

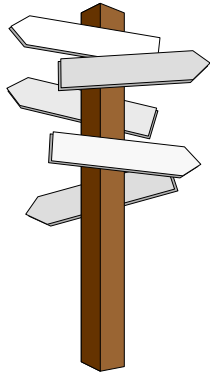
La genèse d'UML

Les diagrammes de classes

Les diagrammes d'objets

Chantal Reynaud

Université Paris X - Nanterre UFR SEGMI - Maîtrise MIAGE



Plan

- I. La genèse d'UML
- II. Les diagrammes de classes
- III. Les diagrammes d'objets

Partie I. La genèse d'UML

- I. Les méthodes d'analyse et de conception
- II. L'unification des méthodes
- III. Spécificités d'UML
- IV. Diagrammes d'UML
- V. Bibliographie

- UML est la forme contractée de « **Unified Modeling Language** ».
- UML est un langage graphique conçu pour *représenter, spécifier, construire et documenter* les artefacts d'un système à dominante logicielle.
- UML est *un langage standard de modélisation objet* qui peut se substituer aux notations d'autres méthodes objets. *Ce n'est pas une méthode.*

I. Les méthodes d'analyse et de conception

I.1. Le besoin de méthodes

- Une méthode permet de **surmonter la complexité**.
- Une méthode définit une démarche reproductible qui fournit des résultats **fiables** :
 - une représentation graphique qui permet de manipuler aisément **des modèles**, faciliter la communication et l'échange d'informations
 - **des règles de mise en œuvre** qui décrit l'articulation des différents points de vue, l'enchaînement des actions, l'ordonnancement des tâches et la répartition des responsabilités.

I. Les méthodes d'analyse et de conception

I.1. Le besoin de méthodes

- Pourquoi des modèles ?

- C'est le pilier de toute activité qui conduit au déploiement de logiciels de qualité.
- Un modèle est **une simplification de la réalité**. Un bon modèle inclut les éléments qui revêtent une grande importance et laisse de côté ceux qui sont inutiles.
- Un modèle **permet de mieux comprendre le système** que l'on développe.
- La modélisation permet d'atteindre **4 objectifs** : (1) aider à *visualiser* un système tel qu'il est ou tel qu'on voudrait qu'il soit, (2) *préciser la structure ou le comportement* d'un système, (3) fournir un canevas qui *guide la construction* d'un système, (4) *documenter* les décisions prises.

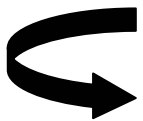
- Les 4 principes de la modélisation

- Le choix des modèles à créer a une très forte influence sur la manière d'aborder un problème et sur la nature de la solution.
- Les modèles peuvent avoir des niveaux de précision très différents.
- Les meilleurs modèles ne perdent pas le sens de la réalité.
- Parce qu'aucun modèle n'est suffisant à lui seul, il est préférable de le décomposer en un ensemble de petits modèles presque indépendants.

I. Les méthodes d'analyse et de conception

I.2. Les méthodes orientées objet

- Entre le milieu des années 70 et la fin 90 : langages de programmation orientés objet
- Entre 89 et 94 : le nombre de méthodes orientées objet est passé de 10 à plus de 50.



Certaines méthodes se sont dégagées du lot : Booch, OOSE, OMT, Fusion, Shlaer et Mellor, Coad et Yourdon, ...

Chacune de ces méthodes constituait une méthode complète tout en présentant des avantages et des inconvénients

- Au milieu des années 90, G. Booch, I. Jacobson et J. Rumbaugh ont chacun commencé à adopter les idées des autres. Les 3 auteurs ont souhaité créer un langage de modélisation unifié.

II. L'unification des méthodes

- Pourquoi ?

- 1) Élimination des différences inutiles et arbitraires entre les 3 méthodes
- 2) Stabilité du marche orienté objet
- 3) Amélioration attendue

- Objectifs

- Modéliser des systèmes au moyen de techniques orientées objet depuis leur conception jusqu'à leur implémentation
- Résoudre des problèmes d'échelle inhérents aux systèmes complexes
- Créer un langage utilisable à la fois par les humains et les machines.

- Mise en oeuvre

- Initiateurs : Booch, Jacobson et Rumbaugh
- Résultat final = fruit de tous les partenaires impliqués, le développement d'UML a été un processus ouvert.

II. L'unification des méthodes

Principales étapes de la définition d'UML (MULLER, 01)

Définition en cours par une commission de révision

Soumission à l'OMG

Standardisation par l'OMG
 Soumission à l'OMG

Soumission à l'OMG

Version bêta OOPSLA '96

OOPSLA '95

UML 2.0

UML 1.3

UML 1.2

UML 1.1

UML 1.0

UML 0.9

Méthode unifiée 0.8

Booch '93

OMT-2

Booch '91

OMT-1

OOSE

Partenaires

Jun 1999

Jun 1998

Novembre 1997
 Septembre 1997

Janvier 1997

Jun 1996

Octobre 1995

Autres méthodes

III. Spécificités d'UML

- Synthèse des points de vue *statique, dynamique, fonctionnel*
- Réconcilie *l'analyse et la conception* en proposant une démarche par raffinements successifs

Analyse : consiste à comprendre complètement le problème à modéliser → on présente le problème et on prépare la couche interne de l'application

Conception : on cherche une solution au problème, tel que dégagé par l'analyse, en prenant en compte les contraintes liées aux logiciels à utiliser → on prépare les couches plus techniques de l'application

IV. Diagrammes d'UML

- UML définit 9 types de diagrammes

- Vues statiques

Diagrammes de classes : représentent la structure statique en termes de classes et de relations

Diagrammes d'objets : représentent les objets et leurs relations

Diagrammes de cas d'utilisation : représentent les fonctions du système du point de vue de l'utilisateur

Diagrammes de composants : représentent les composants physiques d'une application

Diagrammes de déploiement : représentent le déploiement des composants sur des dispositifs matériels

- Vues dynamiques

Diagrammes de séquence : sont une représentation temporelle des interactions entre objets

Diagrammes de collaboration : sont une représentation spatiale des objets et de leurs interactions

Diagrammes d'états-transitions : représentent le comportement d'une classe en termes d'états

Diagrammes d'activité : représentent le comportement d'une opération en termes d'actions

