ENSIAS

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

Programmation Orientée Objet

2023-2024

Programmation Orientée Objet

ENSIAS

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

- **Filière BI&A :**

  Module POO →
  - Init au dev logiciel 35%
  - **POO** 65%

- **Filière GL :**

  Module POOdev1 →
  - Développement web 30%
  - XML 20%
  - **POO** 50%

→ **POO**

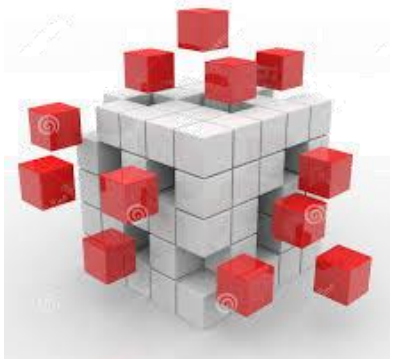
Cours : 14h	TP: 12h	AP: 8h
60%	**40%**	

ENSIAS

Programmation Orientée Objet

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

1. Paradigme et Concepts Objet
2. Instanciation en Java
3. Héritage et interfaces en Java
4. Framework des Collections
5. Swing et IHM
6. Sérialisation et E/S
7. JDBC et Bases de données



ENSIAS

Programmation Orientée Objet

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

1. Paradigme et Concepts Objet

- **Du procédural à l'Objet**
- **Concepts Objet**
  - ***Héritage Simple et multiple***
  - ***Polymorphisme***
  - ***Liaison dynamique***
- **Relations entre classes**
  - ***Association - Agrégation - Composition***

Qualité du logiciel…

1. **Valide**

   respecte le *cahier des charges*

2. **Réutilisable, Flexible**

   s'adapte à de nouvelles *applications*

3. **Maintenable**

   modification, correction, adaptation (choix des SDs)

ENSIAS

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

Histoire ...

