

M1 info

GINF41B2 (Conception et Programmation Orientée Objet)

Cours #2

Classes, délégation et héritage

(aspects conceptuels)

Pierre Tchounikine

<http://membres-liglab.imag.fr/tchounikine/COO/Presentation.pdf>

<http://membres-liglab.imag.fr/tchounikine/COO/Ci.pdf>

(et remplacer le "i" de "Ci" par 1, 2, 3, etc.)

Plan

- Encapsulation
- Classes et objets
- Délégation
- Héritage

Encapsulation

Evolution des langages vers l'abstraction

- Principe :
 - ✗ permettre de travailler au niveau abstrait plutôt qu'au niveau de la mise en œuvre effective (de bas niveau) des concepts
- Exemples
 - ✗ implantation des types simples :
 - caractère / octets
 - booléen / « 0 » et « 1 » (!)
 - ✗ instructions et structures de contrôle
 - instructions / opérations sur des registres
 - Si-Alors-Sinon, Tant Que (etc.) / branchements (go to)
- Objectifs :
 - ✗ rapprocher conception et programmation
 - ✗ simplifier les choses pour le concepteur/programmeur
 - ✗ éviter les situations propices aux erreurs

Type et abstraction

- Type = {valeurs} + {opérations}
 - x {-32768 ... 32767} + {+, -, /, *}
 - x {vrai, faux} + {et, ou, non}
 - x type énuméré + {successeur, prédécesseur}
 - x etc.

- Abstraction et type
 - x définir des types non primitifs
 - x définir les opérations correspondantes

Exemple : « record » + procédures et fonctions associées en Pascal

problème : ce n'est qu'une **juxtaposition**



ne limite pas les opérations que l'on peut faire sur les données
(n'empêche pas de faire des bêtises)

Encapsulation

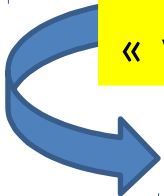
- Idée :
 - x cacher la réalisation effective
 - x empêcher d'autres manipulations que celles prévues
- Principe :
 - x lier les données et leurs opérations
 - x interdire tout autre accès aux données que par les opérations associées



« vue externe » = les services offerts aux utilisateurs de l'objet
« vue interne » = comment les services sont réalisés

- Avantages
 - x dissimulation de l'information
 - x notion d'interface = « contrat », spécification

Avantages de l'encapsulation



« vue externe » = les services offerts aux utilisateurs de l'objet
« vue interne » = comment les services sont réalisés

séparation du « quoi » et du « comment »

pour l'utilisateur du service

simplicité à comprendre

simplicité à utiliser

vision en terme
de services

garantie de
l'intégrité

baisse de la complexité

pour le concepteur du service

flexibilité (possibilité de faire évoluer
l'implantation sans risques, ou même
d'avoir plusieurs implantations
différentes)

meilleures chances de
réutilisabilité

Encapsulation et langages impératifs

L'exemple de ADA

partie publique de
la spécification
(le « contrat » du
composant)

```
generic
  type element is private;
package gest_pile is
  type pile(taille : positive) is private;
  procedure empiler (p : in out pile; e : in element);
  procedure depiler (p : in out pile; e : out element);
  function pile_vide(p : pile) return boolean;
  pilevide, pilepleine : exception;
```






partie privée de
la spécification
(aspects
implantation)

```
private
  type tableau is array (positive range<>) of element;
  type pile(taille : positive) is
    record
      table : tableau(1..taille);
      sommet : natural := 0;
    end record;
end gest_pile;
```

+ partie « corps » : le code des opérations (empiler, dépiler et pile-vide)
= une autre unité de compilation

Encapsulation et langages objets

La classe

nom de la classe
 attribut1 : type = initval  attribut2 : type = initval  attribut3 : type = initval
 opération 1()  opération 2()

Class Toto
+ attribute0 : int # attribute1 : int ~ attribute2 : int - attribute3 : int
+ operation0() : void + operation1() : void + operation2() : void

C1
<i>Attributes</i> private int ac1 private int ac2
<i>Operations</i> public C1() public int getAc1() public void setAc1(int val) public int getAc2() public void setAc2(int val)

```
public class C1 {
    private int ac1;
    private int ac2;

    public C1 () { }

    public int getAc1 () { }
    public void setAc1 (int val) { }
    public int getAc2 () { }
    public void setAc2 (int val) { }
}
```

