoriser les éléments communs (attributs, opérations) d'un ensemble de classes dans une classe plus générale appelée super-classe
    - La spécialisation permet de capturer les particularités d'un ensemble d'objets non discriminés par les classes déjà identifiées





Les concepts objet à travers UML (22)

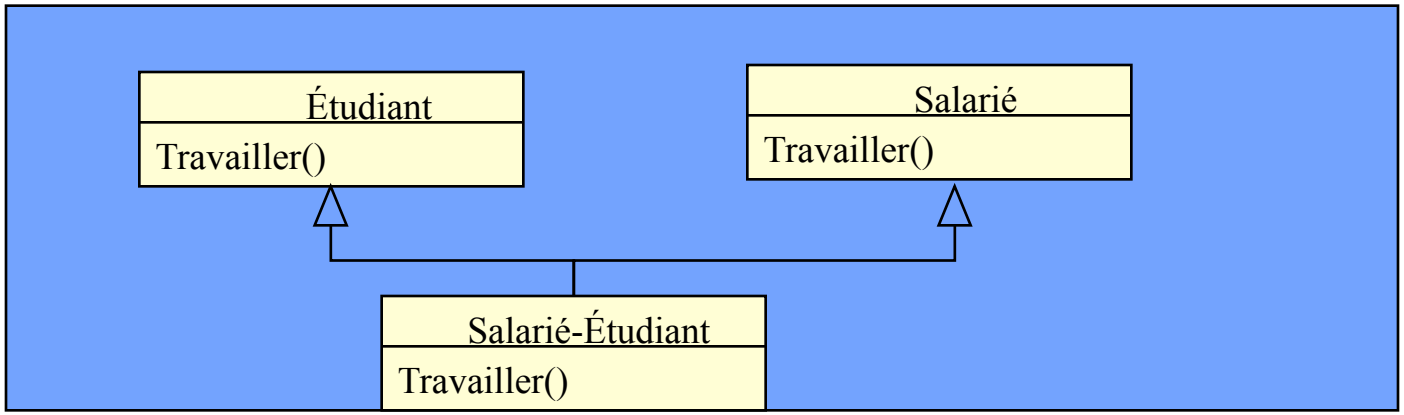
- **Les hiérarchies de classes**
  - Les propriétés d'une sous-classe englobe les propriétés de toutes ses super-classes
    - Ce qui est vrai pour un objet instance d'une super classe est vrai pour un objet instance d'une sous-classe



Les concepts objet à travers UML (23)

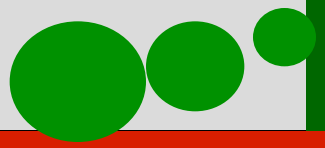
Les hiérarchies de classes

- Généralisation multiple : elle existe entre arbres de classes disjoints
  - Une classe n peut posséder q'une fois une propriété donné



- Problème de classification
  - L'établissement d'une hiérarchie (classification) dépend du point de vue
    - Pas une seule hiérarchie de classe mais des hiérarchies, chacune adaptée à un usage donné
    - Exemple :les animaux
      - Critères de classification : type de nourriture, la protection

Les concepts objet à travers UML (24)



Les hiérarchies de classes

- L'héritage
  - Technique utilisée par les langages de programmation pour réaliser les hiérarchies de généralisation/spécialisation

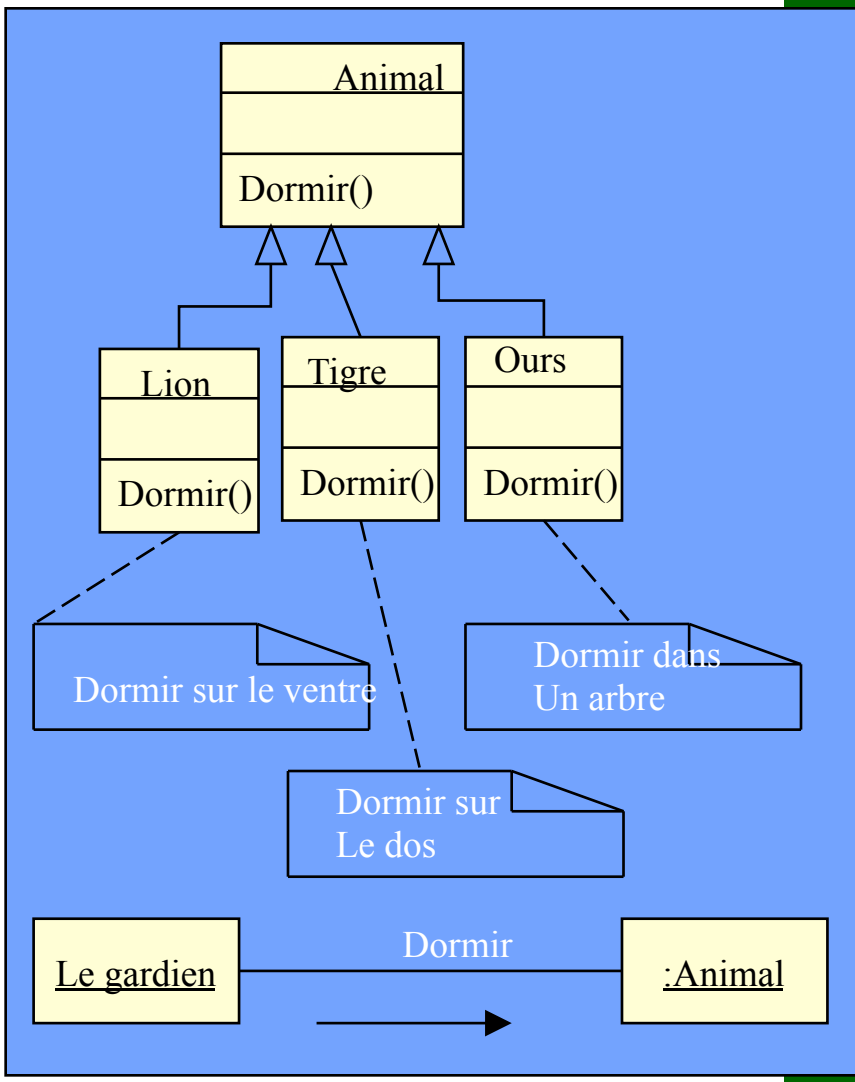
- Le polymorphisme
  - Le terme polymorphisme décrit la caractéristique d'un élément qui peut prendre plusieurs formes, comme l'eau qui se trouve à l'état solide, liquide ou gazeux

  - En informatique : le polymorphisme désigne le fait qu'un nom d'objet peut désigner des instances de classes différentes issues d'une même arborescence

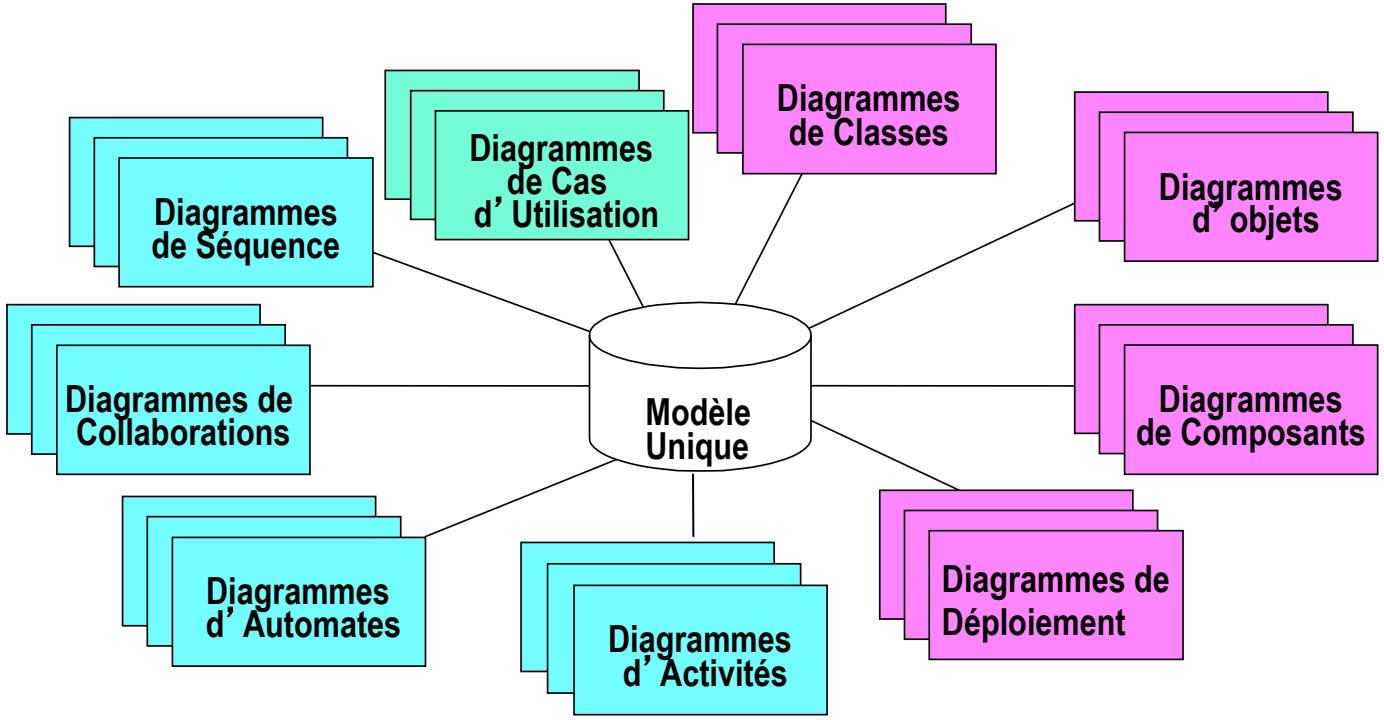
Les concepts objet à travers UML (25)

Les hiérarchies de classes

- Polymorphisme : notion de surcharge
  - Le polymorphisme signifie qu'un même opération peut se comporter différemment sur différents classes
    - L'opération « déplacer » agira de manières différentes sur un fichier, une fenêtre graphique ou un véhicule
  - Le polymorphisme signifie que les différentes méthodes d'une opération ont la même signature
    - Lorsque une opération est invoquée sur un objet, celui-ci « connaît » sa classe et par conséquent est capable d'invoquer automatiquement la méthode correspondante.

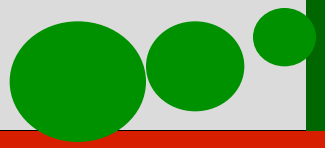


Partie II : Les diagrammes UML



Les vues statiques d'un système en UML

DIAGRAMME DE CLASSES (1)



Sémantique

- Un diagramme de classes est une collection d'éléments de modélisation statiques (classes, paquetages...), qui montre la structure d'un modèle.

