LANGAGE OBJ. AV.
(C++)
MASTER 1

U.F.R. d’Informatique
Université de Paris Cité

Cette semaine

- Notions d'UML - conventions
- Pointeurs, allocation/libération
- Namespaces
- Petit Quizz

LA MODÉLISATION

Le langage couramment employé pour décrire la structure d’une application dans le monde objet est UML (Unified Modeling Langage)

Il s’agit bien d’un **langage** (graphique) permettant d’exprimer différents aspects structurant l’application :

- concepts (ou classes)
- relation entre concepts
- évolution des objets
- interactions entre objets
- etc ...

Nous ne nous intéresserons qu'aux **diagrammes de classes** qui :

- décrivent les concepts pertinents
- décrivent les relations entre concepts

Remarques :

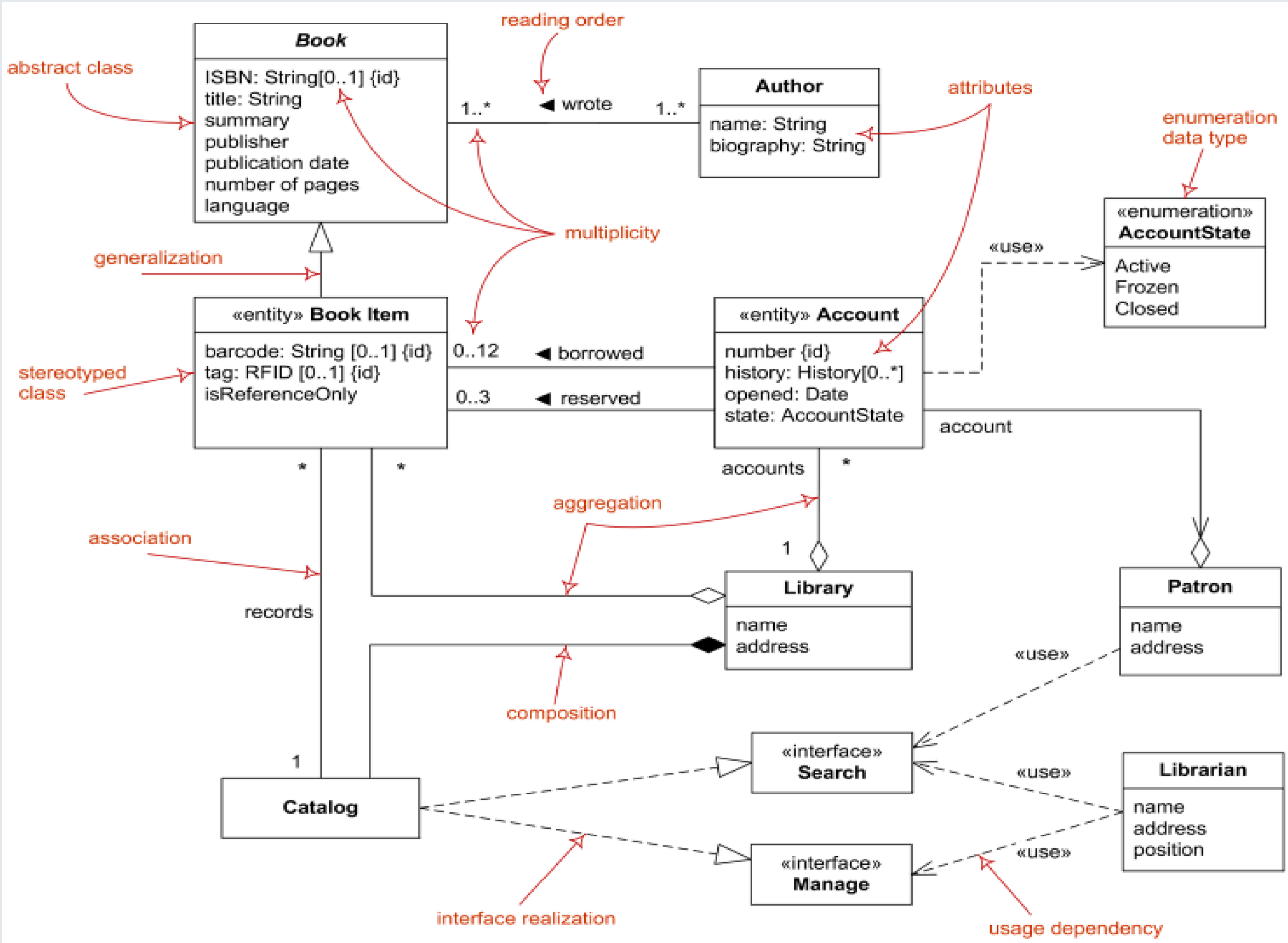
- Ils ne contiennent pas forcément toutes les classes (pas souhaitable), ni toutes les méthodes (faites par exemple une annotation générale précisant que les accesseurs/modifieurs sont implicitement fournis).

- Mais ils doivent contenir tout ce qui est significatif/structurant/utile à la compréhension.

- Un diagramme ne « tombe » pas du ciel! Il est obtenu en faisant (en amont) une liste des objets/concepts du système développé, puis en opérant une classification de ces objets...

- Utiliser un générateur de diagramme à postériori, en ayant d'abord écrit du code est une mauvaise idée : c'est illisible ... Faites le "à la main" !

Exemple :


Book
ISBN: String[0..1] {id}
title: String
summary
publisher
publication date
number of pages
language
abstract class
reading order
wrote
1..*
1..*
Author
name: String
biography: String
attributes
enumeration data type
«enumeration» AccountState
Active
Frozen
Closed
«use»
multiplicity
generalization
«entity» Book Item
barcode: String [0..1] {id}
tag: RFID [0..1] {id}
isReferenceOnly
stereotyped class
0..12
borrowed
0..3
reserved
«entity» Account
number {id}
history: History[0..*]
opened: Date
state: AccountState
account
accounts
*
*
*
*
aggregation
association
1
Library
name
address
Patron
name
address
records
composition
«use»
1
Catalog
«interface» Search
«use»
Librarian
name
address
position
«interface» Manage
«use»
interface realization
usage dependency

UML est un langage qui permet d'exprimer une façon de voir le monde via le prisme de concepts assez bien établis lorsqu'on programme :

- Abstraction
- Encapsulation
- Modularité
- Hierarchisation

L’abstraction :

Elle fait ressortir les caractéristiques essentielles d'un objet du point de vue de l’observateur.

Par exemple on peut décrire ce qu'on appelle une liste sans rentrer dans les détails d'une implémentation.

L’encapsulation :

C'est le fait de cacher l’ensemble des détails d’un objet qui ne font pas partie de ses caractéristiques essentielles.

Dans la vie réelle on parle de boîte noire (un appareil électro-ménager est une boîte noire, i.e. on n’en voit pas les détails de fonctionnement, ni directement tous les constituants).

La modularité :

Propriété d’un système qui a été décomposé en un ensemble de modules cohérents et faiblement couplés.

Ceci permet de nettement différencier les tâches à l’étape de réalisation et de confier celles-ci à des équipes quasi indépendantes.

La Hiérarchisation :

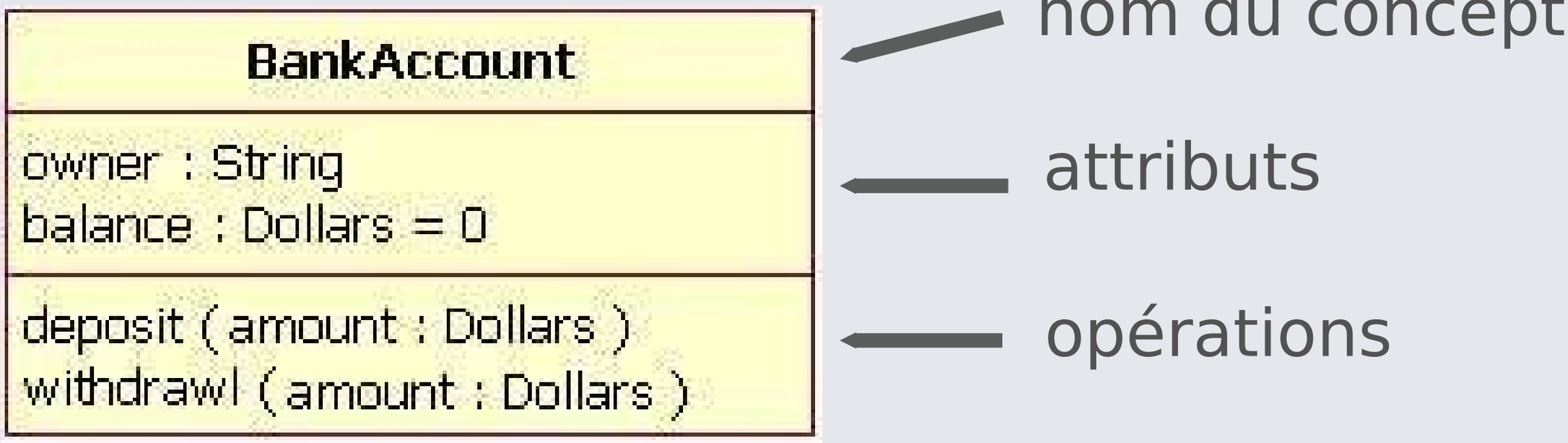
c'est l'ordonnancement des abstractions.