V. Bibliographie

- Modéliser objet avec UML, P.-A. Muller, N. Gaertner, Eyrolles, 2001*
- Intégrer UML dans vos projets, N. Lopez, J. Migueis, E. Pichon, Eyrolles 97
- De Merise à UML, N. Kettany, D.Mignet, P. Paré, C. Rosenthal-Sabroux, Eyrolles 2001 *
- UML pour l'analyse d'un système d'information, C. Morley, J. Hugues, B. Leblanc, Dunod 2000
- The Unified Modeling Language Reference Manual, J. Rumbaugh, I. Jacobson, G. Booch, Addison Wesley, Fall 98
- Le processus unifié de développement logiciel, I. Jacobson, G. Booch, J. Rumbaugh, Eyrolles 2000 *
- Le guide de l'utilisateur UML, G. Booch, J. Rumbaugh, I. Jacobson, Eyrolles 2001 *
- UML par la pratique, P. Roques, Eyrolles 2001 *
- UML en action, P. Roques, F. Valée, Eyrolles 2001 *

Partie II. Les diagrammes de classes

- I. Les classes
- II. Les relations
- III. Caractéristiques avancées

I. Les classes

- Une classe est la description d'un ensemble d'objets qui partagent les mêmes attributs, les mêmes opérations, les mêmes relations et la même sémantique.
- Tous les objets sont modélisés sous forme de classes.
- Représentation graphique :

NOM_DE_CLASSE
Attribut : type = valeur initiale
operation ()

I. Les classes

1. Chaque classe doit avoir un nom différent



CONVENTION : noms en majuscules

2. Un attribut est une propriété nommée d'une classe qui décrit un ensemble de valeurs que les instances de cette propriété peuvent prendre.



CONVENTION : la 1ère lettre de chaque mot est une majuscule

MUR
- Hauteur : real - Largeur : real - Epaisseur : real - EstPorteur : bool = faux

I. Les classes

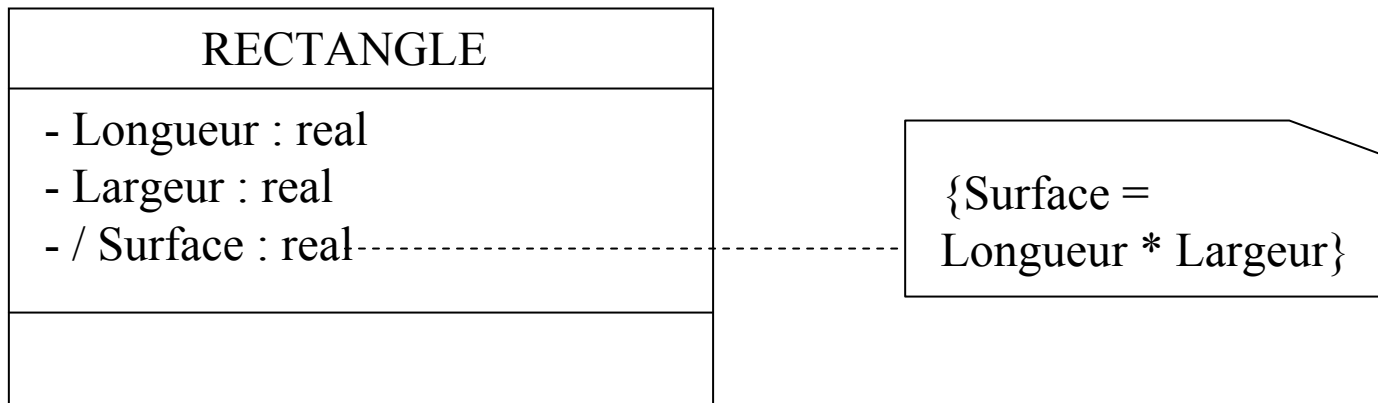
Le type des attributs peut être une classe (rectangle, cercle, ...), un type primitif (entier, chaîne), une expression complexe.

La visibilité et le type existent toujours mais peuvent ne pas être représentés : + public (par défaut), - privé, # protégé.



CONVENTION : en règle générale, les attributs sont privés

On peut également représenter des attributs dérivés à partir d'autres propriétés déjà définies.



I. Les classes

3. Une opération est l'implémentation d'un service qui peut être demandé à tous les objets d'une même classe dans le but de déclencher un comportement. Une classe peut comporter plusieurs opérations ou aucune.

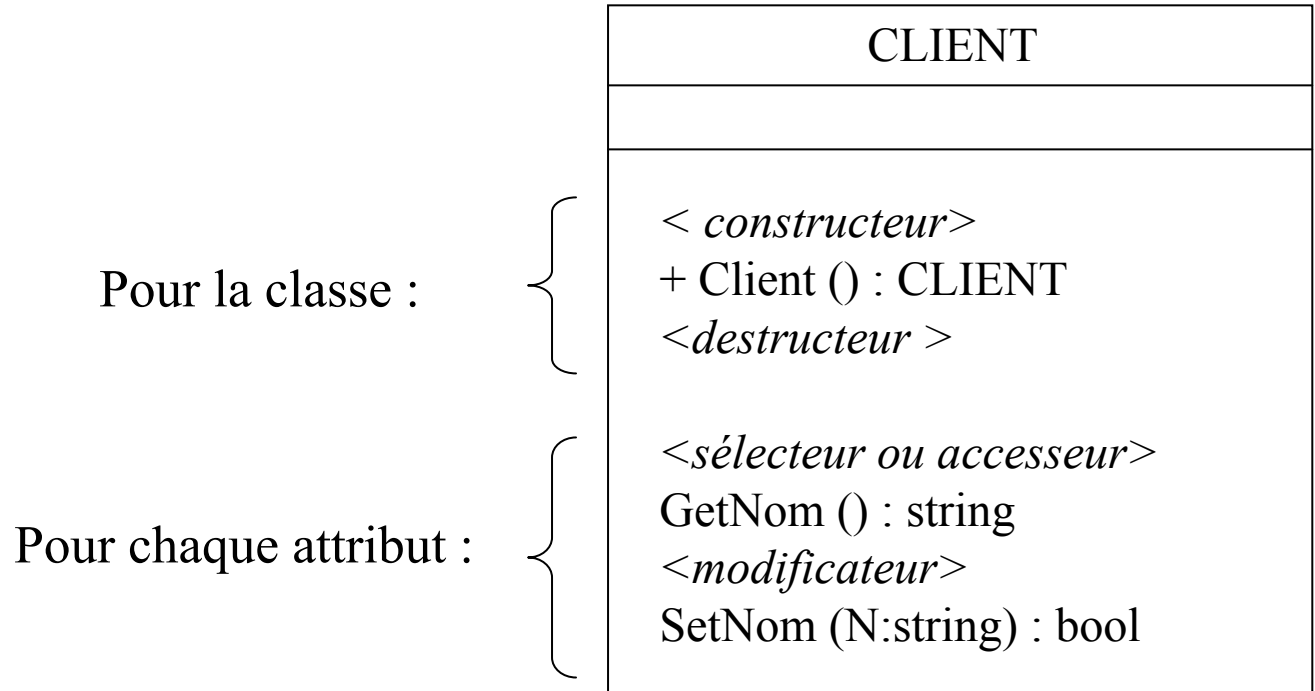
RECTANGLE
+ Ajouter () + Agrandir () + Deplacer ()