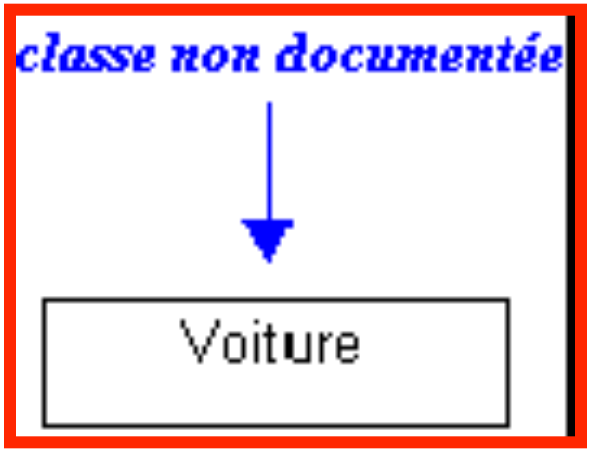
- Un diagramme de classes fait abstraction des aspects dynamiques et temporels

- Pour un modèle complexe, plusieurs diagrammes de classes complémentaires doivent être construits.
  - On peut par exemple se focaliser sur :
    - les classes qui participent à un cas d'utilisation
    - les classes associées dans la réalisation d'un scénario précis,
    - les classes qui composent un paquetage,

- Pour représenter un contexte précis, un diagramme de classes peut être instancié en diagrammes d'objets

DIAGRAMME DE CLASSES (2)

- Documentation d'une classe



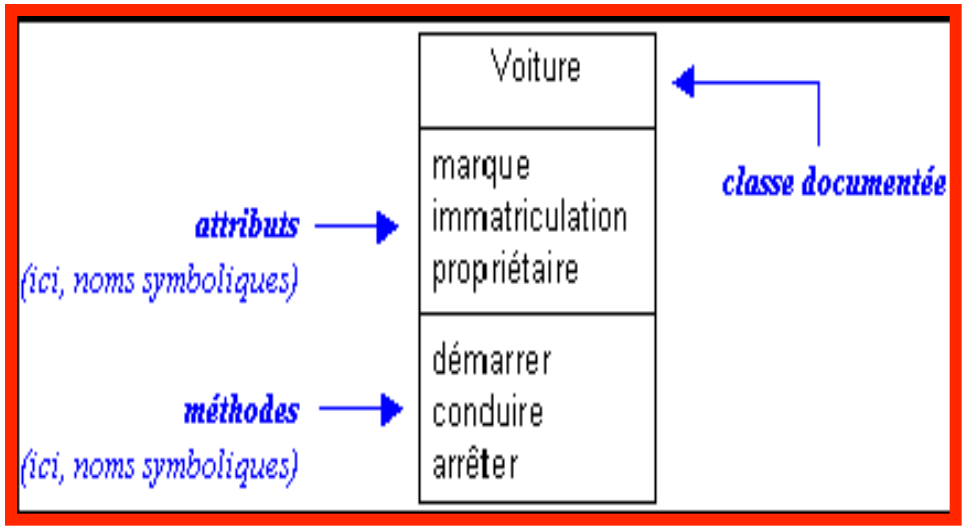


DIAGRAMME DE CLASSES (3)

- Documentation d'une classe

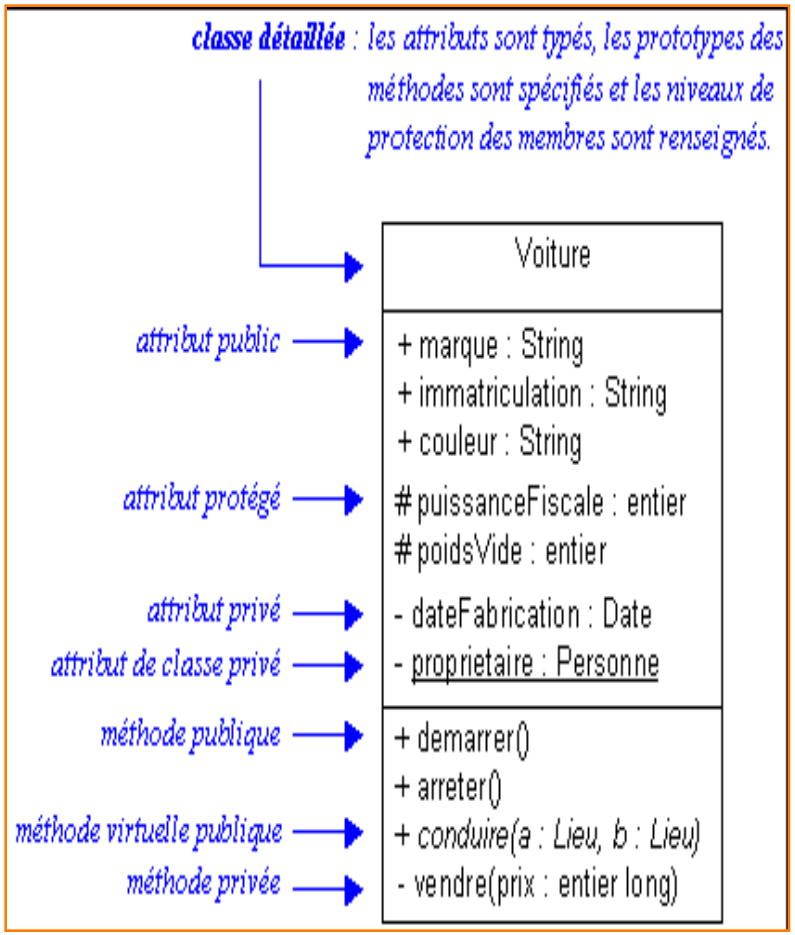
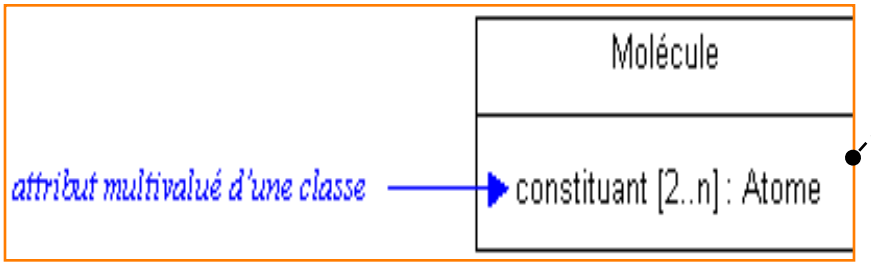


DIAGRAMME DE CLASSES (4)

- Attributs multivalués (attributs de type collection)



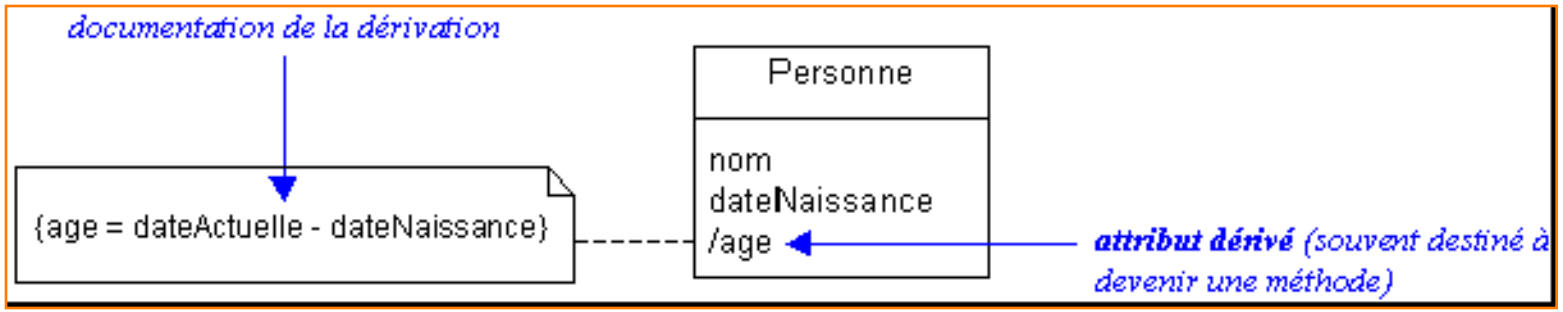
Type Bag : éléments non ordonnés, non uniques

Type OrderedSet : Elts ordonnés et unqiues

Type Set : Elts non ordonnées mais uniques

Type Sequence : Elts ordonnés mais pas uniques

- Attributs dérivés

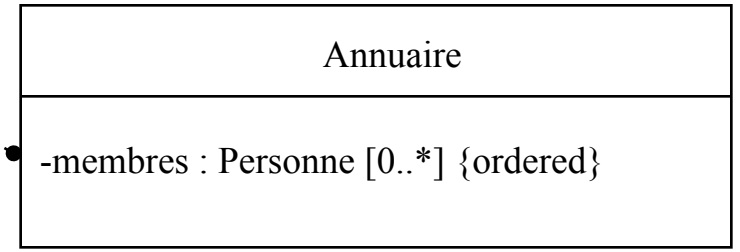


- Typiquement accessible en lecture seule (readOnly)

DIAGRAMME DE CLASSES (5)

- Attributs ordonnés

L'attribut membre sera stocké séquentiellement;
Dans le cas de l'exemple Annuaire l'attribut « membre » sera stocké dans l'ordre alphabétique



- Attributs uniques

Chacun des éléments d'un attribut « unique » doit être unique : ne figure qu'une fois dans l'attribut

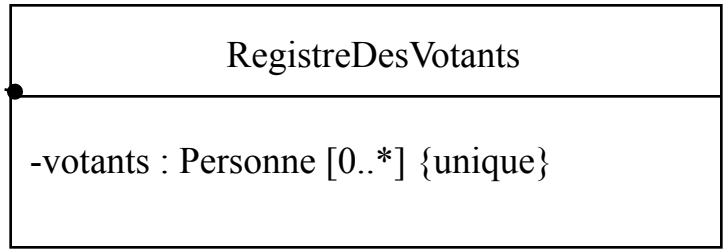
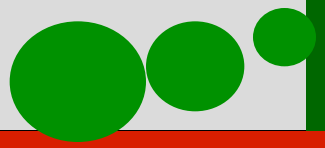


DIAGRAMME DE CLASSES (6)



- Documentation d'une classe : les opérations

Les opérations utilisent la notation suivante

Visibilité nom (paramètres) : type-retourné {propriétés}

Les paramètres sont écrits sous la forme suivante :

direction nom_paramètre : type [multiplicité]
= valeur_par_défaut {propriétés}

Visibilité : public, private, protected, package

Nom : nom permettant de désigner l'opération (généralement un verbe représentant une action)

Type-retourné : type de l'information retournée par l'opération, s'il y a une.

propriétés : spécifie les contraintes et les propriétés associées à l'opération.

Les éléments syntaxiques relatifs aux paramètres sont :

direction : indique la façon dont un paramètre est utilisé par une opération. Prends une des valeurs : IN, INOUT, OUT, RETURN

nom_paramètre : nom du paramètre.

type : type du paramètre

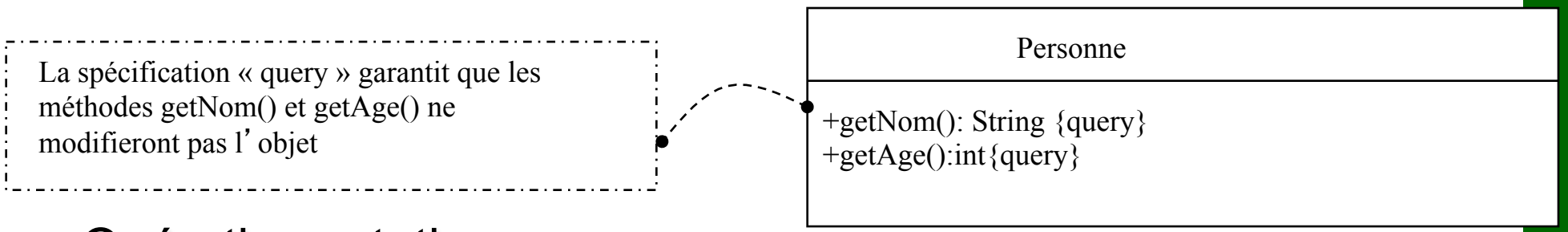
Multiplicité : nombre d'instances (du type spécifié) fournies par le paramètre

Valeur_par_défaut : spécifie la valeur par défaut de ce paramètre

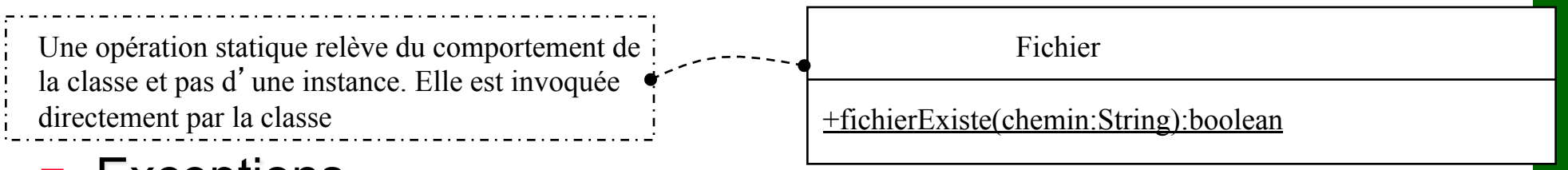
Proprités : spécifie les propriétés liées au paramètre : ORDERED, READONLY, UNIQUE

DIAGRAMME DE CLASSES (7)

- Opérations de type requête (query)



