

# Modélisation objet et diagrammes UML statiques

---

Aurélien Tabard  
Département Informatique  
Université Claude Bernard Lyon 1

# Plan général

---

1. Introduction au langage de modélisation UML
2. Cas d'utilisations
3. Modélisation objet et diagrammes UML statiques
4. Modélisation UML dynamique

# Plan général

---

1. Introduction au langage de modélisation UML
2. Cas d'utilisations
- 3. Modélisation objet et diagrammes UML statiques**
  - Une petite histoire de l'orienté objet
  - Principes de bases
  - Classe
  - Encapsulation
  - Relations entre classes
  - Modèles UML
4. Modélisation UML dynamique

# Objectifs de ce cours

---

- Découvrir les notions de base de l'orienté objet
- Être capable de lire et de construire des diagrammes de classes

# Plan du cours

---

## Modélisation objet et diagrammes UML statiques

- Une petite histoire de l'orienté objet
- Objets et classes
- Encapsulation
- Attributs et opérations
- Relations entre classes
- Diagrammes UML statiques

# Une petite histoire de l'orienté objet

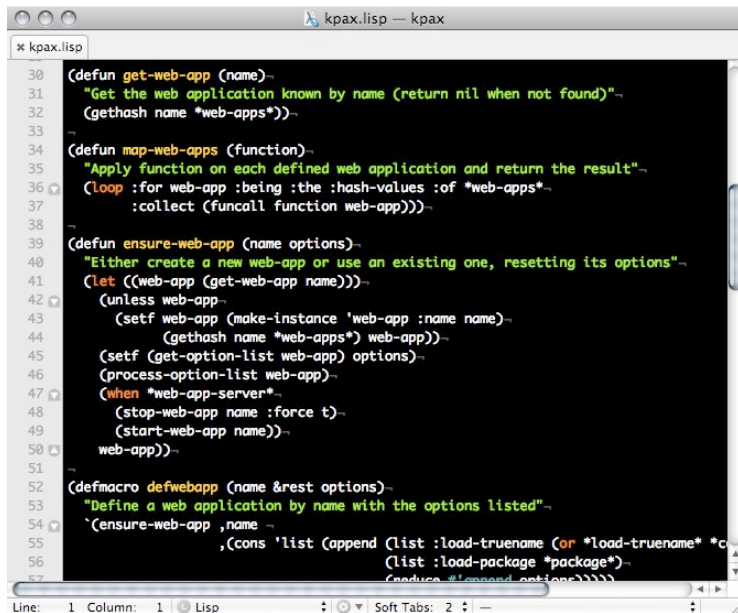
---

## Rappel :

- Méthodes de modélisation par les fonctions
- Méthodes de modélisation par les données
- Méthodes de modélisation orientée-objet

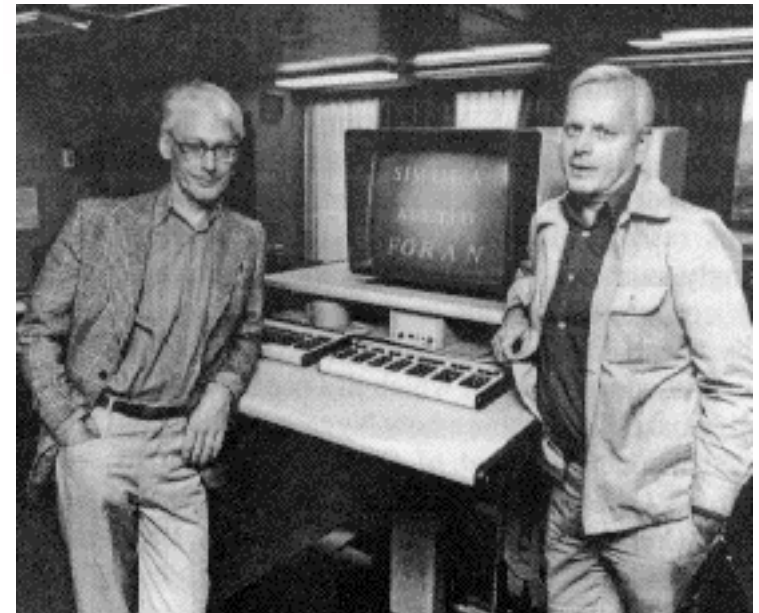
# LISP et Simula – 196x

# LISP



```
kpax.lisp — kpax
x kpax.lisp
30 (defun get-web-app (name)
31   "Get the web application known by name (return nil when not found)"
32   (gethash name *web-apps*))
33
34 (defun map-web-apps (function)
35   "Apply function on each defined web application and return the result"
36   (loop :for web-app :being :the :hash-values :of *web-apps*
37         :collect (funcall function web-app)))
38
39 (defun ensure-web-app (name options)
40   "Either create a new web-app or use an existing one, resetting its options"
41   (let ((web-app (get-web-app name)))
42     (unless web-app
43       (setf web-app (make-instance 'web-app :name name)
44             (gethash name *web-apps*) web-app))
45     (setf (get-option-list web-app) options)
46     (process-option-list web-app)
47     (when *web-app-server*
48       (stop-web-app name :force t)
49       (start-web-app name))
50     web-app))
51
52 (defmacro defwebapp (name &rest options)
53   "Define a web application by name with the options listed"
54   'ensure-web-app ,name
55     ,(cons 'list (append (list :load-truename (or *load-truename* *c
56     (reduce #'load-package *package*)
57     (reduce #'load-package *package*)))))
Line: 1 Column: 1 | Lisp | Soft Tabs: 2
```

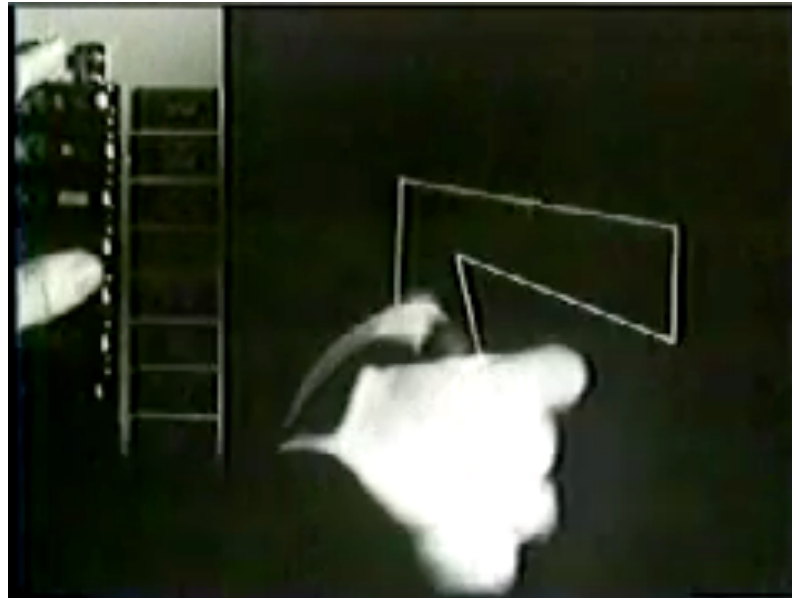
# simula



Ole-Johan Dahl et Kristen Nygaard ©R. Myhre

# Sketchpad - (Sutherland, 1963)

---

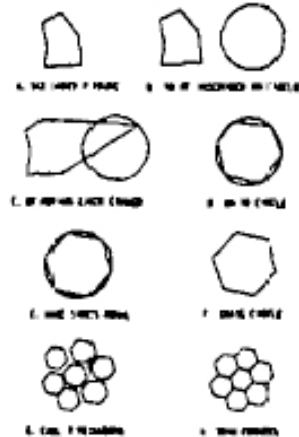




# Sketchpad - (Sutherland, 1963)

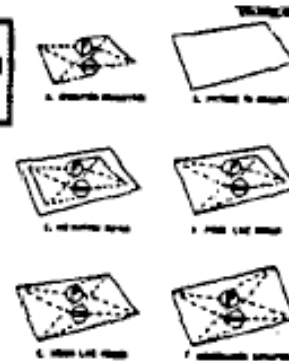


When there was only one personal computer  
Ivan at the TX-2 ca. 1962



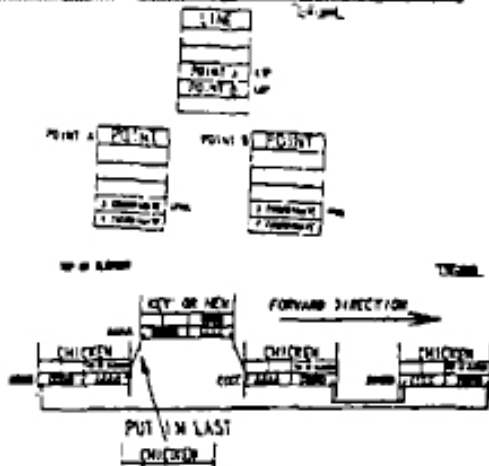
Drawing in Sketchpad

Constraints represented as icons



Constraints merged with picture

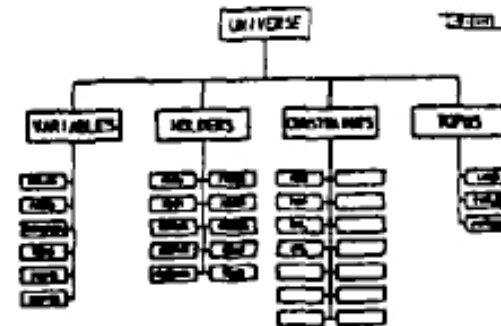
Programming with constraints



Sketchpad Structures

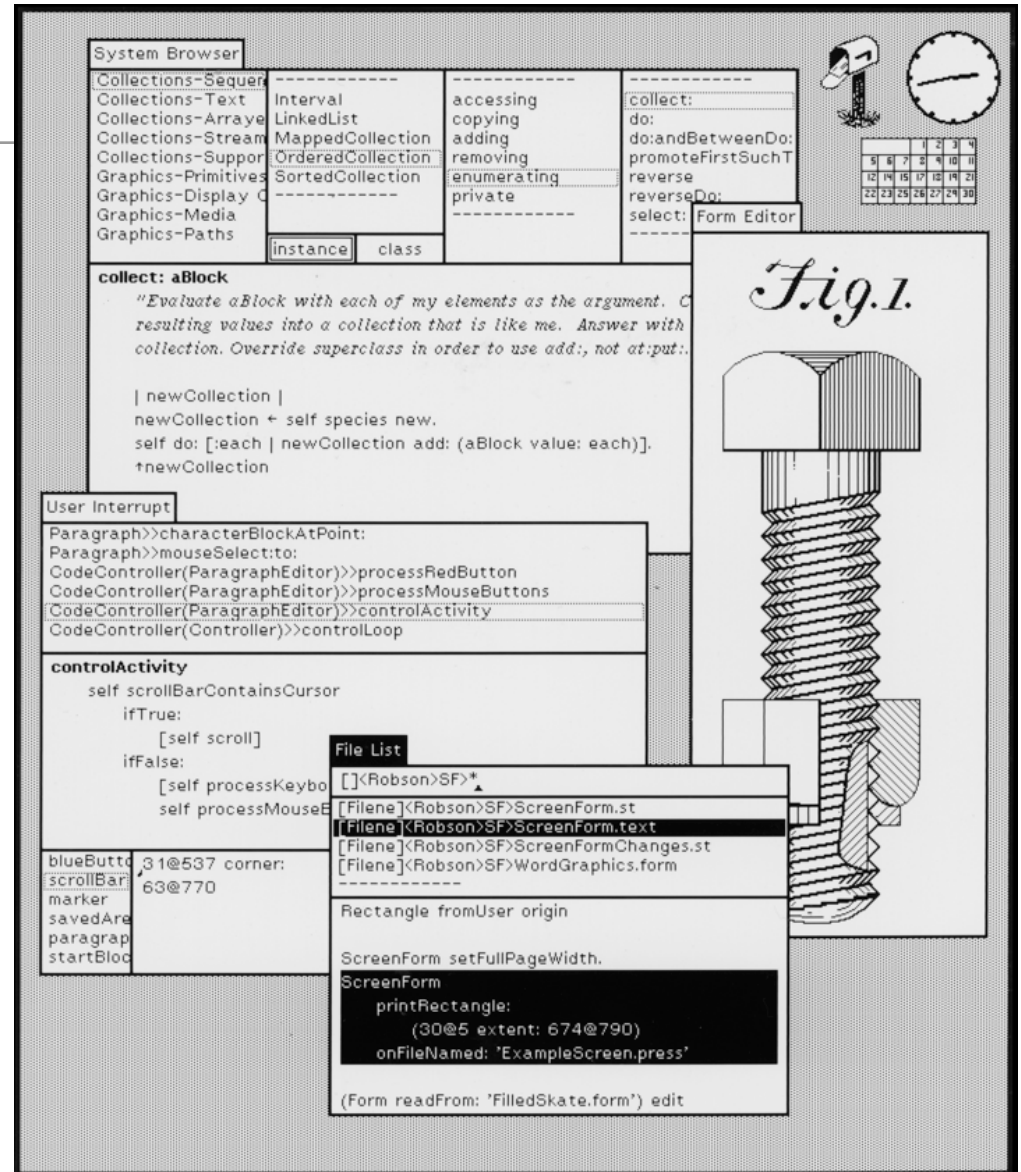
24	4	VARIABLES	TYPE
-2		0000	SPECS
		TYPEWRITER EDGE HAVE	NAME
		SUBROUTINE ENTRY	DISPLAY
		FIT SCOPE AROUND IT	MOVABLE
		APPLY TRANSFORMATION	MOV IT
		N. TL.	SIZE
		NORMAL PICTURE END	KIND
		FOUR COMPONENTS	TUPLE
		VALUE AT TIME	TABLE

