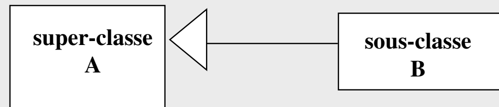
Subsomption

UML ; l 'héritage

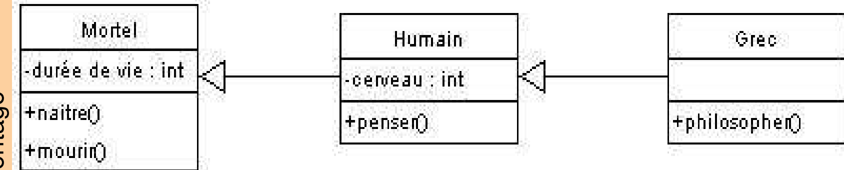


- La classe-concept : Subsomption
  - Le concept A «subsume» celui de B
  - A englobe B
  - B est un cas particulier de A
  - **Un B est un A**

Subsomption

UML ; l 'héritage

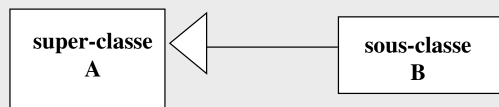
- Exemple

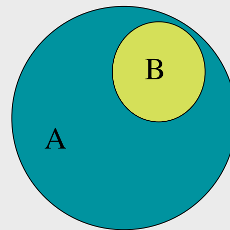


- «mortel» subsume «humain» qui subsume «grec»
- « humain » est plus général que « grec » mais n 'est qu 'un cas particulier de « mortel »

Inclusion

UML ; l 'héritage

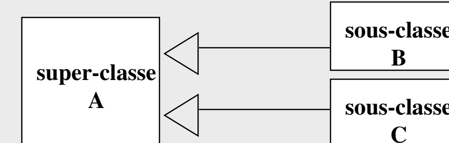




- La classe-ensemble : Inclusion
  - La classe A inclut la classe B
  - B est un sous-ensemble de A
  - Les instances de B sont aussi des instances de A
  - Les instances de A peuvent ou non être des instances de B

Sous-typage

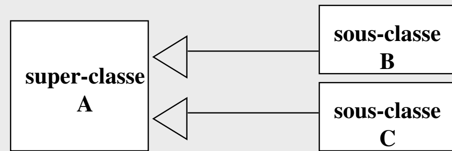
UML ; l 'héritage



- La classe-type abstrait : Sous-typage
  - L'ensemble des propriétés accessibles par la classe A est inclus dans l'ensemble des propriétés accessibles par la classe B
  - Les sous-classes B et C héritent des propriétés de la super-classe A et peuvent en posséder des spécifiques.

Sous-typage

UML : l 'héritage



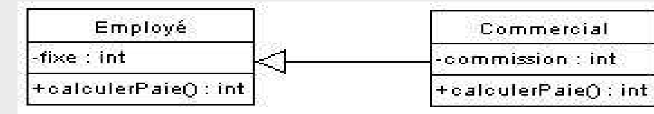
- La classe A factorise des propriétés communes aux classes B et C
- La sous-classe ne peut pas «oublier» des propriétés de sa super-classe mais des opérations héritées peuvent être redéfinies (polymorphisme).
- Dans Poséidon, une opération redéfinie est simplement réécrite
- On évitera de « redéfinir » les attributs (masquage)

Sous-typage

UML : l 'héritage

➔ Exemple 1 : Tout humain possède comme propriétés un attribut «cerveau» et une opération penser(); donc les grecs aussi et donc Socrate aussi

➔ Exemple 2 : «Employé» possède l'opération «calculerPaie», les commerciaux sont des employés et héritent donc de l'opération mais elle est redéfinie pour tenir compte du mode de calcul propre aux commerciaux (commission)



Règles

UML : l 'héritage

➢ Règles

- Lors de la démarche d'analyse, une généralisation doit satisfaire aux trois critères (subsomption, inclusion, sous-typage) ce qui pose quelquefois problème car il peut y avoir contradiction entre eux :
  - ➔ Quand il y a sous-typage et pas subsomption, on parle d ' « héritage par construction » : C 'est le défaut qui consiste à dériver une classe A déjà écrite simplement parce qu 'elle possède des propriétés (méthodes ou attributs) que je veux réutiliser dans une classe B alors que les concepts ne sont pas dépendants, je ne peux pas dire « un B est un A »
  - ➔ Quand il y a subsomption mais pas sous-typage, c 'est le cas des exceptions, un concept B est bien un A mais une (ou plus) des propriétés de A n 'est pas pertinente pour B