- Opérations statiques



- Exceptions

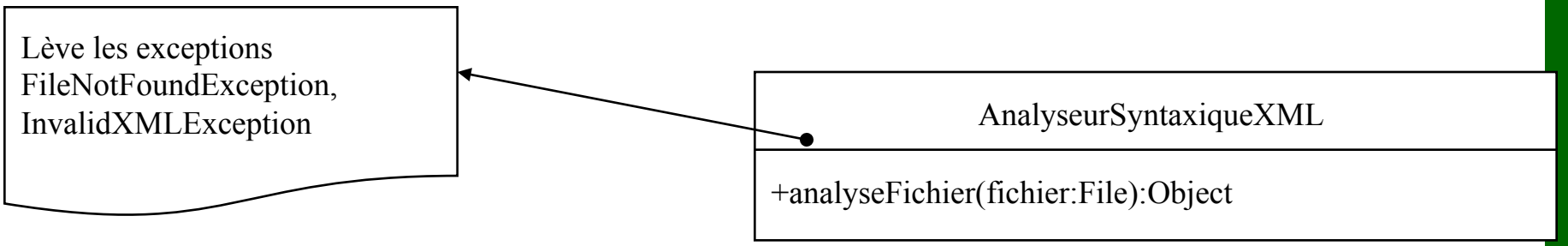


DIAGRAMME DE CLASSES (8)

- Classe abstraite

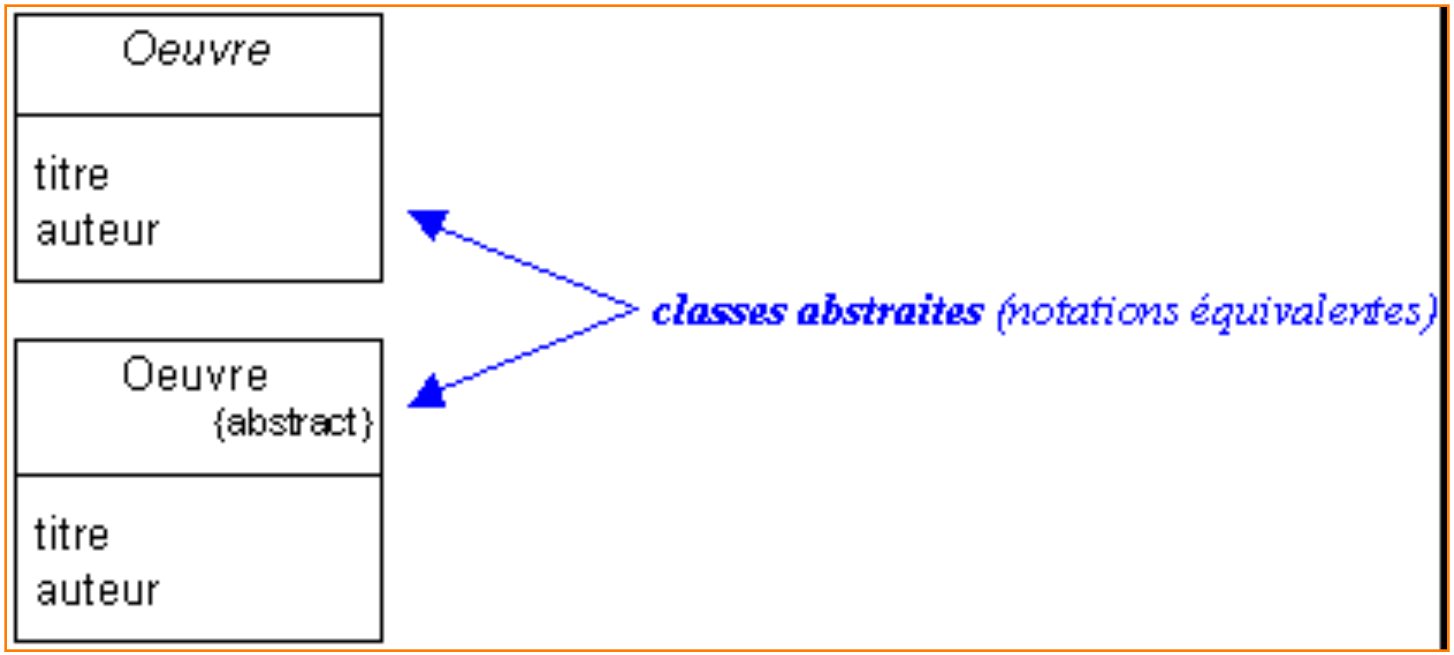


DIAGRAMME DE CLASSES (9)

- Template (classe paramétrable):

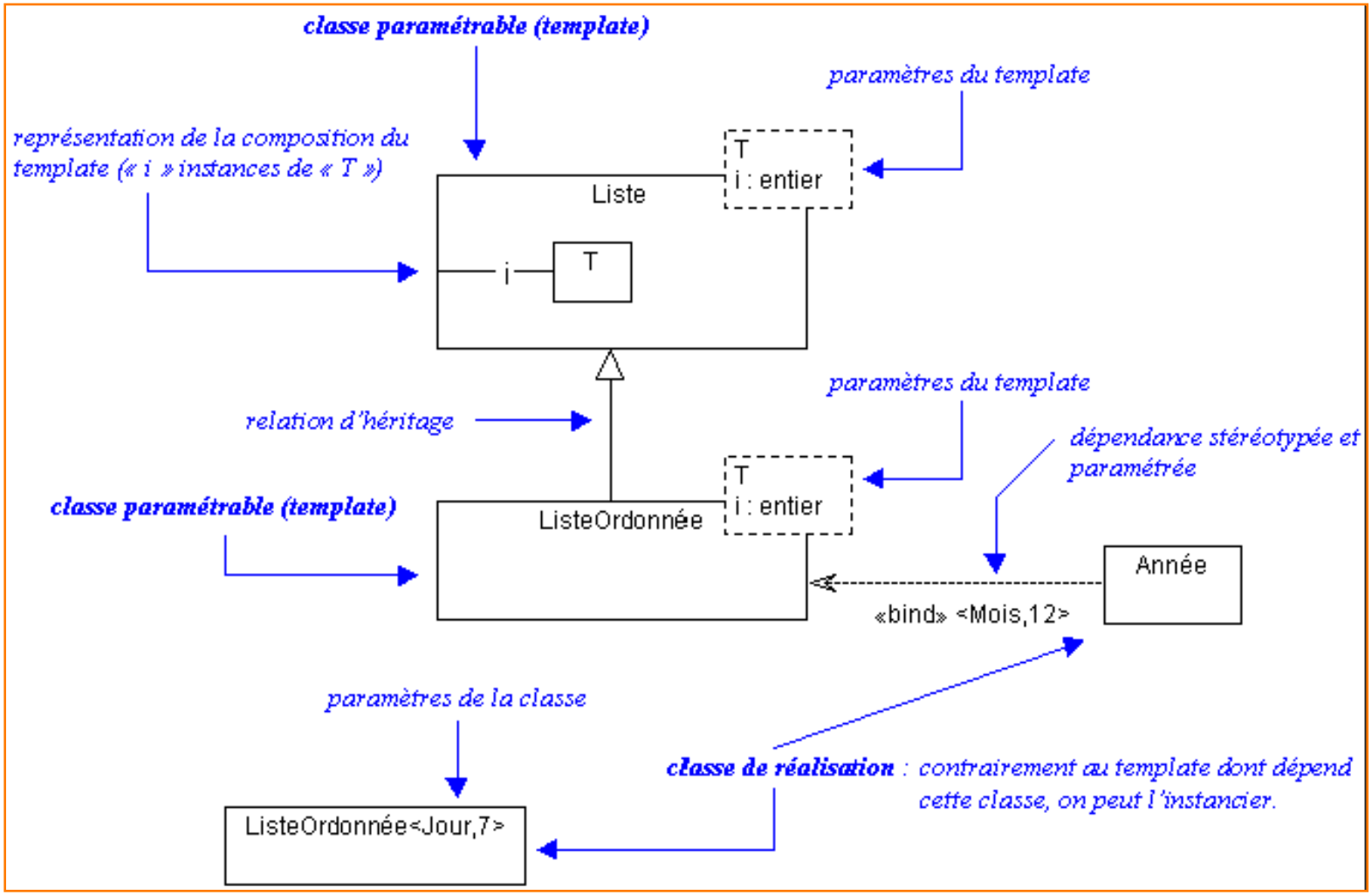
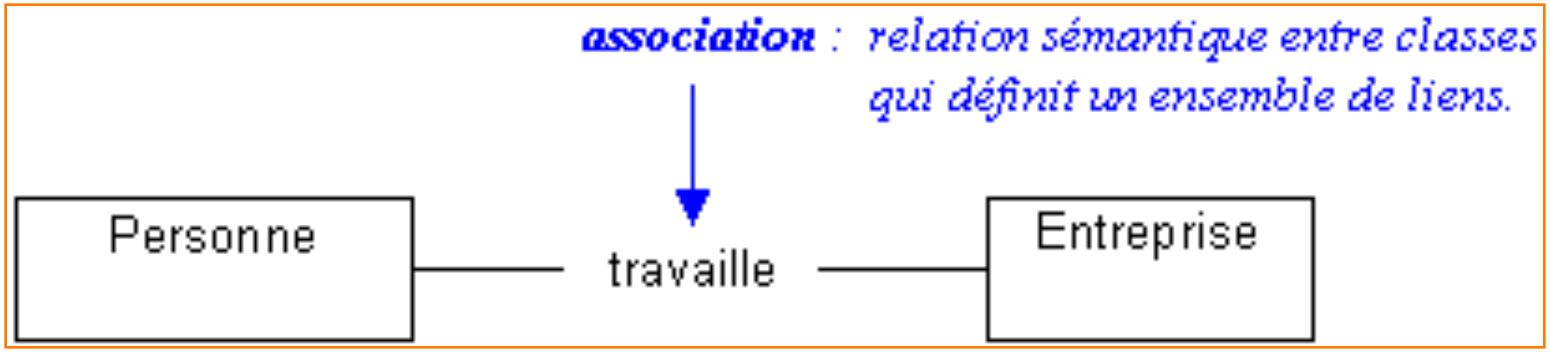


DIAGRAMME DE CLASSES (10)

- Associations entre classes
  - Une association exprime une connexion sémantique bidirectionnelle entre deux classes.



  - L'association est instanciable dans un diagramme d'objets ou de collaboration, sous forme de liens entre objets issus de classes associées.

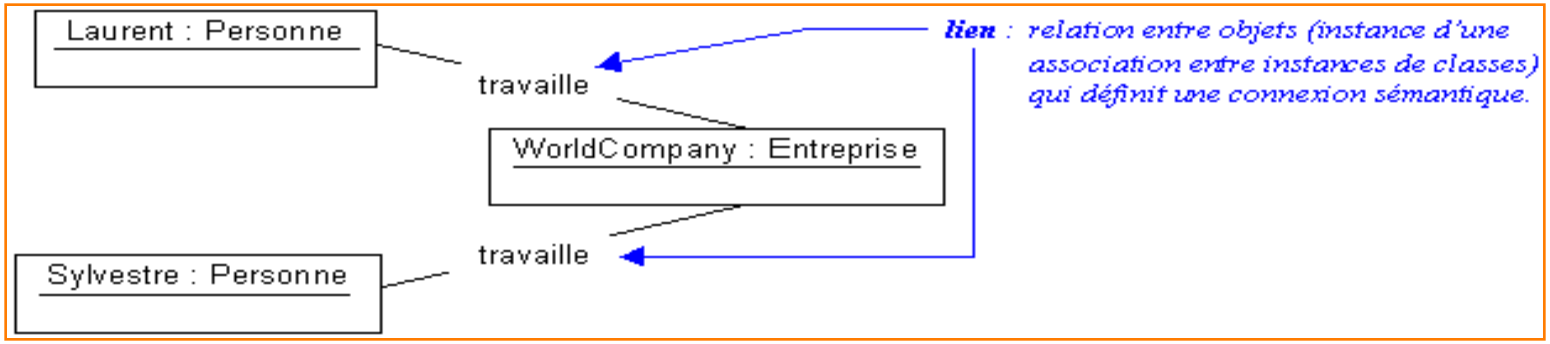


DIAGRAMME DE CLASSES (11)

- Associations entre classes
  - Une association est caractérisée par:
    - un nom, qui peut être omis notamment quand les rôles des classes sont spécifiés
    - un rôle pour chacune des classes qui participent à l'association.
    - une multiplicité