VS

Procedural Programming

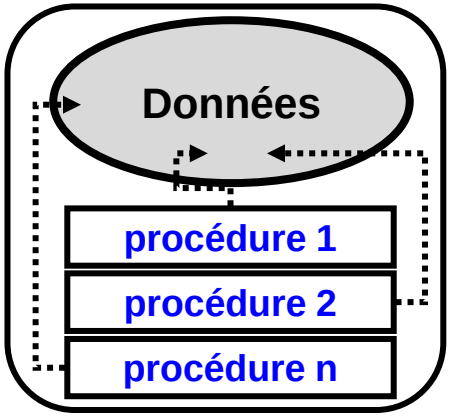
Object-Oriented Programming

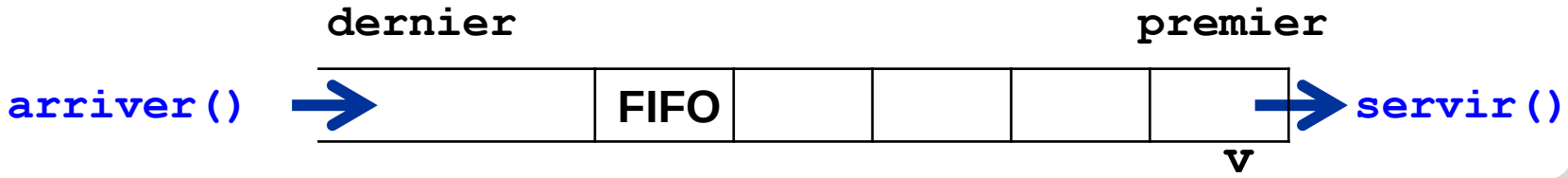
Structurée et modulaire

➔ Equation de **Niclauss Wirth** : **Pascal, C**

Données	+ Procédures

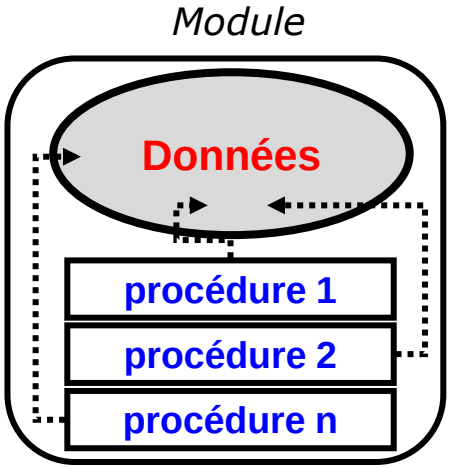
Application Procédurale

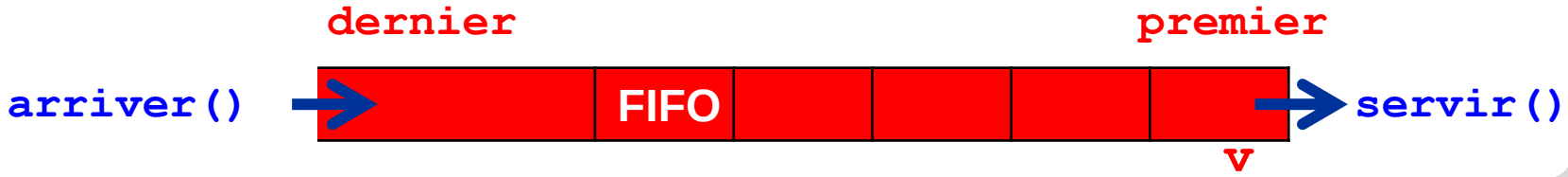




Structurée et modulaire

- Equation de **Niclauss Wirth** : **Pascal, C**
- **Module :** Données privées + Procédures



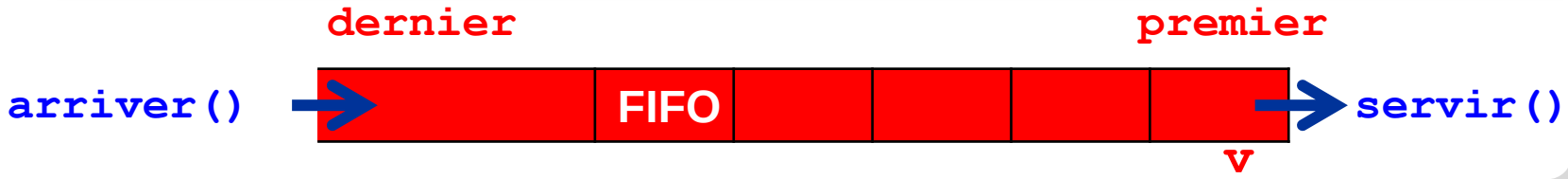


Structurée et modulaire

- Equation de **Niclauss Wirth** : **Pascal, C**
- **Module :** Données privées + Procédures

Représentation séquentielle

```
static client v[100]; //privée (au fichier)
static int premier, dernier;
void arriver(client x){
  if(! filePleine()) { dernier++; v[dernier]=x;}
}
int servir(){/* … */}
```



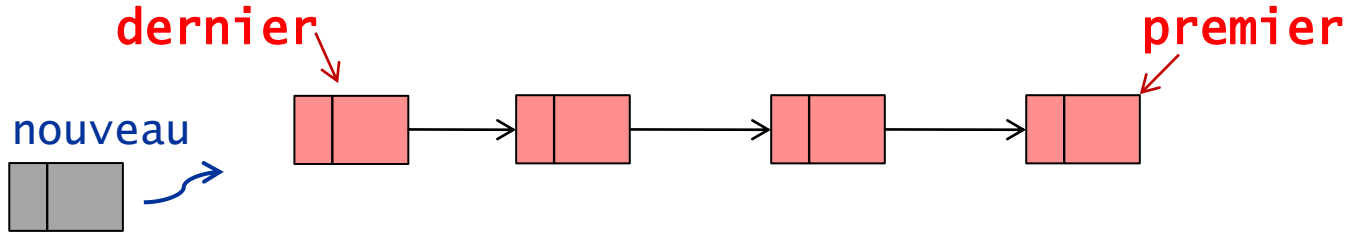
Structurée et modulaire

- **Validité : /à la spécification**
- **Accès à travers une interface**

Structurée et modulaire

Maintenabilité

Représentation en liste chaînée ?