"generic block" showing procedural attachment



Sketchpad's "inheritance" hierarchy

# Smalltalk - 1970/80



# Langages Orientés Objets Actuels

---

Les plus populaires :

- Java
- C++
- Objective C
- C#

# Les hybrides

---

Quelques exemples :

- Python, Ruby -> Objet + Procédural (impératif)
- Scala -> Objet + Fonctionnel

# Plan

---

## Modélisation objet et diagrammes UML statiques

- Une petite histoire de l'orienté objet
- Objets et classes
- Encapsulation
- Attributs et opérations
- Relations entre classes
- Diagrammes UML statiques

# Objets et classes

---

Un *objet* encapsule ses données

Un *objet* a :

- une identité (une référence unique)
  - numéro de sécurité sociale, numéro de passeport
- un état, composé d'attributs
  - taille, forme, couleur...
- un comportement, des opérations possibles (aussi appelées méthodes)
  - manger, dormir...

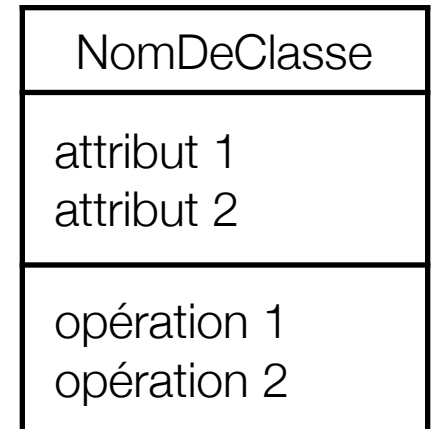
Un *objet* est une *instance* de la *classe*.

# Classes en UML

---

## Représentation

- rectangle à 3 compartiments
  - Nom
  - Attributs
  - Opérations
- plus ou moins de détails suivant les besoins



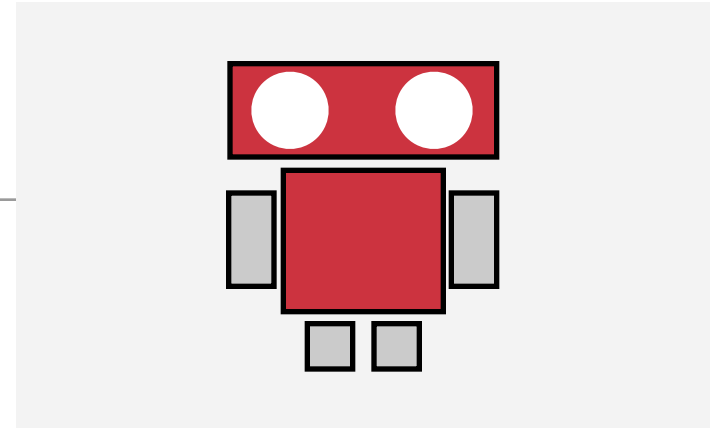
Nom : singulier, majuscule (en général)

- ex. : Fichier, Client, Compte, Chat

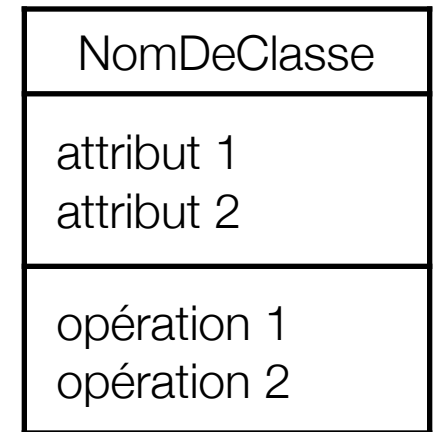
# Exemple / Exercice

---

Représentation objet d'une application de dessin



- **Combien de classes?**
  - combien d'objets
  - noms des classes
- **Quels attributs**
  - noms des attributs?
  - sont ils partagés entre les classes?
- **Quelles opérations (méthodes)?**
  - noms des méthodes
  - sont elles partagés entre les classes?

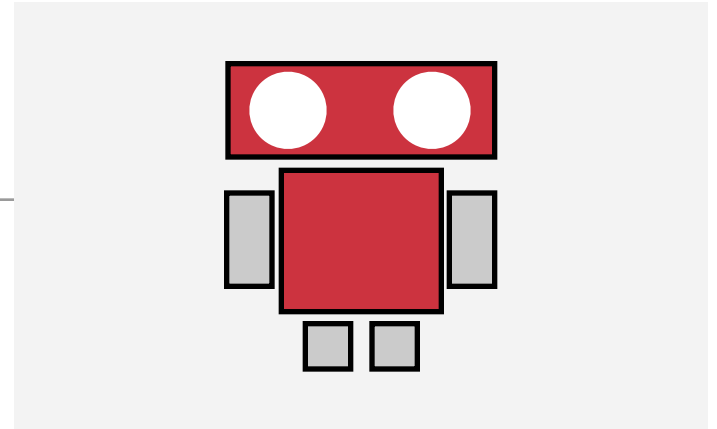




# Exemple / Exercice

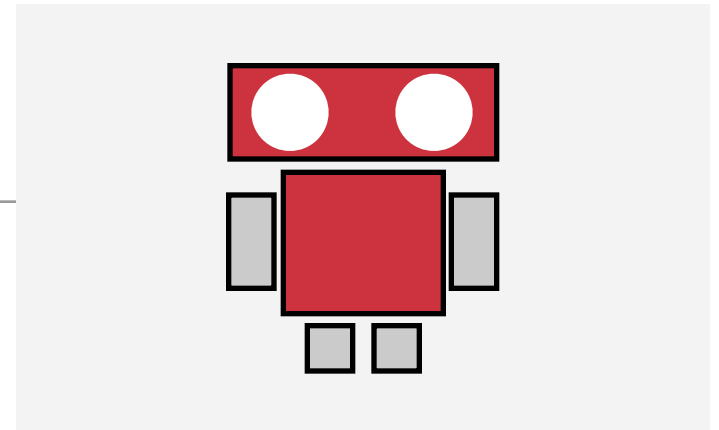
---

Représentation objet d'une application de dessin



# Exemple / Exercice

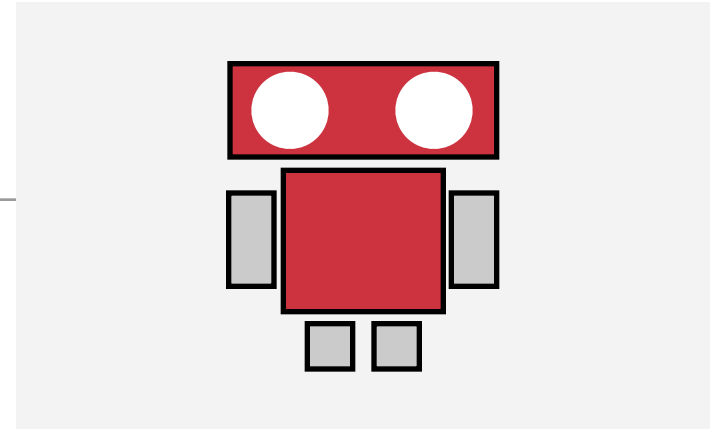
Représentation objet d'une application de dessin



Cercle
x y z diametre couleur
deplacer(x,y,z)

Rectangle
x y z longueur largeur couleur
deplacer(x,y,z)

# Exemple / Exercice



Représentation objet d'une application de dessin

Ou encore :

Cercle
Position Bordure diametre
move(x,y,z) ...

Rectangle
Position Bordure longueur largeur
move(x,y,z); ...

Bordure
largeur Couleur
setLargeur(l) getLargeur() setCouleur(c) getCouleur()

Position
x y z
set(x,y,z) getX(); getY(); getZ():

# Plan

---

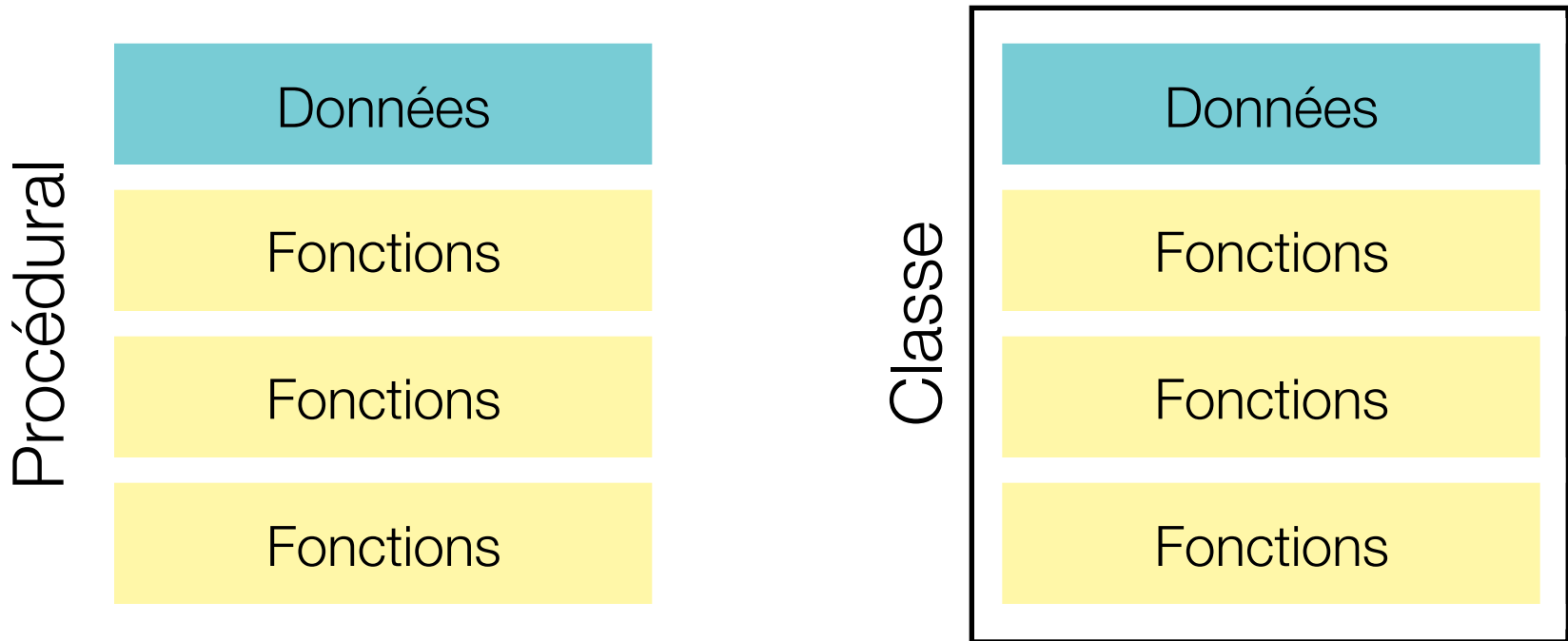
## Modélisation objet et diagrammes UML statiques

- Une petite histoire de l'orienté objet
- Objets et classes
- **Encapsulation**
- Attributs et opérations
- Relations entre classes
- Diagrammes UML statiques