Classes et objets

L'objet et la classe : définitions

- Un objet est une entité définie par
 - ✗ une identité (un nom)
 - ✗ un ensemble d'attributs qui caractérisent son état
 - ✗ un ensemble d'opérations (méthodes) qui définissent son comportement
- Un objet est une instance de classe
- Une classe est un type de données abstrait
 - ✗ caractérisé par des propriétés (attributs et méthodes) communes à des objets
 - ✗ permettant de créer des objets possédant ces propriétés

L'objet et la classe : utilisation

- La notion de classe permet de modéliser le monde ...

- ✗ identifier les objets, les définir, les typer
- ✗ regrouper les objets ayant des points communs
- ✗ établir les relations entre les objets différents
- ✗ etc.

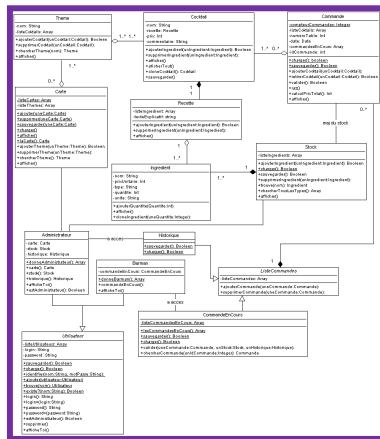
... et de créer des objets par **instanciation**

- Un objet

- ✗ propose des services (ses méthodes)
- ✗ exécute un service lorsqu'il reçoit un message (lorsque sa méthode est invoquée)

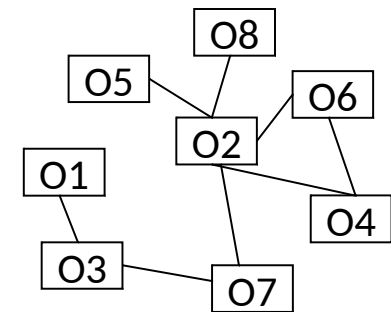
Niveaux modélisation et exécution

- Le travail de modélisation se fait autour de la notion de classe
- L'exécution du programme est réalisée par les interactions entre objets
 - ✗ création d'objets (dynamiquement)
 - ✗ interactions entre les objets (envois de messages)



classes

descriptions formelles d'objets ayant une sémantique et des caractéristiques communes



objets

entités discrètes (<identité, état, comportement>) d'un système en cours d'exécution

Les objets / classe

C1
<i>Attributes</i> private int ac1 private int ac2
<i>Operations</i> public C1() public int getAc1() public void setAc1(int val) public int getAc2() public void setAc2(int val)

- Les méthodes sont définies au niveau de la classe
 - elles existent en 1 exemplaire accessible à tous les objets
- Chaque objet a ses propres variables d'instances
 - chaque objet a ses valeurs
- Par ailleurs :
 - « static » en Java
 - ✗ on peut définir des « variables de classes » (« champs de classe », « champs statiques ») communes à toutes les instances
 - existent en 1 exemplaire accessible à tous les objets
 - ✗ on peut définir des « méthodes de classes », i.e., des méthodes qui ne portent pas sur un objet particulier
 - ✓ exemple : manipulation de variables de classes uniquement
 - ✓ autre exemple : main

Typologie des méthodes

C1
<i>Attributes</i> private int ac1 private int ac2
<i>Operations</i> public C1() public int getAc1() public void setAc1(int val) public int getAc2() public void setAc2(int val)

- Constructeurs : créent des objets
 - ✗ constructeur par défaut
 - ✗ constructeurs spécifiques (initialisation) ; leur usage est recommandé
- Accesseurs : fournissent des informations sur les objets
 - accès aux valeurs des champs
- Altérateurs : modifient les objets
 - modification des valeurs des champs

Accessibilité (bases)

C1
<i>Attributes</i> private int ac1 private int ac2
<i>Operations</i> public C1() public int getAc1() public void setAc1(int val) public int getAc2() public void setAc2(int val)

- Un champ peut être défini comme :
 - ✗ public = accessible partout où la classe est accessible
 - ✗ privé = accessible par les méthodes de l'objet (de la classe) uniquement



l'encapsulation n'est pas nécessairement réalisée !