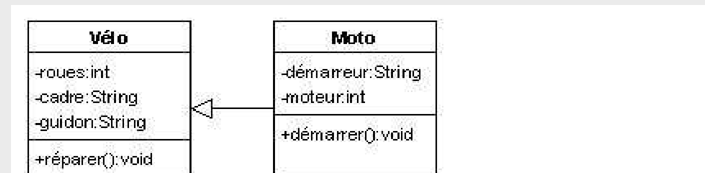
    Les mammifères sont vivipares, l 'ornithorynque est un mammifère mais il est ovipare

Règles

UML : l 'héritage

- Exemple 1 : Sous-typage mais pas subsomption : « héritage par construction »

➔«Moto» a toutes les propriétés de «Vélo» (roues, cadre, guidon) plus d'autres (moteur, démarrer...) donc je pourrais avoir envie de créer un héritage (Moto hérite de Vélo) pour éviter de réécrire les attributs et les méthodes communes mais une moto **n'est pas** un vélo pour les utilisateurs.



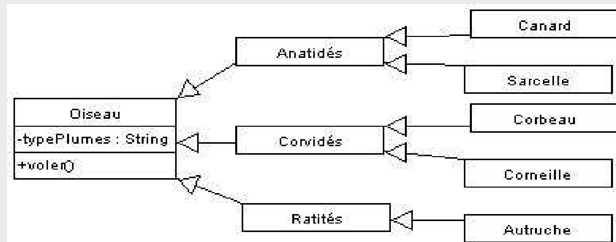
Règles

- Exemple 2 : Subsomption mais pas sous-typage : « les exceptions »

➔Une autruche est un oiseau (au sens commun) mais n'en possède pas toutes les propriétés : opération «voler()».

➔Donc elle n'est pas un oiseau au sens formel

On pourrait penser à redéfinir voler comme opération vide mais voir diapos suivantes



UML ; l 'héritage

Principes

➢ Principe fondamental :

Lors de la démarche d'analyse il faut s'en tenir à l'héritage par classification (vérifiant les 3 critères précédents) et éviter l'héritage par construction et les exceptions.

➢ Principe de substitution (B. Liskov)

« Il doit être possible de substituer n'importe quelle instance d'une sous-classe à n'importe quelle instance d'une super-classe sans que la sémantique du programme écrit dans les termes de la super-classe n'en soit affectée »

UML ; l 'héritage

Exemples

- Exemple 1 : Que va t-il arriver à l 'autruche dans ce programme ?

```
pour tout oiseau o faire :
{o.ouvrirCageAu20étage() ;
o.voler()}
```

- Exemple 2 : Où la Moto qui, par paresse, avait hérité de Vélo se retrouve en infraction :

```
pour tout Velo v faire {v.emprunterPisteCyclable()}
```

- Sujet de réflexion : Rectangle et carré qui hérite de qui ?

➔Le mathématicien : *« Un carré est un cas particulier de rectangle donc Carré hérite de Rectangle»*

➔Le développeur objet : *« Non, les propriétés longueur et largeur de Rectangle ne sont pas pertinentes pour un carré, Rectangle hérite de Carré car il a plus de propriétés »*

UML ; l 'héritage

Principes

➢ Principe de non-mutabilité

- L'arbre d'héritage doit représenter une classification stable : Les objets ne peuvent pas changer de classe dans le cadre de l'application

Ex :	Personne/Homme/Femme
	ObjetEmpruntable/Livre/CD
Contre-Ex :	Personne/Client/Prospect
	Etudiant/Mineur/Majeur
ou même	Enseignant/ChefDeDepartement

NB : Si, malgré tout, on désire rendre un objet mutable, il faut lui appliquer le Design Pattern «mutable»

cf « Design patterns » de Gamma et al.

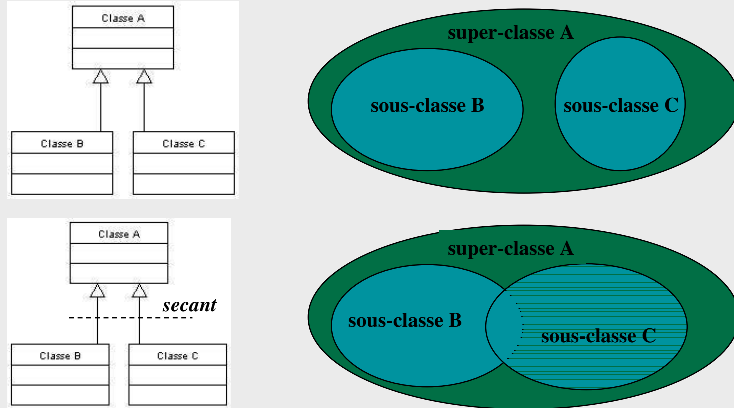
UML ; l 'héritage

Cas particuliers

UML ; l 'héritage

➢ Héritage exclusif

En UML, sans précision particulière, l'héritage est exclusif ; les sous-classes sont disjointes ; s'il ne l'est pas il faut préciser <<secant>>

