

Méthodes pour l'informatisation - compléments

L'analyse orientée objet

Christine Plumejeaud

Doctorante Informatique UJF

CNAM - centre de Grenoble

Plan

1. L'approche orientée objet
2. Genèse d'UML
3. Principes objets

L'approche objet

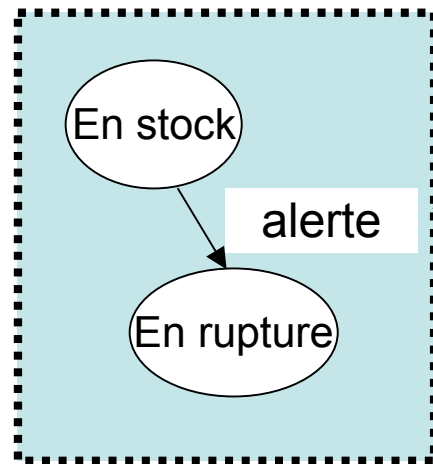
Les « objets » sont une *abstraction* (et une simplification) du monde réel. Avec on peut :

Modéliser le système

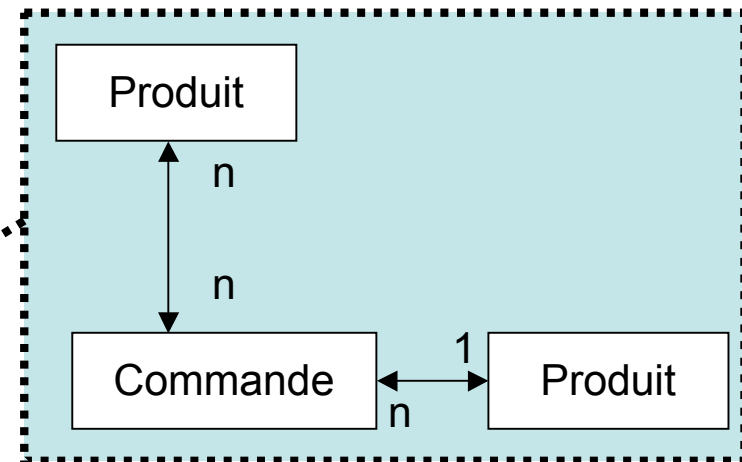
- Visualiser le système
- Spécifier son comportement et sa structure
- Définir un guide de construction
- Expliquer et documenter les décisions d'architecture

Les 3 aspects/vues du système

Vue comportementale /
dynamique

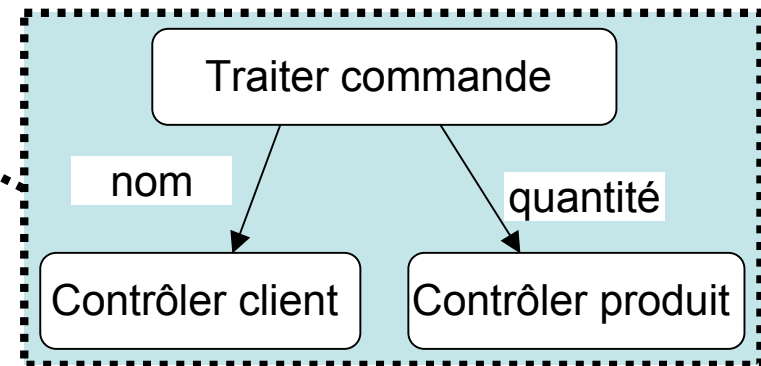


Vue structurelle /
statique



Systeme

Vue fonctionnelle



Intérêts de l'approche objet

1. « *Naturelle* » car basée sur le domaine d'application. Facilite la communication avec les utilisateurs.
2. Meilleur support pour *l'évolution* des besoins
 - Modélisation plus stable
 - Les évolutions du système ne remettent pas l'architecture en cause
3. Facilite la *réutilisation* de composants

Encapsulation	→ systèmes modulaires
Héritage	→ système extensibles
Association	→ systèmes complexes

La genèse d'UML

Unified Modeling Language

1970 : méthodes cartésiennes (HIPO, SADT, SA, AXIAL)

- décomposition fonctionnelle et hiérarchique du système
- Programmation structurée et modulaire

1980 : approche systémique (Merise, Remora)

- Modélisation des données, des actions et des événements
- Approche conceptuelle globale du SI

La genèse d'UML

Unified Modeling Language

1990 : émergence des méthodes objets

- Encapsulation des données et des traitements : synthèse des méthodes systémiques et cartésiennes
- Capacité à modéliser des objets complexes
- Plus de 50 méthodes objet sont apparues en 5 ans (Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE, ...)
- Confusion autour de l'analyse et la conception

La genèse d'UML

Auteurs	Concepts introduits
Booch	Catégories et sous-systèmes
Embley	Classes singleton et objets composites
Fusion	Description des opérations, numération des messages
Gamma	Frameworks, patterns et notes
Harel	Automates (<i>statecharts</i>)
Jacobson	Cas d'utilisation (<i>use cases</i>)
Meyer	Pré et post conditions
Odell	Classification dynamique, éclairage sur les évènements
OMT	associations
Schlear-Mellor	Cycle de vie des objets
Wirfs-Brook	Responsabilités

La genèse d'UML

L'expérience a permis de faire le tri parmi les méthodes existantes avec la constatation que les différences entre les méthodes s'amenuisent