On utilisera en UML :

- l'association,
- l'héritage,
- l'agrégation,
- la composition

SYNTAXE

Une classe / entité simple :



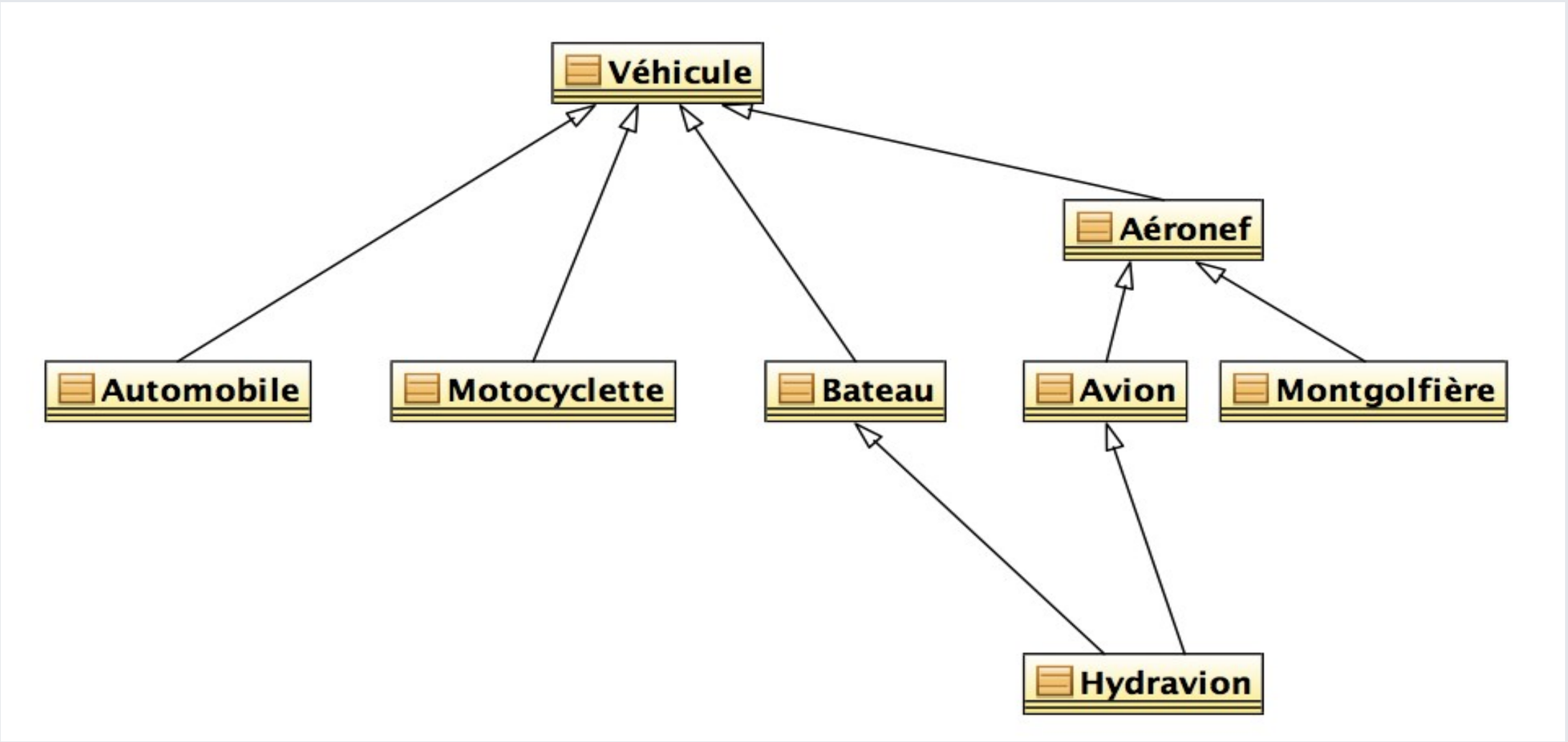
On peut utiliser un nom en italique pour une classe abstraite

On peut y ajouter des annotations de visibilité :

- `+` pour public
- `#` pour protected
- `-` pour private
- `~` pour package

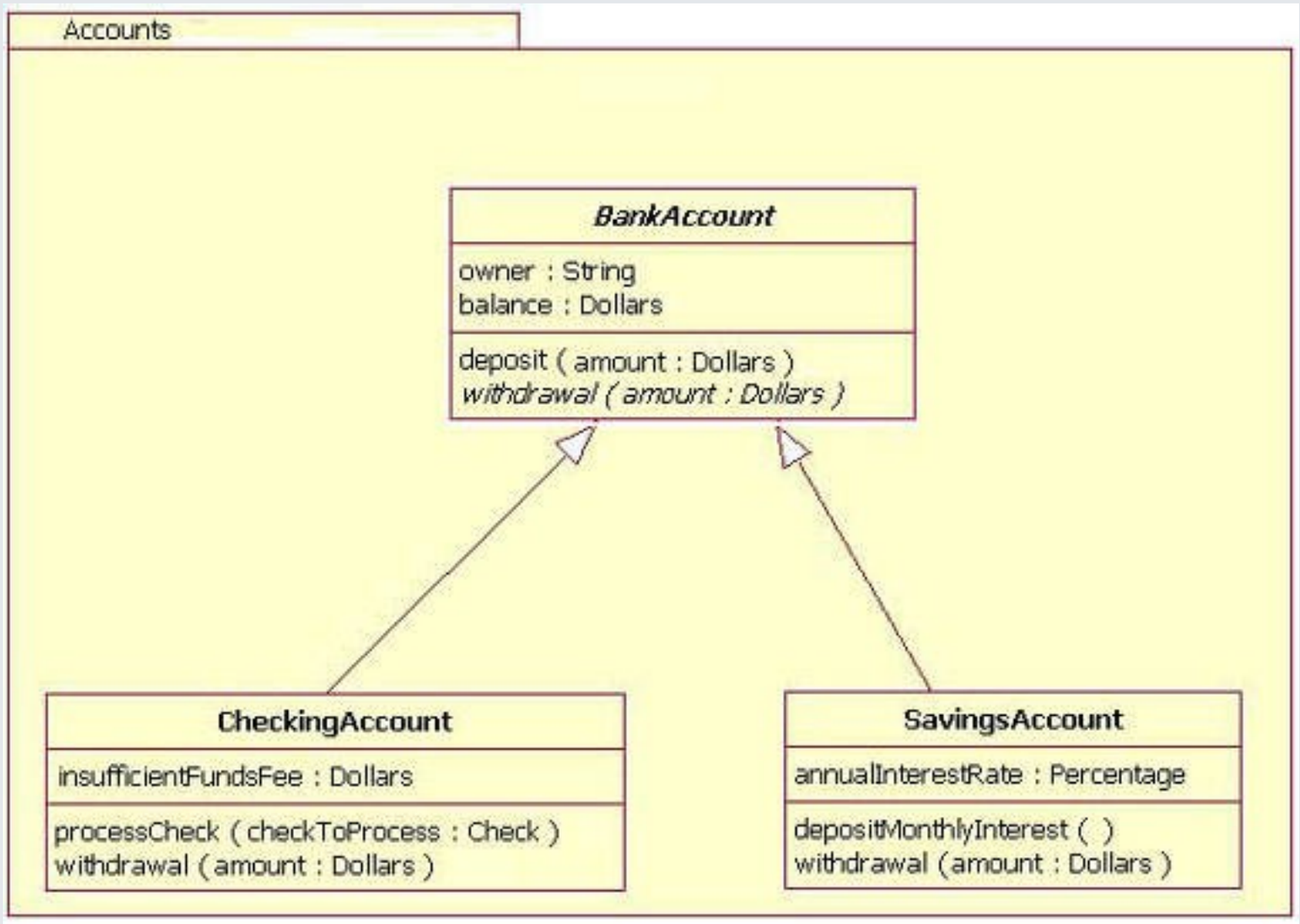
Héritage :

tout le monde comprend assez naturellement :



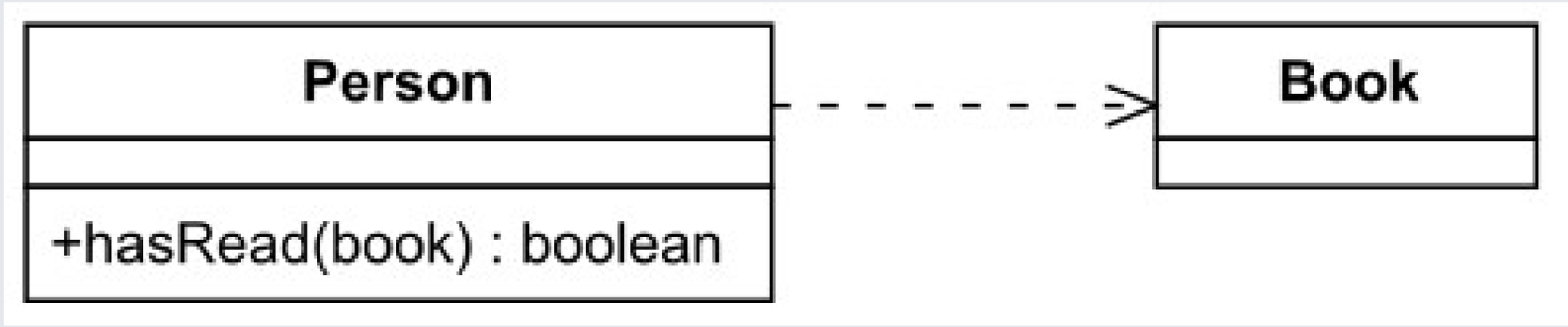
Notez la forme de la flèche et du trait

Classification dans un package :



Rq : ici BankAccount (et parfois withdrawal) sont en italique

Lien d'utilisation (use dependance) :



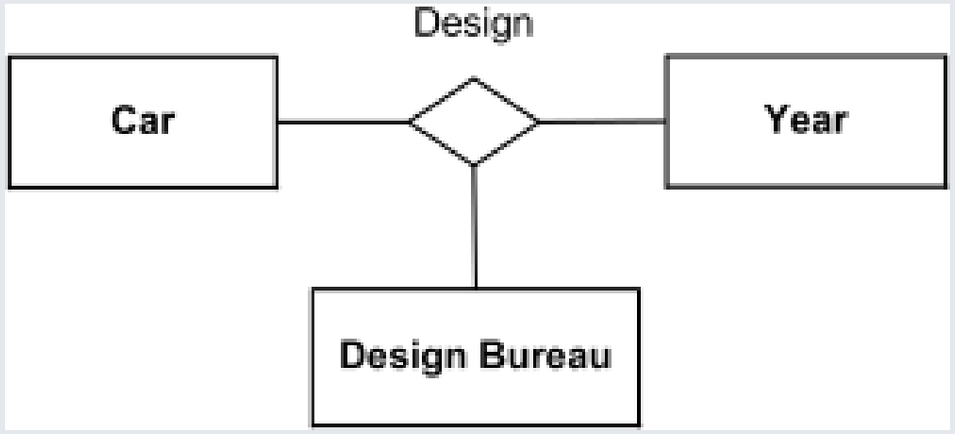
dans cet exemple une personne peut être amenée à utiliser un objet d'une autre classe (un livre) via la méthode hasread(Book).