DETECTEUR_CHALEUR
+ Reinitialiser () + PositionnerAlarme (t:Temperature) + ObtenirValeur () : Temperature

CONVENTIONS :

- (1) Le nom des opérations est un verbe court ou une phrase verbale représentant un comportement. La 1ère lettre de chaque mot est une majuscule.
- (2) Les opérations sont publiques.
- (3) Les opérations qui permettent d'accéder aux attributs sont de la forme GetAttribut() : GetNom(). Les opérations qui modifient un attribut sont de la forme SetAttribut() : SetAdresse().

4. Opérations implicites



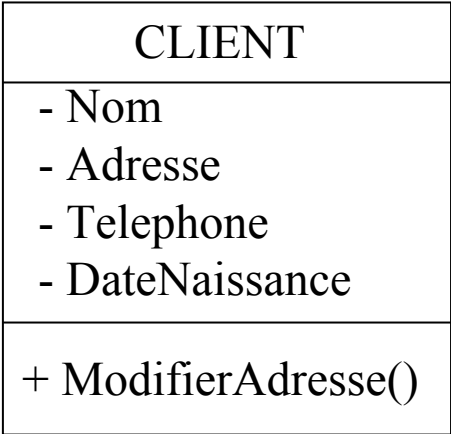
I. Les classes

5. Il est possible de faire des choix dans ce qui est représenté au niveau d'une classe. Un compartiment vide ne signifie donc pas forcément qu'il n'y a pas d'attributs ou d'opérations mais simplement que l'on a décidé de ne pas les montrer. Les choix de représentation seront également différents selon que l'on est en phase d'analyse ou de conception.

Niveau *sans détail*



Niveau *détail analyse*



Au niveau de *détail d'implémentation*, on précisera :

- le type des variables,
- les valeurs par défaut
- les signatures des opérations
- év. le niveau de visibilité

I. Les classes

6. Remarques

- On peut avoir une classe avec une seule instance
- Un attribut peut être multivalué
- Les identificateurs explicites (identifiants) ne sont pas indispensables. On peut les préciser à l'aide d'une note.

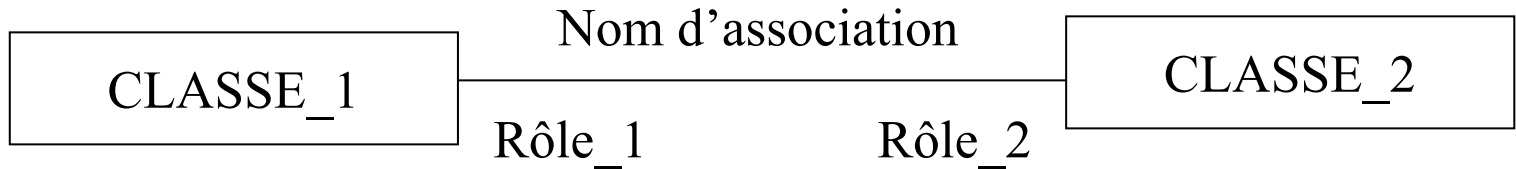
ENTREPRISE
- Nom
- Adresse
- NumérosTelephone
+ ModifierAdresse()
+ AjouterPersonne()

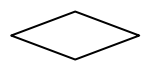
{identifiant}



II. Les relations

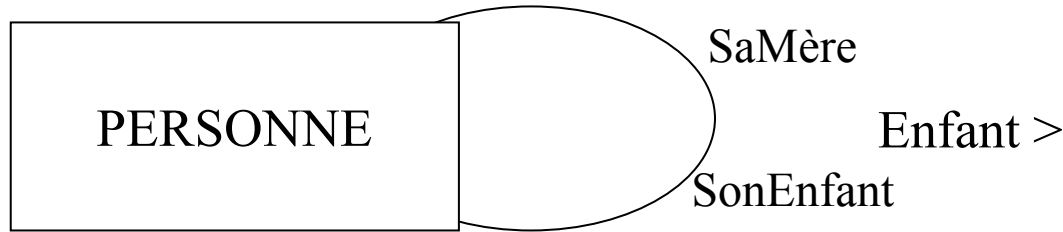
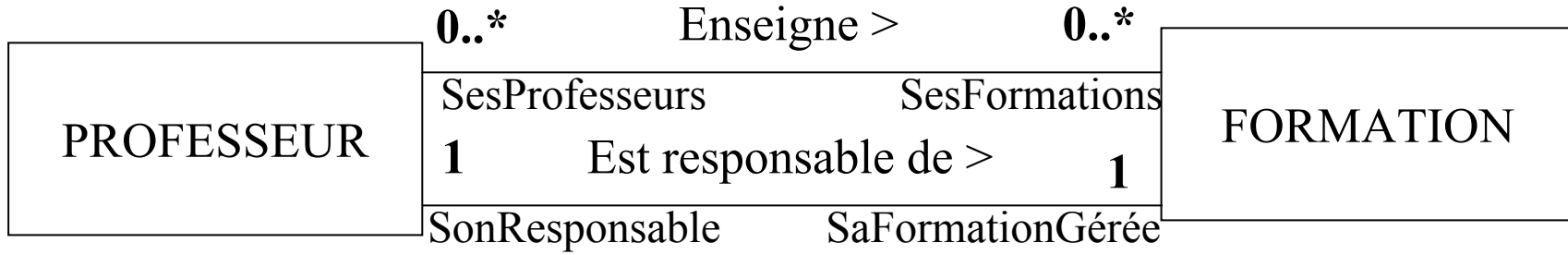
II.1. Associations entre classes