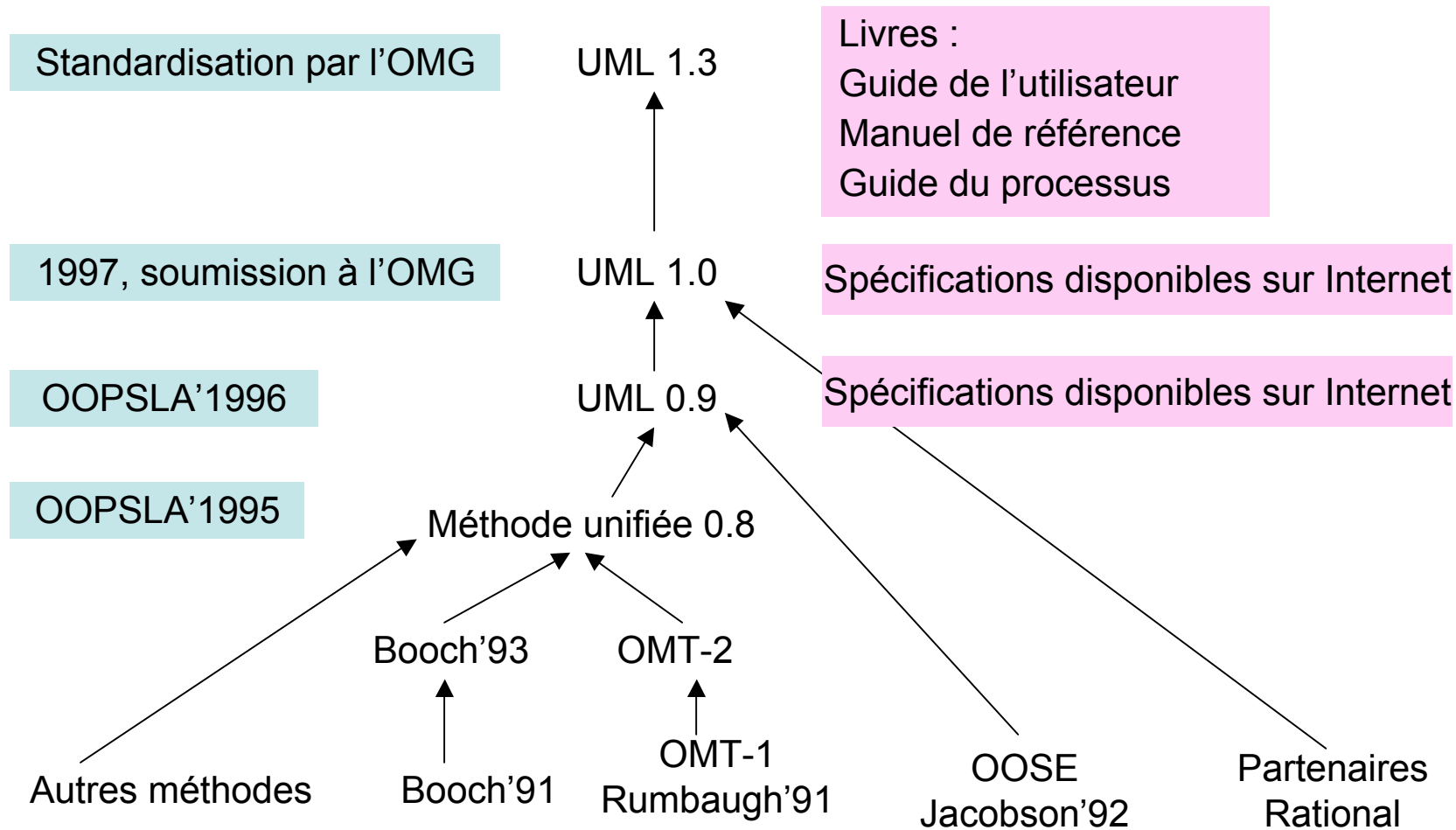
1994 : Rumbaugh et Booch décident d'unifier leurs travaux au sein d'une méthode unique : la méthode unifiée

1995 : première version v0.8

1996 : langage de modélisation unifié

1997 : v1.0 soumis à la standardisation

La genèse d'UML



L'unification des méthodes

OMG (*Object Management Group*)
(<http://www.omg.org/>) fondé en 1989.

Consortium d'industriels pour la
standardisation des méthodes objets, à
l'origine du standard CORBA

Objectifs : maximiser la portabilité et la portabilité
des logiciels par la définition de standards
industriels, pour le développement
d'applications commerciales orientées objet.

570 membres en 1995

Les objectifs de UML

1. UML

- notation graphique de modélisation à objets (9 types de diagrammes)
- UML n'est PAS une méthode de développement, ni un langage de programmation

2. Objectifs :

- Représenter des systèmes entiers par des concepts objets (pas juste les logiciels)
- Lier explicitement les concepts et le code
- Modéliser à différents niveaux de granularité
- Langage utilisable à la fois par les humains et les machines par un formalisme bien établi (→ génération de code)
- Encourager l'utilisation des outils Orientés Objets
- Offrir des mécanismes d'extension et de spécialisation pour pouvoir étendre les concepts de base

Concepts objets

1. L'objet
2. Les classes d'objets
3. L'encapsulation
4. L'héritage
5. Association (par composition, agrégation)
6. Le polymorphisme

L'objet

« Une entité capable de mémoriser un **état** (information) et qui offre des opérations (**comportement**) pour à la fois explorer et modifier cet état. » (*Jacobson, 1992*)

Exemples :

- Objets matériels : table, chaise, voiture
- Objets immatériels : compte en banque, voyage, équation, point, ligne
- Objets conceptuels : groupe de travail, idée politique

Attention à la **subjectivité** des points de vue :

L'objet étant une abstraction faite par un humain d'un phénomène réel observé, 2 personnes peuvent donner 2 modèles différents du même phénomène réel.