# Encapsulation : protéger l'information

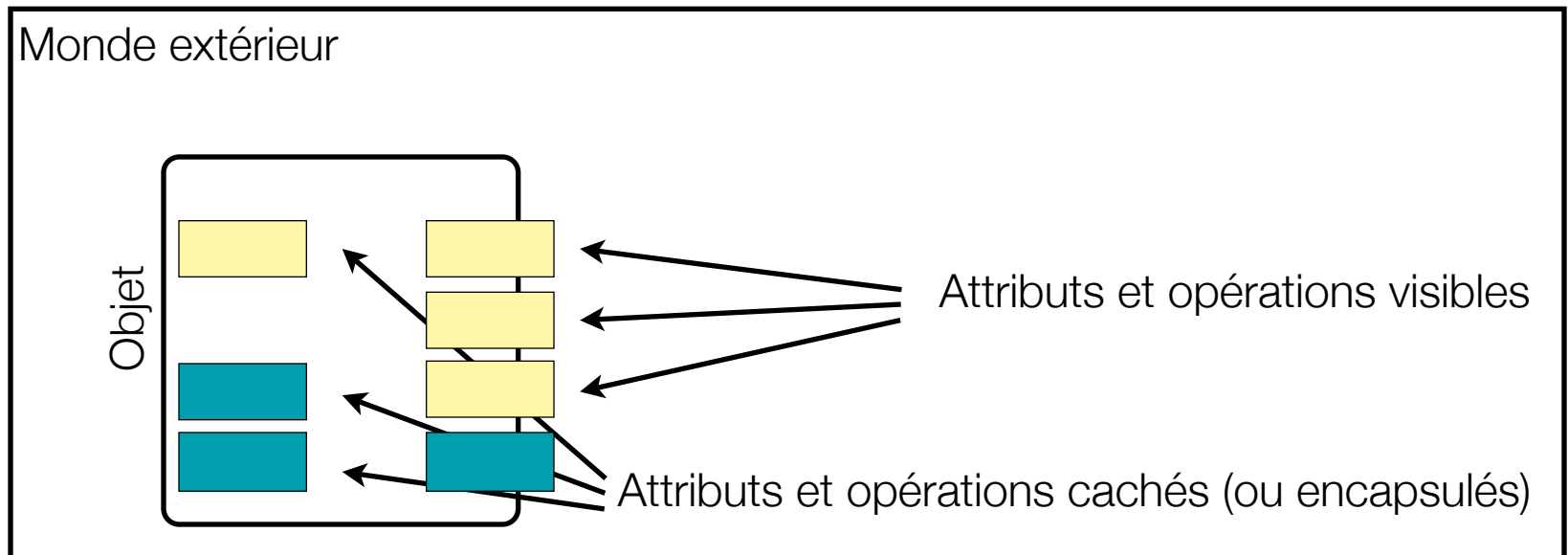
---

- Les données peuvent être encapsulées pour être invisibles aux monde extérieur
- Les données sont alors seulement accessibles par des méthodes



# Encapsulation : protéger l'information

- Le “monde extérieur” ne peut dépendre que de ce qu’il voit!
- L’objet offre une protection entre ses données et le monde extérieur
- Les données et méthodes cachées peuvent changer sans affecter le monde extérieur.



# Plan

---

## Modélisation objet et diagrammes UML statiques

- Une petite histoire de l'orienté objet
- Objets et classes
- Encapsulation
- **Attributs et opérations**
- Relations entre classes
- Diagrammes UML statiques

# Attributs de classes

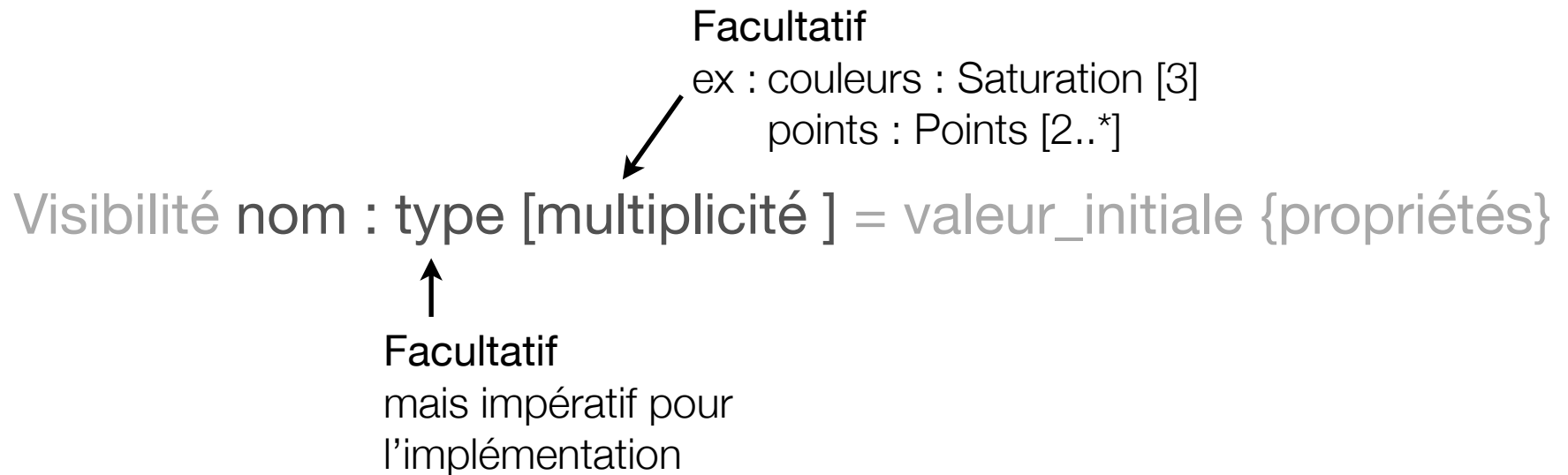
---

Visibilité nom : type [multiplicité ] = valeur\_initiale {propriétés}



# Attributs de classes

---



# Attributs de classes

---

Visibilité nom : type [multiplicité ] = valeur\_initiale {propriétés}

# Attributs de classes

---

Visibilité nom : type [multiplicité ] = valeur\_initiale {propriétés}



public +

privé -

protégé #

paquetage ~

# Attributs de classes

---

Visibilité nom : type [multiplicité ] = valeur\_initiale {propriétés}

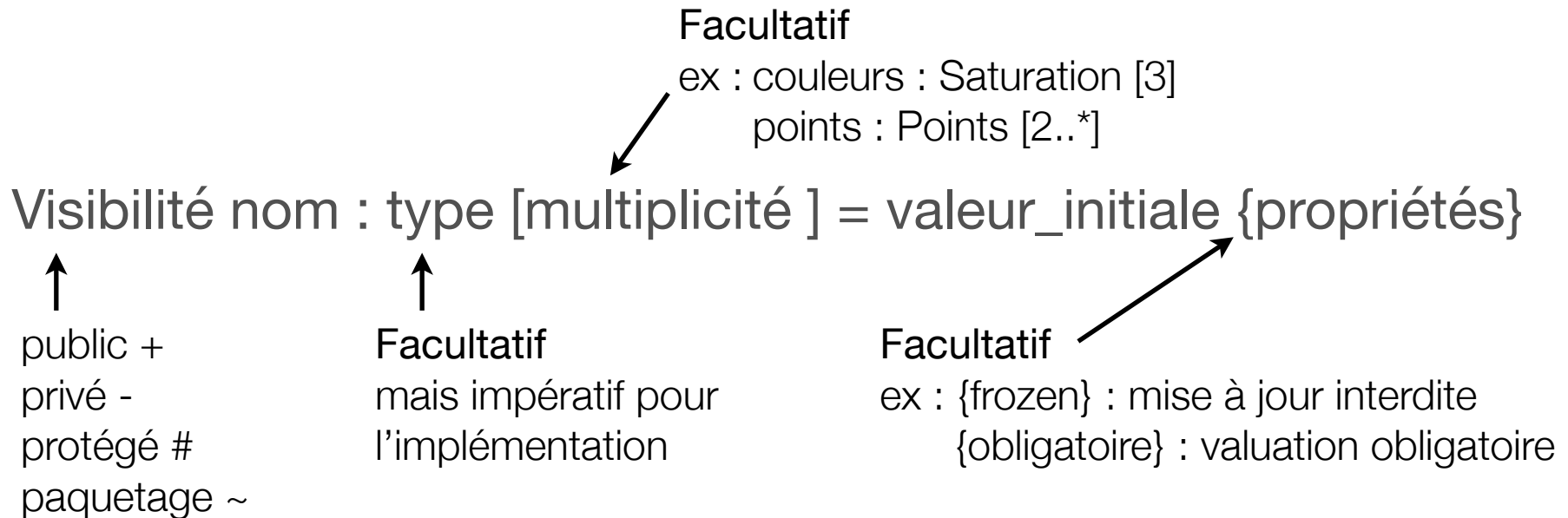
Facultatif

ex : {frozen} : mise à jour interdite

{obligatoire} : valuation obligatoire

# Attributs de classes

---

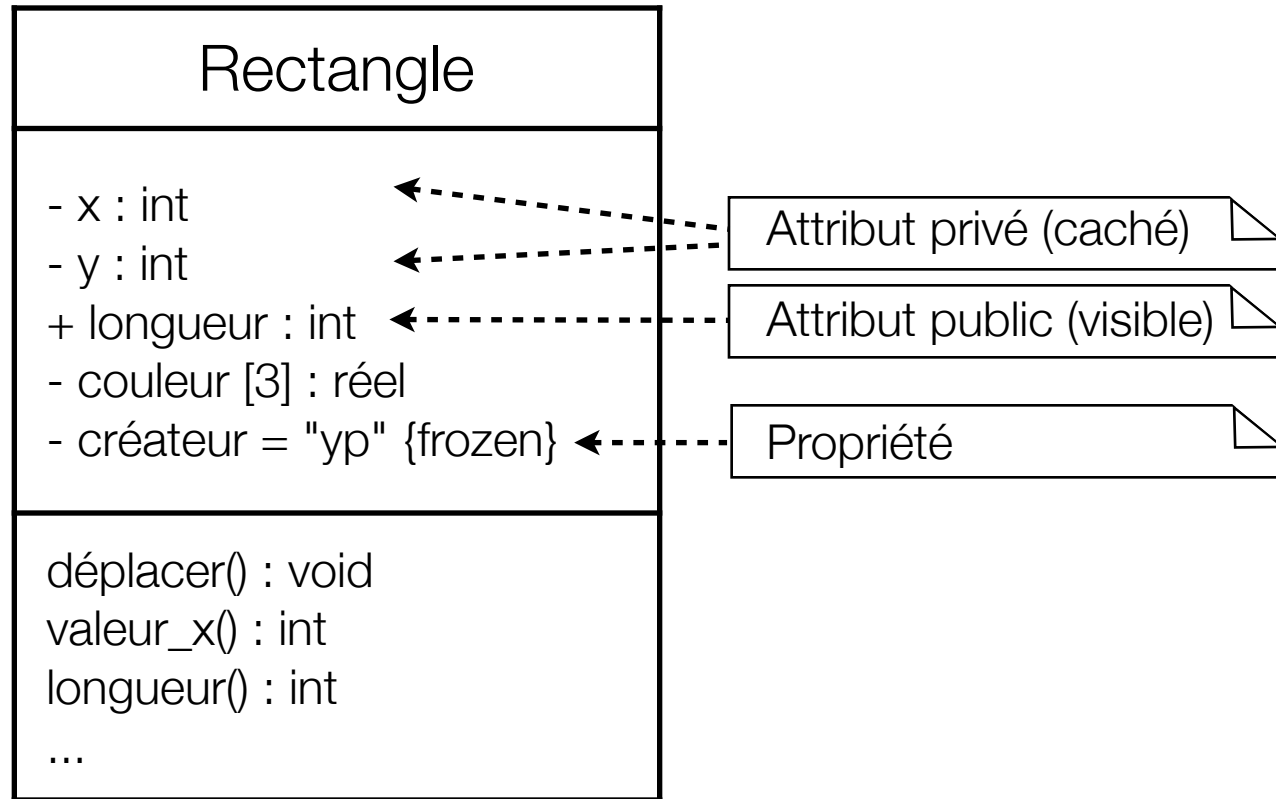


## Remarques

- /nom : attribut dérivé (calculé)
- souligné : attribut statique (de classe)
- {frozen} : disparu de UML2 ; à utiliser quand-même