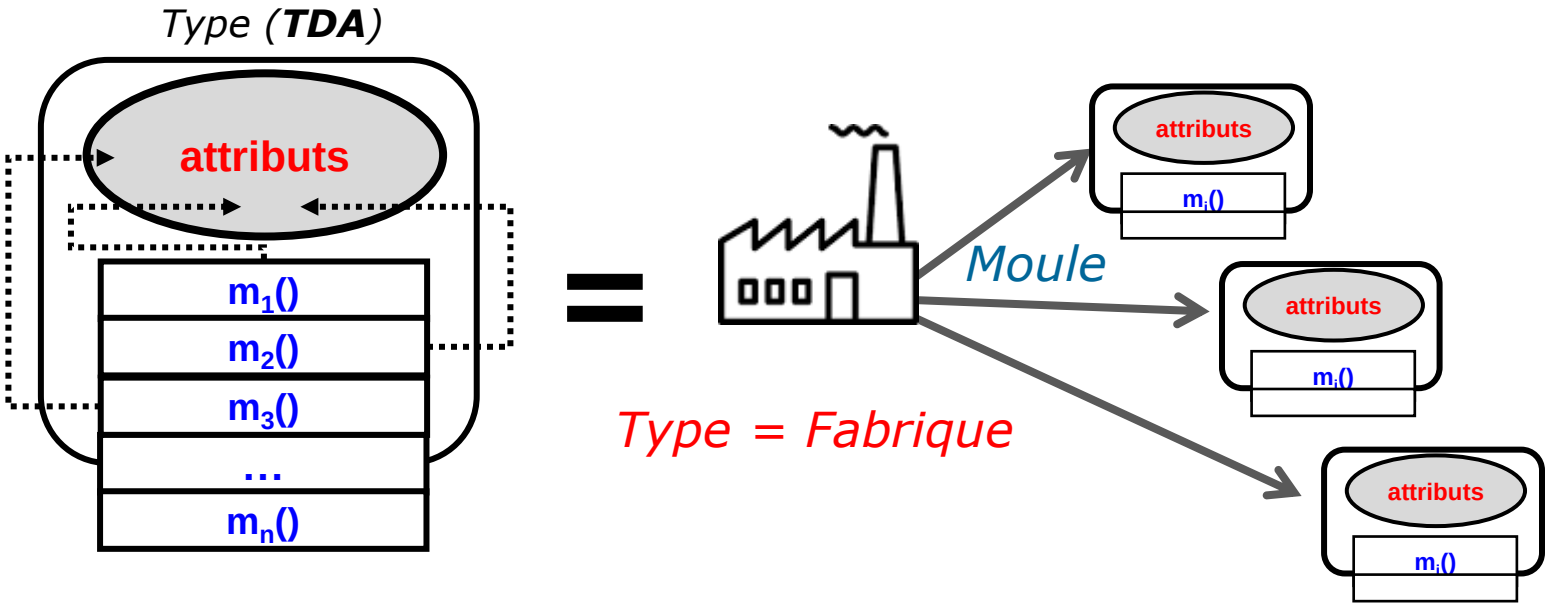



```
static client * premier, dernier;

void arriver(client * nouveau){
   nouveau -> suivant = dernier;
   dernier = nouveau;
}
```