L'objet

Entité pertinente du domaine technique

Définition OO : « une structure de données et un ensemble de fonctions associées à cette structure de données »

Caractérisé par :

- Une *identité* (ou un nom)
- Un *état* interne: la valeur de ses attributs
- Un *comportement* : ses méthodes ou opérations

Identité de l'objet

Elle caractérise l'existence propre de l'objet

- Immuable
- Permanente : permet de distinguer tout objet de façon non ambiguë indépendamment de son état
- Concept différent de l'égalité dans les approches orientés objet
- Deux objets peuvent être égaux, mais correspondrent chacun à 2 instances différentes.

Sac à main
Couleur
Texture
Forme
Marque

2008

NFA013 - CNAM Grenoble -
16/42

Christine

Sylvie



Les attributs

Les attributs représentent les caractéristiques de l'objet
(son *état*)

Un attribut est constitué du couple (nom, valeur)

Exemple :

Sac à main
Couleur : {rose, rouge, noir, marron, blanc, ...}
Texture : {Cuir, plastique, tissu, ...}
Forme : {carrée, enveloppe, sac à dos, ronde, ...}
Marque : {Gucci, Dior, Hermes, ...}

Remarque : on définit un domaine pour la valeur de l'attribut. Le plus souvent, c'est un *type* de base (entier, réel, chaîne de caractères, booléen) mais cela peut aussi être un type énuméré (ensemble fini de valeurs)

Comportement de l'objet

Il regroupe toutes les compétences de l'objet et il est décrit par l'ensemble des *opérations* applicables à l'objet.

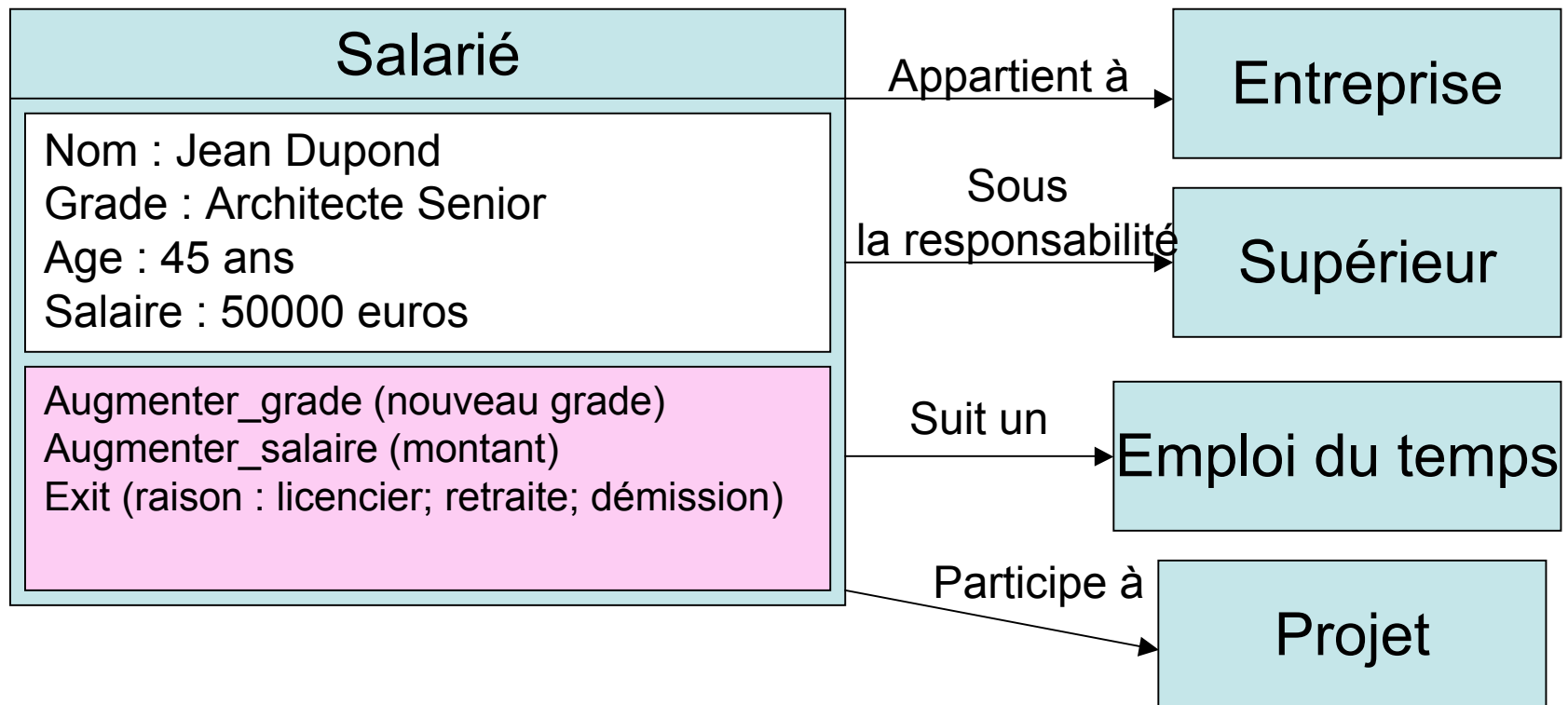
Décrit les actions et les réaction d'un objet (sa *dynamique*)

- Les méthodes (opérations)
- Les messages (évènements)

Une opération est déclenchée suite à une stimulation externe.