- Une méthode peut être définie comme :
 - ✗ publique = accessible partout où la classe est accessible
 - ✗ privée = accessible au sein de la classe uniquement

Classe et encapsulation

- Le « contrat » défini par la classe (les services proposés), c'est :
 - ✗ les en-têtes des méthodes publiques
 - ✗ le comportement des méthodes
- Remarques 1 :
 - ✗ attention à la définition de la portée des attributs
 - ✗ attention à la définition de la portée des méthodes
 - ✗ attention aux altérateurs
- Remarques 2 :
 - ✗ accessibilité des champs/méthodes, en-têtes : vérification syntaxique
 - ✗ comportement des méthodes : ...

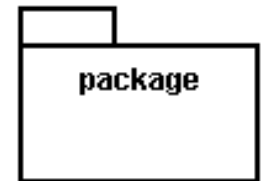
private !

→ intérêt de pouvoir réifier des contraintes explicites, cf. plus tard

pré-conditions, post-conditions, invariants

Paquetage et impact / accessibilité

- Élément de structuration classique des classes : le paquetage
 - x paquetage = regroupement logique de classes (≈ bibliothèque)



→ Différentes accessibilités des classes :

- x public = accessible à toutes les classes qui la référencent
- x paquetage = accessible aux classes du même paquetage seulement

→ Différentes accessibilités des champs et méthodes :

- x public = accessible partout où la classe est accessible
- x privé = accessible par les méthodes de l'objet (de la classe) uniquement
- x paquetage = accessible par toutes les classes du même paquetage

Utilisation de la notion de classe

- La notion de classe peut être utilisée pour représenter différents types de choses :
 - ✗ des éléments concrets (ex : un étudiant)
 - ✗ des éléments abstraits (ex : une commande)
 - ✗ des composants d'une application (ex : un bouton, une fenêtre)
 - ✗ des structures informatiques
 - une table d'index
 - une tâche
 - un traitement (une comparaison, un tri)
 - ...



notion de réification

(= transformation en « objet »)

Relations entre classes

- La COO/POO repose sur
 - × la définition de classes
 - × la définition des relations entre classes

les relations vont correspondre au niveau implantation/exécution à des structurations particulières et/ou à des « envois de messages »

- Il existe différents types de relations dont, en particulier :
 - × les relations d'**agrégation/composition** (notion de *délégation*)
 - × la relation d'**héritage**

Agrégation (et délégation)

Agrégation et délégation

- Point de vue « objet = structure »

- ✗ un objet est défini comme une agrégation d'autres objets
- ✗ un objet (la classe) est défini *à l'aide* d'autres objets (...)



vilain terme indéfini qui peut correspondre à différentes choses

- Point de vue « objet = service »

- ✗ un objet « utilise » un autre objet (→ un « client » et un « serveur »)
- ✗ un objet délègue à un autre objet une partie de ce qu'il devait faire

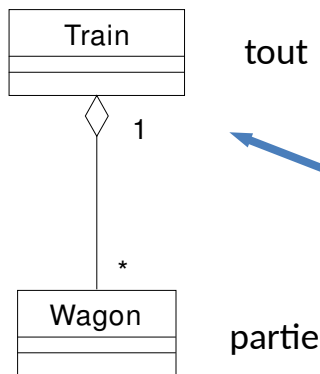
- Point de vue « type de relation entre les objets / classes » :

- ✗ relation de « tout » à « partie »

Agrégation vs. Composition

point de vue « conceptuel »

agrégation

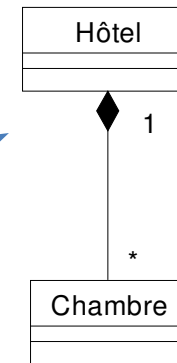


tout

partie

syntaxe graphique

composition



tout

agrégation « forte »
(« par valeur »)

partie

les durées de vies des objets ne sont pas liées

on peut « détruire » un train, les wagons
existeront toujours par ailleurs (la « partie »
survit à la destruction du « tout »)