- Le nom de l'association n'est pas indispensable lorsque les noms de rôles sont explicites.
- La plupart des associations sont binaires. Des arités supérieures peuvent néanmoins exister. Elles se représentent avec le symbole : 

II. Les relations

II.1. Associations entre classes

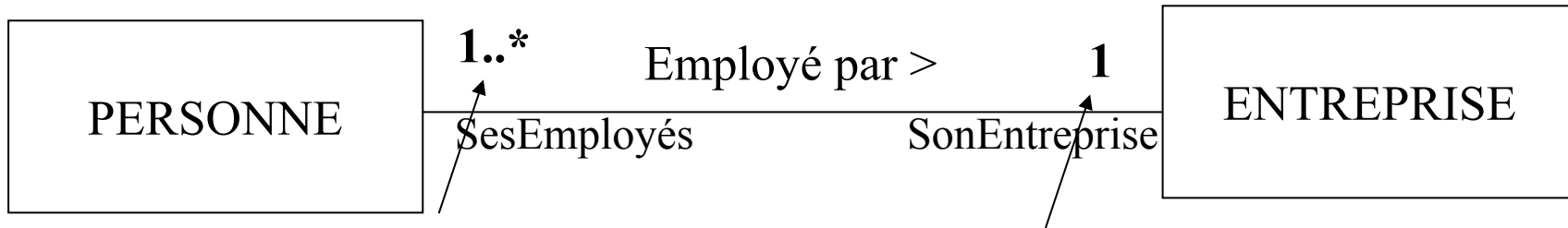


CONVENTION :

- (1) Le nom d'une association commence toujours par une majuscule.
- (2) On précisera toujours les noms des rôles, le nom de l'association est facultatif.

II. Les relations

II.1. Associations entre classes - cardinalités



Nombre d'employés
d'une entreprise

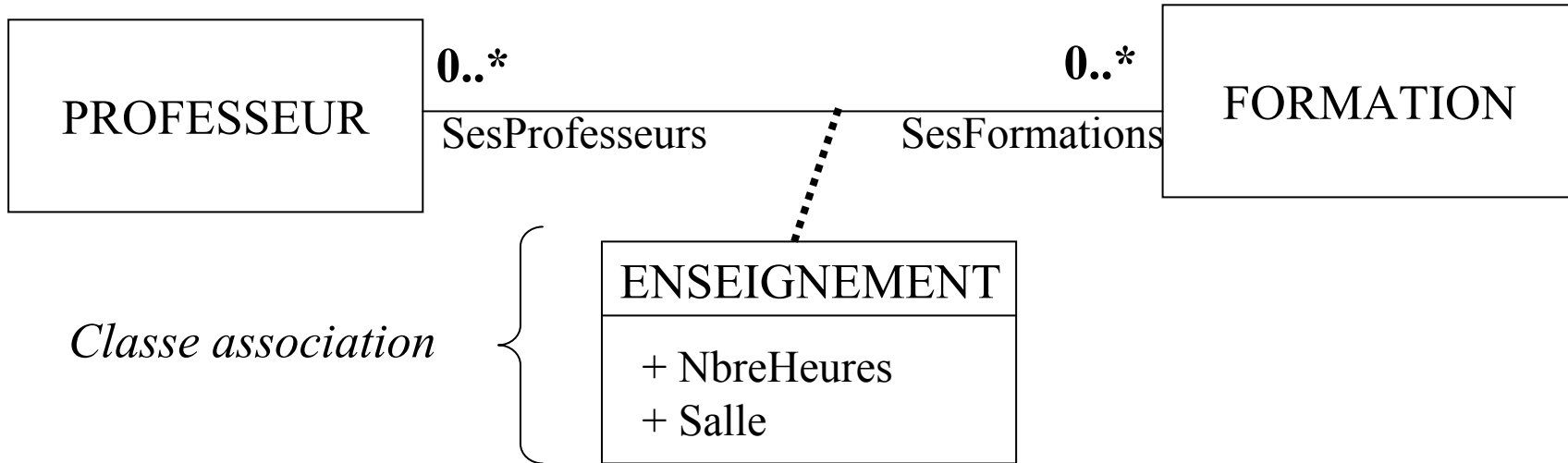
Nombre d'entreprises dans lesquelles
une personne est employée

Une entreprise emploie plusieurs personnes (sous-entendu, elle peut aussi, à un moment donné, ne pas avoir d'employés). Une personne n'est employée que par une seule entreprise.

Exactement 1	1
Exactement n	n
Plusieurs (0 ou plus)	0..*
Au plus 1	0..1
1 ou plus	1..*
Cardinalité spécifiée	1..2 4

II. Les relations

II.1. Associations entre classes - attributs



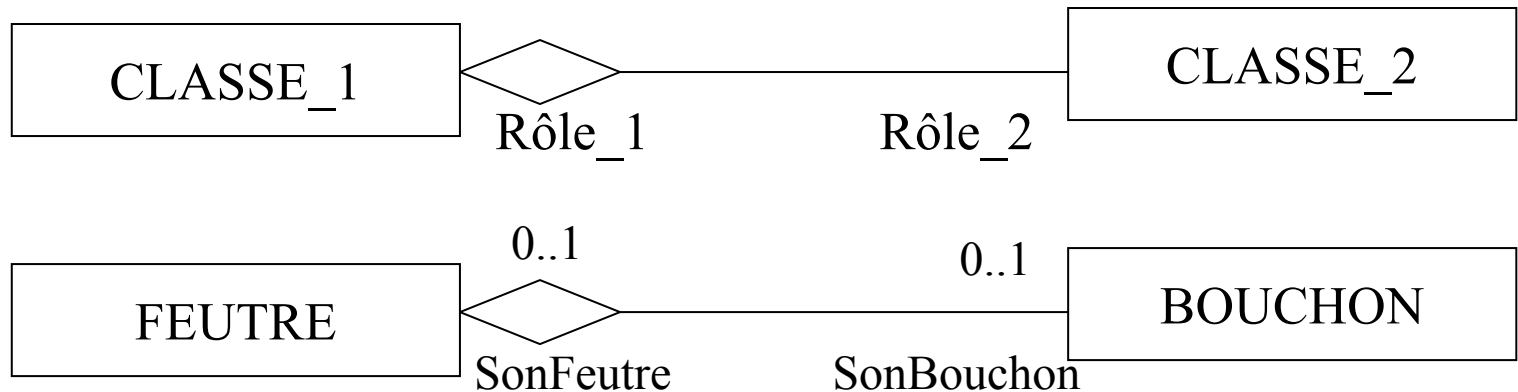
CONVENTION

- Eviter les classes d'associations difficiles à gérer.
- Les noms de rôles avec cardinalité $0..*$ ou $1..*$ commencent par Ses.
- Les noms des rôles avec cardinalité $0..1$ ou 1 commencent par Son ou Sa.