Exemple

Un salarié dans le système entreprise



Les associations

Elles symbolisent les liens d'un objet vers un autre

1. Elles peuvent être soit :

- des agrégations ou compositions
- des références (=pointeurs sur une instance).

2. Elles peuvent être aussi de nature :

- statique : lien durable entre les objets. Exemple : salarié appartient à une entreprise
- dynamique : liés établis par les opérations et liés au comportement. Exemple : changer de projet

Les rôles des objets

Lors d'échanges de messages, un objet peut être :

- Un *acteur*
 - objet à l'origine d'une interaction
 - actif : il possède un fil d'exécution, et émet des messages
- Un *serveur*
 - est en attente d'interaction
 - passif : destinataire de messages
- Un *agent*
 - acteur et serveur à la fois
 - base du mécanisme de délégation

L'encapsulation

«L'encapsulation est le processus qui consiste à compartimenter des éléments d'une abstraction pour constituer sa structure et son comportement » (*Booch, 1995*)

L'abstraction se focalise sur les caractéristiques **observables** et le comportement de l'objet

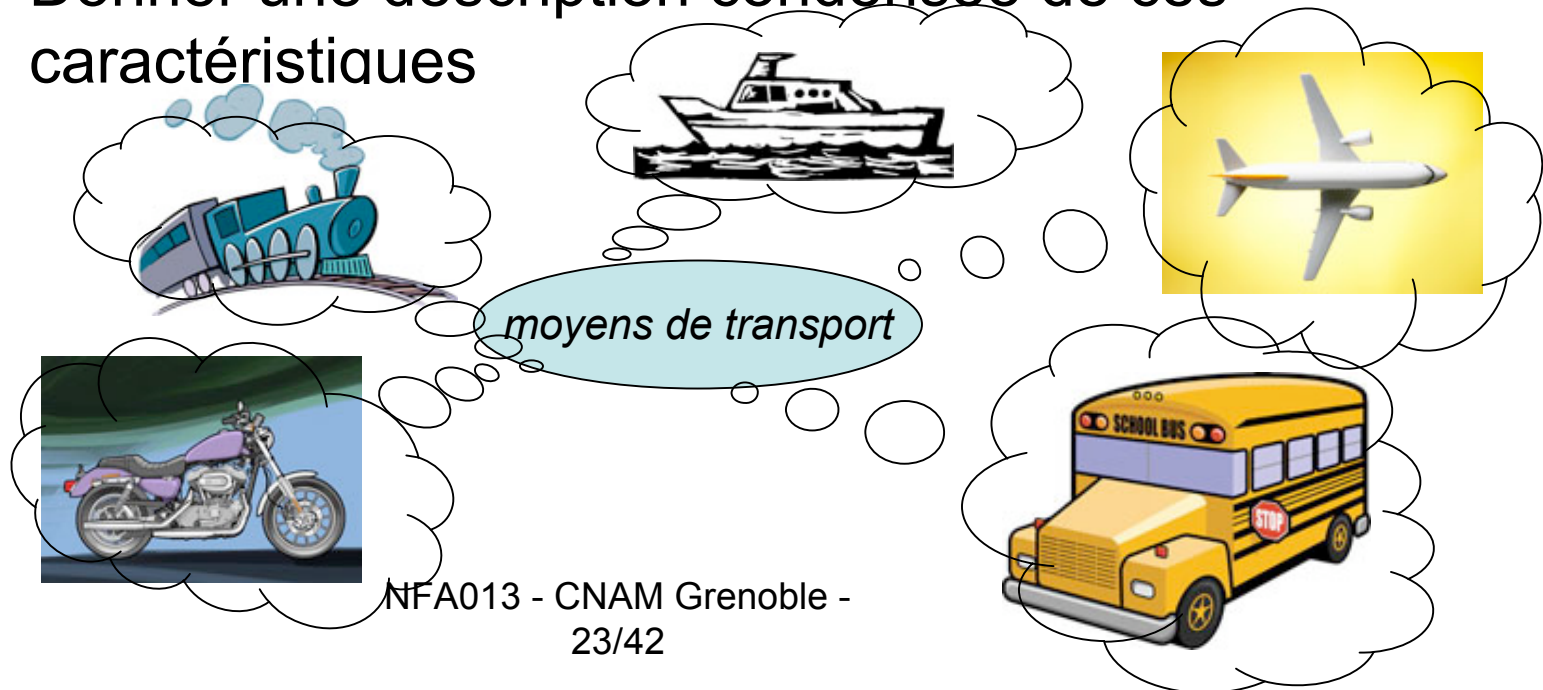
Encapsulation : masquer les détails de la réalisation

- Masquer la structure sous-jacente et l'implémentation des opérations
- Les plus gros bénéfices obtenus par une approche orientée objet proviennent du respect du principe d'encapsulation.

Comment abstraire ?

Regrouper des objets ou concepts qui se rassemblent

- Identifier des caractéristiques communes à un ensemble d'éléments
- Donner une description condensée de ces caractéristiques



La classe

1. Spécification :

- décrit le domaine de définition et les propriétés des instances de cette classe (équivalent de la notion de **type** dans les langages classiques)

2. Réalisation :

- décrit comment la spécification est réalisée et qui contient le corps des opérations et les données nécessaires à leur fonctionnement.

3. Catégories :

- Classes instanciables
- Classes non instanciable : *abstract*
- Classes dérivables
- Classes utilitaires : *System*

Les instances de classes

- Tout objet est une instance de classe
- La classe définit les comportements et les structures possibles de ces instances qui sont les objets
- Différentes instances (objets) peuvent avoir leurs opérations invoquées à différents moments, de différentes manières, et être dans différents états.
- Le mécanisme d'instanciation permet de générer des objets (instances) d'une classe.