les durées de vies des objets sont liées

si l'hôtel est détruit les chambres sont
détruites avec

*cf. approfondissements
plus loin dans le cours*

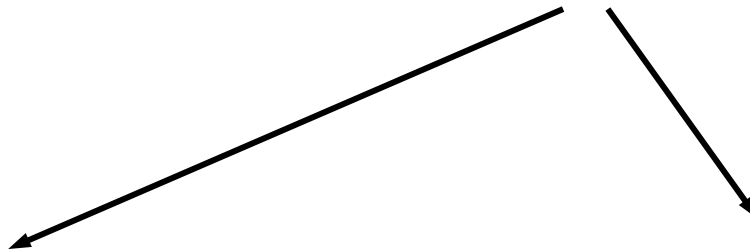
Remarques :

- la différence est un point de vue de modélisation → lié à l'intention
- la différence peut être « peu claire » (...)
- pas de consensus sur l'utilisation de ces 2 relations (cf. différents cours)

Agrégation vs. Composition

point de vue « structure »

- Point de vue « objet = structure »
 - un objet (→ la classe) est définie **à l'aide** d'autres objets



**l'objet O1 de la Classe 1 contient
une référence à un objet O2 de la Classe 2**

**l'objet O1 de la Classe 1 contient
un objet O2 de la Classe 2**

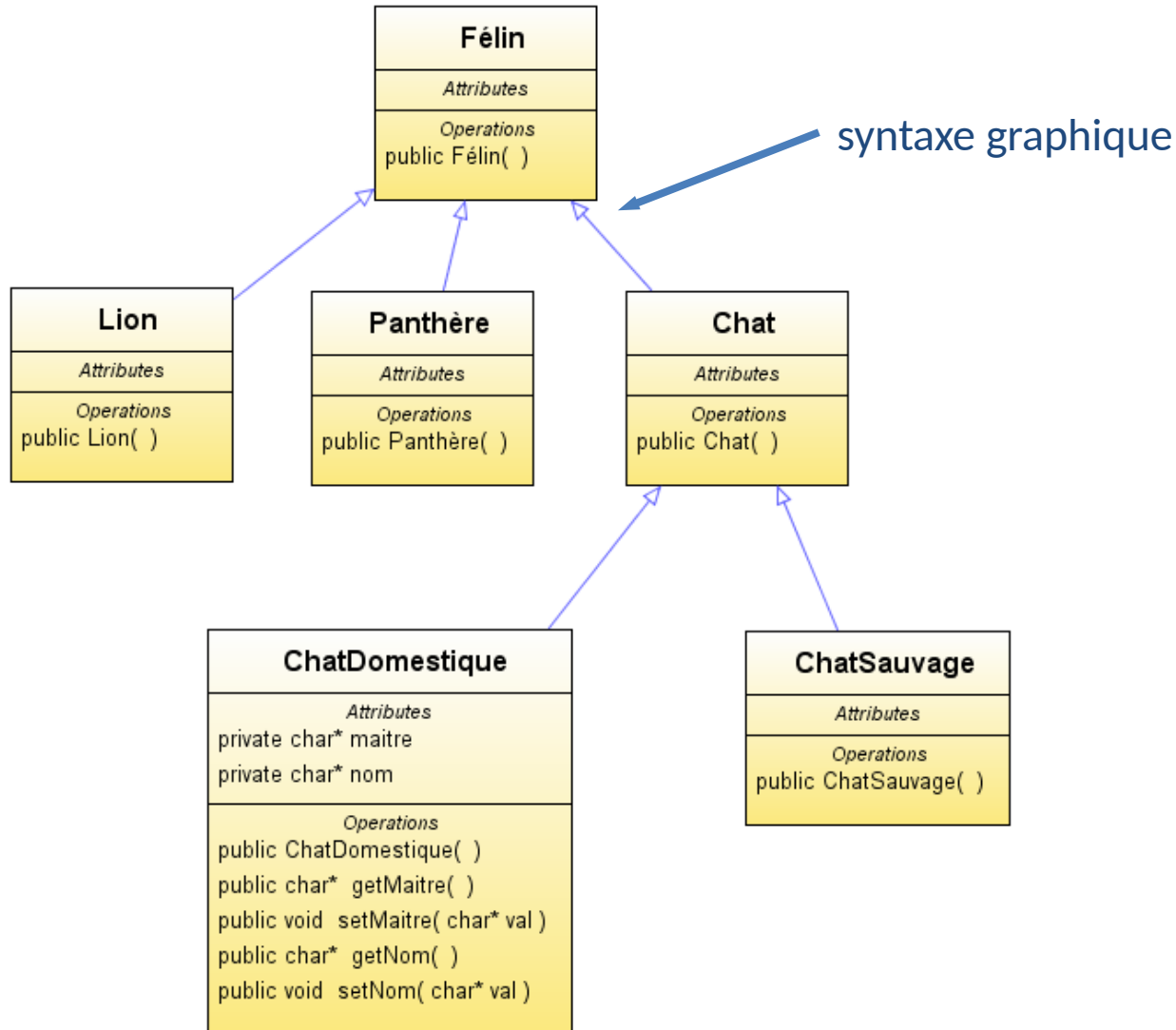
O2 n'« appartient » pas à O1
→ il peut être partagé