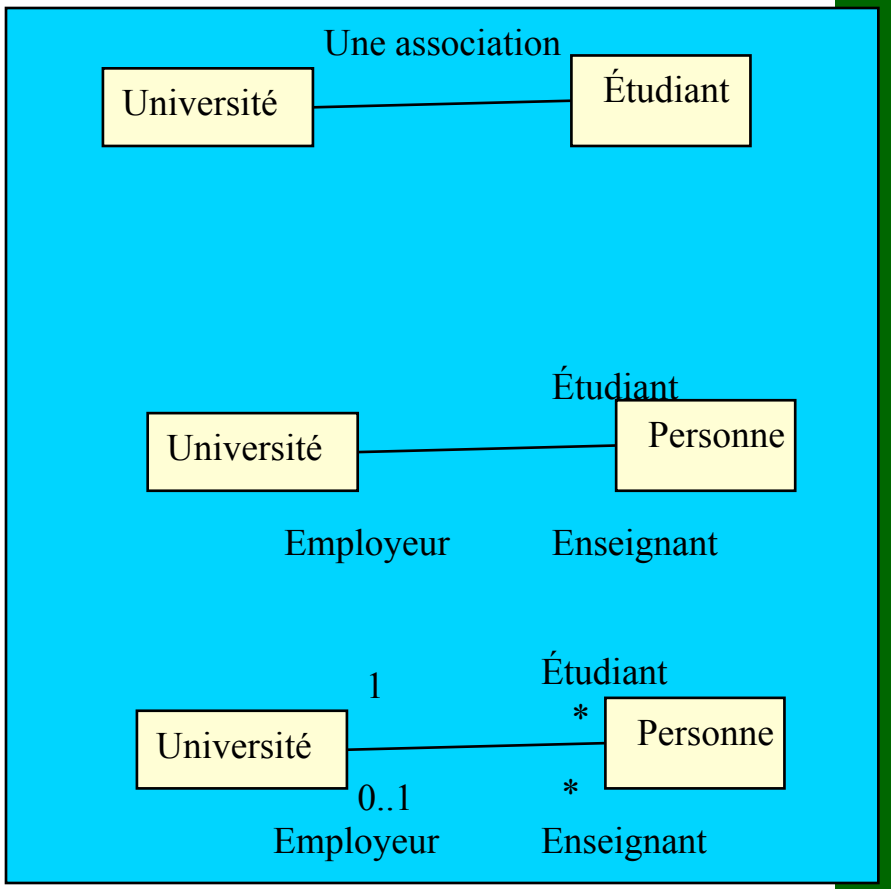


DIAGRAMME DE CLASSES (12)

- Associations entre classes
  - Multiplicité
    - Chaque extrémité d'une association peut porter une indication de multiplicité (nombre d'occurrences) qui montre combien d'objets de la classe considérée peuvent être liés à un objet de l'autre classe.

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

DIAGRAMME DE CLASSES (13)

- Associations entre classes
  - Le nom de l'association peut être suivi d'une flèche de direction (triangle plein) qui précise le sens de lecture
  - Il existe des associations ternaires (ou d'ordre supérieur) qui sont représentées par un diamant relié à chacune des classes participant à la relation.

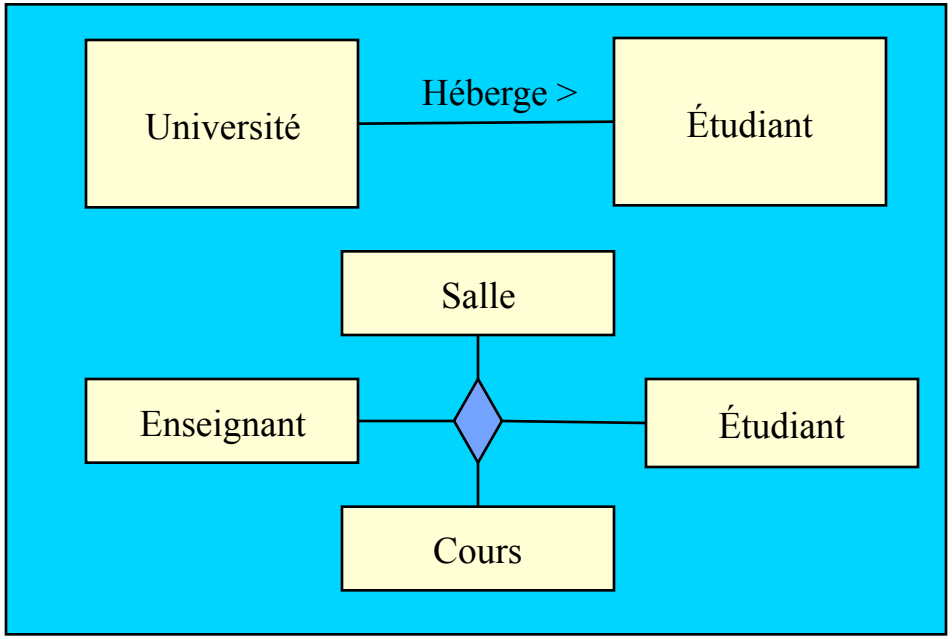


DIAGRAMME DE CLASSES (14)

- Associations entre classes
  - Documentation d'une association et types d'associations
    - Association en forme verbale active : précise le sens de lecture principal d'une association.

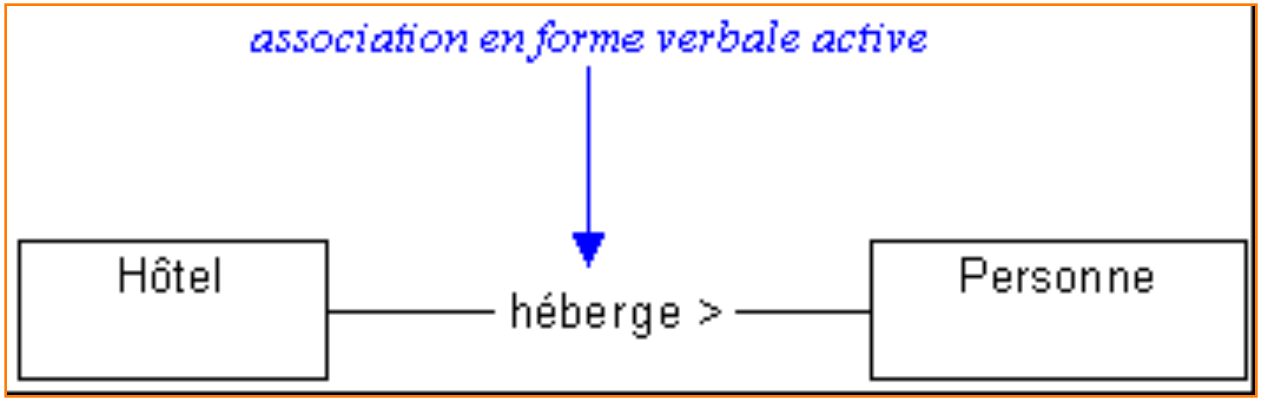
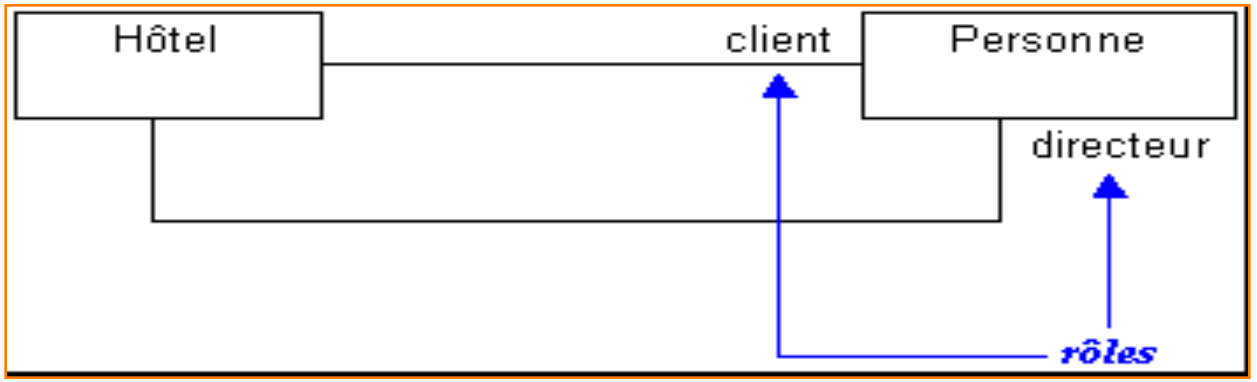


DIAGRAMME DE CLASSES (15)

- Associations entre classes
  - Rôles
    - Spécifie la fonction d'une classe pour une association donnée



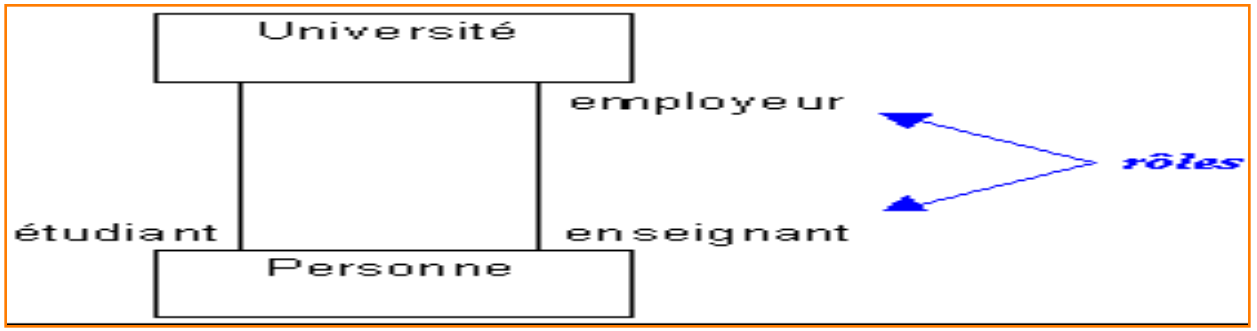


DIAGRAMME DE CLASSES (16)

- Associations entre classes
  - Relation de dépendance
    - Relation d'utilisation unidirectionnelle et d'obsolescence
      - Une modification de l'élément dont on dépend, peut nécessiter une mise à jour de l'élément dépendant.

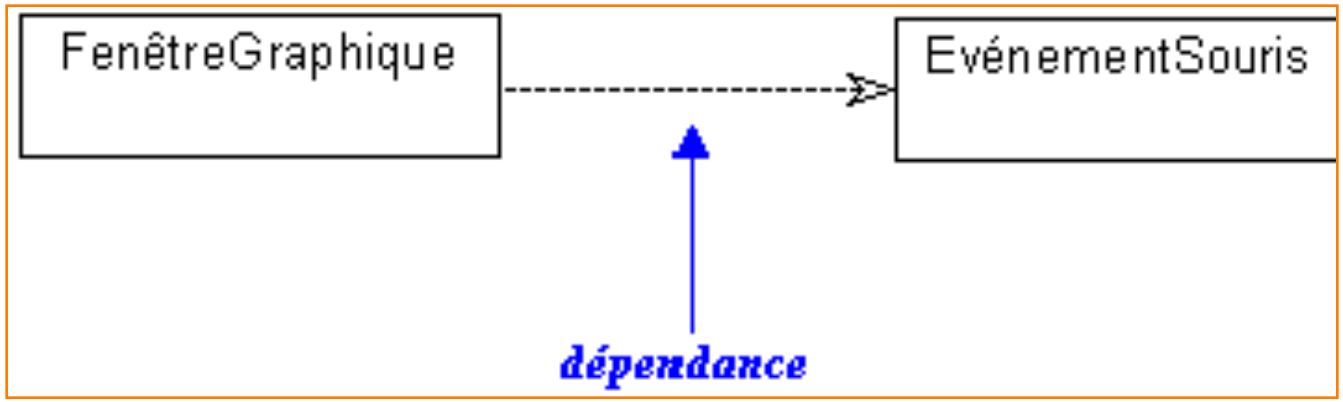


DIAGRAMME DE CLASSES (17)

- Associations entre classes
  - Association à navigabilité restreinte
    - Par défaut, une association est navigable dans les deux sens.
    - La réduction de la portée peut aussi être exprimée dans un modèle pour indiquer que les instances d'une classe ne "connaissent" pas les instances d'une autre.