II. Les relations

II.2. Relations d'agrégation « simples »

Association composé-composant ou partie-de ne liant pas la durée de vie du tout et de ses parties



Un feutre peut perdre son bouchon et un bouchon peut perdre le corps de son feutre d'origine. Il n'y a pas forcément cohérence entre les cycles de vie des feutres et des bouchons.

II. Les relations

II.3. Relations de composition

Cas particulier d'agrégation avec un couplage plus important



Le composé n'existe qu'en tant que partie du composite. La destruction du composite entraîne la destruction des composés. Un objet ne fait partie que d'un seul composite à la fois.

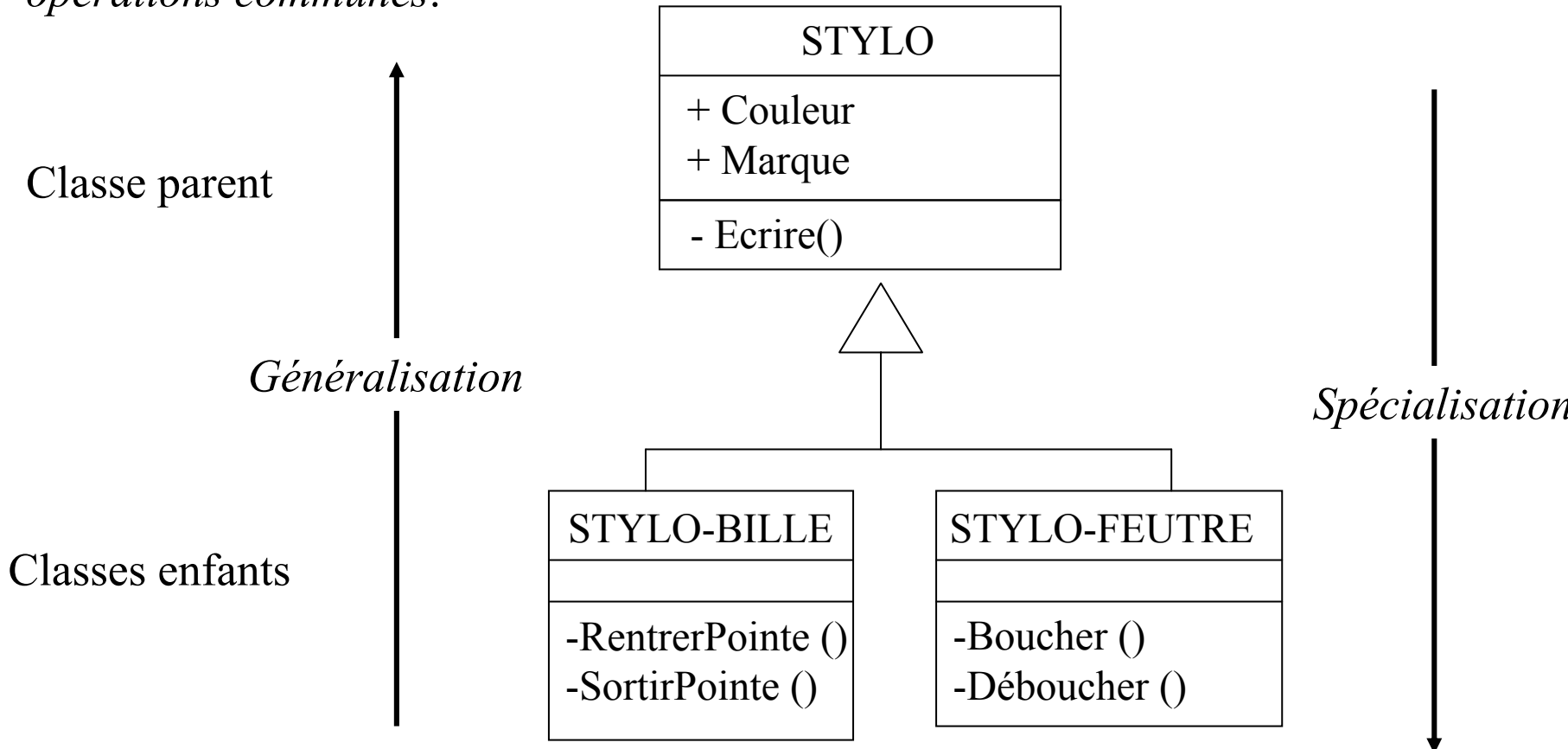


Un corps n'existe qu'en tant que partie d'un crayon. La destruction d'un crayon entraîne la destruction de son corps.

II. Les relations

II.4. Relations de généralisation/spécialisation

La classe enfant hérite de la structure et du comportement de la classe parent. Les relations de G/S permettent ainsi de regrouper des attributs communs et des opérations communes.

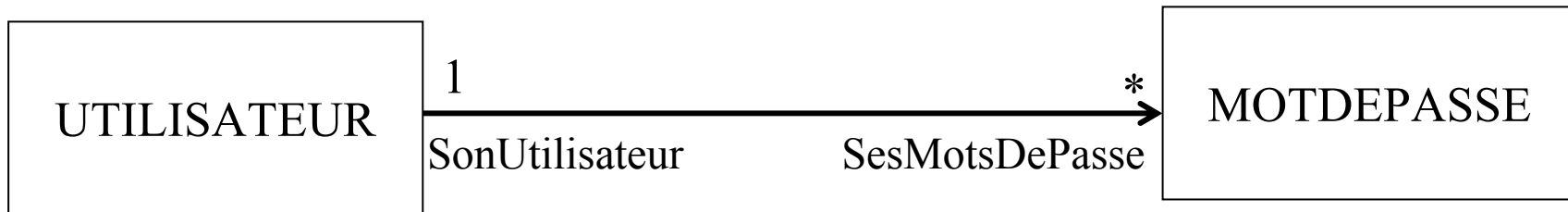


III. Caractéristiques avancées

III.1. Navigabilité d'une association

Qualité d'une association qui permet le passage d'un objet d'une sorte à un objet d'une autre sorte. Sauf indication contraire, la navigabilité est bi-directionnelle.