O2 « appartient » à O1

*cf. approfondissements
plus loin dans le cours*

Héritage

Héritage : exemple



Héritage : notions sous-jacentes

- Classification

- × identification d'objets ayant des caractéristiques communes



notion de type

- Spécialisation, généralisation, différenciation

- × définition d'objets ayant des caractéristiques plus spécifiques



un Machin est un Truc
et un Machin est un Truc particulier
et cette particularité mérite d'être dénotée

- Principe d'économie

- × définir, représenter, opérationnaliser une seule fois les choses



mise en commun ; transmission

Héritage en COO/POO

- Principe :

- x définir une nouvelle classe à partir de la définition d'une classe existante
- x la classe « fille » est une spécialisation de la classe « mère »

relation « est un »

- Effets :

- x les objets de la classe fille sont aussi des objets de la classe mère
- x les objets de la classe fille héritent des caractéristiques (attributs, méthodes) de leur classe mère

- Remarque :

- x la relation « est-un » est transitive (mais pas bijective !)



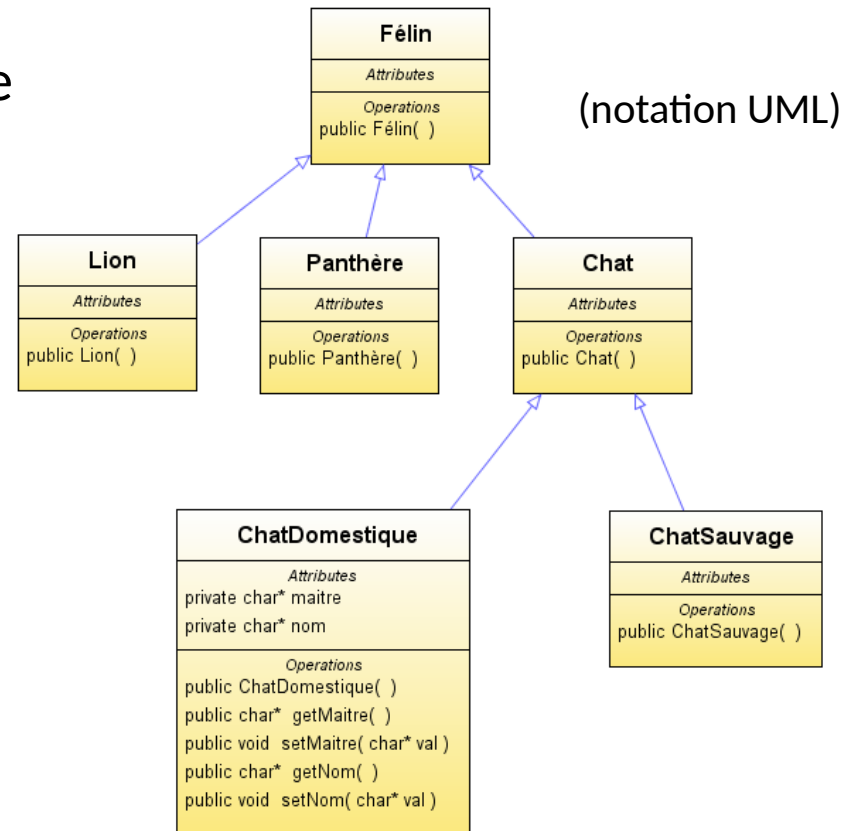
hiérarchies de classes

Héritage en COO/POO

- La relation « est-un » est transitive



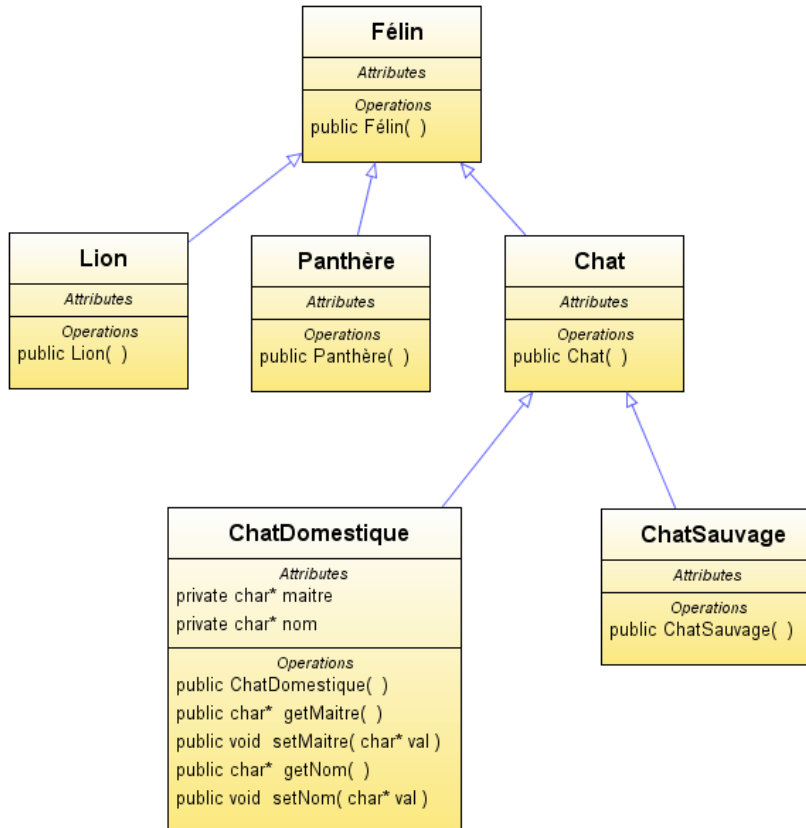
hiérarchies de classes



pas de limitation en profondeur a priori

dans certains langages (dont Java) on peut indiquer qu'une classe ne peut pas être spécialisée
(Java : mot clé « `final` »)