Concrètement cela correspondra à un import.
Attention à ne pas surcharger votre diagramme, vous pouvez par exemple faire un "calque" n'exprimant que les dépendances.

Cette relation n'exprime rien sur le fait que les livres concernés sont ou pas stockés dans les attributs d'une personne. Elle exprime simplement que les personnes ont besoin de savoir ce qu'est un livre.

(Peut-être répondra t'il en consultant une base de donnée non représentée ici)

Associations :



Les associations ternaires ou plus doivent rester rares.

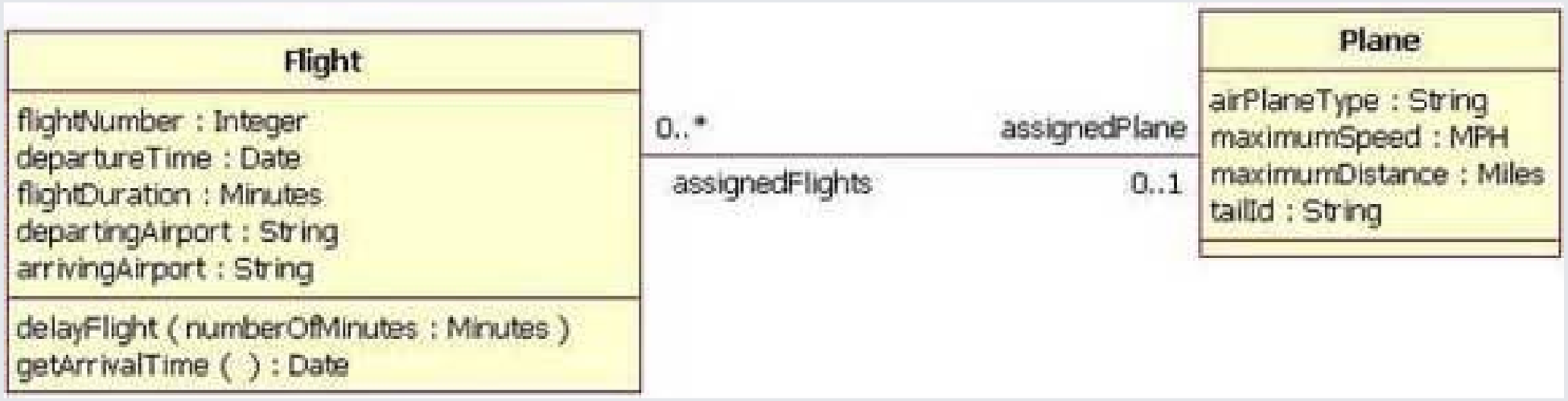
Les plus fréquentes sont les binaires, où on peut omettre le losange

Dans le cas binaire, le nom de l'association peut être orienté pour faciliter la construction de la phrase "l'enseignant a écrit le livre". On aurait pu choisir WrittenBy, et l'orienter dans l'autre sens.

Des rôles facultatifs peuvent être ajoutés en étiquette à la relation pour préciser que l'enseignant apparaissant dans cette relation est un auteur, et que le livre est un manuel scolaire.

Les multiplicités ...

Les multiplicités :

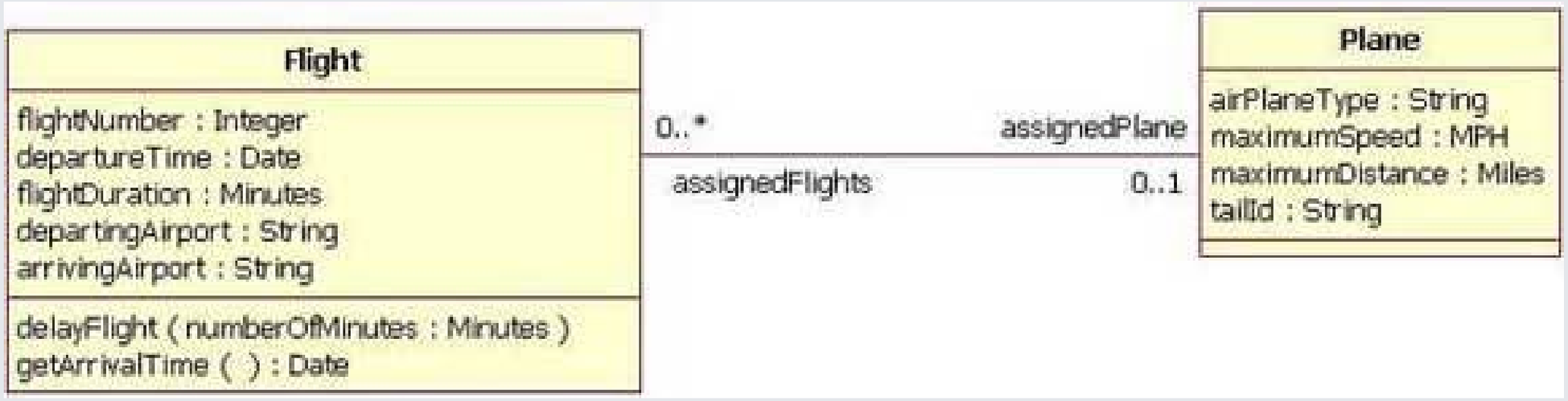


C'est un sujet de confusion pour les étudiants ... peut être à cause de merise/crowfoot ou simplement parce que ça a été mal expliqué

L'idée est d'exprimer combien de fois tel objet peut être en relation avec d'autres.

La définition officielle est la suivante :
"For an association with N ends, choose any N-1 ends and associate specific instances with those ends. Then the collection of links of the association that refer to these specific instances will identify a collection of instances at the other end. The multiplicity of the association end constrains the size of this collection."

Les multiplicités :

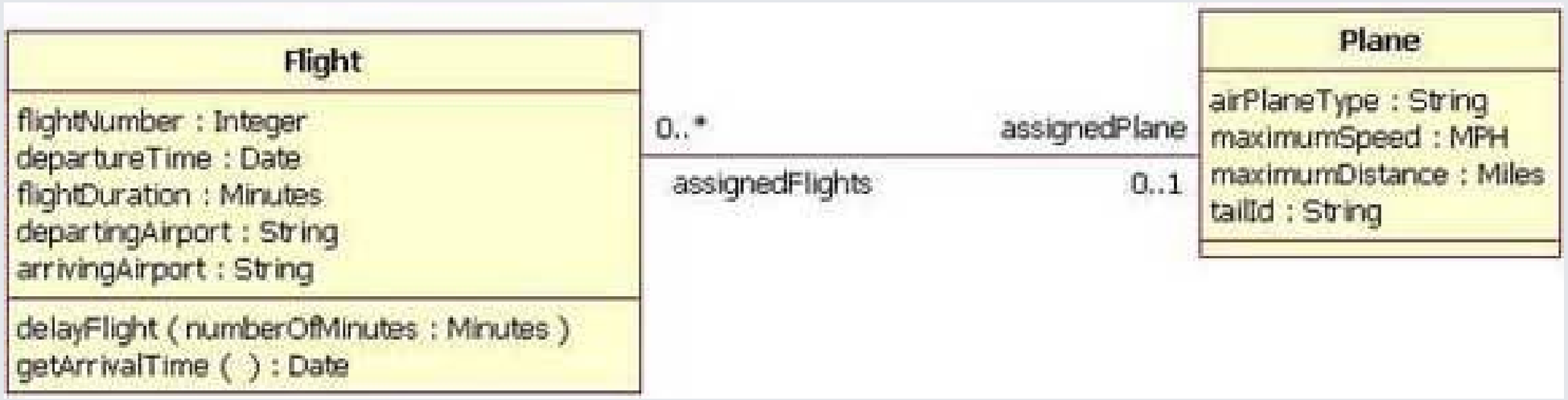


C'est un sujet de confusion pour les étudiants ... peut être à cause de merise/crowfoot ou simplement parce que ça a été mal expliqué

L'idée est d'exprimer combien de fois tel objet peut être en relation avec d'autres.