Point $p = \textbf{new Point}(10, 5);$

Règles de la classification

1. Identification de chaque objet
2. Appartenance EXPLICITE de chaque objet à UNE classe. Implicite à plusieurs classes.
3. Importances des objets PERSISTANTS : mécanismes de sérialisation
4. UNICITE des noms de classe
5. Au moins UNE propriété (attribut ou méthode) par classe
6. Objet \neq valeur

Valeur : entité mathématique, éternelle, immuable, jamais créée ni détruite, ni mise à jour ! $2+5$ n'implique pas que 7 vient d'être créé, ni que 2 et 5 ont été consommés

Objet : entité modifiable tout en gardant son identité ; son état peut changer au cours du temps. Il peut être créé, détruit, abandonné, partagé, mis à jour.

Les relations entre les classes

1. L'association est une **connexion sémantique bidirectionnelle** entre classes
2. C'est une abstraction des liens qui existent entre les objets des classes associées.
3. Dès lors qu'un objet a un lien vers un autre objet, il peut émettre un message pour déclencher une de ses opérations.

Exemple : Entreprise E est une composition de Salarié S_i .
Une augmentation globale se traduit ainsi :

```
Liste salariés = E.listeDesSalariés();  
Pour i de 1 à E.nombreSalariés,  
    Salarié  $S_i$  = salariés.getElement(i);  
     $S_i$ .Augmenter_salaire (100);
```

L'agrégation et la composition

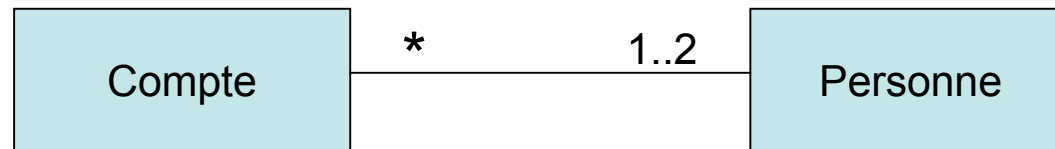
Relations de classes bidirectionnelle dissymétriques

Un mariage est composé d'une homme et d'une femme.

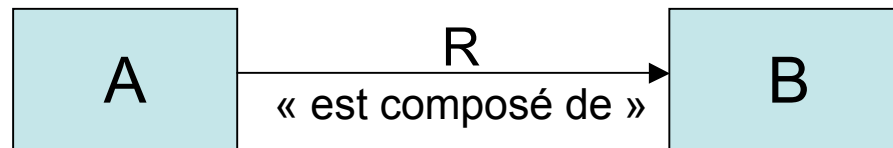
Un train est composé de voitures.

Une course agrège des participants.

Ces relations possèdent une **arité** (ou multiplicité) exprimé par un entier : $n \in \mathbb{N}$

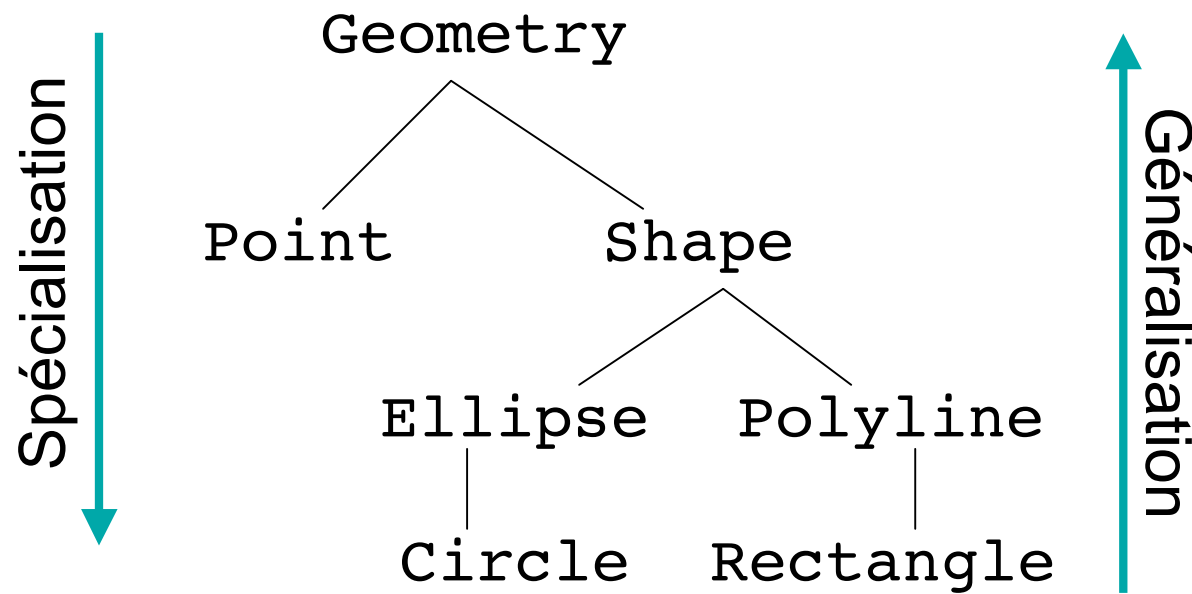


Une association peut avoir une **orientation**.



L'héritage

L'héritage est un mécanisme de spécialisation d'objets plus « généraux » vers des objets plus « spécifiques ». Les objets qui héritent sont une sous-catégorie de leur classe mère.