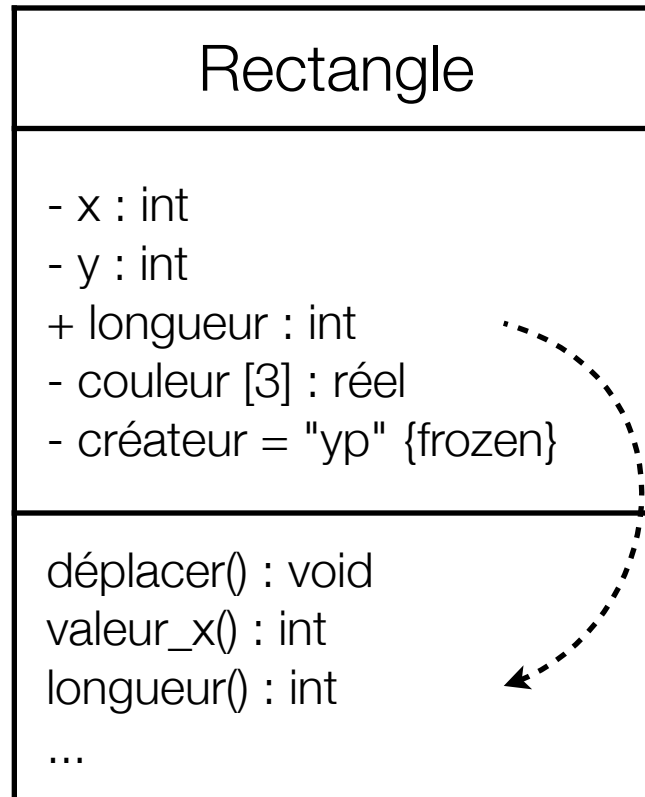
# Exemple

---

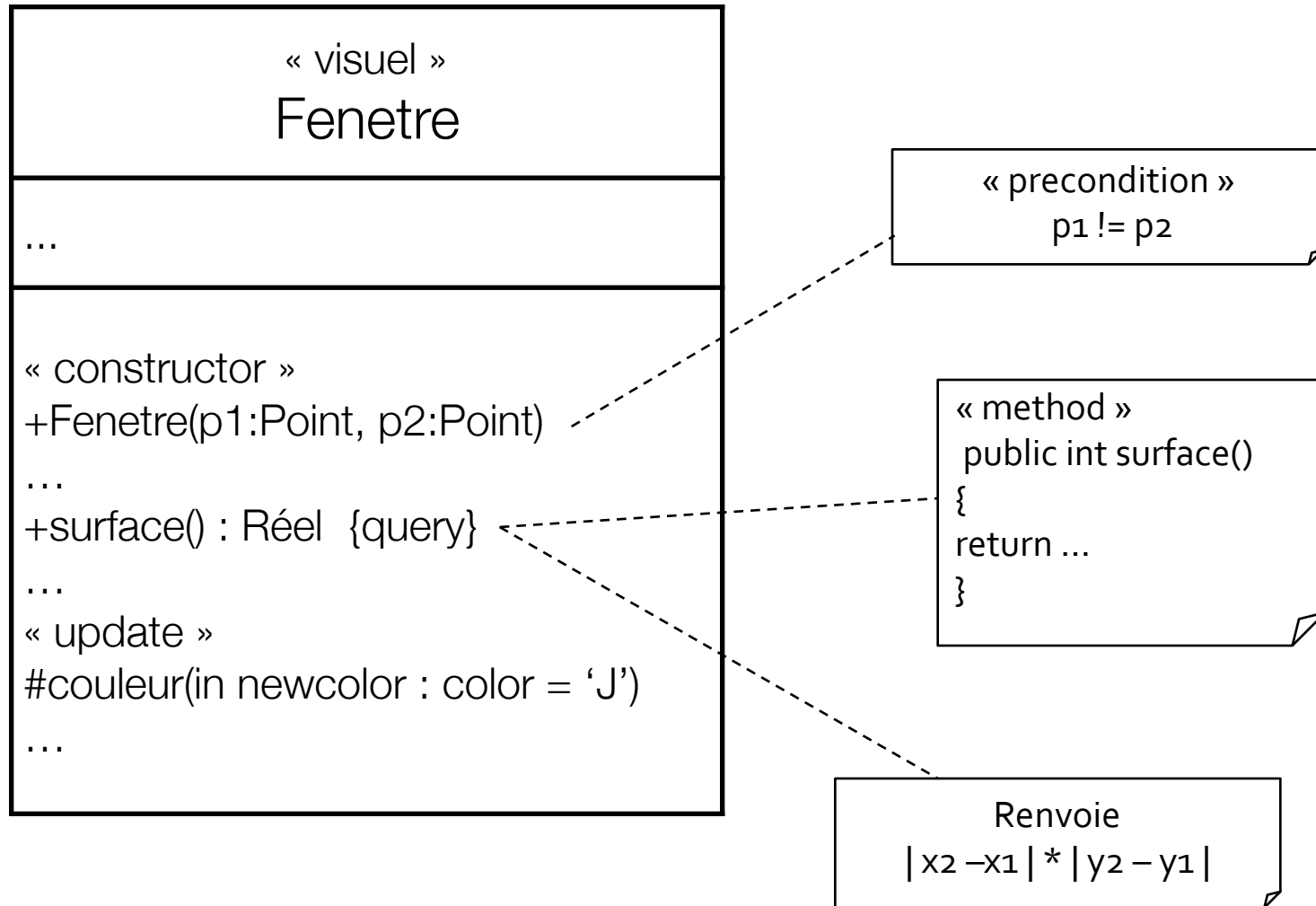


# Exemple

---

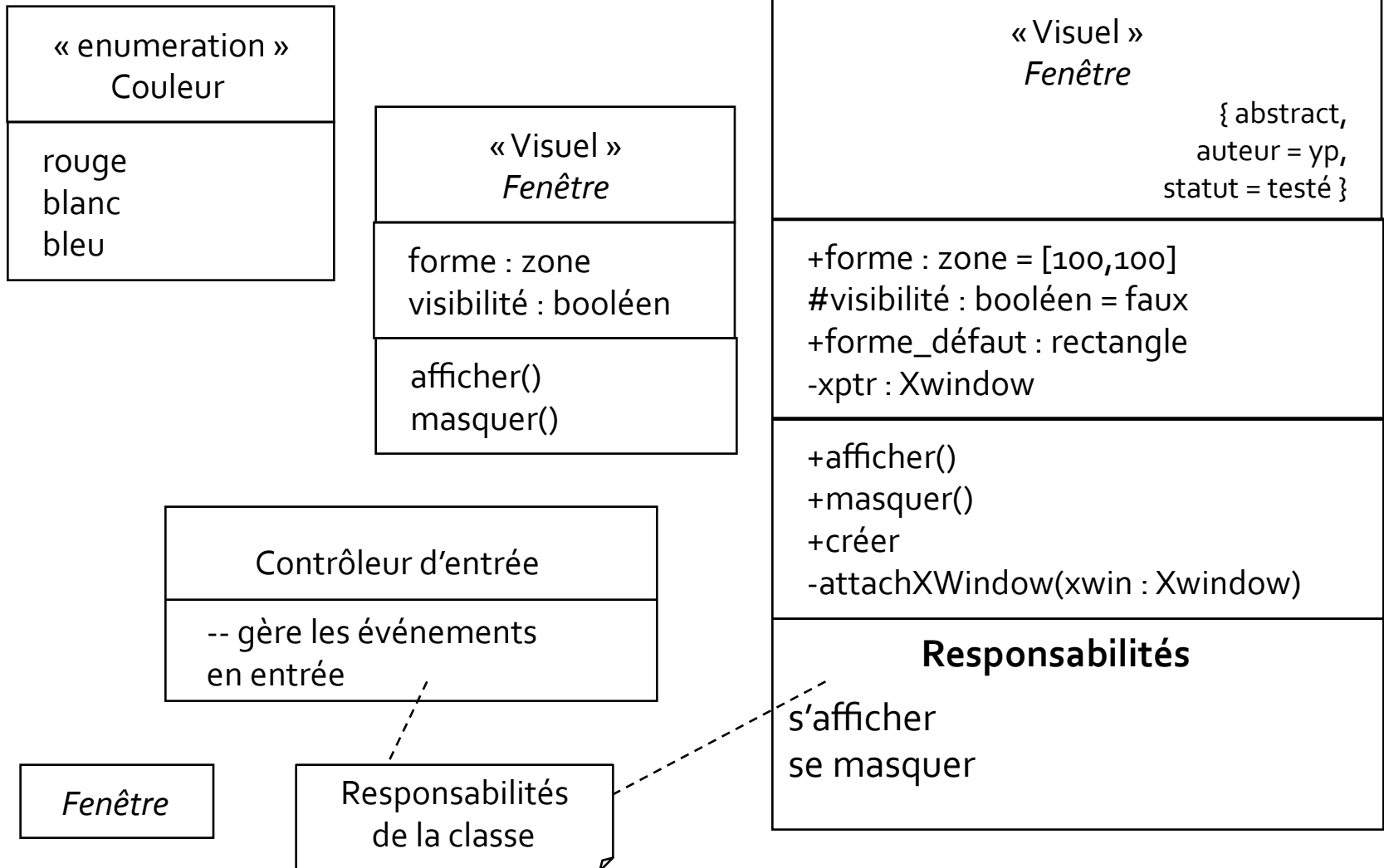


# Exemple





# Autres exemples de classes



# Plan

---

## Modélisation objet et diagrammes UML statiques

- Une petite histoire de l'orienté objet
- Objets et classes
- Encapsulation
- Attributs et opérations
- Relations entre classes
- Diagrammes UML statiques

# Relations entre classes / liens entre objets

---

- Association

- les instances des classes sont liées
- possibilité de communication entre objets
- relation forte : composition



- Généralisation/spécialisation

- les instances de la sous-classe sont des instances de la super-classe (niveau conceptuel)
- héritage (niveau implémentation)



- Dépendance

- la modification d'une classe peut avoir des conséquences sur une autre



- Réalisation

- une classe réalise une interface



# Relations entre classes / liens entre objets

---

- Association

- les instances des classes sont liées
- possibilité de communication entre objets
- relation forte : composition



- Généralisation/spécialisation

- les instances de la sous-classe sont des instances de la super-classe (niveau conceptuel)
- héritage (niveau implémentation)



- Dépendance

- la modification d'une classe peut avoir des conséquences sur une autre



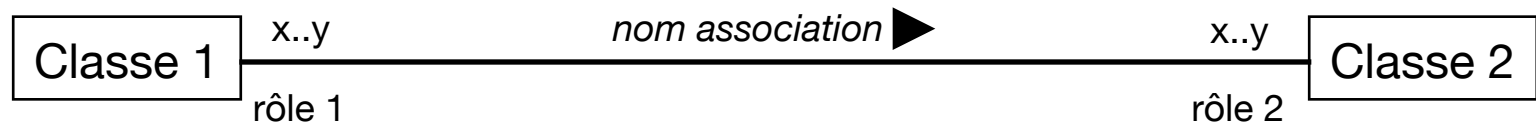
- Réalisation

- une classe réalise une interface



# Associations

---

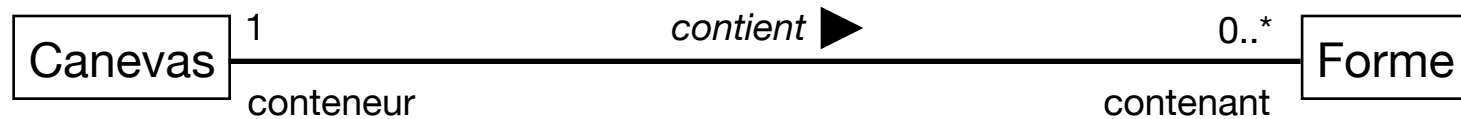


- Nom : forme verbale, sens de lecture avec flèche
- Rôles : forme nominale, identification extrémité association
- Multiplicité : 1, 0..1, 0..\*, 1..\*, n..m
- Mots-clés : set, ordered set (uniques) ; bag, list (doublons)

# Associations

---

- Les associations ont une durée de vie, sont indépendantes les unes des autres, sont héritées, comme les attributs.