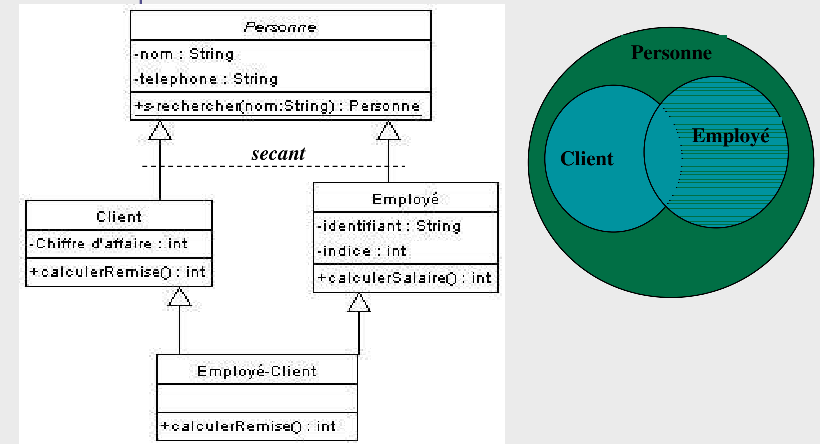


Cas particuliers

UML ; l 'héritage

➢ Héritage multiple

- L'héritage non exclusif peut donner lieu à l'héritage multiple



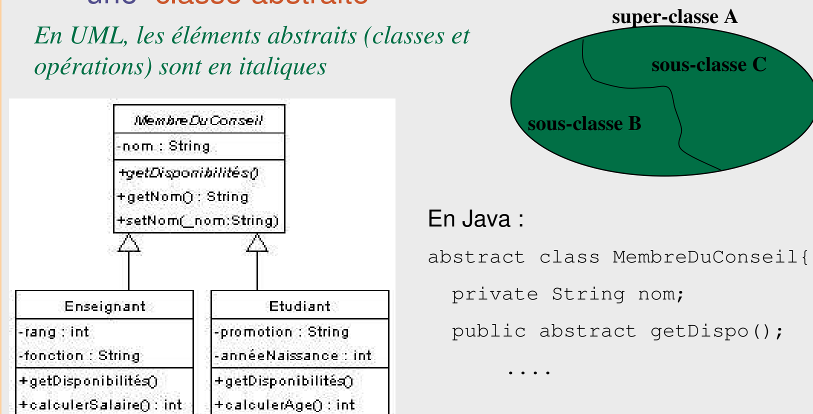
Cas particuliers

UML ; l 'héritage

➢ Classe abstraite

- Lorsqu'un héritage fait intervenir une notion de totalité, la super-classe n'est jamais instanciée, c'est une classe abstraite

En UML, les éléments abstraits (classes et opérations) sont en italiques



En Java :

```
abstract class MembreDuConseil{
  private String nom;
  public abstract getDispo();
     ....
```


Cas particuliers

UML ; l 'héritage

➢ Quelques rappels Java

- Une classe abstraite n'est jamais instanciée (directement)
- Une classe abstraite peut contenir des méthodes abstraites et des méthodes concrètes
- Les méthodes abstraites doivent être redéfinies dans les sous-classes
- Une classe qui contient au moins une méthode abstraite doit être déclarée abstraite
- Une classe abstraite peut définir un constructeur qui n 'est jamais déclenché directement mais qui est hérité par ses sous-classes (et souvent surchargé)

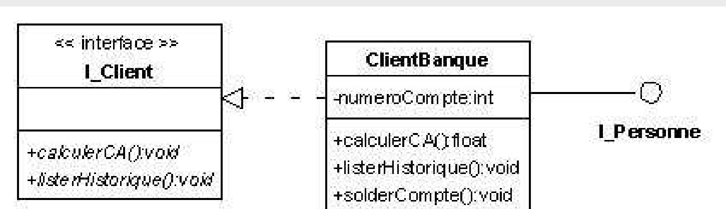
Interfaces

➢ Interface

Ensemble d'opérations abstraites utilisées pour décrire un contrat (une vue) d'une classe sans en préciser la réalisation

Deux formalismes UML sont possibles :

- ➔ **Forme détaillée : Comme une classe mais avec le stereotype <<Interface>>**
- ➔ **Forme condensée : représentée par un cercle avec son nom**



La classe «ClientBanque» implémente les interfaces «I_Client» et «I_Personne»

Interfaces en Java

➢ En Java

- Une interface est une classe qui ne contient que des méthodes abstraites et des constantes `(static final)`
- Java n'autorise que l'héritage simple de classes. Il remplace l'héritage multiple par la notion d'interface.
- Permet de spécifier des protocoles génériques, des comportements abstraits.
- Une classe peut implémenter une ou plusieurs interfaces

```
class ClientBanque implements I_Personne, I_Client{
...}
```

- Elle doit alors fournir une implémentation à chacune des méthodes abstraites héritées
- Une interface peut étendre une autre interface (extends)