La traduction pratique :
"Pour exprimer les multiplicités des associations N-aire, choisissez N-1 extrémités et considérez y une instance spécifique (quelconque). Les associations liées à ces N-1 instances considérées fixées forment un ensemble dont les N-uplets ne se distinguent que par les valeurs de l'extrémité non choisie. La multiplicité de cette extrémité contraint la taille de cet ensemble"

Les multiplicités :



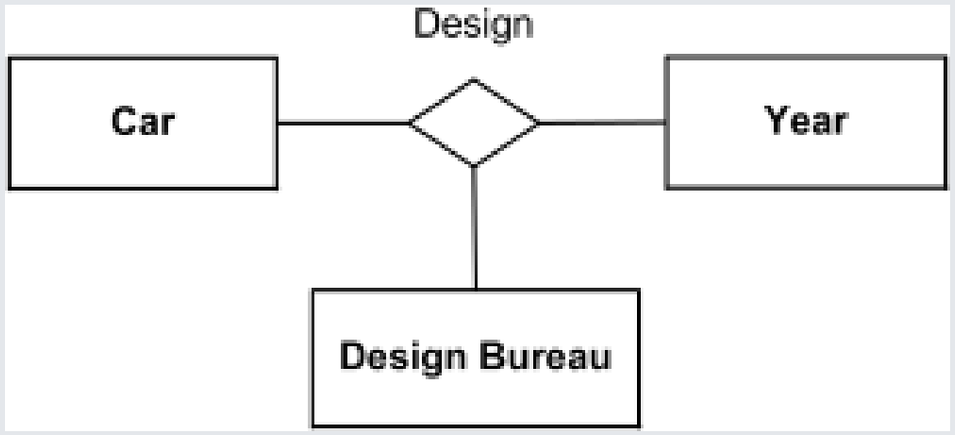
"For an association with N ends, choose any N-1 ends and associate specific instances with those ends. Then the collection of links of the association that refer to these specific instances will identify a collection of instances at the other end. The multiplicity of the association end constrains the size of this collection."

Sur cet exemple :

- pour la multiplicité coté Plane : on fixe un vol (Flight) quelconque, on comprend bien qu'alors il y a 0 ou 1 avion (Plane) qui lui sera affecté

- pour la multiplicité coté Flight : si on fixe un avion quelconque, il est clair qu'il pourra être utilisé dans plusieurs vols (ou aucun s'il est tout neuf)

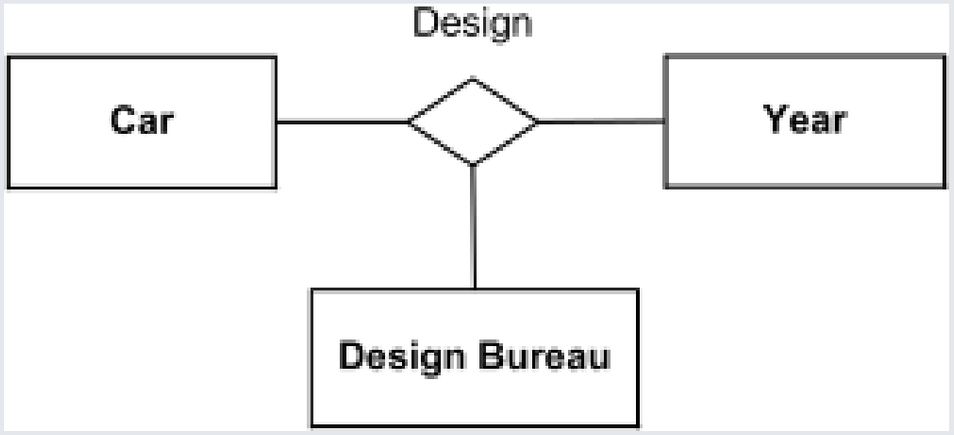
Les multiplicités :



"For an association with N ends, choose any N-1 ends and associate specific instances with those ends. Then the collection of links of the association that refer to these specific instances will identify a collection of instances at the other end. The multiplicity of the association end constrains the size of this collection."

Note : pour une association n-aire, les bornes basses des multiplicités sont typiquement 0. En effet si par exemple on avait 1, cela signifierait qu'un lien au moins devrait exister pour chaque combinaison possible des autres extrémités.

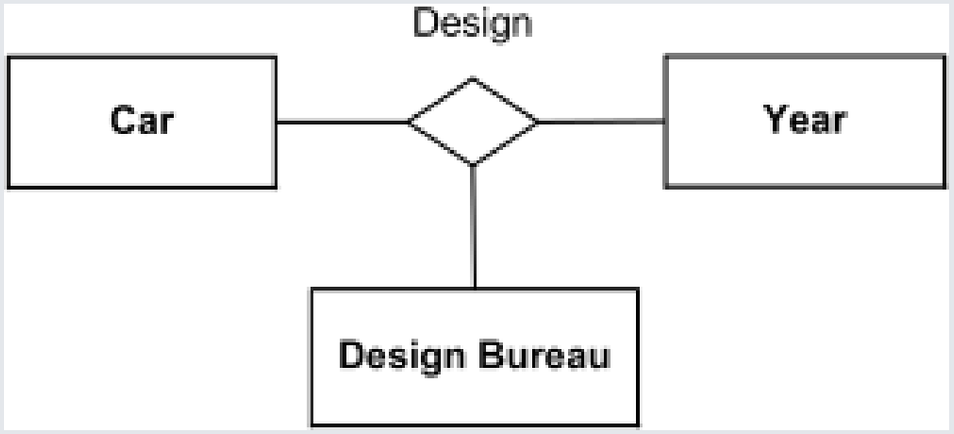
Les multiplicités :



"For an association with N ends, choose any N-1 ends and associate specific instances with those ends. Then the collection of links of the association that refer to these specific instances will identify a collection of instances at the other end. The multiplicity of the association end constrains the size of this collection."

Coté Bureau d'étude, on pourrait avoir :

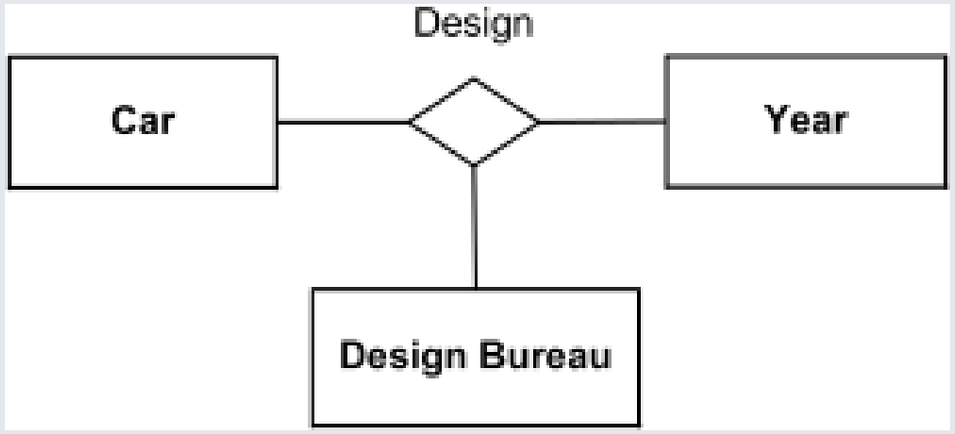
Les multiplicités :



"For an association with N ends, choose any N-1 ends and associate specific instances with those ends. Then the collection of links of the association that refer to these specific instances will identify a collection of instances at the other end. The multiplicity of the association end constrains the size of this collection."

Coté Bureau d'étude, on pourrait avoir : 0..1
Coté Année, on pourrait avoir :

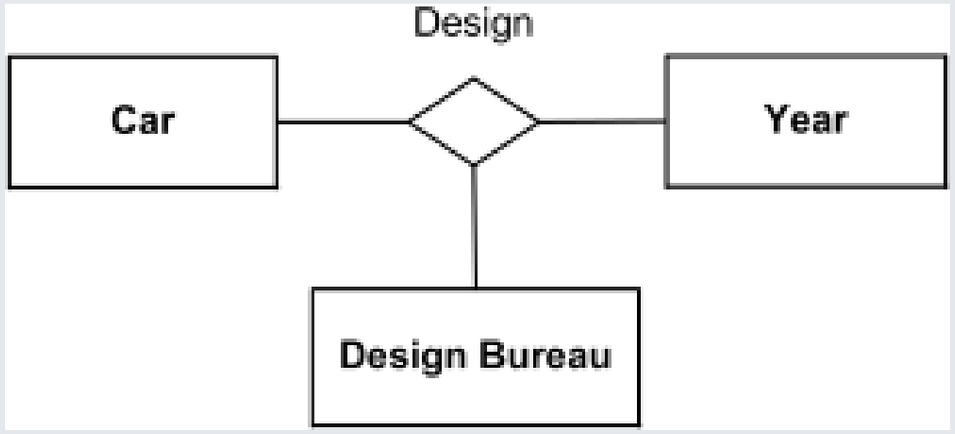
Les multiplicités :



"For an association with N ends, choose any N-1 ends and associate specific instances with those ends. Then the collection of links of the association that refer to these specific instances will identify a collection of instances at the other end. The multiplicity of the association end constrains the size of this collection."

Coté Bureau Etude, on pourrait avoir : 0..1
Coté Année, on pourrait avoir : 0..*
Coté Voiture, on pourrait avoir :

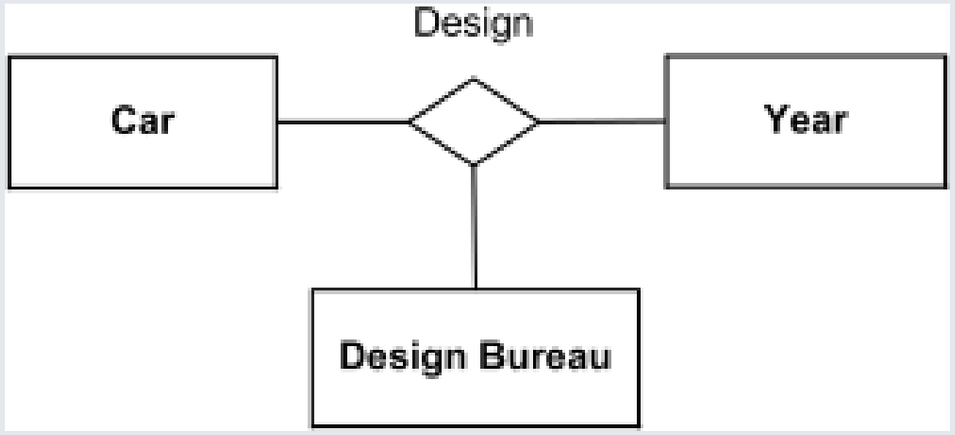
Les multiplicités :



"For an association with N ends, choose any N-1 ends and associate specific instances with those ends. Then the collection of links of the association that refer to these specific instances will identify a collection of instances at the other end. The multiplicity of the association end constrains the size of this collection."