Circle hérite de Ellipse
Rectangle est une spécialisation de Polyline
Geometry est l'abstraction la plus générale de tous ces objets

Hiérarchie de classes

Avec l'héritage, on met en évidence des relations de **sous-typage**.

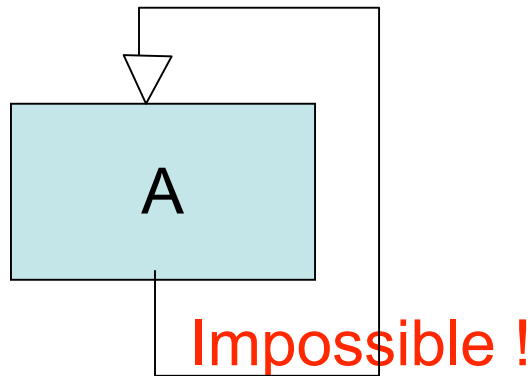
« Si une classe B hérite d'une classe A, alors aussi bien les opérations que la structure (=attributs) de A deviennent ceux de B » (*Jacobson, 1992*)

La généralisation signifie toujours « est un » ou « est une sorte de ». Ainsi, si Y hérite de X, on peut **substituer** une instance de X par une instance de Y.

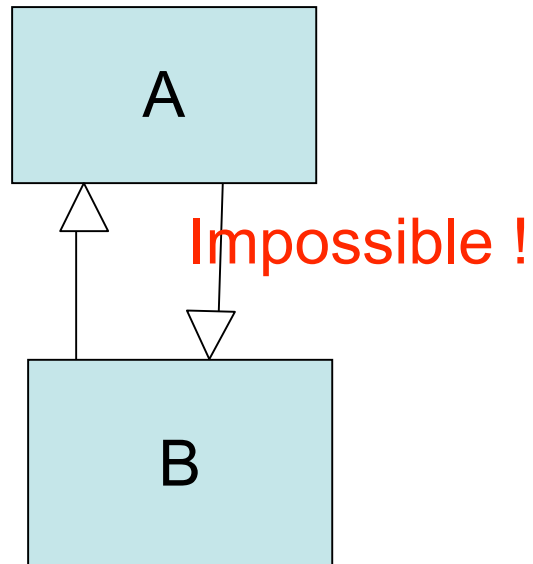
```
Geometry x = new Point();
```

Généralisation : propriétés de la relation

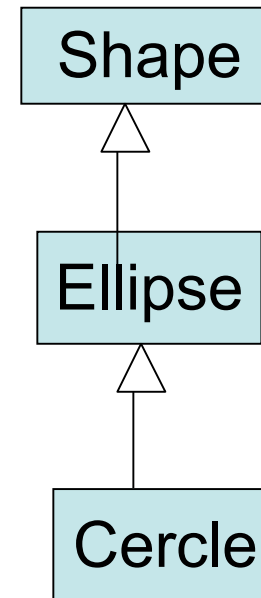
Relation
non réflexive



Relation
non symétrique



Relation transitive



Cercle est une Ellipse,
puisque qu'il hérite de Ellipse.
C'est aussi une forme
géométrique (*Shape*)
par transitivité

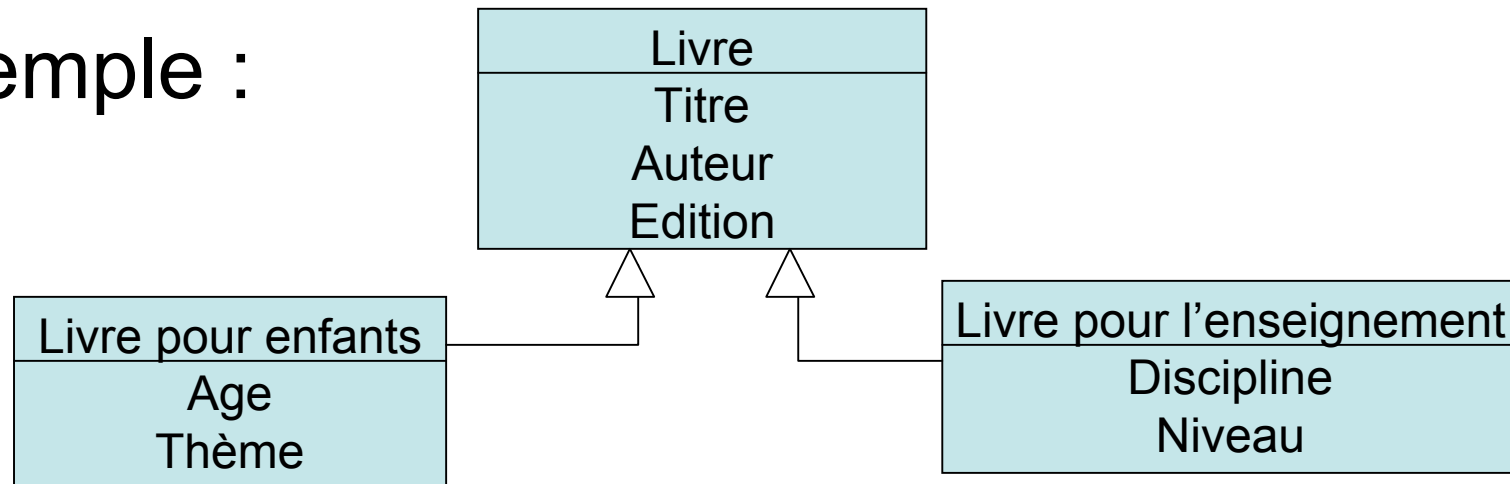
Des ensembles aux classes

- Les objets instance d'une classe partagent des caractéristiques générales, exprimées sous forme d'attribut, d'opération et de contrainte. C'est la **propriété caractéristique** de l'ensemble des instances.
- La propriété caractéristique de **X** est notée **P(X)**
- L'ensemble X peut être divisé en sous-ensembles.
- Dans une hiérarchie, l'ensemble des instances d'un sous-type sont incluses dans l'ensemble des instances du super type