Abstraction des données

On passe du module au Type

Ensemble complet d'opérations



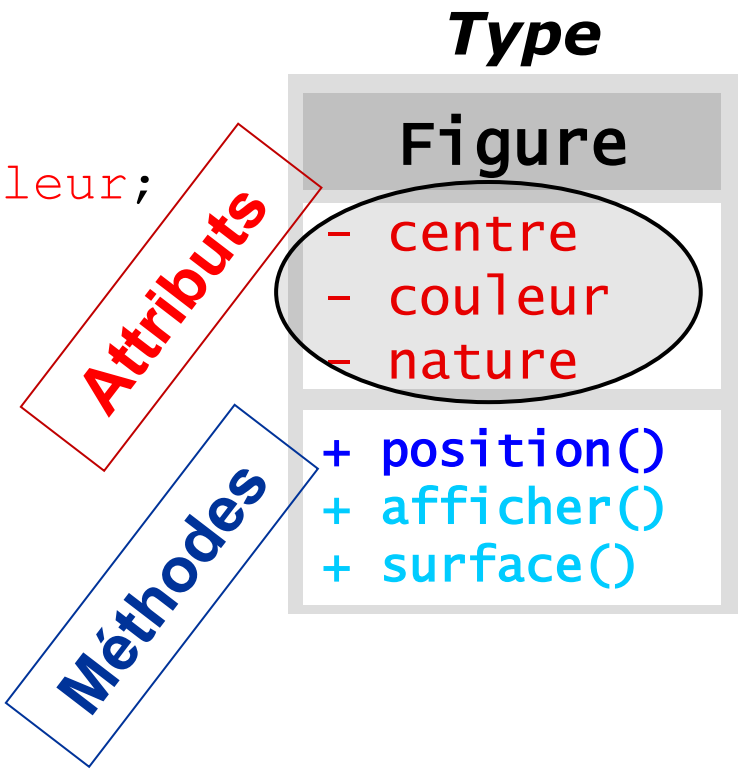
ENSIAS

Abstraction des données

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

On passe du module au Type

```
enum Nature {cercle,triangle,rectangle}

class Figure {

  private :
      Point centre; Couleur couleur;
      Nature nature;

  public :

      Point position() {
         return centre;
      }

      void afficher(int);
      float surface();
}
```

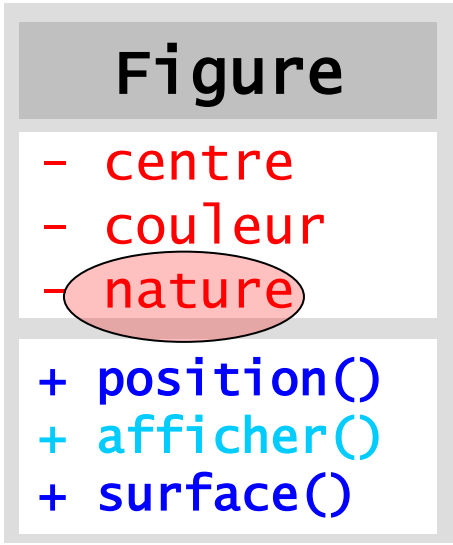


Abstraction des données

On passe du module au Type

Boite noire Il faut "connaître" la nature

```
float surface() {
  switch ( nature ) {
    case Cercle : /* code spécifique */
    case Triangle :/* code spécifique */
    case Rectangle: /* code spécifique */
  }
}
```

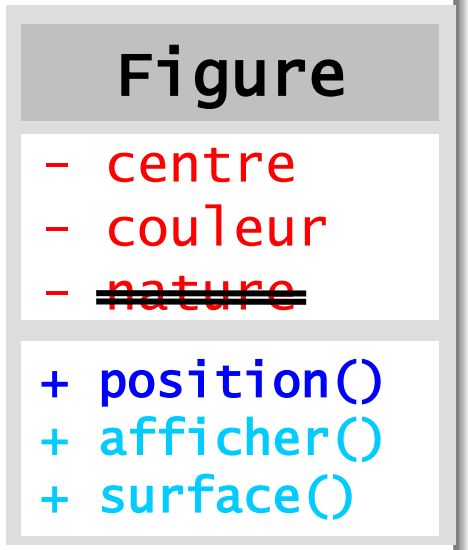


Paradigme Objet

On ajoute la notion d'héritage

➔ ***D'abord Propriétés générales***

```
abstract class Figure {
    private Point centre;
    private Couleur couleur;
    public Point position() {
        return centre;
    }
    public abstract void afficher();
    public abstract float surface();
}
```