- Il est parfois nécessaire de limiter la navigabilité à un seul sens.



Pour chaque utilisateur, il faut trouver les mots de passe correspondants, mais aucun des mots de passe ne doit permettre d'identifier un utilisateur

- Indiquer un sens de passage ne signifie pas nécessairement qu'il doit être impossible de naviguer dans l'autre sens. Cela peut aussi signifier qu'il doit être possible de passer *facilement* et *directement* aux objets de l'autre extrémité (généralement parce que l'objet source comporte des références aux objets de la cible).

III. Caractéristiques avancées

III.2. Attributs et opérations de classes

Un *attribut de classe* décrit une valeur commune à une classe d'objets dans son ensemble.

Une *opération de classe* est une opération sur la classe elle-même. La plus commune est celle qui crée des nouvelles instances de classe.

Attributs et opérations de classe sont soulignés.

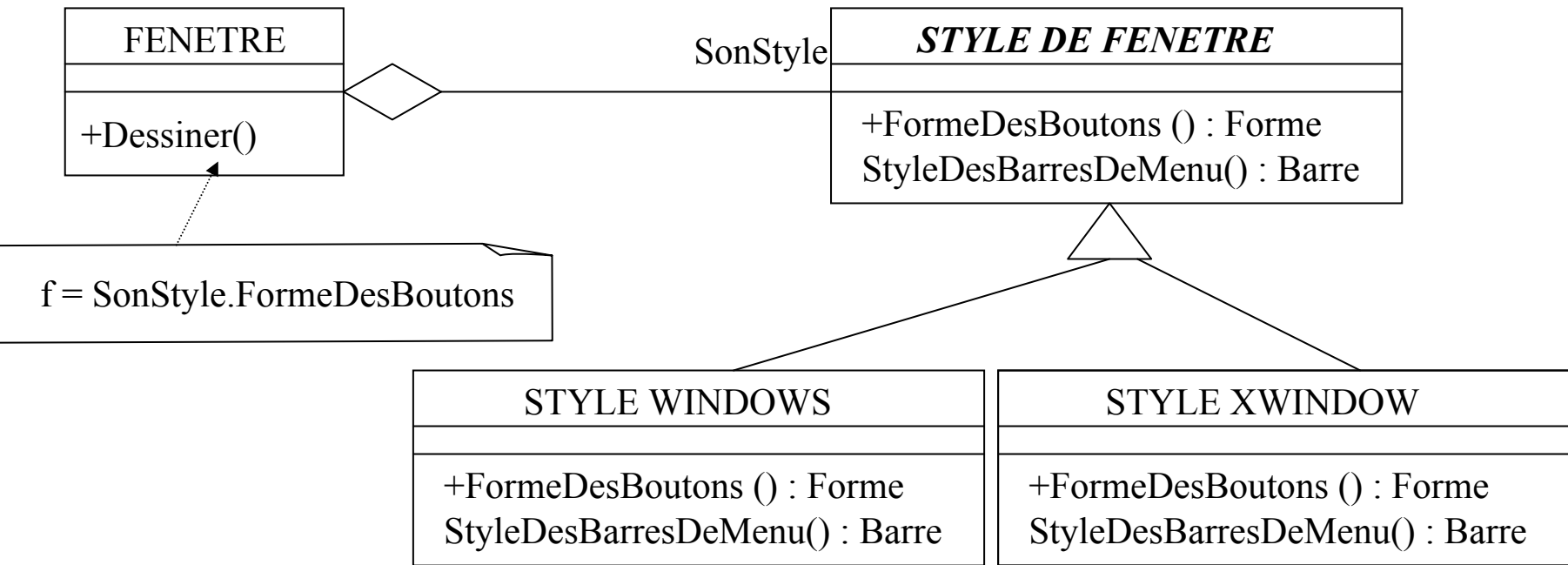
CLIENT
- Nom - Adresse - <u>NbClients</u>
<u>+ Client () : CLIENT</u> <u>+ CompterClients ()</u>

III. Caractéristiques avancées

III.3. Classes et opérations abstraites / Polymorphisme

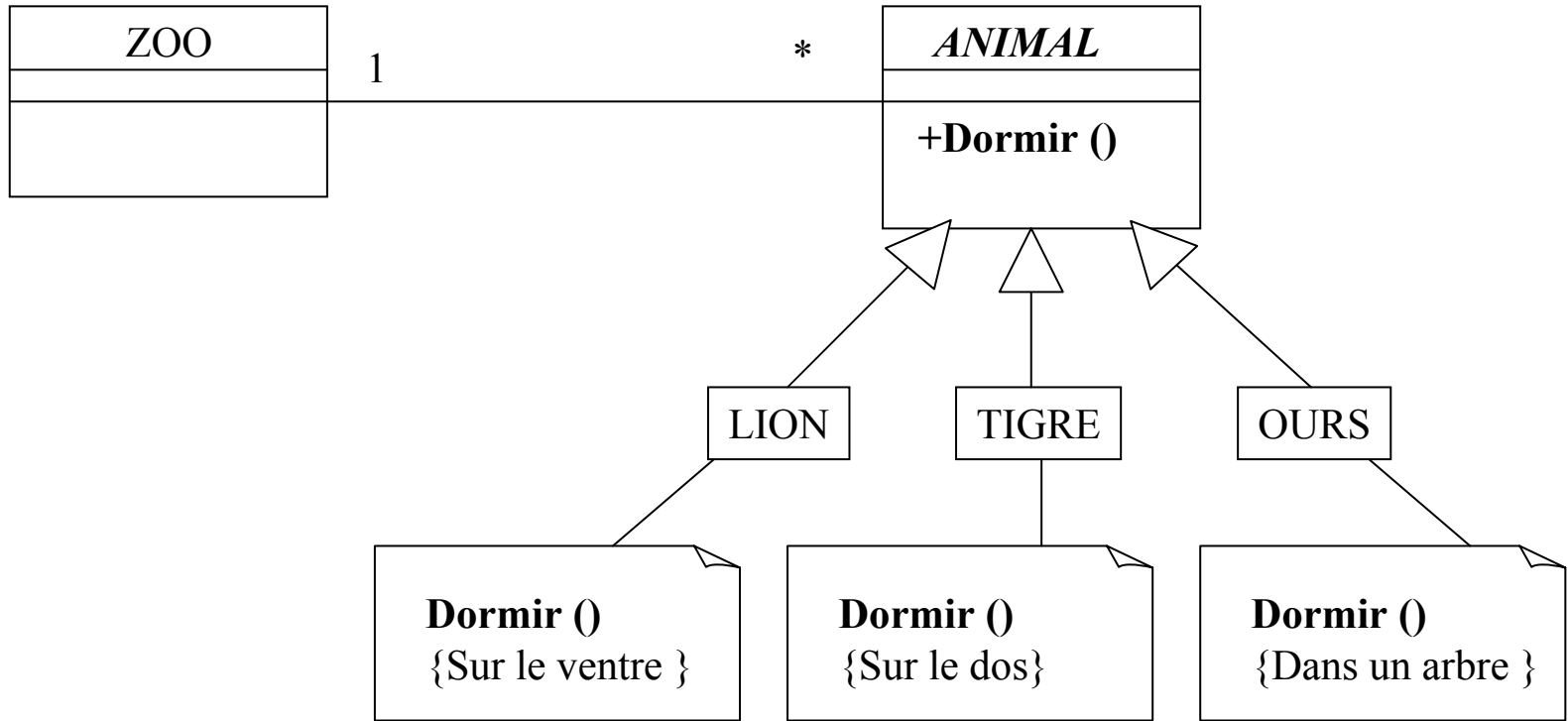
Une *classe abstraite* est une classe qui ne peut avoir aucune instance directe. On écrit son nom en italique (notation manuelle : classe préfixée par Abs).