Des ensembles aux classes

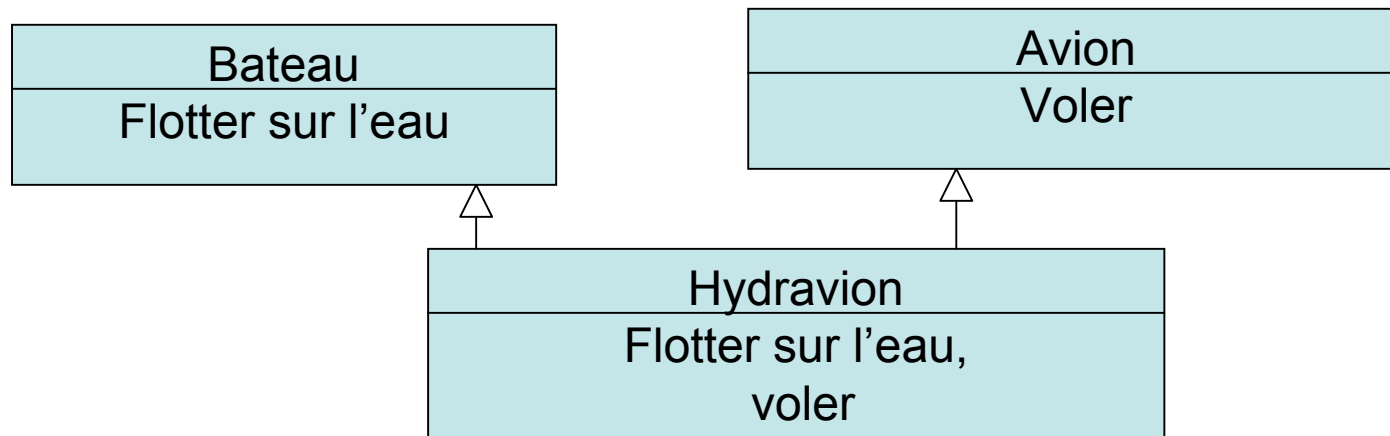
- La généralisation des classes correspond à la relation d'inclusion des ensembles.
- Les objets instances d'une classe donnée sont décrits par la propriété caractéristique de leur classe, mais également par les propriétés caractéristiques de toutes les classes parents de leur classe.

Exemple :



Des ensembles aux classes

1. La généralisation - sous sa forme multiple - existe également entre arbres de classes disjoints
2. Elle permet à une classe d'hériter de deux classes ou plus (NB : interdit en Java)



Notion de classe abstraite

- Une **classe abstraite** (*abstract*) ne peut être instanciée.
- Une opération abstraite est une méthode déclarée sans implémentation.

Conséquences :

- Les instances d'une classe abstraite sont forcément des instances de ses sous-classes.
- Une classe abstraite ne donne pas directement des objets : il est nécessaire de la spécialiser pour obtenir des instances.
- Utilisée pour regrouper des attributs et opérations communes à plusieurs classes.

NB : en Java, l'**interface** est l'équivalent d'une classe abstraite avec seulement des opérations

Polymorphisme

« Polymorphe » désigne un élément qui peut prendre plusieurs formes.
En informatique, cela décrit un nom d'objet qui peut désigner des instances de classe différentes issues d'une même arborescence.

Principe général :

Les interactions des objets sont décrites selon les termes des spécifications définies dans leurs superclasses.

Dans ce cas, on parle de **polymorphisme d'opération** : déclencher des opérations différentes en réponse à un même message. Une sous-classe peut modifier localement le comportement d'une opération héritée d'une super-classe.

Application du polymorphisme

Exemple

```
Rectangle x = new Rectangle ();  
Circle y = new Circle();  
Triangle z = new Triangle();  
CollectionGeometry[3] = { x, y, z};  
For (int i=0; i<3; i++) {  
    CollectionGeometry[i].draw();  
}
```

Collection polymorphe, avec spécialisation d'une opération.

Polymorphisme

Le polymorphisme existe dans des environnements typés et non typés. Seuls les environnements typés garantissent une exécution sans surprise.

Attention :