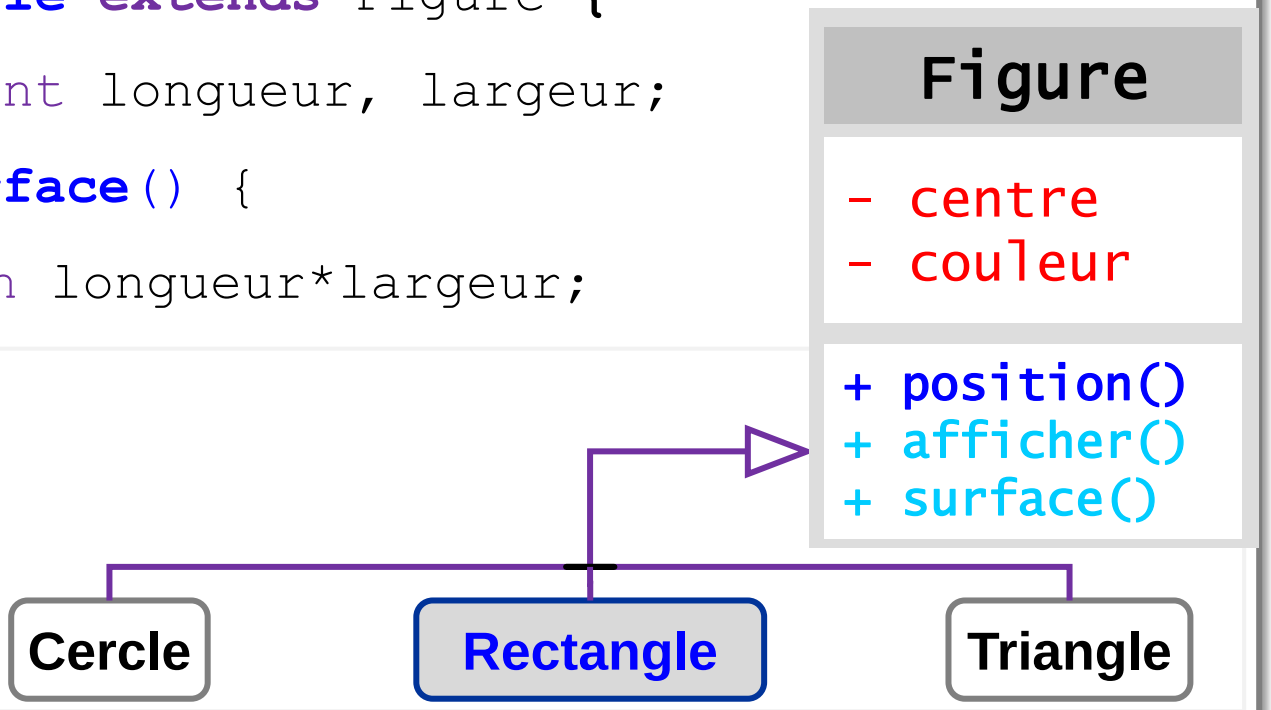


Paradigme Objet

On ajoute la notion d'héritage

➔ ***Ensuite propriétés particulières***

```
class Rectangle extends Figure {
    private int longueur, largeur;
    float surface() {
        return longueur*largeur;
    }
}
```



Paradigme Objet

Intérêt

➔ **Réduction du coût de développement**

Par Réutilisation du Code

➔ **Réduction du coût de maintenance**

Réduction des dépendances entre classes

Paradigme Objet

Démarche

➔ **Décider quelles Procédures** (2) **Que veut-on faire ?**

Meilleurs algorithmes

➔ **Décider quels Modules**

Masquer les données

Plus persistent ?

➔ **Décider quels Types**

Ensemble complet d'opérations

(1) **De quoi parle-t-on ?**

➔ **Décider quelles Classes**

Expliciter les propriétés communes à l'aide de ***l'héritage***

Paradigme Objet

Démarche

1. **De quoi parle-t-on ?**
2. Que veut-on faire ?

➔ ***Ex : Simuler un carrefour***

- Rues, Feux, Véhicules, Piétons, Accidents
- Embouteillages, Manifestations

Paradigme Objet

Démarche

① De quoi parle-t-on ?