Une *opération abstraite* est une opération incomplète qui a besoin de sa classe fille pour fournir une implémentation.



III. Caractéristiques avancées

III.3. Classes et opérations abstraites / Polymorphisme



Chaque sous-classe hérite de la spécification des opérations de ses super-classes mais a la possibilité de modifier localement le comportement de ces opérations afin de mieux prendre en compte les particularismes liés à un niveau d'abstraction donné.

Partie III. Les diagrammes d'objets

- I. Représentation
- II. Règles gouvernant la transition entre les deux types de diagrammes
- III. Exemples

I. Représentation

- Instance et objet sont largement synonymes.
- On représente une instance en soulignant son nom.
- Chaque instance doit avoir un nom différent des autres instances dans son contexte.

- Instances nommées

NomInstance:NOMCLASSE

Dupont:CLIENT

- Instances anonymes

:NOMCLASSE

- Instances orphelines

NomInstance:

- Instances avec valeurs d'attributs

Bouton2 : RECTANGLE

Nom : string = « bouton-poussoir »

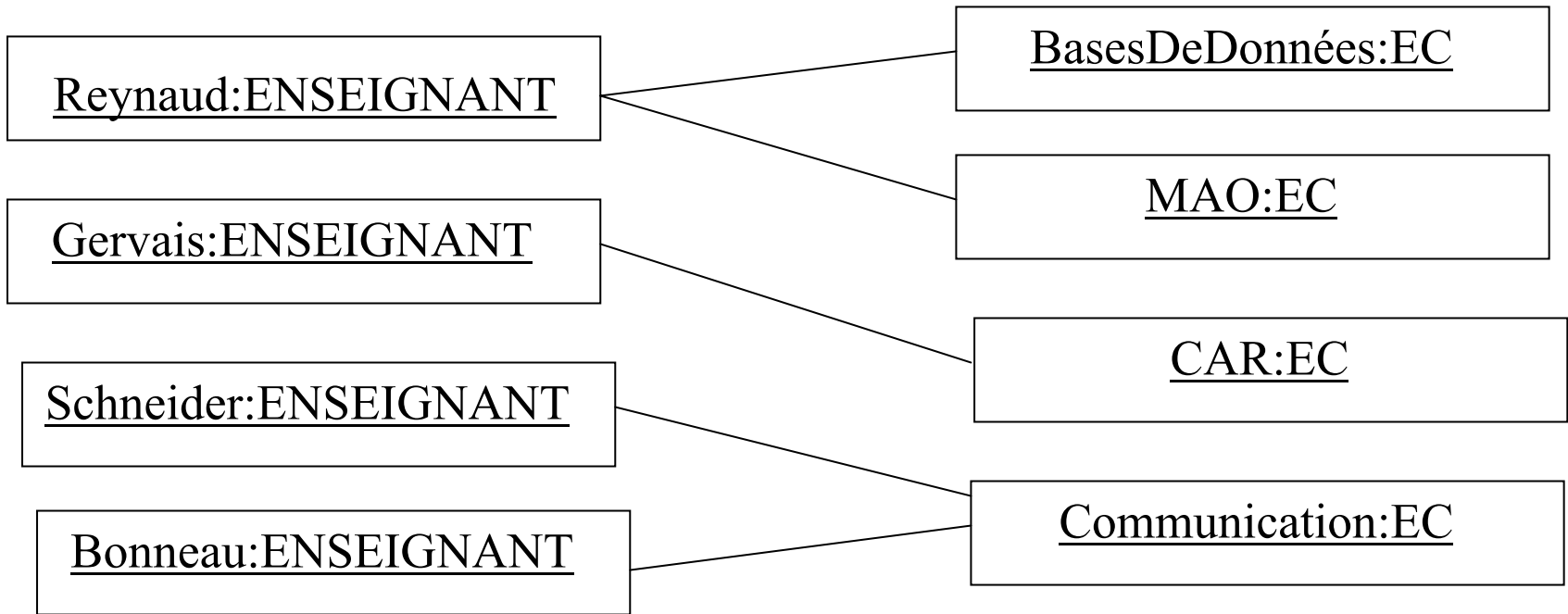
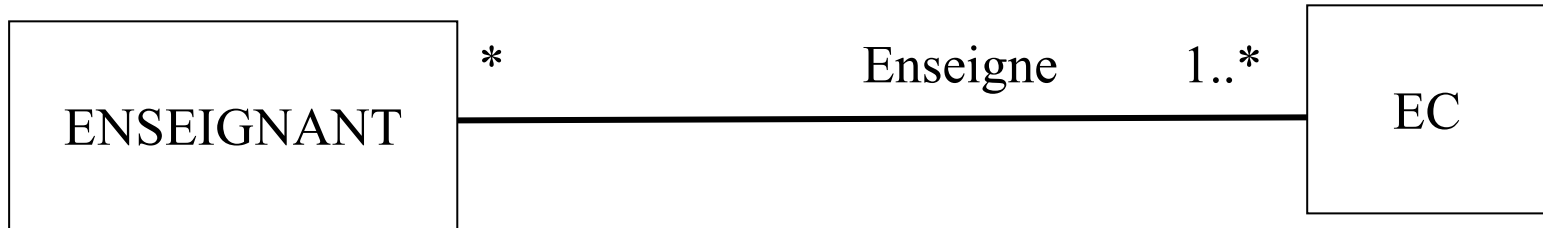
Longueur : float = 13.5