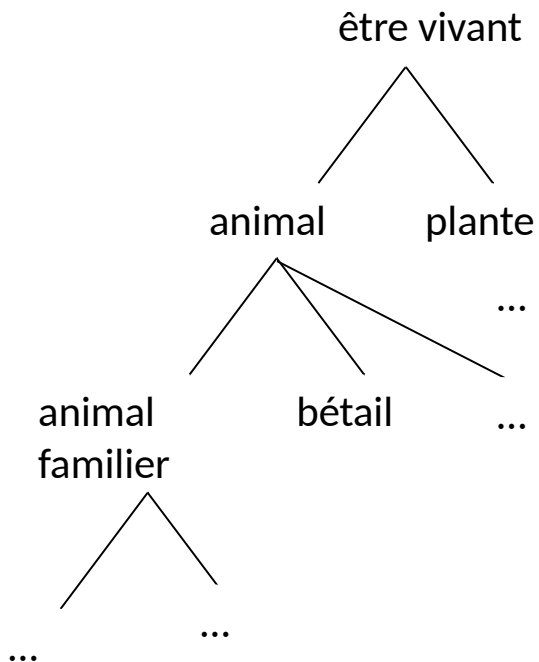
Héritage en Java



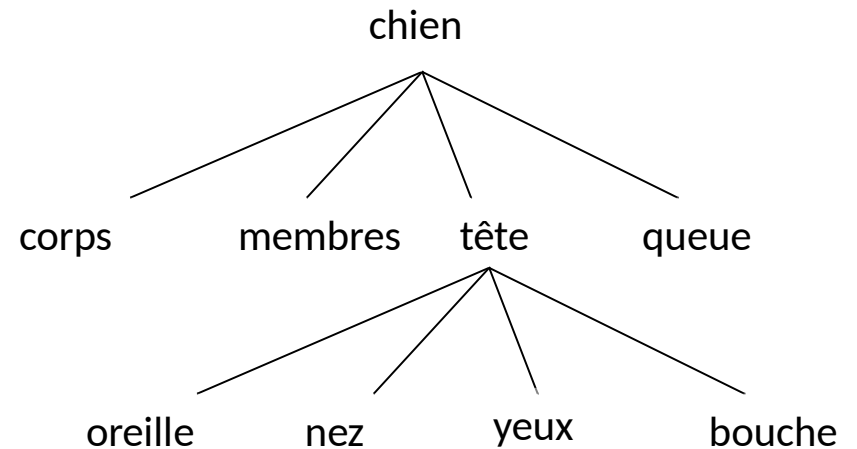
```
public class Chat extends Félin {  
...  
}
```

```
public class ChatDomestique extends Chat {  
...  
}
```

Héritage et agrégation



relation « est un »



relation « est une partie de »

Intérêts de la notion d'héritage

- Faciliter la modélisation (\neq niveaux)
- Concentrer les propriétés (**factorisation**) (...)
- Éviter la duplication
- Réutiliser et encourager la réutilisation par
 - × utilisation directe
 - × raffinement (**extension, spécialisation**) (...)

mais il n'y a pas que des avantages, par exemple les propriétés d'un objet peuvent être plus compliquées à comprendre

(différents niveaux, différents fichiers \rightarrow rôle des IDE)

Factorisation

- Une classe hérite de sa classe mère :
 - ✗ les objets de la classe fille ont tous les attributs de la classe mère
 - ✗ les objets de la classe fille ont toutes les méthodes de la classe mère

**tout ce qui est propre aux Félin est défini dans la classe Félin et bénéficie aux sous-classes Lion, Panthère et Chat
(et donc, par transitivité, aux ChatDomestique et aux ChatSauvage)**



- ✓ économie de définition / code
- ✓ modifications plus faciles

Remarque : tous les objets ont un comportement de base identique car ils héritent tous (ils sont tous une dérivation) de la classe `object`

Extension

- Une classe fille peut étendre sa classe mère :
 - ✗ les objets de la classe fille peuvent avoir des attributs spécifiques supplémentaires de ceux de la mère
 - ✗ les objets de la classe fille peuvent avoir des comportements spécifiques supplémentaires de ceux de la mère

un ChatDomestique a toutes les propriétés et tous les comportements des Félin et il a en plus un nom, un maitre et des « comportements » (méthodes) supplémentaires : demanderDesCaresses, etc.



- ✓ économie de définition / code
- ✓ modifications plus faciles

Spécialisation

- Une classe fille peut spécialiser sa classe mère, i.e., redéfinir les comportements de sa classe mère :
 - × les méthodes de la classe fille peuvent définir une interprétation propre à la sous-classe des méthodes de la classe mère

Supposons que la classe Félin définisse une méthode « manger »

La classe ChatDomestique peut redéfinir localement la méthode « manger » pour représenter le comportement particulier des ChatDomestique pour « manger »