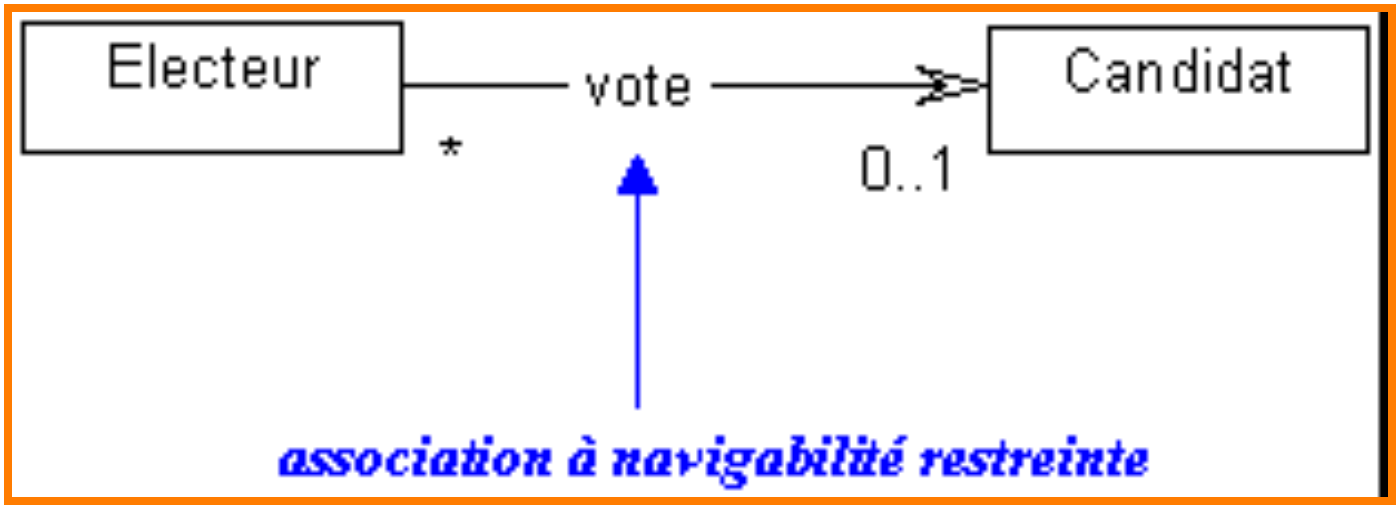


DIAGRAMME DE CLASSES (18)

- Associations entre classes
  - Association n-aire
    - Il s'agit d'une association qui relie plus de deux classes

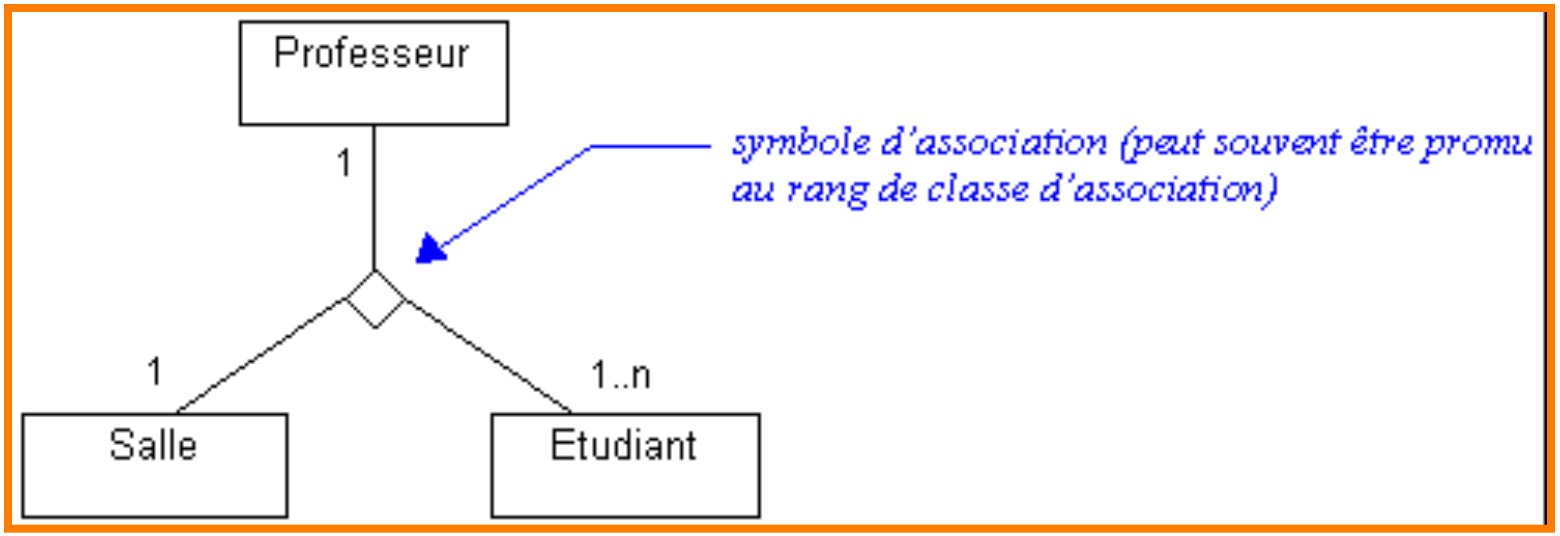
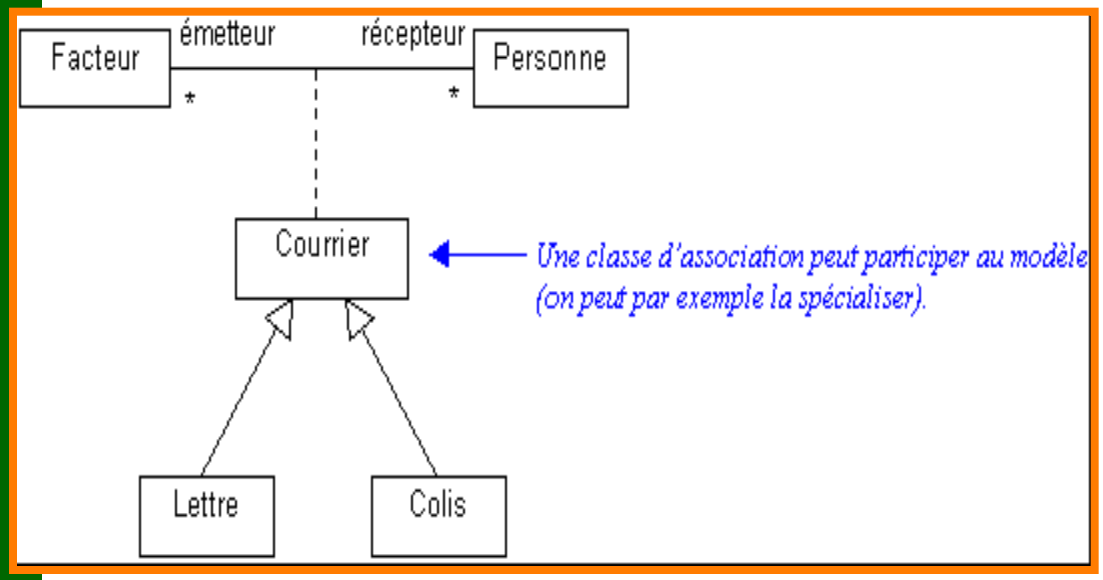


DIAGRAMME DE CLASSES (19)

- Associations entre classes
  - Classe d'association
    - Il s'agit d'une classe qui réalise la navigation entre les instances d'autres classes



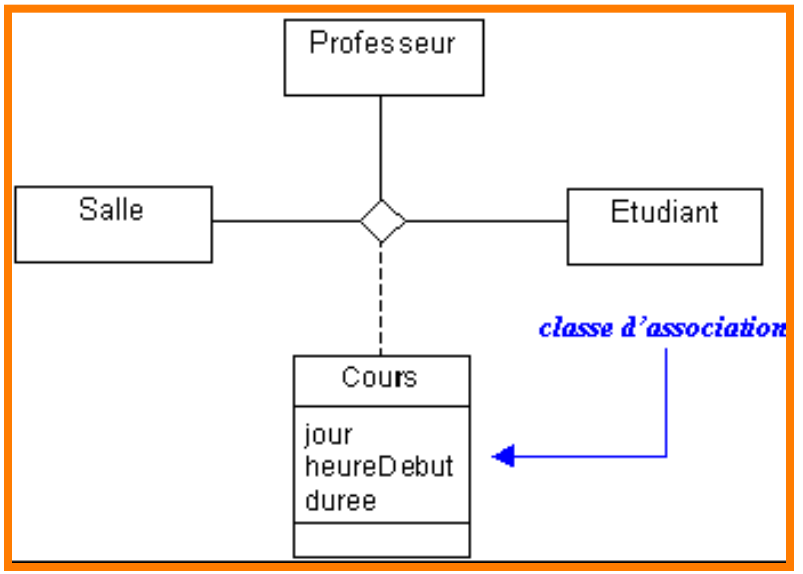


DIAGRAMME DE CLASSES (20)

- Associations entre classes
  - Qualification
    - Permet de sélectionner un sous-ensemble d'objets, parmi l'ensemble des objets qui participent à une association.
    - La restriction de l'association est définie par une clé, qui permet de sélectionner les objets ciblés.

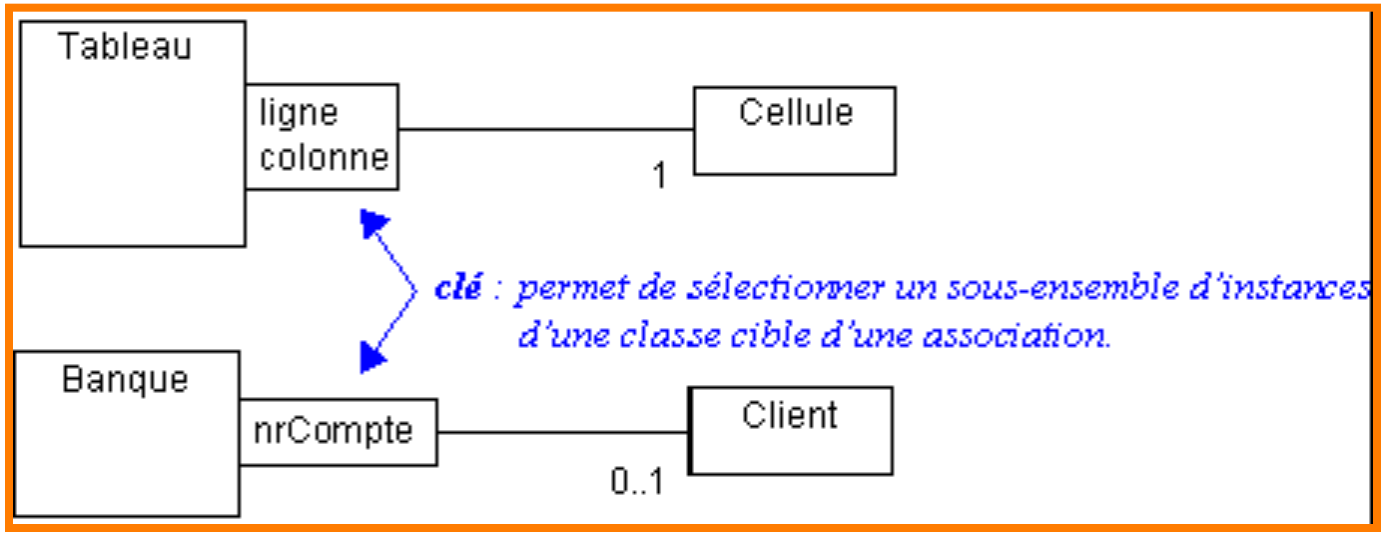


DIAGRAMME DE CLASSES (21)

- Associations entre classes
  - Association dérivée
    - Les associations dérivées sont des associations redondantes, qu'on peut déduire d'une autre association ou d'un ensemble d'autres associations.
    - Elles permettent d'indiquer des chemins de navigation "calculés", sur un diagramme de classes
    - Elles servent beaucoup à la compréhension de la navigation (comment joindre telles instances d'une classe à partir d'une autre).