Largeur : float = 3.2

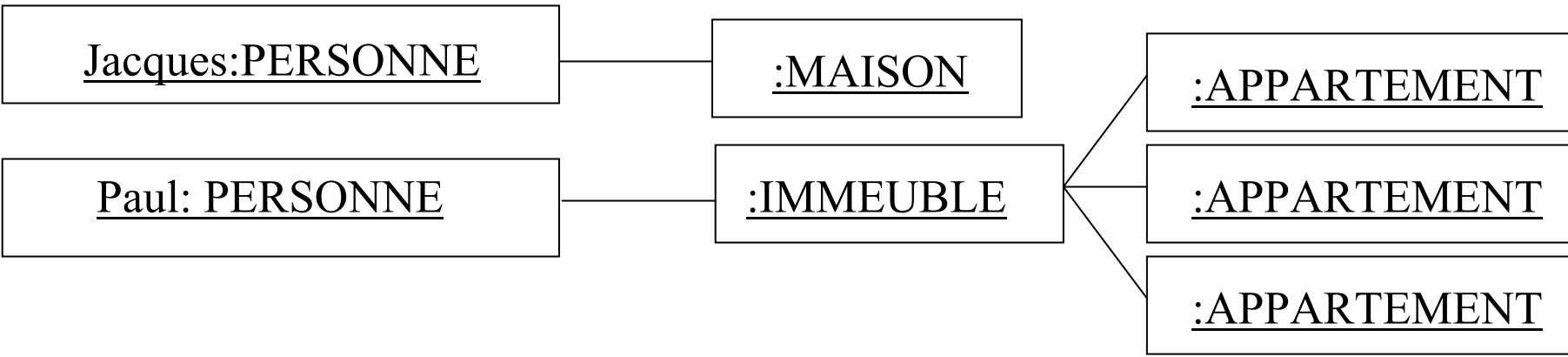
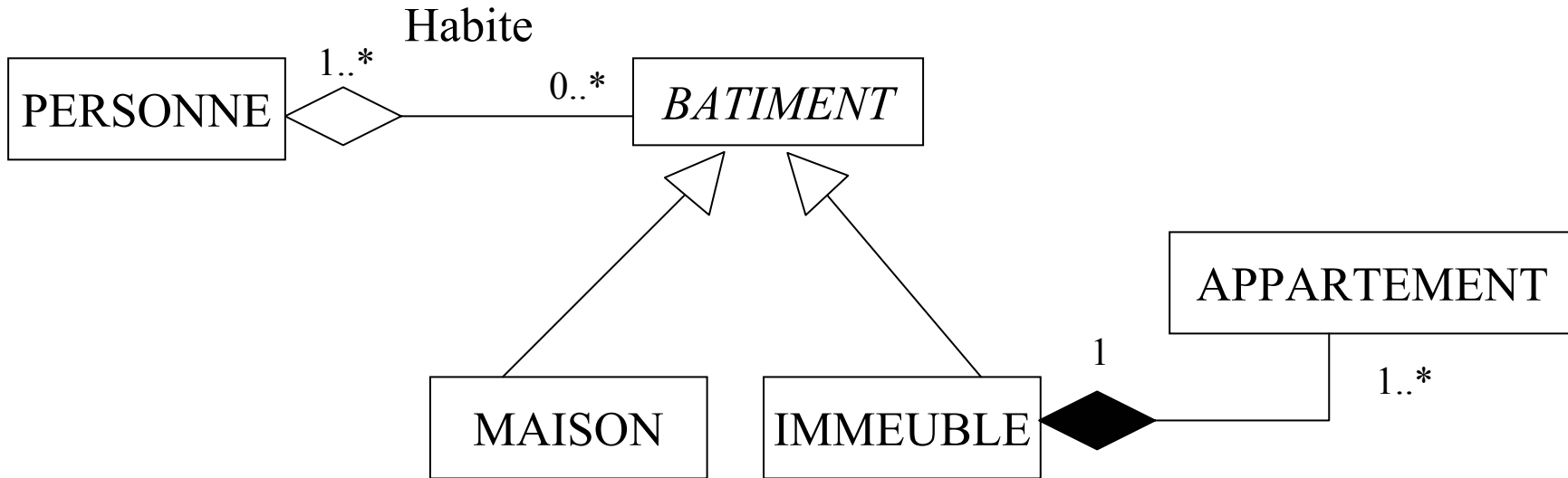
II. Règles gouvernant la transition entre les deux types de diagrammes

- Chaque objet est instance d'une classe et la classe de l'objet ne change pas durant la vie de l'objet.
- Les classes abstraites ne peuvent pas être instanciées.
- Les liens relient les objets et les relations relient les classes.
- Chaque lien est instance d'une relation (association, agrégation, composition)
- Un lien entre deux objets implique une relation entre les classes (ou super-classes) des 2 objets.
- Un lien entre 2 objets indiquent qu'ils se connaissent et qu'ils peuvent s'échanger des messages.
- Les diagrammes d'objets qui contiennent des objets et des liens sont instances des diagrammes de classes qui contiennent des classes et des relations.

III. Exemples



III. Exemples



Conclusion

- Les diagrammes de classes et d'objets doivent être cohérents les uns par rapport aux autres.
- Ils se construisent en parallèle. Le processus de modélisation objet n'est pas linéaire. Il n'est pas souhaitable de construire un type de diagramme et, ensuite, d'en dériver l'autre.
- Pas de méthode portant sur ce processus de construction. Dans certains cas, la structure de classes est évidente. Dans d'autres cas, les objets sont plus faciles à identifier que les classes.