# Associations : remarques

---

Tout objet doit être accessible via un lien

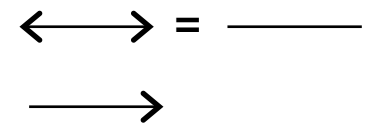
- ne peut recevoir de messages sinon
- liens plus ou moins permanents : voir “Visibilités”

## Multiplicité

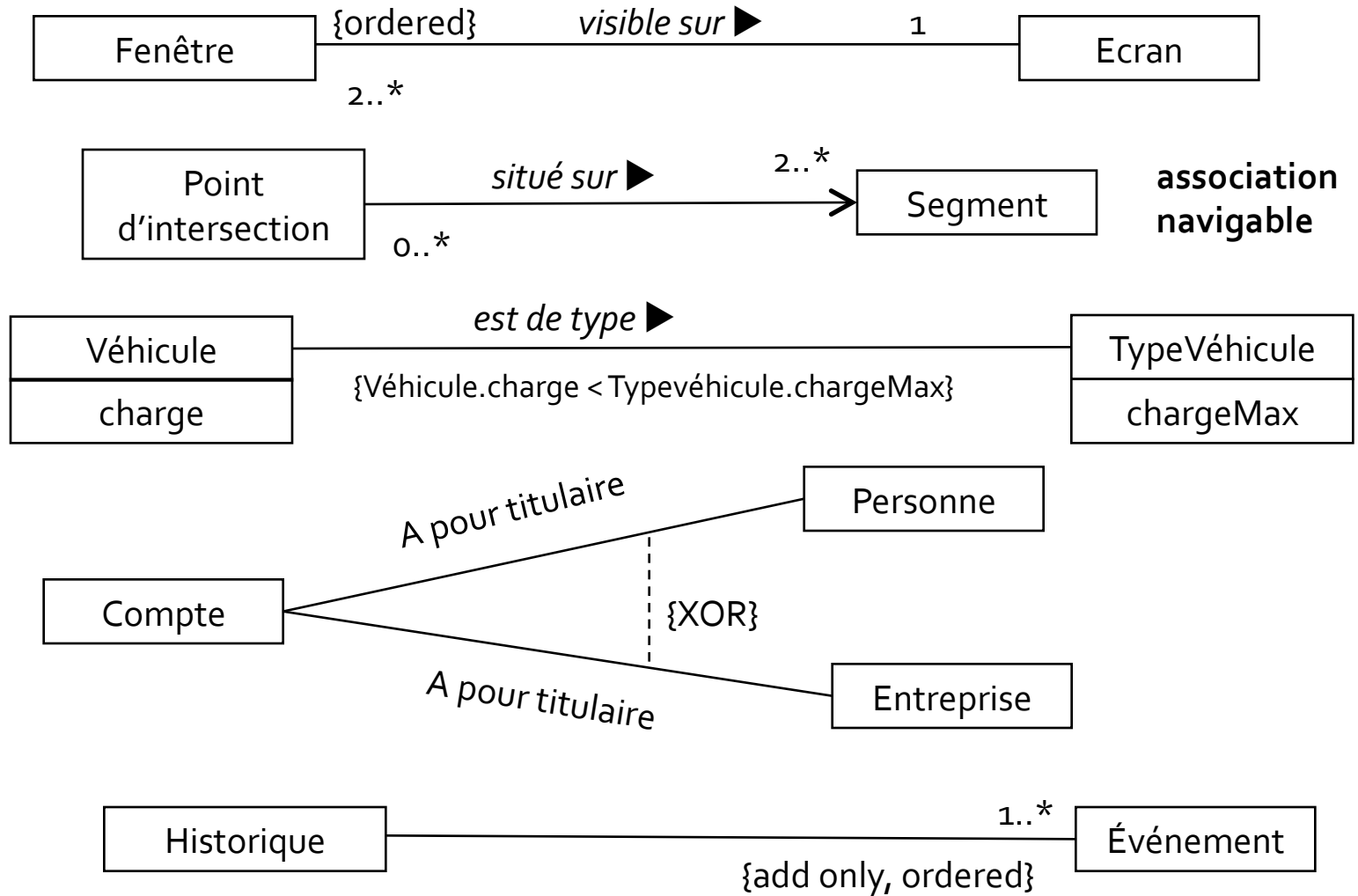
- nombre d’instances d’une classe en relation avec une instance d’une autre classe
- pour chaque association
  - deux décisions à prendre : deux extrémités

## Directionnalité

- bi-directionnalité par défaut, evt explicitée
- restriction de la navigation à une direction



# Associations et contraintes





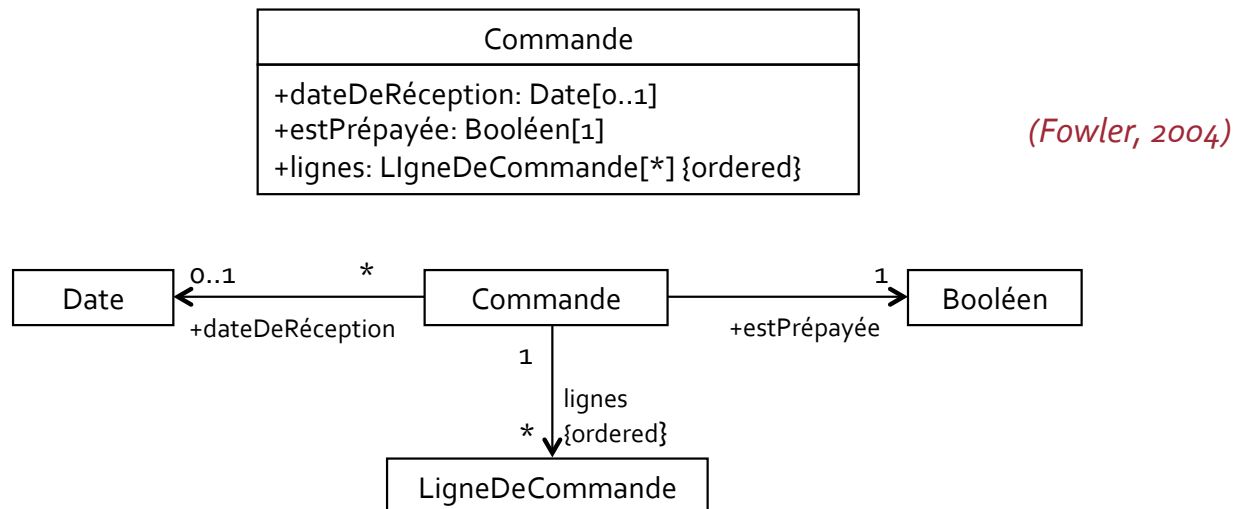
# Propriétés : caractéristiques structurelles des classes

---

Concept unique regroupant attributs et associations mono-directionnelles : équivalence des représentations

Pour choisir

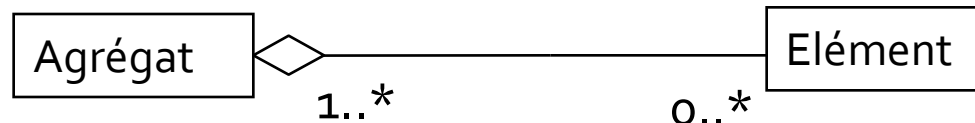
- attribut (texte) pour les types de données
  - objets dont l'identité n'est pas importante
- association pour insister sur les classes



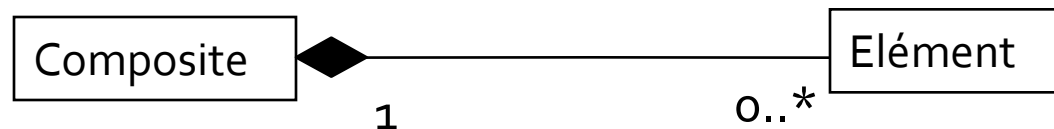
# Agrégations et compositions

---

- Associations asymétriques, fortes
- Agrégation
  - non nommée, structure d'arbre sous-jacente (pas de cycle), rôle prépondérant d'une extrémité

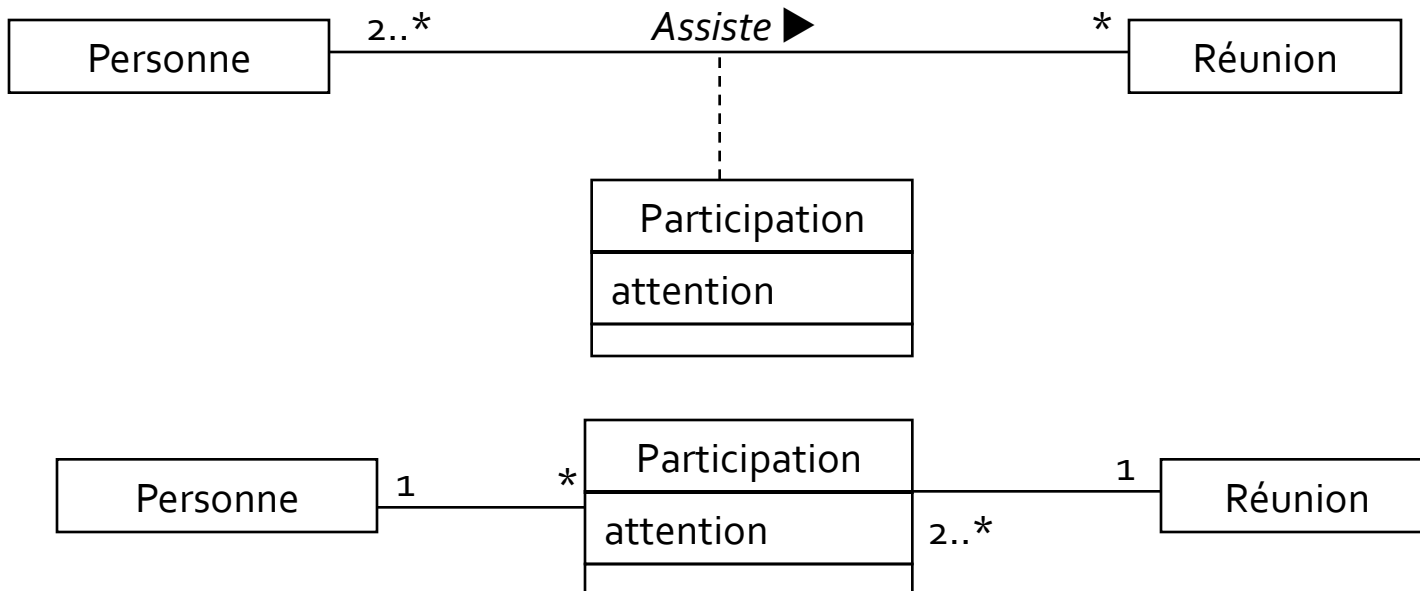


- Composition
  - non partage des éléments composants, création et destruction des composants avec le composite



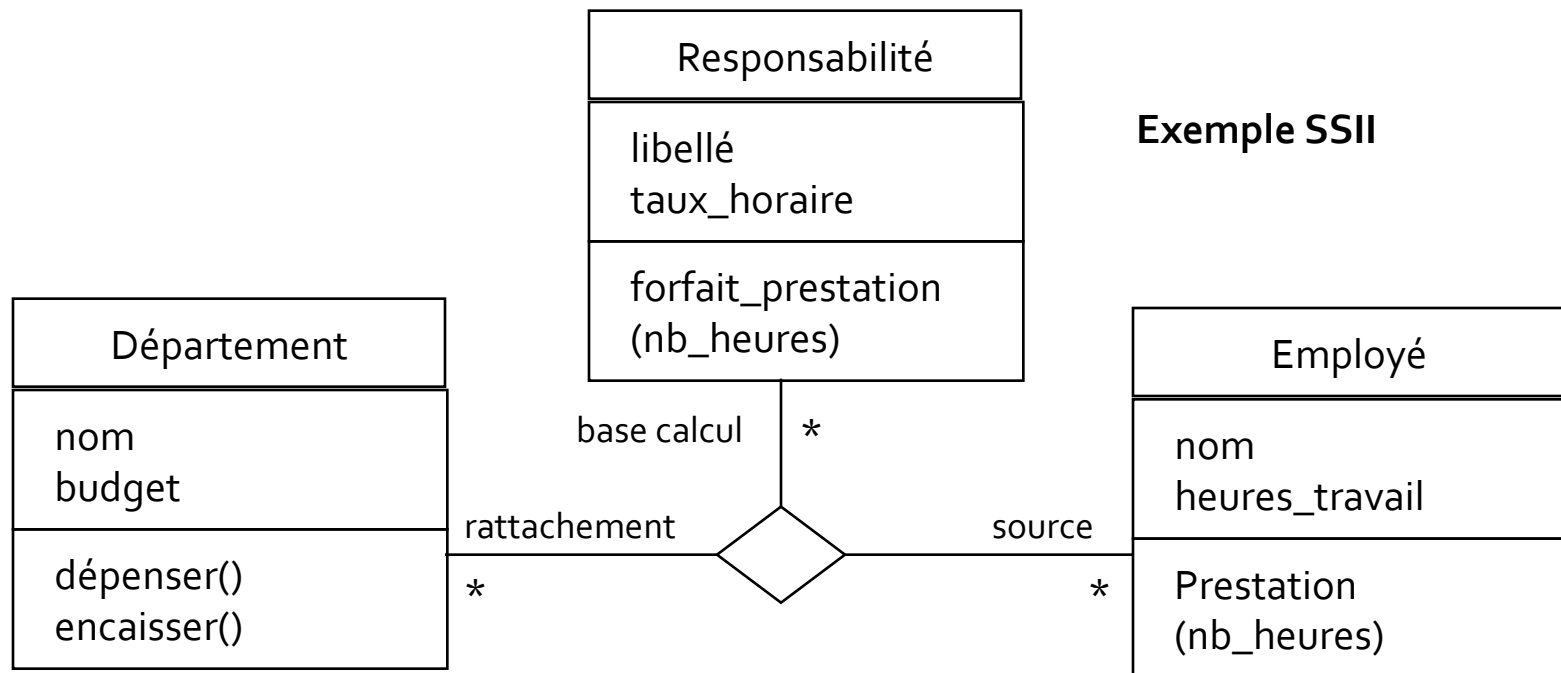
# Classes d'association

- Pour ajouter attributs et opérations à des associations
- Quelques indices pour l'utilisation
  - un attribut est lié à une association
  - la durée de vie des instances de la CA dépend de l'association
  - association N..N entre deux classes + informations liées à l'association