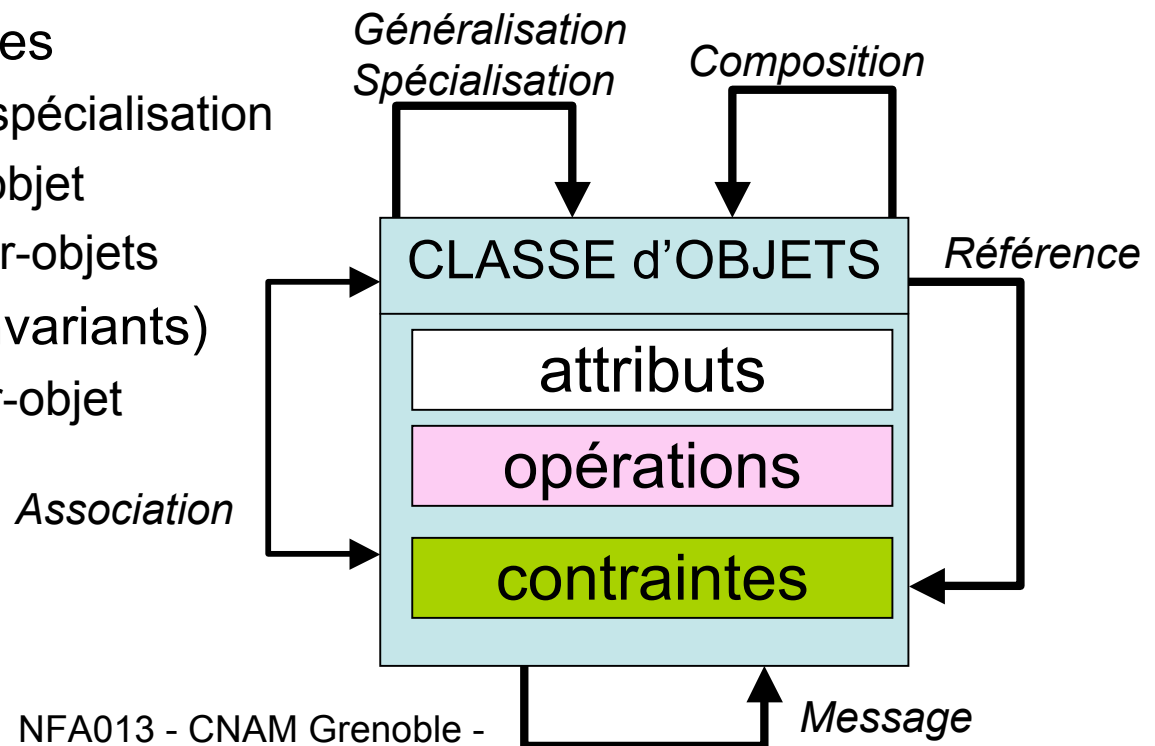
- confusion entre polymorphisme et surcharge des opérations (= profil de paramètres différents pour des opérations de même noms).
- Confusion avec la liaison dynamique qui est résolue dynamiquement à l'exécution.

Conclusion

- Objet : entité qui combine les caractéristiques abstraites à partir du domaine système
- Classe : expression des caractéristiques communes aux objets
- Encapsulation : mécanisme permettant de masquer les détails de l'implémentation
- Association (par agrégation/composition) permet de construire des objets plus complexes et l'échanges de messages
- Héritage : relations entre classes et sous-classes décrivant des ressemblances, et permettant de factoriser du code
- Polymorphisme : mécanisme permettant l'invocation « aveugle » de méthodes sur un objet

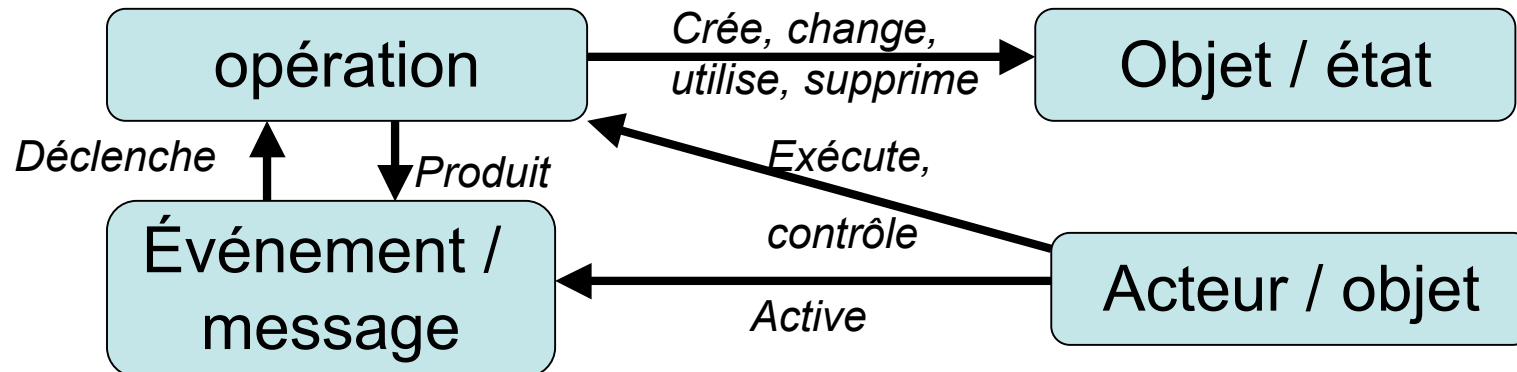
Aspects statiques

- Identification de l'objet
- Valeurs de l'objet
- Structuration complexe des données : produit cartésien, ensemble, liste
- Graphes de classes
 - Généralisation/spécialisation
 - Composition d'objet
 - Association inter-objets
- Contraintes (ou invariants)
 - Intra-objet, Inter-objet



Aspects dynamiques

- Donner une vie à des objets qui naissent, évoluent et meurent
- États pour contrôler l'évolution
- Évènements pour déclencher des changements d'états
- Messages pour supporter des évènements
- Opérations pour gérer les objets en réaction aux évènements
- Acteurs pour contrôler et exécuter les opérations
- Flux d'information pour communiquer
- Architecture des applications



Les diagrammes UML

1. Besoin des utilisateurs : use cases
2. Aspects statiques
 - diagrammes de classes
 - diagrammes d'objets
3. Aspects dynamiques
 - diagrammes de séquence,
 - diagrammes de collaboration,
 - automates d'états-transitions
 - Diagrammes d'activités
4. Aspects physiques
 - Les diagrammes de composants
 - Les diagrammes de déploiement