Coté Bureau Etude, on pourrait avoir : 0..1
Coté Année, on pourrait avoir : 0..*
Coté Voiture, on pourrait avoir : 0..*

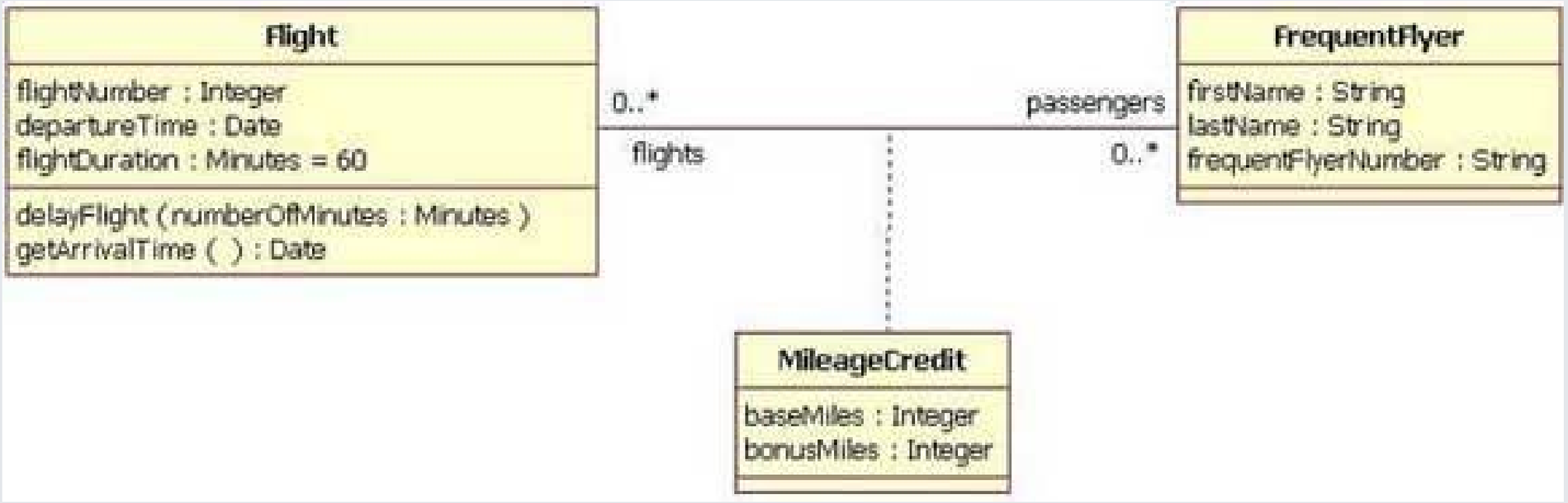
On peut parfois éviter des relations N-aires, en organisant autrement les N entités.



Par exemple on pourrait penser qu'une voiture associée à une année défini le concept CarModel, et que ce sont ces modèles qui ont été confiés à des bureaux d'étude, etc ...

.

Notion de Classe d'association :



Dans cette représentation, l'association MileageCredit :

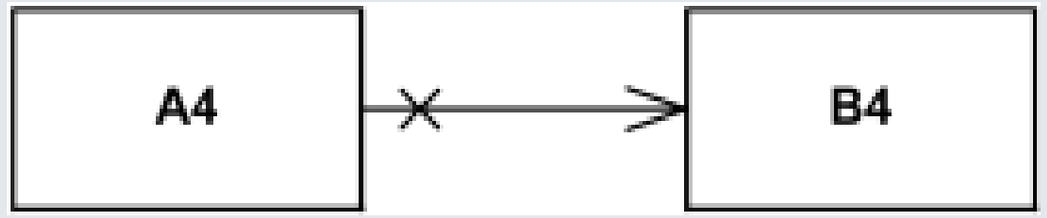
- porte des informations entre le vol et le passager abonné

- elle peut aussi servir de point d'entrée à d'autre association. Ainsi c'est le fait qu'une personne a vraiment effectué un vol qui sera pris en compte (par exemple pour ses remboursements de frais, son bilan carbone, etc)

Notion de Navigabilité

la flèche qui termine une association précise que ces objets peuvent être connus facilement des autres extrémités (sans préciser comment).
Naturellement la façon la plus simple de le faire est que Person possède des attributs qui lui permettent une connaissance directe.

Le contraire (non navigable) est modélisé par une croix interdisant l'arc.



Sous cette forme on expliciterait clairement que Book ne peut pas (facilement) naviguer vers Personne

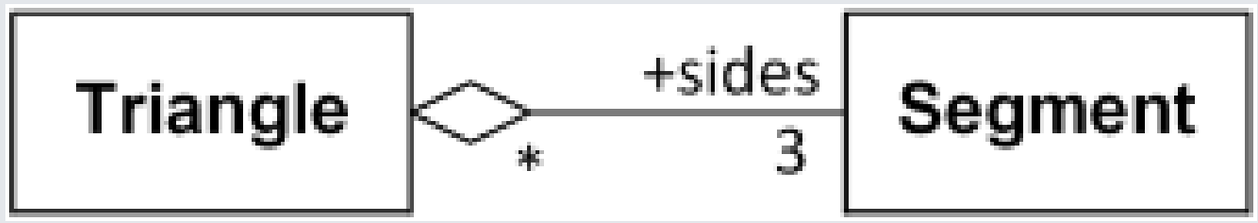
Si par contre le nom du propriétaire est inscrit sur le livre, alors vous pourriez utiliser cette forme d'association :

Notez qu'alors Book aurait un attribut owner de type Person (simple, pas de tableau)

Cas particulier d' associations binaires très fréquentes :

- Agrégation
- Composition

L’agrégation :



l’agrégation est une relation binaire reliant des composants à un composé

la **vie** des "éléments" est **indépendante** de celle de "l’agrégat"

un élément peut appartenir à plusieurs agrégats

ex : ici un triangle agrège 3 segments (qu'il appelle cotés), les segments peuvent servir à définir d'autres choses (des carrés par exemple, mais également d'autres triangles)

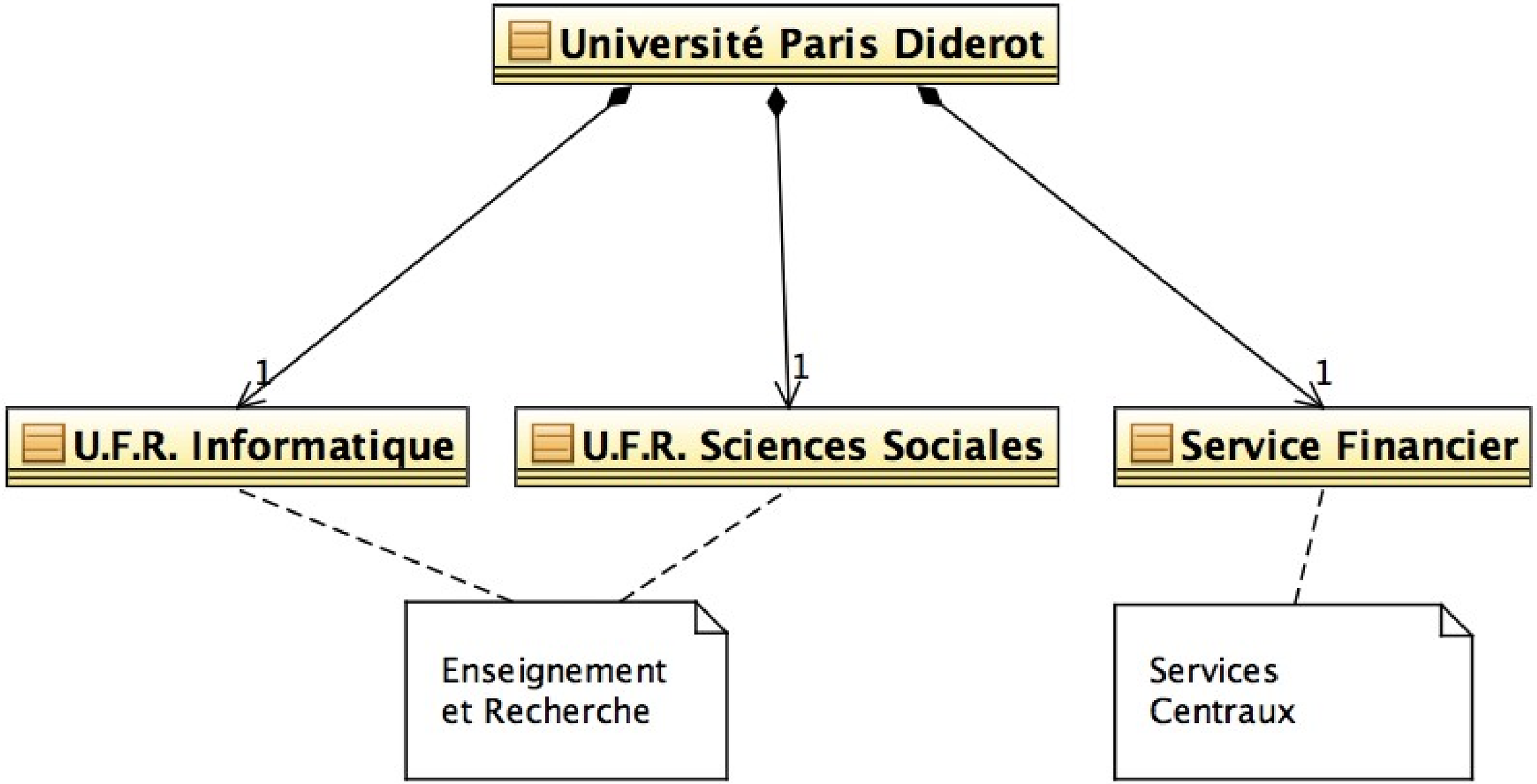
La navigabilité peut théoriquement venir surcharger cette notation (j'aurais tendance à la masquer coté triangle et la rendre visible coté segment)