# Associations n-aires

- Groupe de liens entre au moins trois instances
- Instance de l'association = n-uplet des attributs des instances impliquées



# Composition, agrégation et association

---

## Quelques questions à se poser

- asymétrie et lien de subordination entre instances des deux classes (agrégation/composition) ou indépendance des objets (association) ?
- propagation d'opérations ou d'attributs du tout vers les parties ? (agrégation/composition)
- création et destruction des parties avec le tout ? (composition)

## Remarques importantes

- dans le doute, toujours utiliser une association : c'est la moins contrainte
- **pour certains experts, il faut oublier l'agrégation**  
agrégation = “placebo dénué de sens”

# Relations entre classes / liens entre objets

---

- Association

- les instances des classes sont liées
- possibilité de communication entre objets
- relation forte : composition



- Généralisation/spécialisation

- les instances de la sous-classe sont des instances de la super-classe (niveau conceptuel)
- héritage (niveau implémentation)



- Dépendance

- la modification d'une classe peut avoir des conséquences sur une autre



- Réalisation

- une classe réalise une interface



# Généralisation / spécialisation

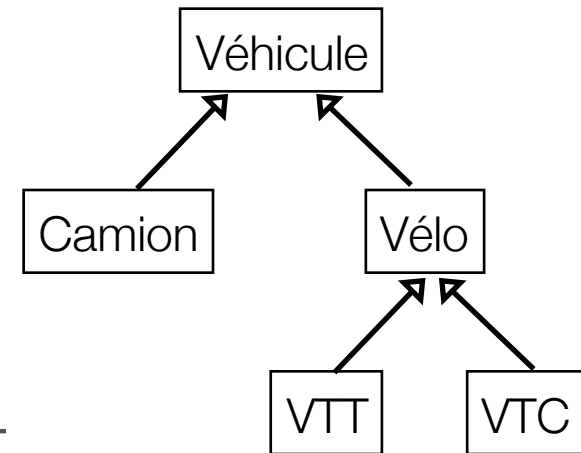
---

## Deux interprétations

- niveau conceptuel
  - organisation : un concept est plus général qu'un autre
- niveau implémentation
  - héritage des attributs et méthodes

## Pour mieux voir :

- Une *généralisation (ou agrégation)* est un ensemble de sous concepts qui collectivement forme un nouveau concept.
- Une *spécialisation (ou décomposition)* divise un concept en sous-concepts.



# Généralisation/spécialisation

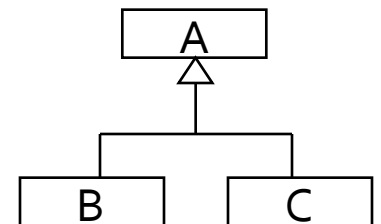
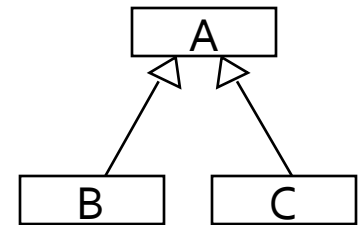
---

## Pour une bonne classification conceptuelle

- principe de substitution / conformité à la définition
  - toutes les propriétés de la classe parent doivent être valables pour les classes enfant
- « A est une sorte de B » (mieux que « A est un B »)
  - toutes les instances de la sous-classe sont des instances de la super-classe (définition ensembliste)

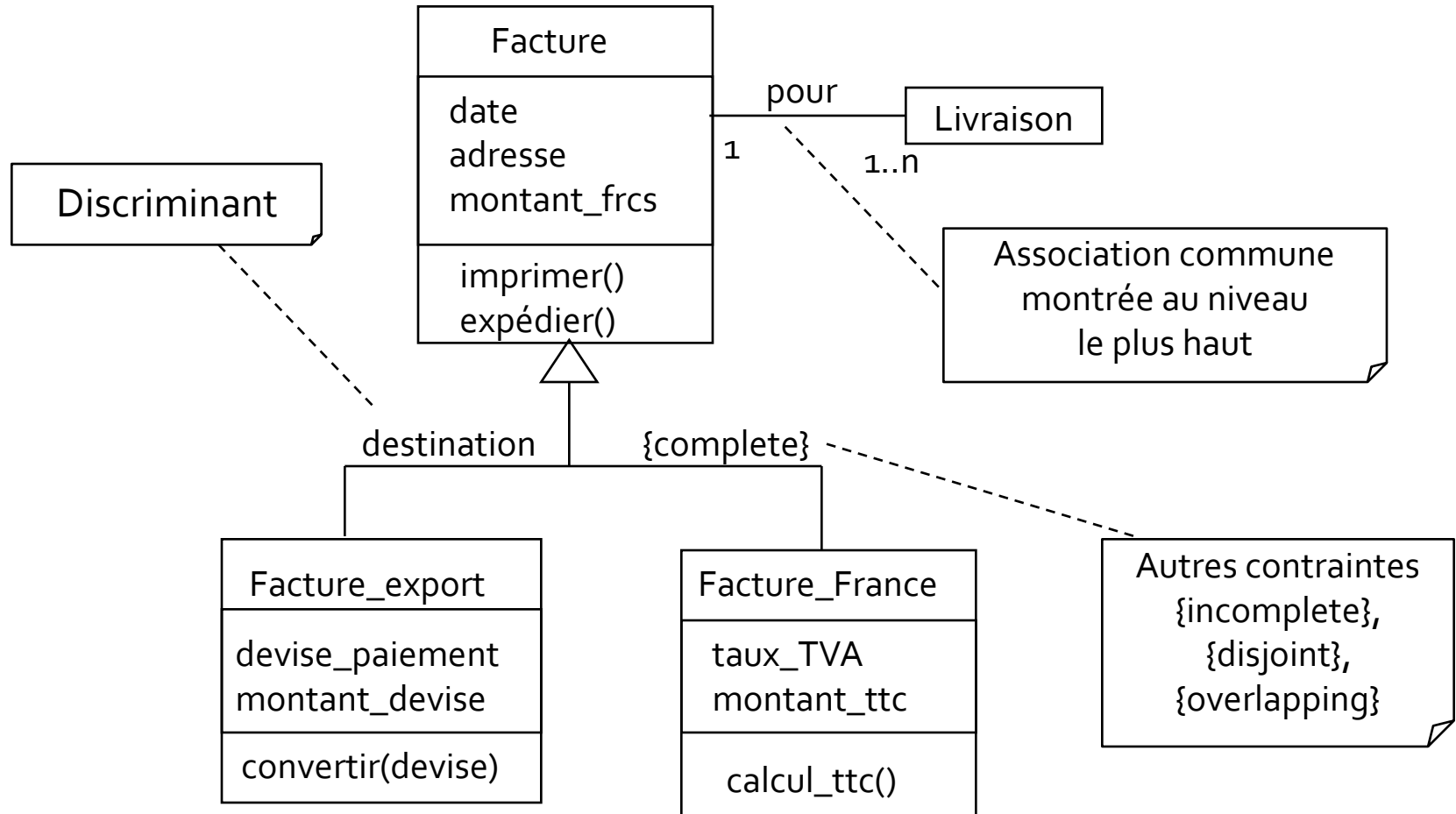
## Spécialisation

- relation inverse de la généralisation





# Hiérarchie des classes



# Conseils pour la classification conceptuelle 1/2

---

## Partitionner une classe en sous-classes

- la sous-classe a des attributs et/ou des associations supplémentaires pertinents
- par rapport à la super-classe ou à d'autres sous-classes, la sous-classe doit être gérée, manipulée, on doit agir sur elle ou elle doit réagir différemment, et cette distinction est pertinente
- le concept de la sous-classe représente une entité animée (humain, animal, robot) qui a un comportement différent de celui de la super-classe, et cette distinction est pertinente

# Conseils pour la classification conceptuelle 2/2

---

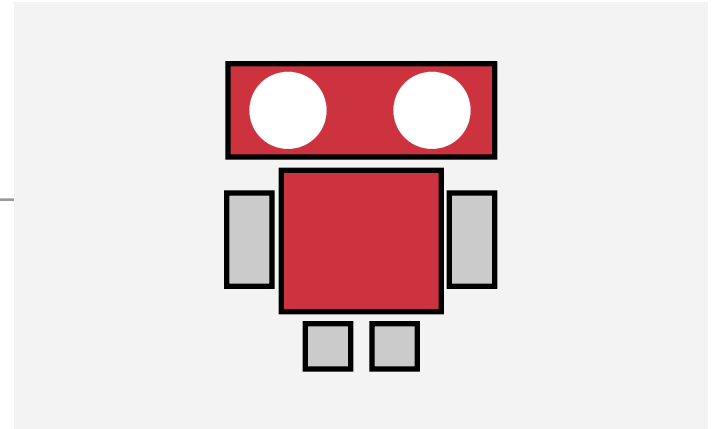
## Définir une super-classe

- les sous-classes sont conformes aux principes de substitution et « sorte-de »
- toutes les sous-classes ont au moins un même attribut et/ou une même association qui peut être extrait et factorisé dans la superclasse

# Exemple / Exercice

---

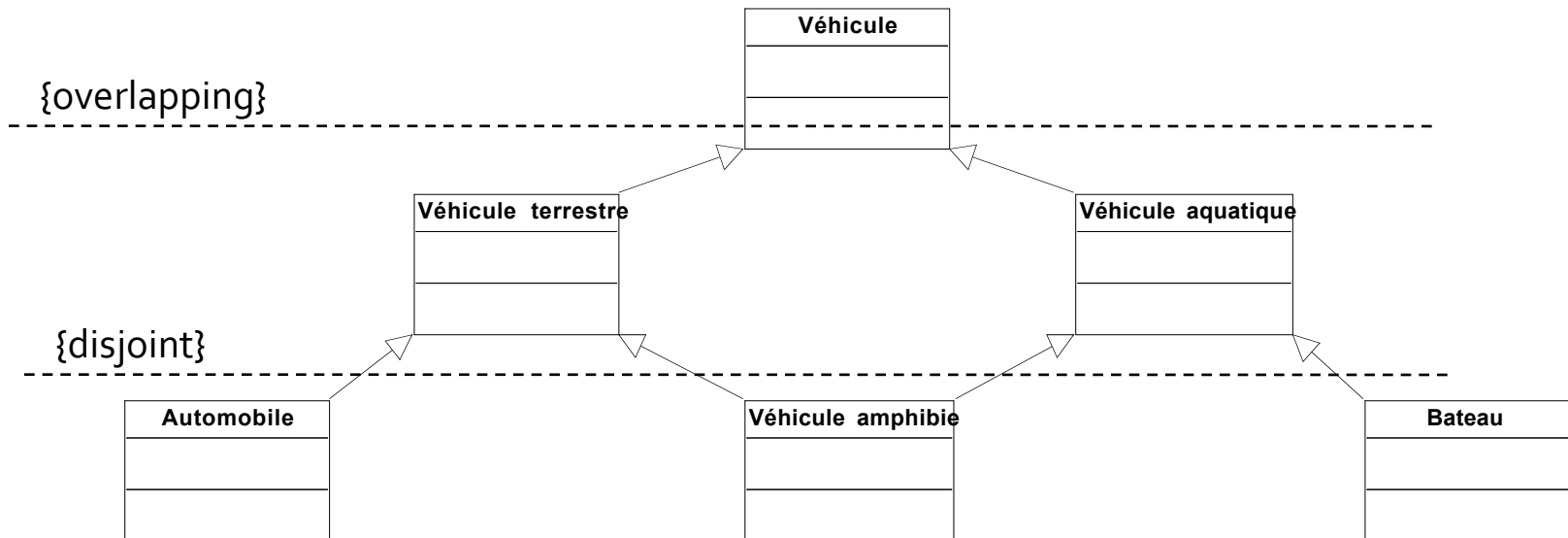
Application de l'héritage



# Généralisation multiple

---

- Autorisée en UML
- Attention aux conflits : il faut les résoudre
- Possibilité d'utiliser aussi délégations ou interfaces



# Relations entre classes / liens entre objets

---

- Association

- les instances des classes sont liées
- possibilité de communication entre objets
- relation forte : composition



- Généralisation/spécialisation

- les instances de la sous-classe sont des instances de la super-classe (niveau conceptuel)
- héritage (niveau implémentation)