notions de
redéfinition et
de surcharge



- ✓ économie de définition / code
- ✓ modifications plus faciles

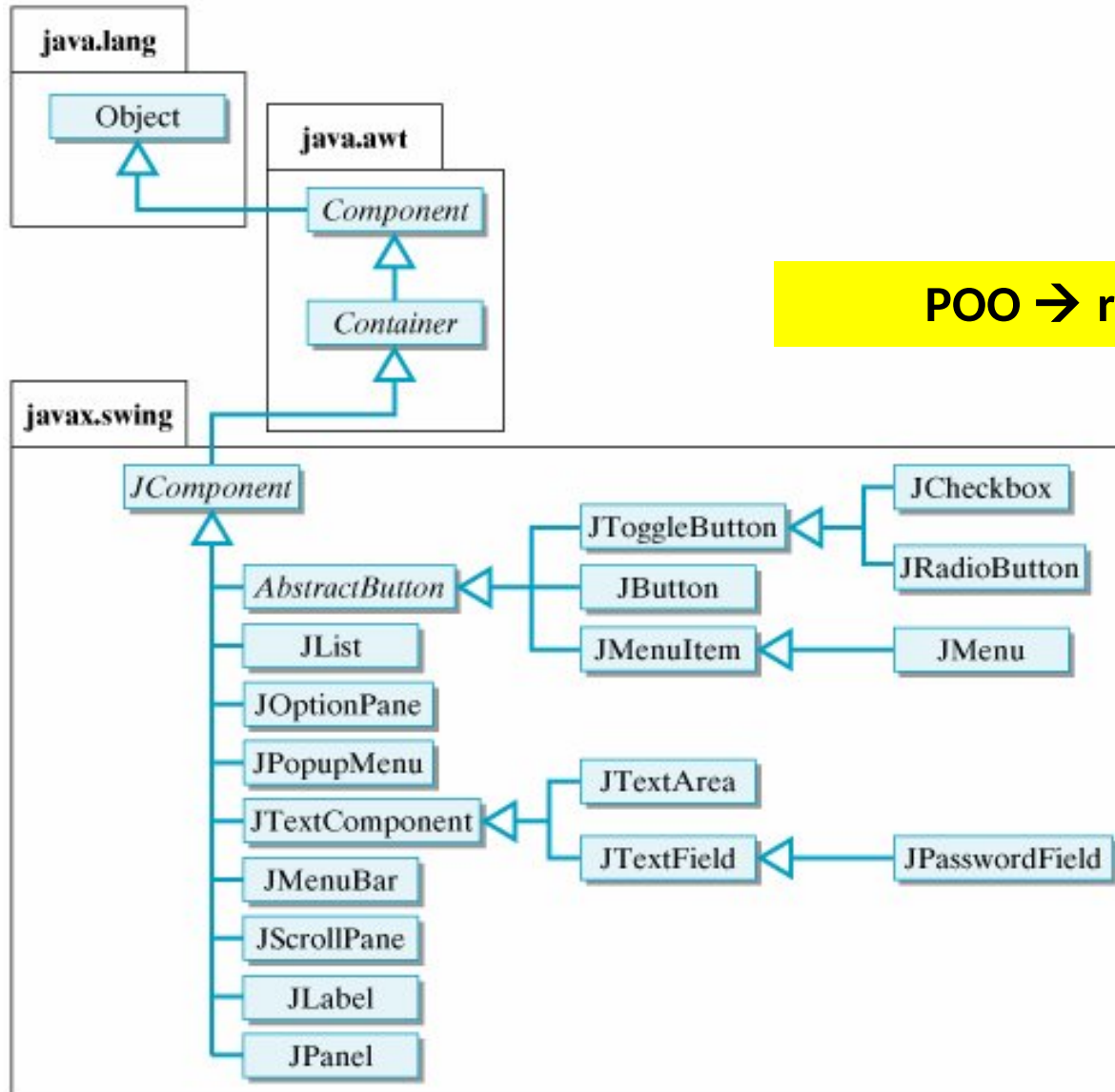
surcharge (masquage) d'attributs : à éviter !

Impact héritage / accessibilité

Modificateur	Classe	Sous-Classes	Classes du même paquetage	Reste du monde
public (+)	oui	oui	oui	oui
protégé (#)	oui	oui	oui	non
privé (-)	oui	non	non	non
paquetage ou « friendly » ()	oui		oui	non

« protégé » : généralement déconseillé, en particulier pour les attributs
(éventuellement, classes d'implantation nécessaires aux sous-classes)

Héritage et réutilisation

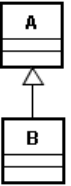


POO → réutilisation !

Héritage : différents points de vue

taxonomie de B. Meyer, approfondi plus tard

B hérite de A



Point de vue	Interprétation	Notion sous jacente
Conceptuel	B est plus spécifique que A, B est un A plus spécifique	Classification
Ensembliste	Les éléments de B sont aussi des A, B est inclus dans A	Type
Logique	Si i est un B, i est un A	Polymorphisme
Comportemental	Les méthodes applicables aux A le sont aussi aux B, les B savent faire ce que savent faire les A	Classification
Structurel	Les attributs de A font partie des attributs de B, les instances de B ont les attributs de A	Classification