La composition :

- dépendance forte : une destruction du composé entraîne la destruction des composants

- un composant ne peut appartenir qu’à un seul composé

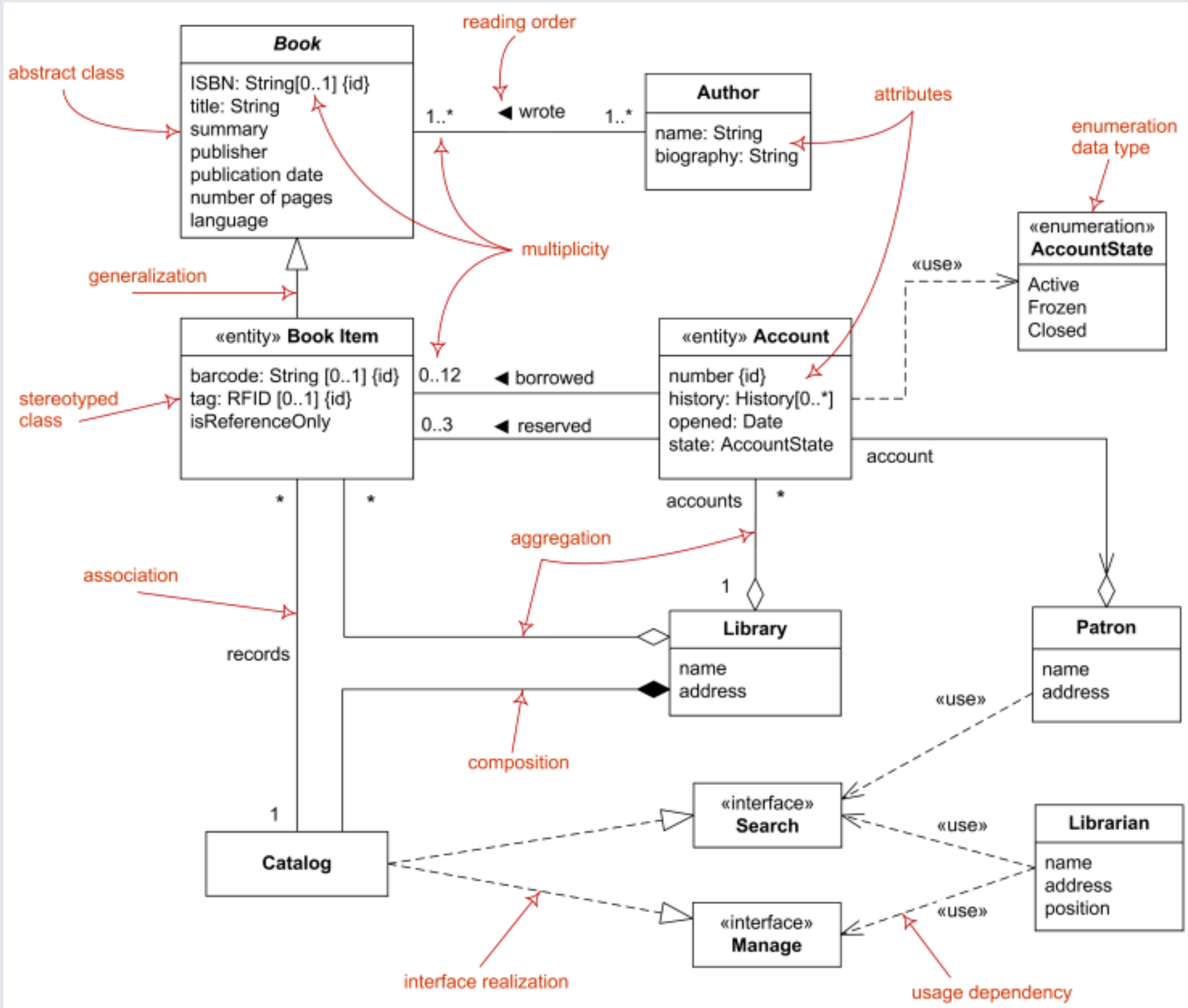
Rq : ici 1..1 n'est intéressant que pour le différencier de 0..1, où un fichier pourrait exister sans répertoire (par exemple temporairement)


Université Paris Diderot
1
1
1
U.F.R. Informatique
U.F.R. Sciences Sociales
Service Financier
Enseignement
et Recherche
Services
Centraux

discussions ...

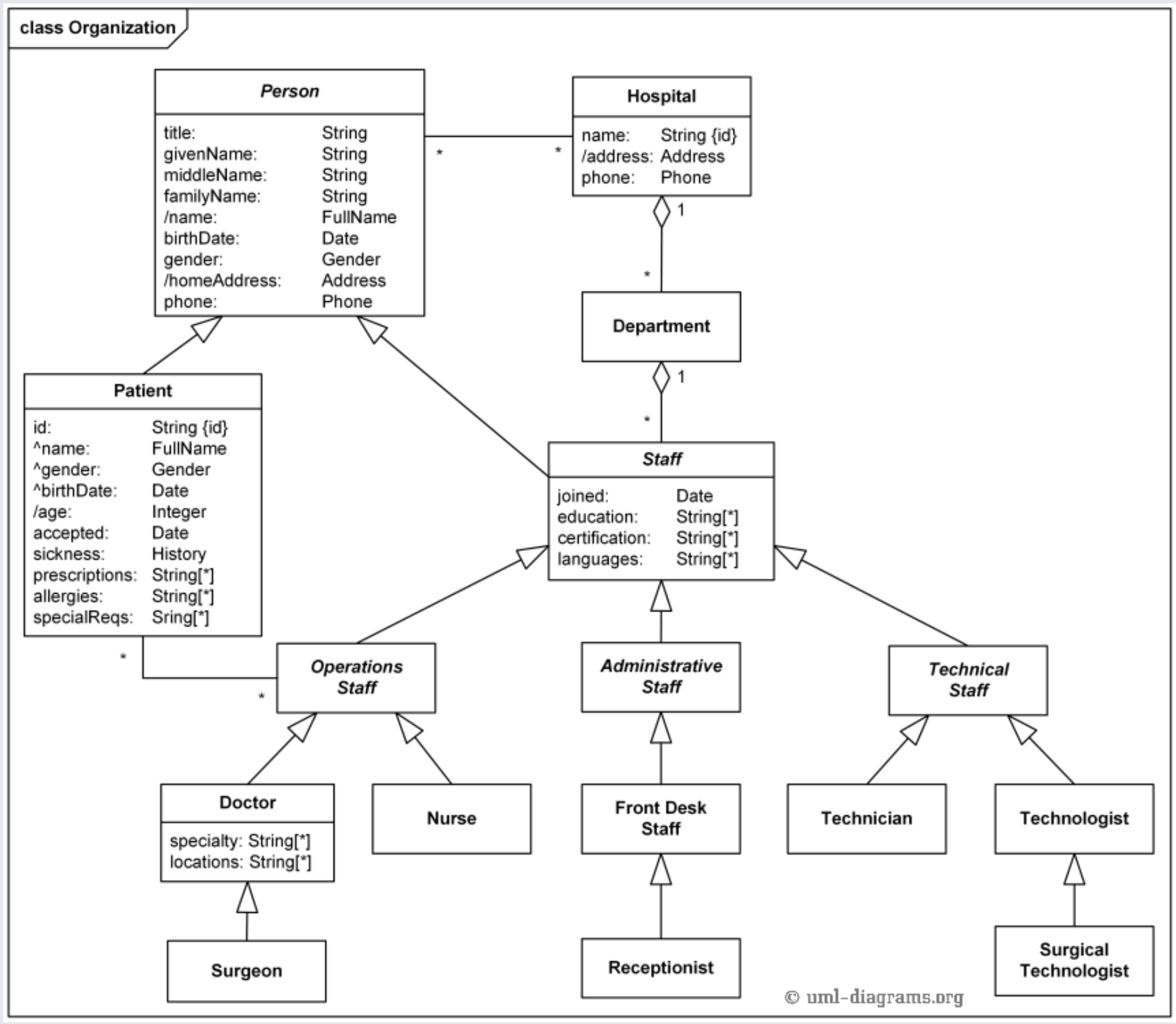
History ?
Multiplicité ?