DIAGRAMME DE CLASSES (22)

- Associations entre classes
  - Association dérivée
    - Exemple

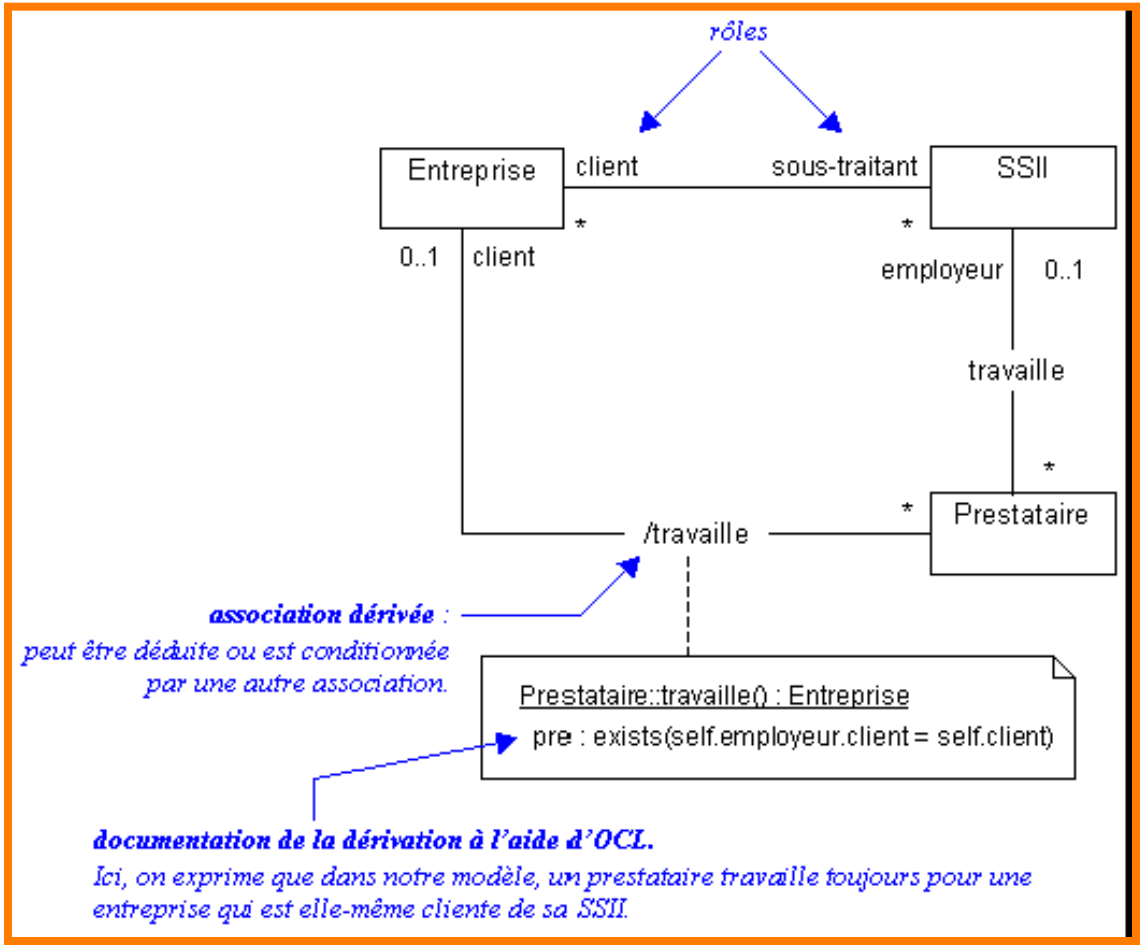
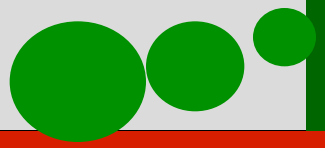


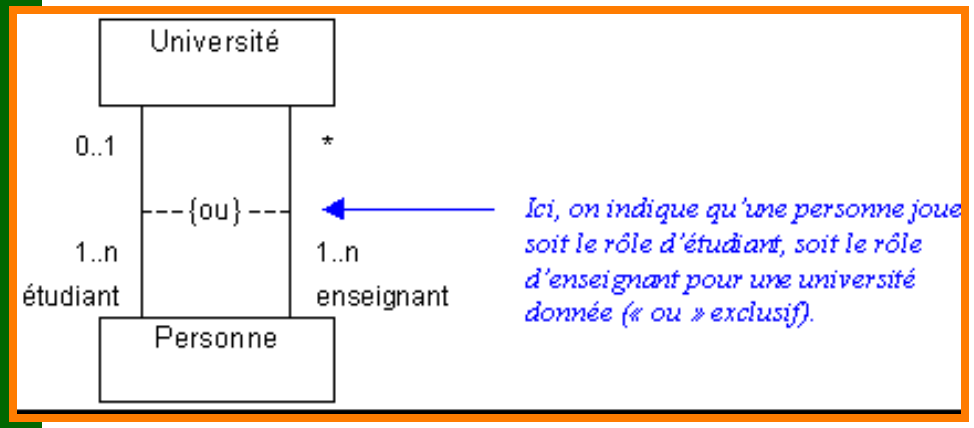
DIAGRAMME DE CLASSES (23)

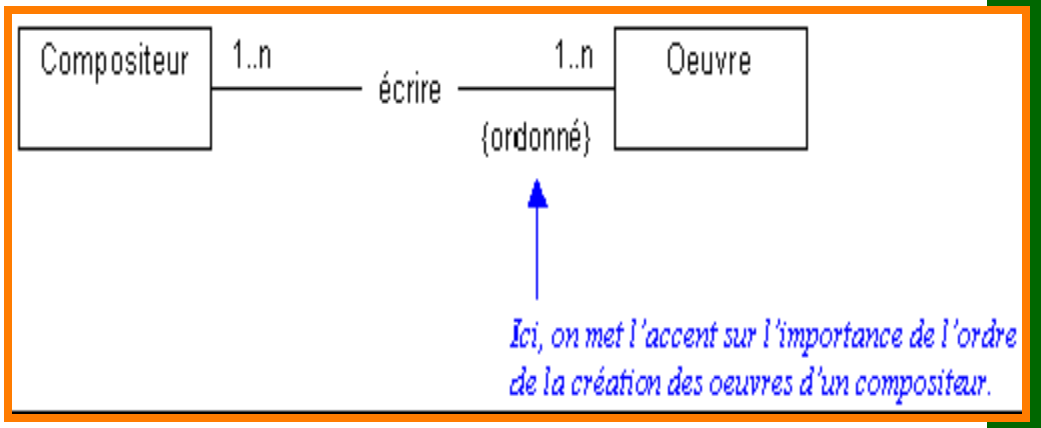


- Associations entre classes
  - Contrainte sur une association
    - Les contraintes sont des expressions qui précisent le rôle ou la portée d'un élément de modélisation
      - Elles permettent d'étendre ou préciser sa sémantique
    - Sur une association, elles peuvent par exemple restreindre le nombre d'instances visées
    - Les contraintes peuvent s'exprimer en langage naturel
      - Graphiquement, il s'agit d'un texte encadré d'accolades.

DIAGRAMME DE CLASSES (24)

- Associations entre classes
  - Contrainte sur une association





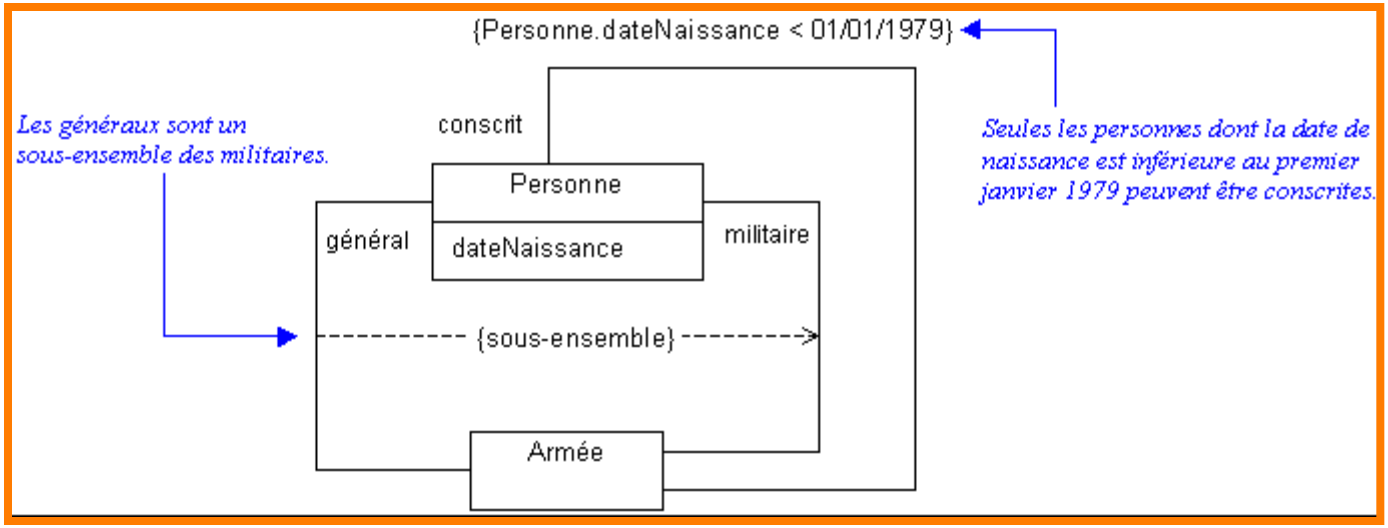


DIAGRAMME DE CLASSES (25)

- Associations entre classes
  - Contrainte sur une association : OCL
    - UML formalise l'expression des contraintes avec OCL (Object Constraint Language).
    - OCL est une contribution d'IBM à UML 1.1.
    - Ce langage formel est volontairement simple d'accès et possède une grammaire élémentaire (OCL peut être interprété par des outils).
    - Il représente un juste milieu, entre langage naturel et langage mathématique. OCL permet ainsi de limiter les ambiguïtés, tout en restant accessible.

DIAGRAMME DE CLASSES (26)

- Associations entre classes
  - Contrainte sur une association
    - Exemple

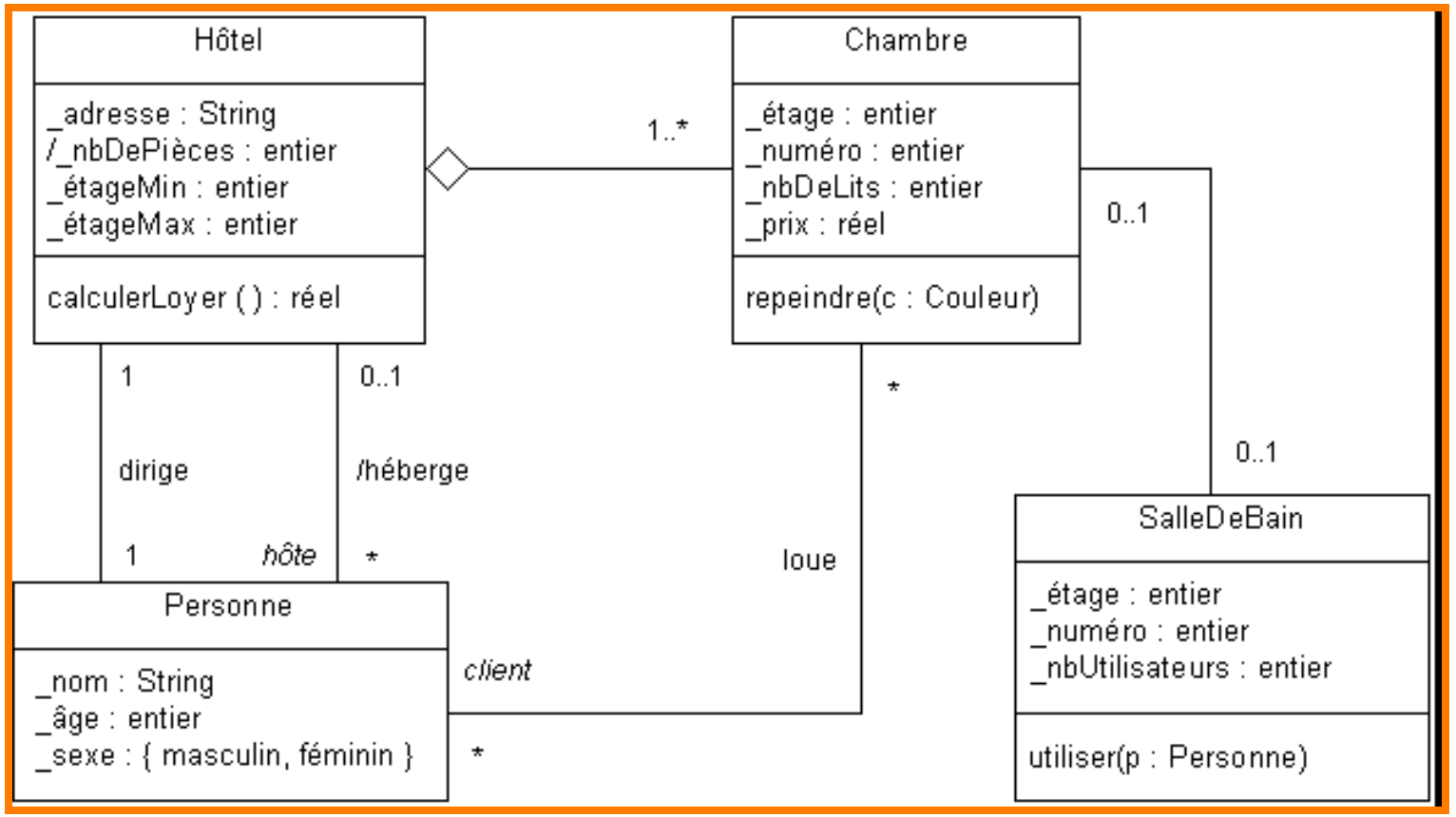
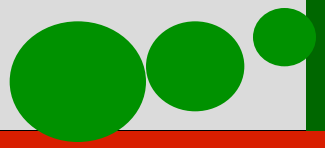