- Dépendance

- la modification d'une classe peut avoir des conséquences sur une autre



- Réalisation

- une classe réalise une interface



# Relations de dépendance

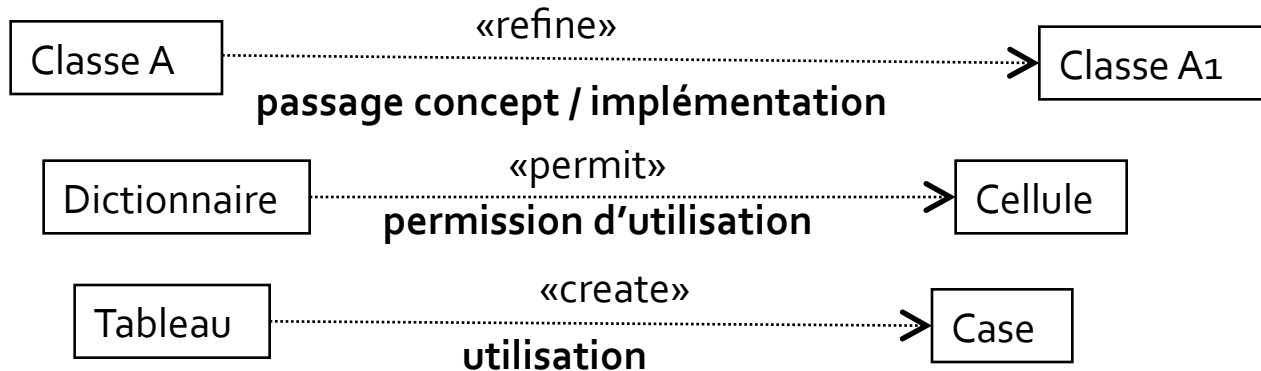
---

## 3 grands types

- abstraction : différents niveaux d'abstraction
  - ex. «refine», «trace», «derive»
- permission d'utilisation (cf. friend en C++)
  - ex. «permit»
- utilisation
  - ex. «use», «create», «call», «parameter»

## Conseil

- utiliser une dépendance pour tout ce qui n'est pas spécifié



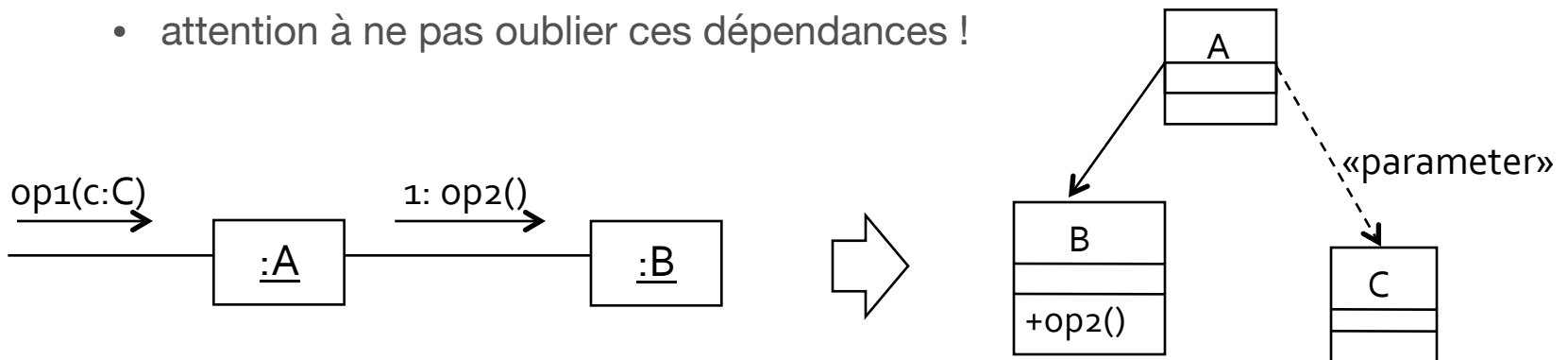
# Liens, visibilité et dépendances

---

Lien : possibilité d'envoyer un message d'un objet à un autre

## Deux types de liens

- Lien durable : visibilité d'attribut ou globale
  - se matérialise par une association entre classes
- Lien temporaire : visibilité paramètre ou locale
  - résulte d'une utilisation temporaire d'un objet par un autre
  - se matérialise par une dépendance entre classes
  - ex. passage de paramètre : « parameter », variable locale à une méthode : « local »
  - attention à ne pas oublier ces dépendances !





# Relations entre classes / liens entre objets

---

- Association

- les instances des classes sont liées
- possibilité de communication entre objets
- relation forte : composition



- Généralisation/spécialisation

- les instances de la sous-classe sont des instances de la super-classe (niveau conceptuel)
- héritage (niveau implémentation)



- Dépendance

- la modification d'une classe peut avoir des conséquences sur une autre



- Réalisation

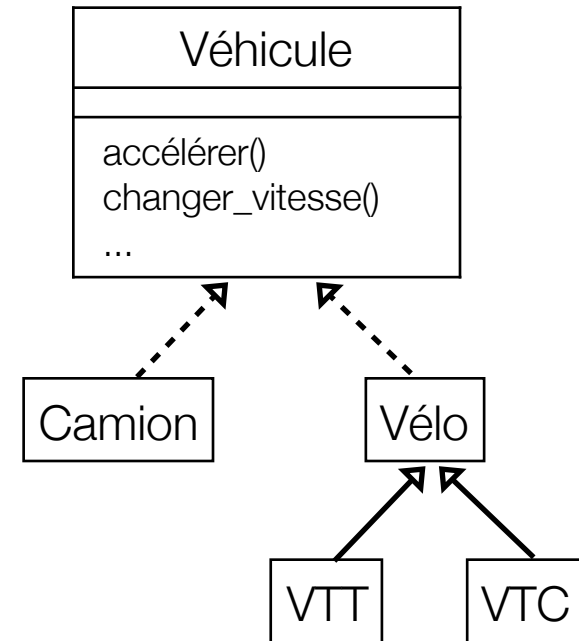
- une classe réalise une interface



# Réalisation, classe abstraite

Une classe abstraite est une classe possédant des méthodes abstraites.

- Une méthode abstraite
  - ne possède pas d'implémentation.
  - est utilisée par la classe elle-même ou une autre classe.
- Il n'est pas possible de créer et d'utiliser une instance directe d'une classe abstraite.
- Il faut utiliser une sous-classe qui implémente toutes les méthodes abstraites.

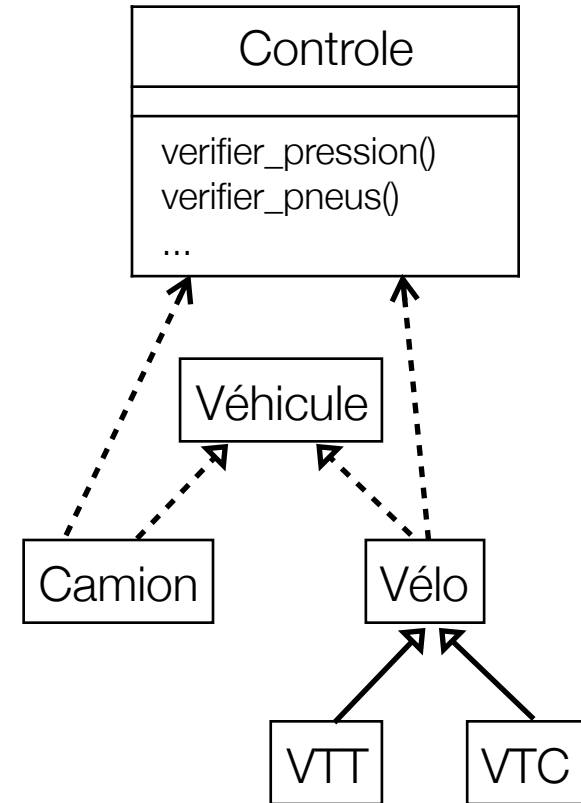


# Réalisation, interface

---

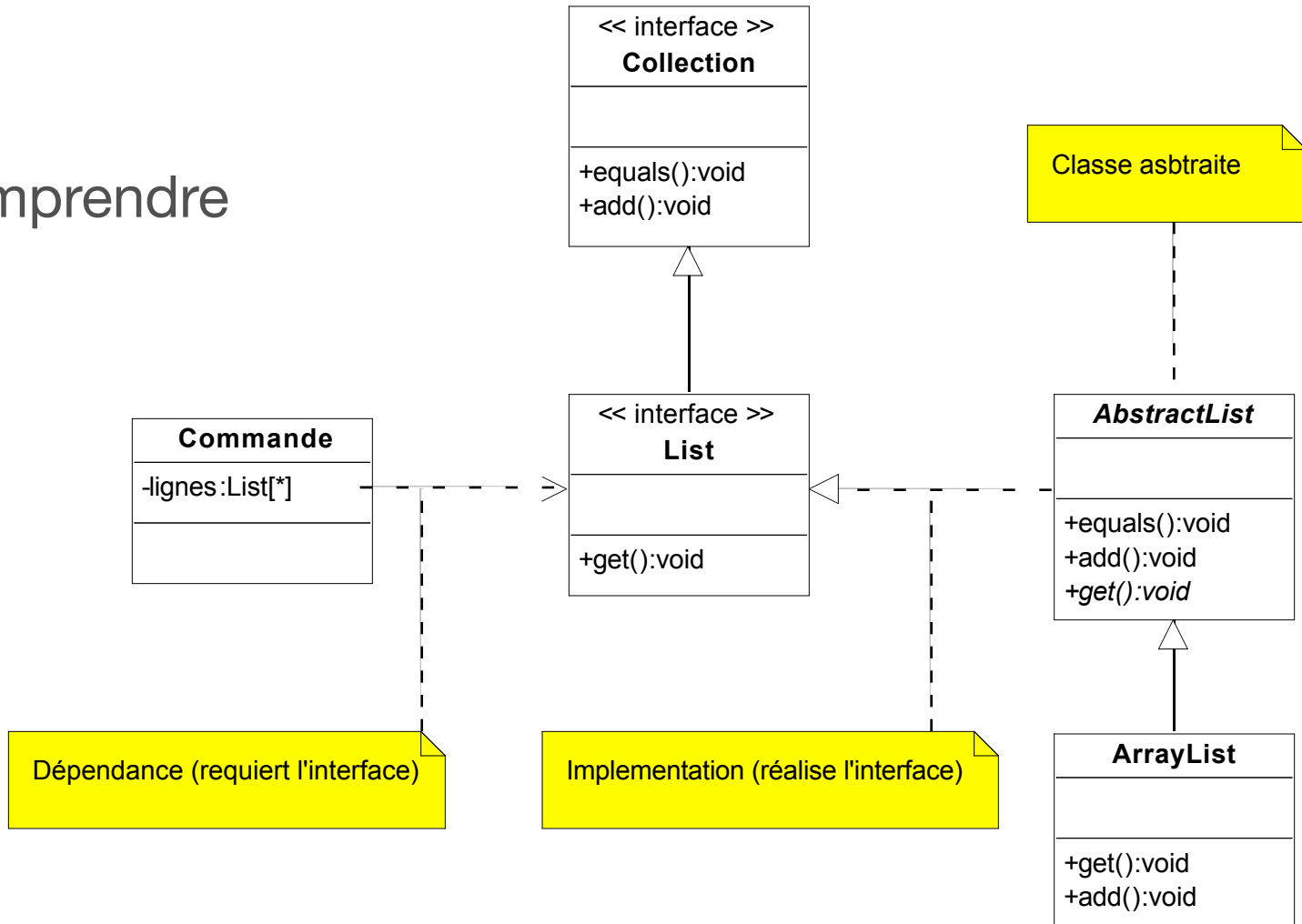
Une interface décrit un ensemble de méthodes qui doivent être définies par les classes qui l'implante.