Redondance ?
(Livre, BookItem, Catalog)

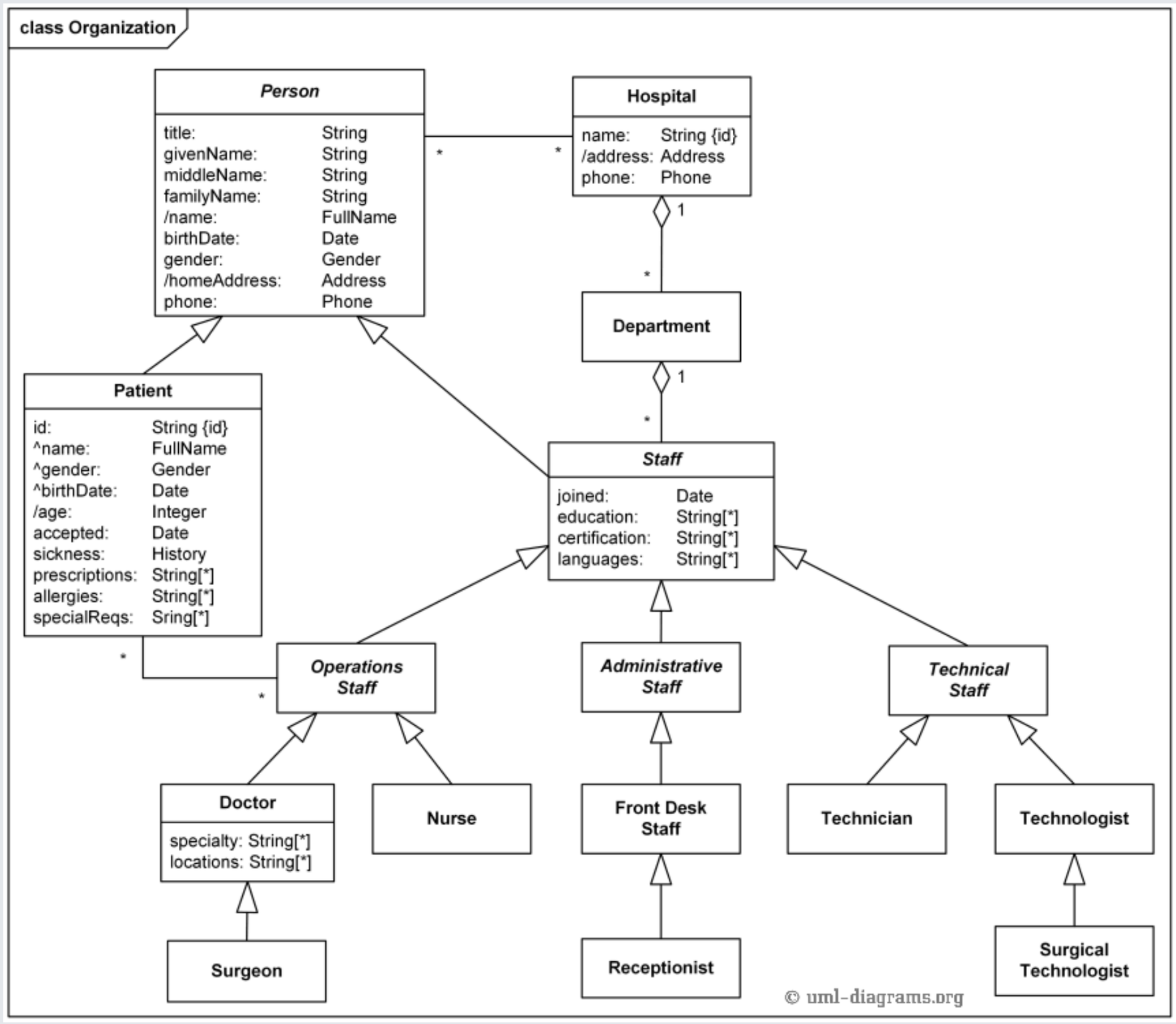


Qu'est ce que Catalog ? n'est ce pas une erreur ... qui devrait être modélisé en classe-Association ?

Discussions ...



Discussions ...



Person-Hospital ?? .... (pose la question du rôle du Patient ?)
Agregation ou Composition pour Department
Surgical Technologist ? (pas lié à opérations Staff ?)

LES POINTEURS, L'ALLOCATION DYNAMIQUE, LES TABLEAUX []

Les pointeurs - syntaxe

```
int a{0}, b{0};
int *pa, *pb;
pa = &a; pb = &b;
cout << (a==b) << (pa==pb) << endl;
```

remarquez que :

* ne fait pas vraiment partie du type (sinon on aurait écrit int * pa, pb)

& permet de récupérer l'adresse (il a cependant d'autres usages : expr. logique, type référence)

Le zéro d'un pointeur est nullptr.

Il n'y a pas d'affectation par défaut pour les pointeurs

Les pointeurs - syntaxe (2)

```
int a{0}, *pa{&a};
cout << a << *pa << endl;
```

c'est * qui permet de récupérer l'objet pointé

Que fait :

```
int a{0}, *pa{&a};
cout << a << *pa << endl;
*pa++;
cout << a << *pa << endl;
```

... je ne sais pas ...

Les pointeurs - syntaxe (2)

```
int a{0}, *pa{&a};
cout << a << *pa << endl;
```

c'est * qui permet de récupérer l'objet pointé

Que fait :

```
int a{0}, *pa{&a};
cout << a << *pa << endl;
*pa++;
cout << a << *pa << endl;
```

... je ne sais pas ...

++ est prioritaire sur * ... si on voulait ici (*pa)++ on a eu *(pa++)

Allocation dynamique :

```
int *p1 = new int    // allocation d'un espace libre
int *p2 = new int{10}; // allocation + initialisation
```

```
int *q = new int[10]; // 10 entiers consécutifs
```

Qu'on utilise ainsi :

```
for (int i=0;i<10;i++) *(q+i)=i;
for (int i=0;i<10;i++) q[i]=-i;
```

Pensez à libérer l'espace alloué dynamiquement
(sous votre responsabilité)

```
int *p = new int;
delete p;
```

```
int *q = new int[10];
delete [] q;
```

LES PORTÉES DES NOMS

L’opérateur :: permet d’accéder à un nom dans un contexte donné (espace de noms)

```
// une variable globale
int v;
int f(int v) {
    v = 3;
    return ::v;
}
```

un paramètre (var. locale)

accès à la variable locale

accès à la variable contextuelle

```
// dans A.hpp
namespace A {
    double x = 0.123;
}
```

```
// dans B.hpp
namespace B {
   string x = “truc“;
}
```

```
// ailleurs.cpp
#include “A.hpp“
#include “B.hpp“
....
....
if (A::x >0)
    cout << B::x << endl;
```

```
// ailleurs.cpp
#include “A.hpp“
#include “B.hpp“
....
using namespace A; // pas les 2...
if (x >0)
    cout << B::x << endl;
```

```
namespace two {
   namespace one { const int zero = 0; }
}
// qu'on utilisera ainsi
two::one::zero;
```

UN PETIT QUIZZ POUR FINIR

JOUER AVEC CONST ET * LES PRIORITÉS ENTRE * ET []

Quizz 1

```
int *t =new int[10];
int t2 [10];
t=t2; // ok ou non ok ?
t2=t; // ok ou non ok ?
```

Quizz 1

```
int *t =new int[10];
int t2 [10];
t=t2; // ok
t2=t; // non ok
```

incompatible types in assignment of ‘int*’ to ‘int [10]

Quizz 2

```
int *x[10], y[10], z{4};
// comment décrire le type de x et de y ?

y[0]=123;
x[0]=y;
x[1]= &z;
y[1]=2*z;
cout  << x[0] << endl;  //
cout << *x << endl; //
cout << y << endl;  //
cout << *(x[0]) << endl;  //
cout << (*x)[0] << endl; //
cout << *(x[1]) << endl; //
cout << (*x)[1] << endl; //
cout << *(x[1]) << endl; //
```

Quizz 2

```
int *x[10], y[10], z{4};
// comment décrire le type de x et de y ?

y[0]=123;
x[0]=y;
x[1]= &z;
y[1]=2*z;
cout  << x[0] << endl;  //
cout << *x << endl; //
cout << y << endl;  //
cout << *(x[0]) << endl;  //
cout << (*x)[0] << endl; //
cout << *(x[1]) << endl; //
cout << (*x)[1] << endl; //
cout << *(x[1]) << endl; //
```

y est un(e adresse de) tableau contenant 10 entiers

x est un(e adresse de) tableau contenant 10 pointeurs vers des entiers

Quizz 2

```
int *x[10], y[10], z{4};
// comment décrire le type de x et de y ?

y[0]=123;
x[0]=y;
x[1]= &z;
y[1]=2*z;
cout  << x[0] << endl;  // du genre 0xbfb22154
cout << *x << endl; //
cout << y << endl;  //
cout << *(x[0]) << endl;  //
cout << (*x)[0] << endl; //
cout << *(x[1]) << endl; //
cout << (*x)[1] << endl; //
cout << *(x[1]) << endl; //
```

y est un(e adresse de) tableau contenant 10 entiers

x est un(e adresse de) tableau contenant 10 pointeurs vers des entiers

Quizz 2

```
int *x[10], y[10], z{4};
// comment décrire le type de x et de y ?

y[0]=123;
x[0]=y;
x[1]= &z;
y[1]=2*z;
cout  << x[0] << endl;  // du genre 0xbfb22154
cout << *x << endl; // idem
cout << y << endl;  //
cout << *(x[0]) << endl;  //
cout << (*x)[0] << endl; //
cout << *(x[1]) << endl; //
cout << (*x)[1] << endl; //
cout << *(x[1]) << endl; //
```

y est un(e adresse de) tableau contenant 10 entiers

x est un(e adresse de) tableau contenant 10 pointeurs vers des entiers

Quizz 2

```
int *x[10], y[10], z{4};
// comment décrire le type de x et de y ?

y[0]=123;
x[0]=y;
x[1]= &z;
y[1]=2*z;
cout  << x[0] << endl;  // du genre 0xbfb22154
cout << *x << endl; // idem
cout << y << endl;  // idem
cout << *(x[0]) << endl;  //
cout << (*x)[0] << endl; //
cout << *(x[1]) << endl; //
cout << (*x)[1] << endl; //
cout << *(x[1]) << endl; //
```

y est un(e adresse de) tableau contenant 10 entiers

x est un(e adresse de) tableau contenant 10 pointeurs vers des entiers

Quizz 2

```
int *x[10], y[10], z{4};
// comment décrire le type de x et de y ?

y[0]=123;
x[0]=y;
x[1]= &z;
y[1]=2*z;
cout  << x[0] << endl;  // du genre 0xbfb22154
cout << *x << endl; // idem
cout << y << endl;  // idem
cout << *(x[0]) << endl;  // 123
cout << (*x)[0] << endl; //
cout << *(x[1]) << endl; //
cout << (*x)[1] << endl; //
cout << *(x[1]) << endl; //
```