DIAGRAMME DE CLASSES (27)



- Associations entre classes
  - Contrainte sur une association
    - Exemple
      - *Le nombre de personnes par chambre doit être inférieur ou égal à* 3.
      - *l'étage de chaque chambre est compris entre le premier et le dernier étage de l'hôtel.*

- context Chambre inv: self.client <= 3
- context Hôtel inv:
  self.chambre→forAll(c : Chambre | c._étage<= self._étageMax and
  c._étage >= self._étageMin)

- *Un hôtel Formule 1 ne contient jamais d'étage numéro 13*

context Chambre inv: self._étage <> 13
context SalleDeBain inv: self._étage <> 13

DIAGRAMME DE CLASSES(28)

- Classification, spécialisation, généralisation et héritage
  - Exemple (voir transparents partie I)

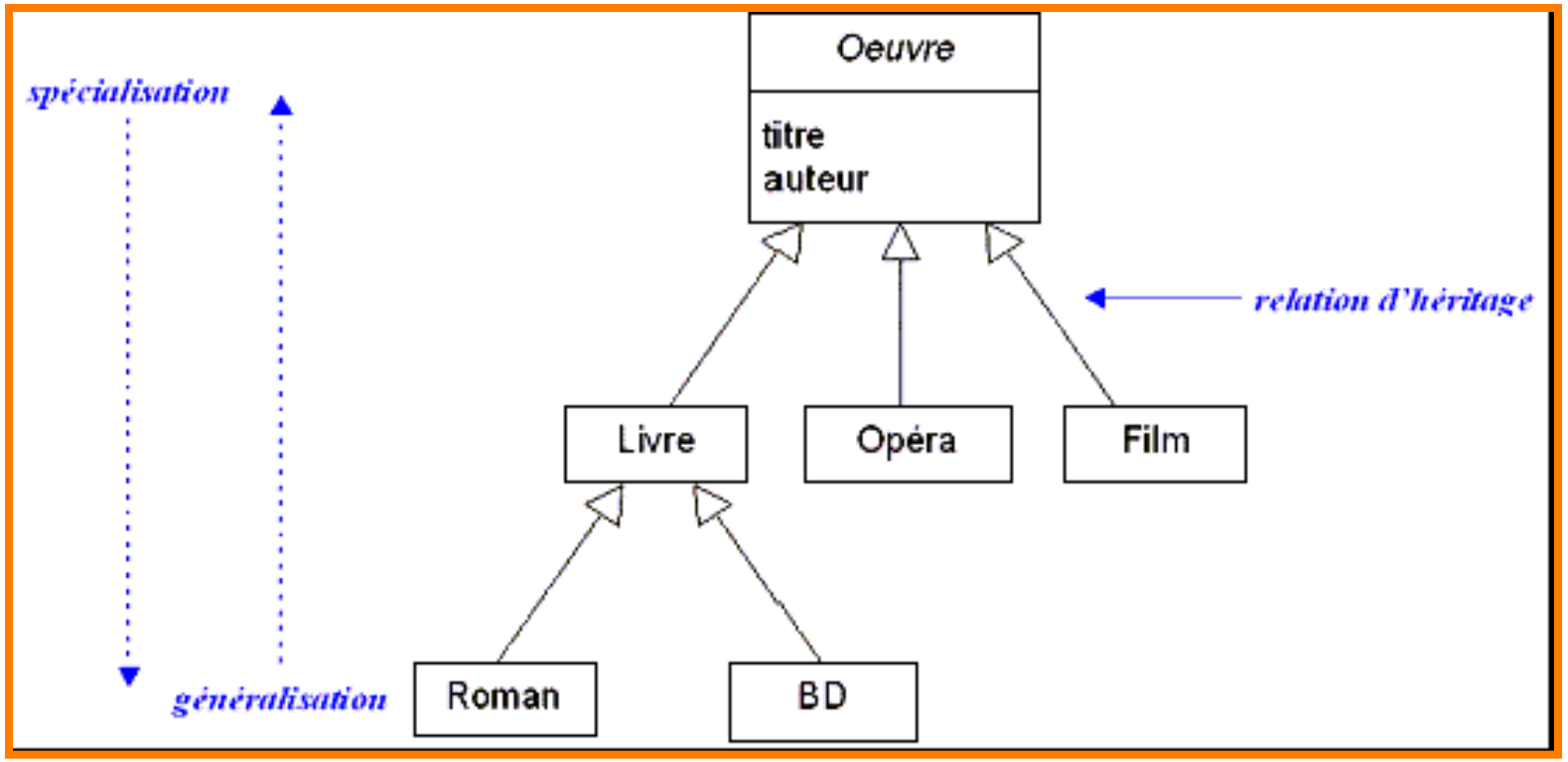


DIAGRAMME DE CLASSES(29)

- Agrégation
  - L'agrégation est une association non symétrique, qui exprime un couplage fort et une relation de subordination.
  - Elle représente une relation de type "ensemble / élément".
  - UML ne définit pas ce qu'est une relation de type "ensemble / élément", mais il permet cependant d'exprimer cette vue subjective de manière explicite.

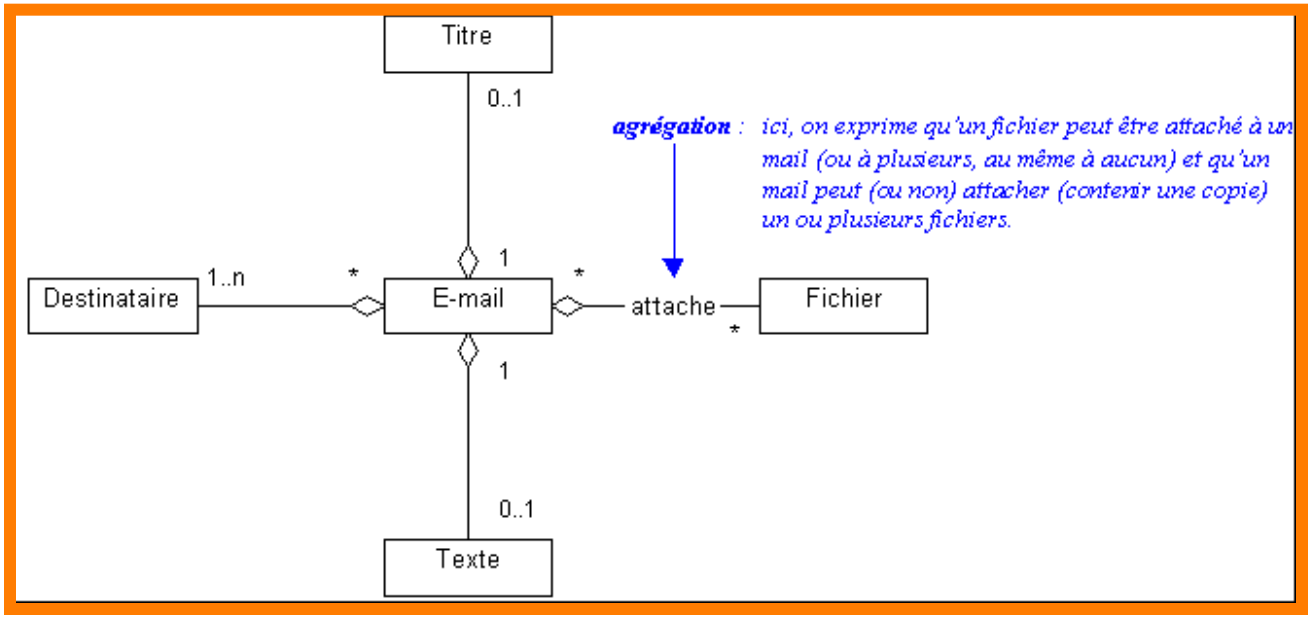


DIAGRAMME DE CLASSES (30)

- Composition
  - La composition est une agrégation forte (agrégation par valeur).
  - Les cycles de vies des éléments (les "composants") et de l'agrégat sont liés : si l'agrégat est détruit (ou copié), ses composants le sont aussi.
  - A un même moment, une instance de composant ne peut être liée qu'à un seul agrégat.
  - Les "objets composites" sont des instances de classes composées.

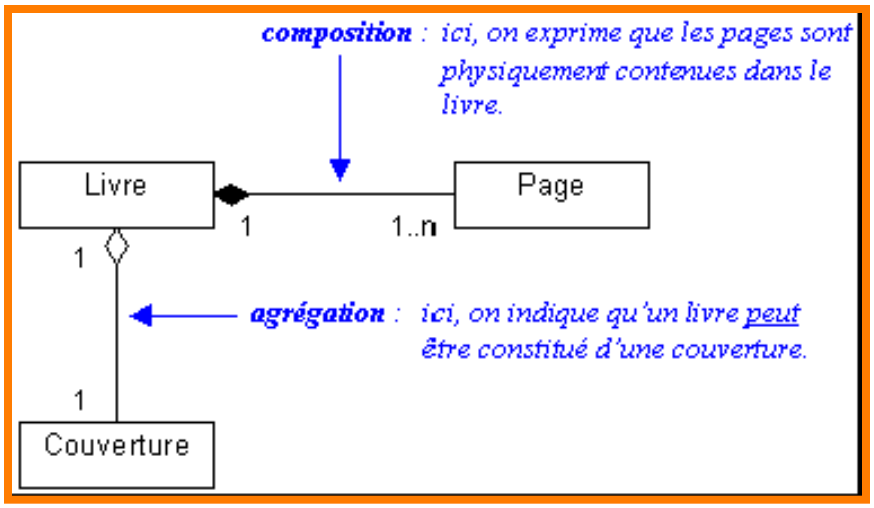
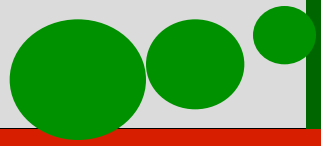


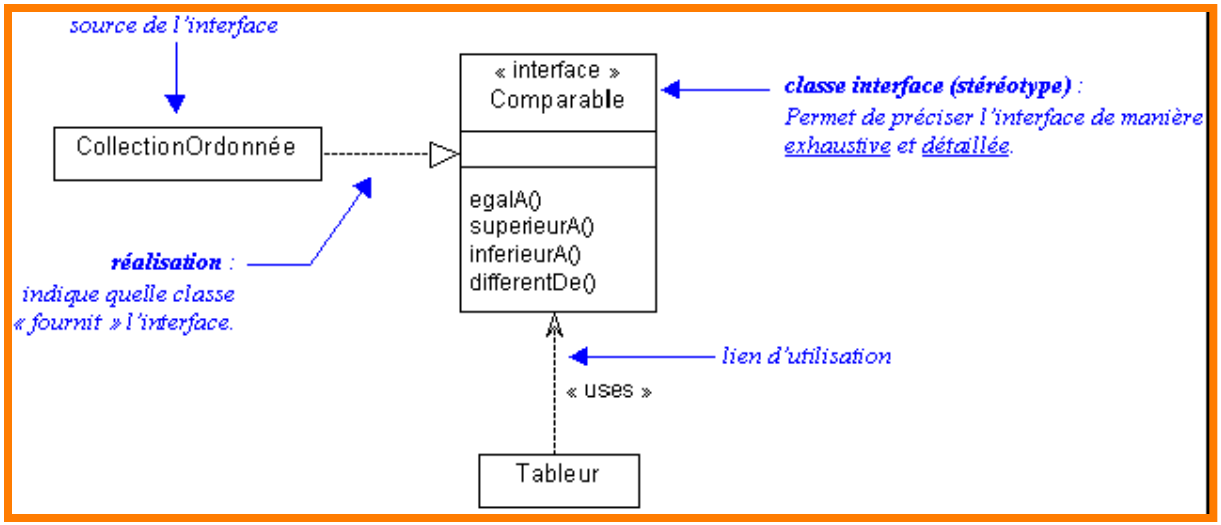
DIAGRAMME DE CLASSES(31)