Elle définit un contrat avec le monde extérieur qui doit être respecté



# Interfaces et classes abstraites

A comprendre



# Plan

---

## Modélisation objet et diagrammes UML statiques

- Une petite histoire de l'orienté objet
- Objets et classes
- Encapsulation
- Attributs et opérations
- Relations entre classes
- **Diagrammes UML statiques**

# Diagrammes UML statiques

---

- Diagrammes de classes
- Diagrammes d'objets
- Diagrammes de paquetages

# Diagrammes de classes

---

*Tout ce qui a été vu jusqu'à maintenant*

# Utilisation des diagrammes de classes

---

- Expression des besoins
  - modélisation du domaine
- Conception
  - spécification : gros grain
- Construction
  - implémentation : précis
  - rétro-ingénierie
- Les diagrammes de classes permettent de représenter toute modélisation en classes, que ce soit des classes implantées en machine ou non
- On peut modéliser n'importe quel domaine avec des classes



# Diagrammes UML statiques

---

- Diagrammes de classes
- Diagrammes d'objets
- Diagrammes de paquetages

# Diagrammes d'objets

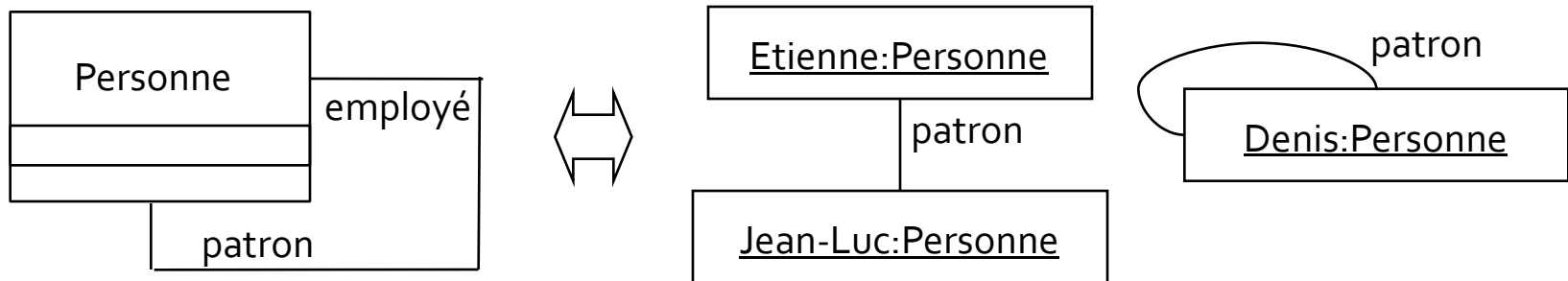
---

Pour représenter un instantané du système

- les objets et leurs liens
- objets = spécification d'instances

Quand les utiliser ?

- pour montrer un contexte
  - collaborations sans messages
- quand une structure complexe est trop difficile à comprendre avec un diagramme de classe
  - ex. : récursivité, associations multiples, etc.



# Objets

---

- Nom de l'objet souligné
- Objets anonymes ou non
- Objets classifiés ou non

Nom objet

Nom objet : classe

: classe

Rex

Rex : Chien

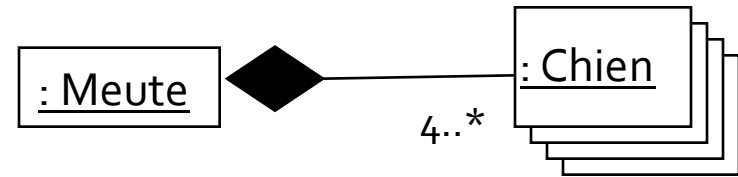
: Chien

# Objets dans une collection

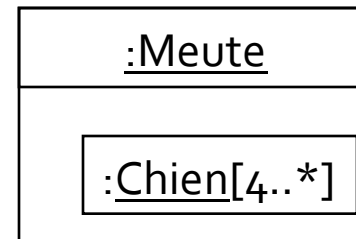
---

- Multi-objet (UML1)

- modéliser un jeu
- comme un objet unique avec des opérations sur le jeu
- comme jeu d'objets individuels avec leurs opérations



- Classe structurée (UML2)



- Utiles pour les diagrammes de communication pour s'adresser

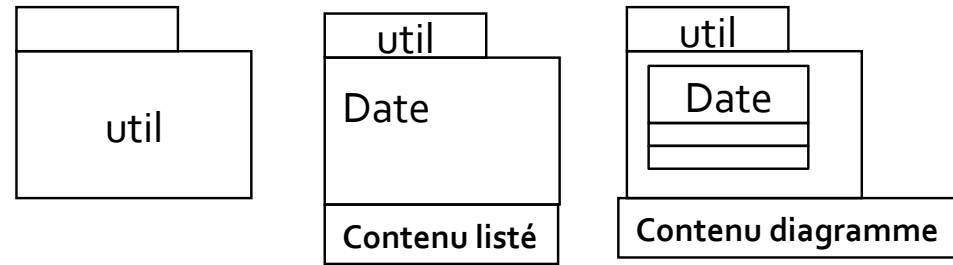
- à l'objet qui représente la collection (Meute)
- aux objets dans la collection (Chien)

# Diagrammes UML statiques

---

- Diagrammes de classes
- Diagrammes d'objets
- Diagrammes de paquetages

# Paquetage

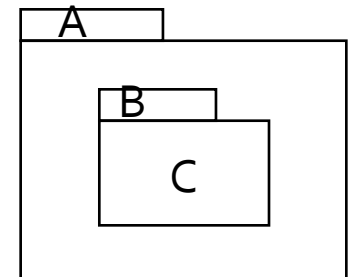


## Mécanisme général pour

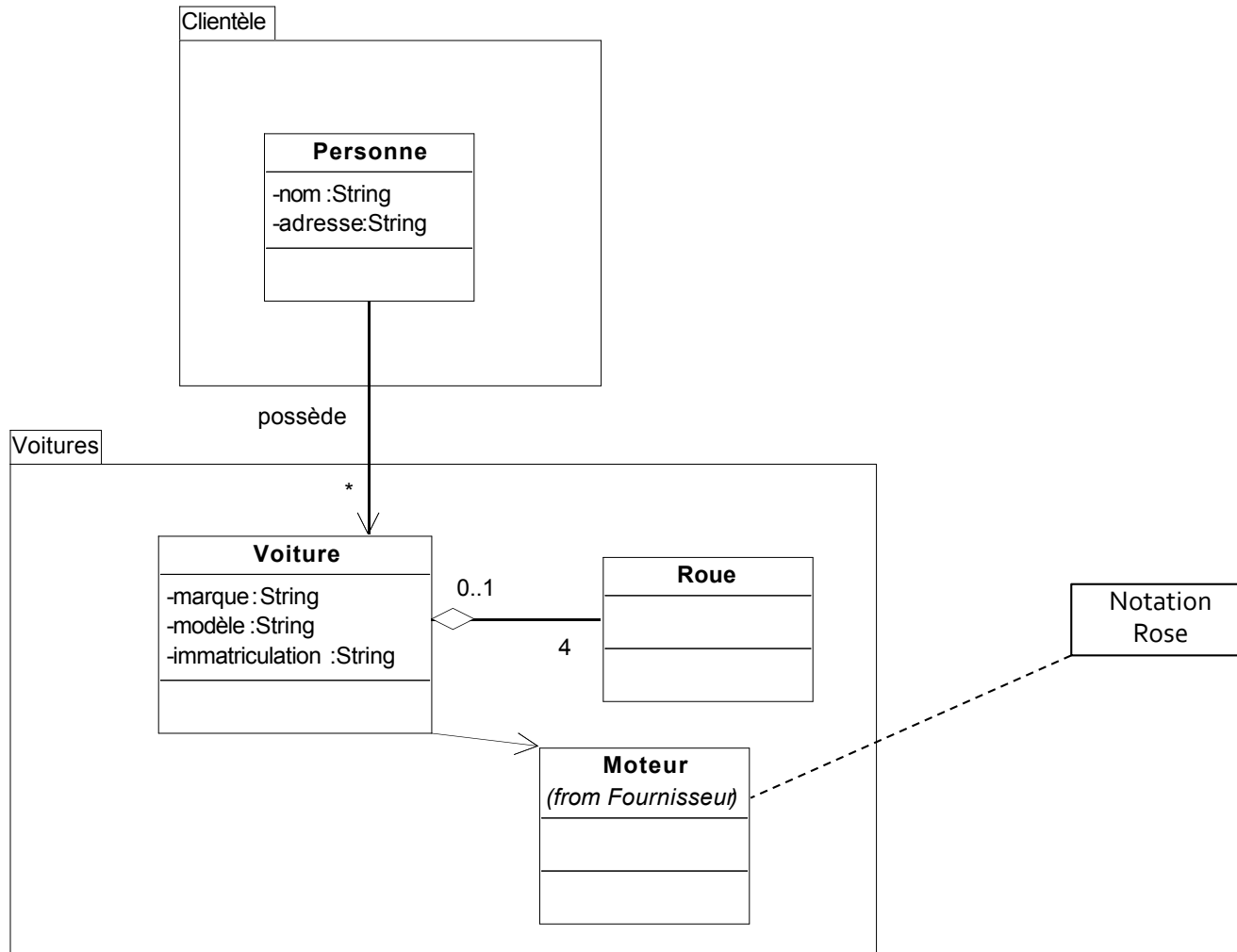
- organiser les éléments et les diagrammes du modèle, notamment les classes
  - partitionner, hiérarchiser
  - clarifier
- les nommer
  - un paquetage définit un espace de nom
  - deux éléments ne peuvent avoir le même nom dans un paquetage

## Un paquetage

- contient des éléments
- y compris d'autres paquetages : hiérarchie
- peut importer d'autres paquetages
- peut posséder des interfaces



# Diagramme de paquetage



# Dépendances entre paquetages

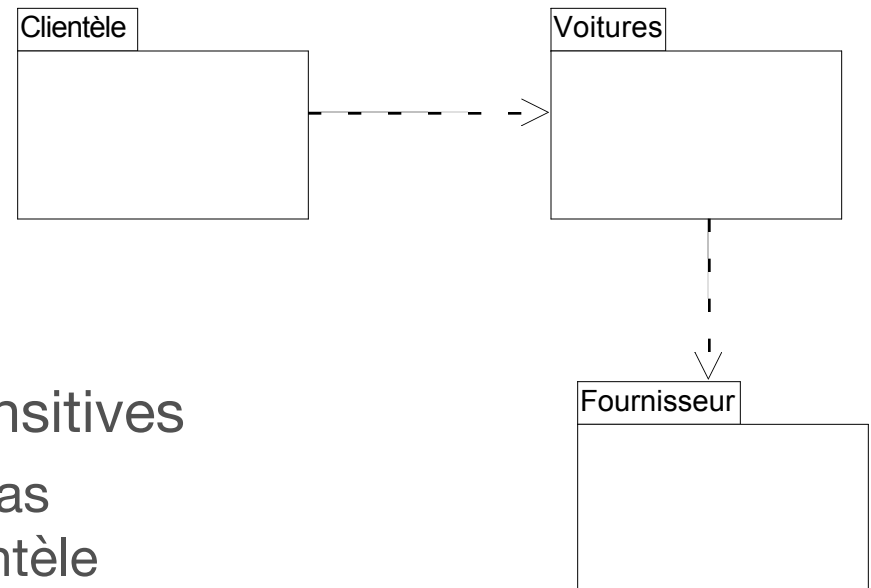
---

Découlent des dépendances entre éléments des paquetages

- notamment les classes

Les dépendances ne sont pas transitives

- modifier Fournisseur n'oblige pas obligatoirement à modifier Clientèle





# Utilisation des diagrammes de paquetages

---

## Organisation globale du modèle mis en place

- hiérarchies de paquetages contenant diagrammes et éléments

## Organisation des classes en paquetages pour

- contrôler la structure du système
  - comprendre et partager
  - obtenir une application plus évolutive et facile à maintenir
    - ne pas se faire déborder par les modifications
    - viser la généricité et la réutilisabilité des paquetages
- avoir une vue claire des flux de dépendances entre paquetages
  - il s'agit de les minimiser

# Principes du découpage en paquetage

---

Cohérence interne du paquetage : relations étroites entre classes

- fermeture commune
  - les classes changent pour des raisons similaires
- réutilisation commune
  - les classes doivent être réutilisées ensemble

Indépendance par rapport aux autres paquetages

Un paquetage d'analyse contient généralement moins de 10 classes

# Bien gérer les dépendances

---

Les minimiser pour maintenir un couplage faible (voir patterns)

- dépendances unidirectionnelles
  - cf. associations navigables
- pas de cycles de dépendances
  - ou au moins pas de cycles inter-couches
- stabilité des dépendances
  - plus il y a de dépendances entrantes, plus les interfaces de paquetage doivent être stables

# Paquetages : remarques variées

---

- Les paquetages peuvent être considérés comme
  - soit simples regroupements
  - soit des véritables sous-systèmes opérationnels
    - comportement + interfaces
- Paquetage vu de l'extérieur
  - classe publique gérant le comportement externe (cf. pattern Façade)
  - interfaces
- Pour un Paquetages utilisé partout (très stable)
  - mot-clé « global »
- Utilité pratique d'un paquetage Commun
  - regrouper les concepts largement partagés, ou épars
- Lien entre paquetages et couches (niveaux)
  - une couche est composée de paquetages

# Diagrammes UML statiques

---

- Diagrammes de classes
- Diagrammes d'objets
- Diagrammes de paquetages