y est un(e adresse de) tableau contenant 10 entiers

x est un(e adresse de) tableau contenant 10 pointeurs vers des entiers

Quizz 2

```
int *x[10], y[10], z{4};
// comment décrire le type de x et de y ?

y[0]=123;
x[0]=y;
x[1]= &z;
y[1]=2*z;
cout  << x[0] << endl;  // du genre 0xbfb22154
cout << *x << endl; // idem
cout << y << endl;  // idem
cout << *(x[0]) << endl;  // 123
cout << (*x)[0] << endl; // 123
cout << *(x[1]) << endl; //
cout << (*x)[1] << endl; //
cout << *(x[1]) << endl; //
```

y est un(e adresse de) tableau contenant 10 entiers

x est un(e adresse de) tableau contenant 10 pointeurs vers des entiers

Quizz 2

```
int *x[10], y[10], z{4};
// comment décrire le type de x et de y ?

y[0]=123;
x[0]=y;
x[1]= &z;
y[1]=2*z;
cout  << x[0] << endl;  // du genre 0xbfb22154
cout << *x << endl; // idem
cout << y << endl;  // idem
cout << *(x[0]) << endl;  // 123
cout << (*x)[0] << endl; // 123
cout << *(x[1]) << endl; // 4
cout << (*x)[1] << endl; //
cout << *(x[1]) << endl; //
```

y est un(e adresse de) tableau contenant 10 entiers

x est un(e adresse de) tableau contenant 10 pointeurs vers des entiers

Quizz 2

```
int *x[10], y[10], z{4};
// comment décrire le type de x et de y ?

y[0]=123;
x[0]=y;
x[1]= &z;
y[1]=2*z;
cout  << x[0] << endl;  // du genre 0xbfb22154
cout << *x << endl; // idem
cout << y << endl;  // idem
cout << *(x[0]) << endl;  // 123
cout << (*x)[0] << endl; // 123
cout << *(x[1]) << endl; // 4
cout << (*x)[1] << endl; // 8
cout << *(x[1]) << endl; //
```

y est un(e adresse de) tableau contenant 10 entiers

x est un(e adresse de) tableau contenant 10 pointeurs vers des entiers

Quizz 2

```
int *x[10], y[10], z{4};
// comment décrire le type de x et de y ?

y[0]=123;
x[0]=y;
x[1]= &z;
y[1]=2*z;
cout  << x[0] << endl;  // du genre 0xbfb22154
cout << *x << endl; // idem
cout << y << endl;  // idem
cout << *(x[0]) << endl;  // 123
cout << (*x)[0] << endl; // 123
cout << *(x[1]) << endl; // 4
cout << (*x)[1] << endl; // 8
cout << *(x[1]) << endl; // 4
```

y est un(e adresse de) tableau contenant 10 entiers

x est un(e adresse de) tableau contenant 10 pointeurs vers des entiers

Quizz 2

```
int *x[10], y[10], z{4};
// comment décrire le type de x et de y ?

y[0]=123;
x[0]=y;
x[1]= &z;
y[1]=2*z;
cout  << x[0] << endl;  // du genre 0xbfb22154
cout << *x << endl; // idem
cout << y << endl;  // idem
cout << *(x[0]) << endl;  // 123
cout << (*x)[0] << endl; // 123
cout << *(x[1]) << endl; // 4
cout << (*x)[1] << endl; // 8
cout << *(x[1]) << endl; // 4
cout << *x[1] << endl; // ??
```

y est un(e adresse de) tableau contenant 10 entiers

x est un(e adresse de) tableau contenant 10 pointeurs vers des entiers

Quizz 2

```
int *x[10], y[10], z{4};
// comment décrire le type de x et de y ?

y[0]=123;
x[0]=y;
x[1]= &z;
y[1]=2*z;
cout  << x[0] << endl;  // du genre 0xbfb22154
cout << *x << endl; // idem
cout << y << endl;  // idem
cout << *(x[0]) << endl;  // 123
cout << (*x)[0] << endl; // 123
cout << *(x[1]) << endl; // 4
cout << (*x)[1] << endl; // 8
cout << *(x[1]) << endl; // 4
cout << *x[1] << endl; // 4
```

Notez la priorité de [] sur *

(qu'on avait déjà compris sur les types ...)

y est un(e adresse de) tableau contenant 10 entiers

x est un(e adresse de) tableau contenant 10 pointeurs vers des entiers

Quizz 2

```
int *x[10], y[10], z{4};
// comment décrire le type de x et de y ?

y[0]=123;
x[0]=y;
x[1]= &z;
y[1]=2*z;
cout  << x[0] << endl;  // du genre 0xbfb22154
cout << *x << endl; // idem
cout << y << endl;  // idem
cout << *(x[0]) << endl;  // 123
cout << (*x)[0] << endl; // 123
cout << *(x[1]) << endl; // 4
cout << (*x)[1] << endl; // 8
cout << *(x[1]) << endl; // 4
cout << *(x[1]) << endl; // 4
cout << *x[1] << endl; // 4
cout << **x << endl; // ??
```

Encore un ...

Quizz 2

```
int *x[10], y[10], z{4};
// comment décrire le type de x et de y ?

y[0]=123;
x[0]=y;
x[1]= &z;
y[1]=2*z;
cout  << x[0] << endl;  // du genre 0xbfb22154
cout << *x << endl; // idem
cout << y << endl;  // idem
cout << *(x[0]) << endl;  // 123
cout << (*x)[0] << endl; // 123
cout << *(x[1]) << endl; // 4
cout << (*x)[1] << endl; // 8
cout << *(x[1]) << endl; // 4
cout << *(x[1]) << endl; // 4
cout << *x[1] << endl; // 4
cout << **x << endl; // 123
```

Encore un ...

Quizz 3

```
int const a {1};
const int b {1};

const int *p {&a};
p = &b; //
*p = 2;  //
```

Règle générale pour const :
il s'applique à ce qui se trouve directement à sa gauche.
Tolérance :
s'il n'y a rien qui le précède, il s'applique au premier type à sa droite.

Quizz 3

```
int const a {1};
const int b {1};

const int *p {&a};
p = &b; // ok
*p = 2;  //
```

Règle générale pour const :
il s'applique à ce qui se trouve directement à sa gauche.
Tolérance :
s'il n'y a rien qui le précède, il s'applique au premier type à sa droite.

Quizz 3

```
int const a {1};
const int b {1};

const int *p {&a};
p = &b; // ok
*p = 2;  // interdit

int c {3}, d = 4.5;
int *const q = &c;

*q = 4;  //
q = &d; //
q = &a; //
```

Règle générale pour const :
il s'applique à ce qui se trouve directement à sa gauche.
Tolérance :
s'il n'y a rien qui le précède, il s'applique au premier type à sa droite.

Quizz 3

```
int const a {1};
const int b {1};

const int *p {&a};
p = &b; // ok
*p = 2;  // interdit

int c {3}, d = 4.5;
int *const q = &c;

*q = 4;  // ok
q = &d; //
q = &a; //
```

Règle générale pour const :
il s'applique à ce qui se trouve directement à sa gauche.
Tolérance :
s'il n'y a rien qui le précède, il s'applique au premier type à sa droite.

Quizz 3

```
int const a {1};
const int b {1};

const int *p {&a};
p = &b; // ok
*p = 2;  // interdit

int c {3}, d = 4.5;
int *const q = &c;

*q = 4;  // ok
q = &d; // interdit
q = &a; //
```

Règle générale pour const :
il s'applique à ce qui se trouve directement à sa gauche.
Tolérance :
s'il n'y a rien qui le précède, il s'applique au premier type à sa droite.

Quizz 3

```
int const a {1};
const int b {1};

const int *p {&a};
p = &b; // ok
*p = 2;  // interdit

int c {3}, d = 4.5;
int *const q = &c;

*q = 4;  // ok
q = &d; // interdit
q = &a; // interdit

const int * const r {&a};
r = &c; //
*r = 5; //
```

Règle générale pour const :

il s'applique à ce qui se trouve directement à sa gauche.

Tolérance :

s'il n'y a rien qui le précède, il s'applique au premier type à sa droite.

Quizz 3

```
int const a {1};
const int b {1};

const int *p {&a};
p = &b; // ok
*p = 2;  // interdit

int c {3}, d = 4.5;
int *const q = &c;

*q = 4;  // ok
q = &d; // interdit
q = &a; // interdit

const int * const r {&a};
r = &c; // interdit
*r = 5; // interdit
```

Règle générale pour const :

il s'applique à ce qui se trouve directement à sa gauche.

Tolérance :

s'il n'y a rien qui le précède, il s'applique au premier type à sa droite.

Quizz 4

```
int b[10]={0,1,2,3,4,5,6,7,8,9};
const int * const x[10] {b};  //
```

Quizz 4

```
int b[10]={0,1,2,3,4,5,6,7,8,9};
const int * const x[10] {b};  // ok !

cout << *x[0] << endl;  // 0
(*x[0])++;  // ?
```

Quizz 4

```
int b[10]={0,1,2,3,4,5,6,7,8,9};
const int * const x[10] {b};  //

cout << *x[0] << endl;  // 0
(*x[0])++;  // erreur
b[0]++;      // ?
```

Quizz 4

```
int b[10]={0,1,2,3,4,5,6,7,8,9};
const int * const x[10] {b};  //

cout << *x[0] << endl;  // 0
(*x[0])++;  // erreur
b[0]++;      // ok
cout << *x[0] << endl; // ?
```

Quizz 4

```
int b[10]={0,1,2,3,4,5,6,7,8,9};
const int * const x[10] {b};  //

cout << *x[0] << endl;  // 0
(*x[0])++;  // erreur
b[0]++;      // ok
cout << *x[0] << endl; // 1
```

il faut comprendre que le type d'une variable contraint/indique les autorisations de modification faites via cette variable, cela ne dit rien sur la mémoire sous-jacente.