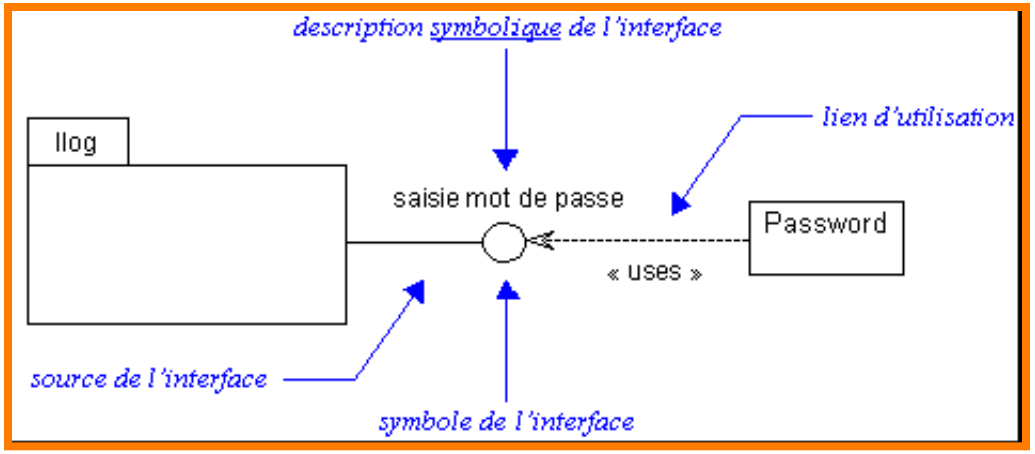


- Interfaces
  - Une interface fournit une vue totale ou partielle d'un ensemble de services offerts par une classe, un paquetage ou un composant. Les éléments qui utilisent l'interface peuvent exploiter tout ou partie de l'interface
  - Dans un modèle UML, le symbole "interface" sert à identifier de manière explicite et symbolique les services offerts par un élément et l'utilisation qui en est faite par les autres éléments.

DIAGRAMME DE CLASSES(32)

- Interfaces





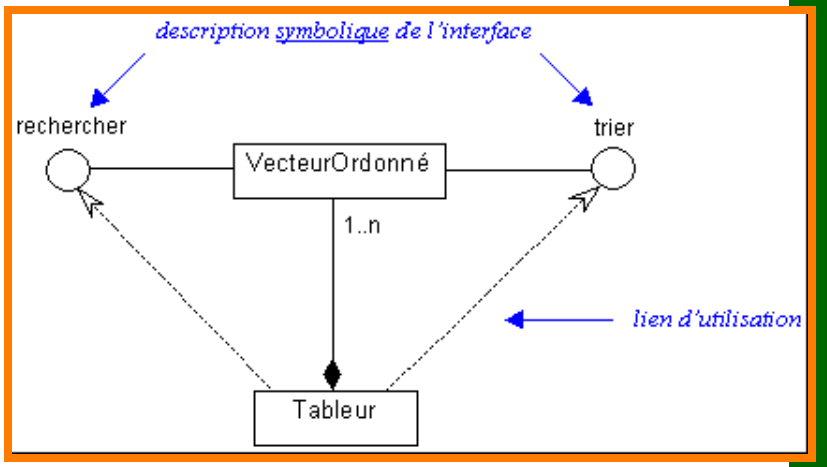


DIAGRAMME D'OBJETS (1)

- Représentation des instances **(des objets)**
  - Un objet est une instance (avec un état précis) d'une classe

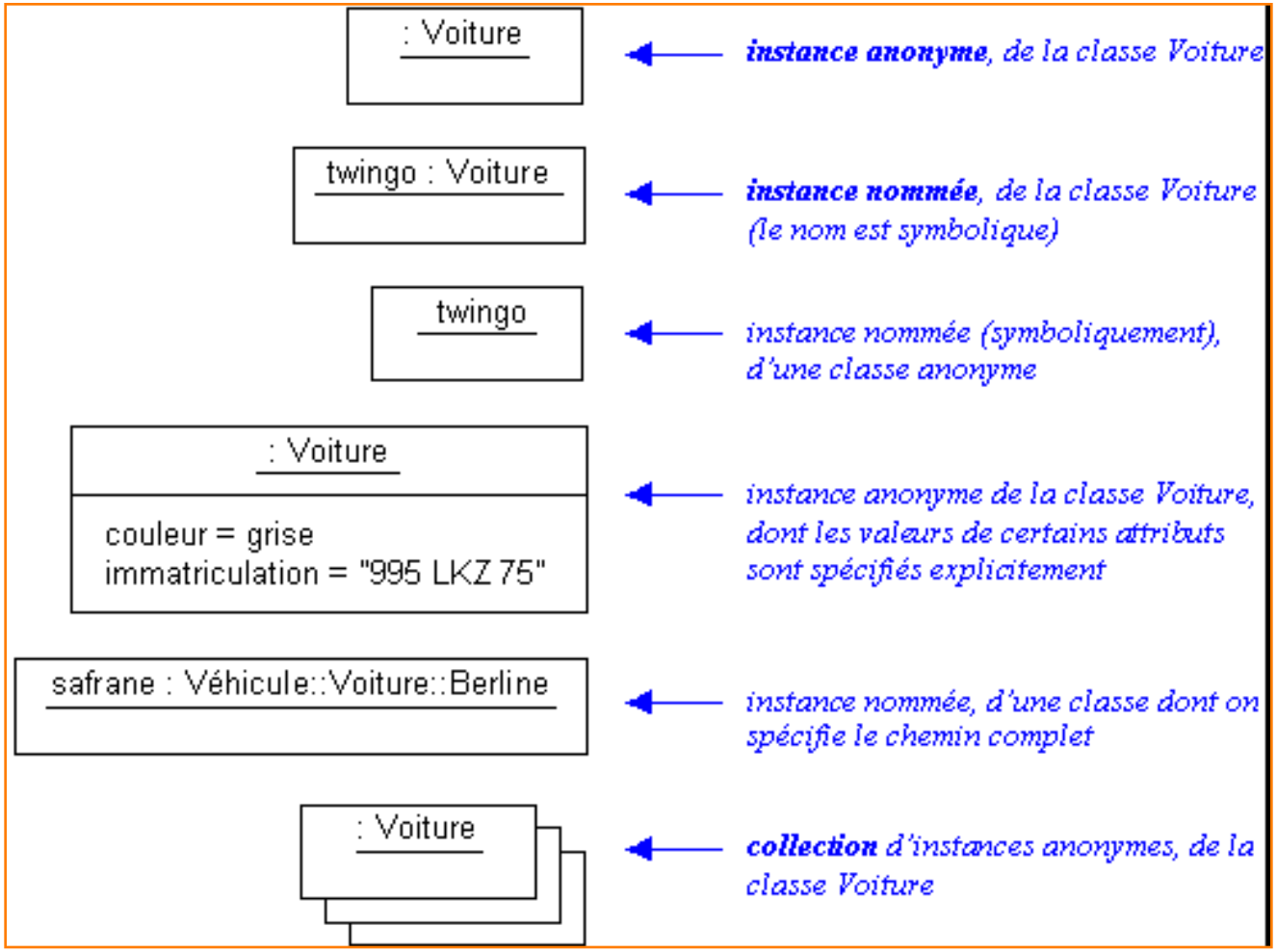
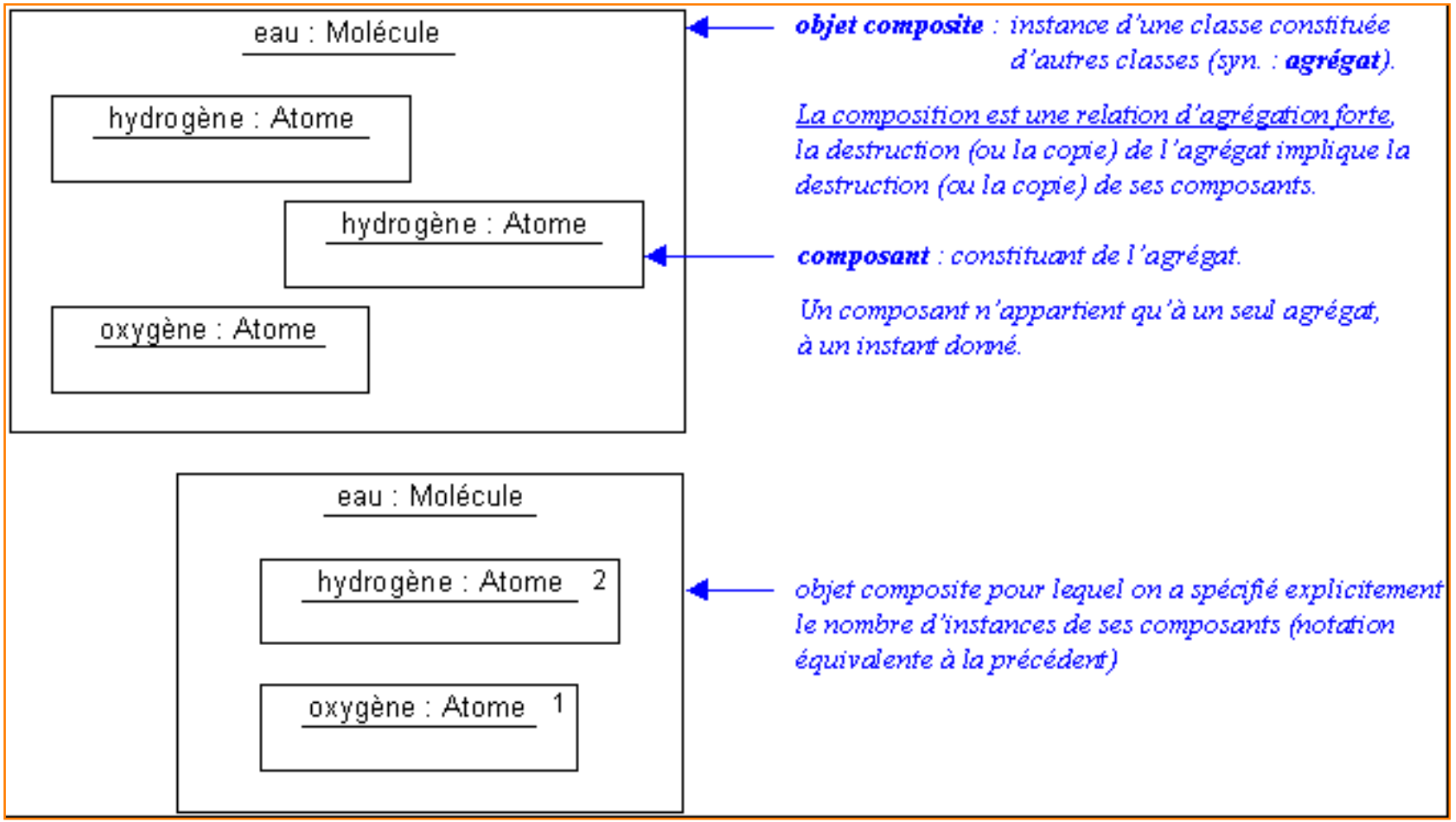


DIAGRAMME D'OBJETS (2)

- Objets composites



objet composite : *instance d'une classe constituée d'autres classes (syn. : **agrégat**).*

La composition est une relation d'agrégation forte, la destruction (ou la copie) de l'agrégat implique la destruction (ou la copie) de ses composants.

composant : *constituant de l'agrégat.*

Un composant n'appartient qu'à un seul agrégat, à un instant donné.

objet composite pour lequel on a spécifié explicitement le nombre d'instances de ses composants (notation équivalente à la précédent)

DIAGRAMME D'OBJETS (3)

- Sémantique
  - Un diagramme d'objets montre des objets et des liens entre ces objets
  - Les diagrammes d'objets s'utilisent pour montrer un contexte (avant ou après une interaction entre objets par exemple)