② **Que veut-on faire ?**

➔ ***Ouvrir()*** **une porte ?**

C'est la ***Porte*** *qui s'****ouvre()*** *!! (être vivant)*

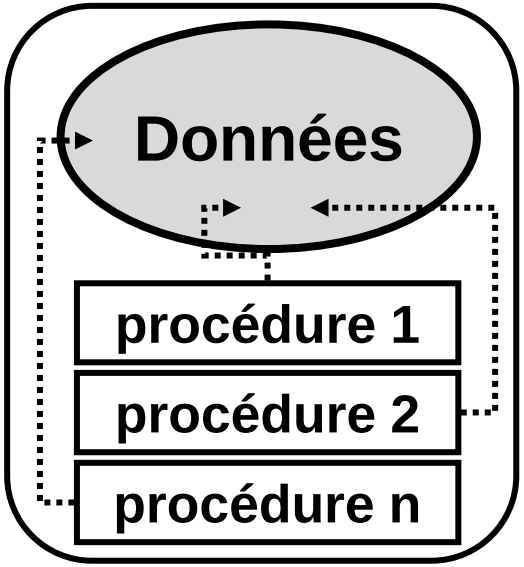
- ***Double battants***
- ***Coulissante, à relever, …***

Porte d'Ali Baba

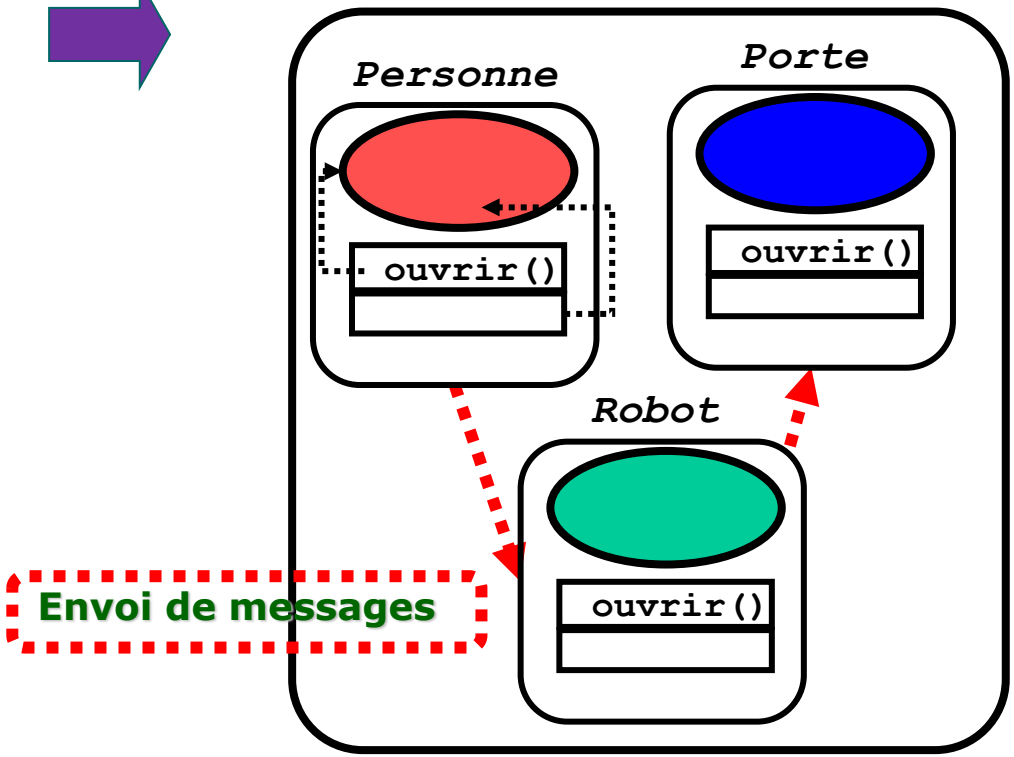
Paradigme Objet

Démarche

Application Procédurale



Application Objet



Paradigme Objet

Concepts

Encapsulation

Masquer les détails d'implémentation d'un objet, en exposant uniquement **ses fonctionnalités**

Account

Attributs

name : String

balance: double

Méthodes

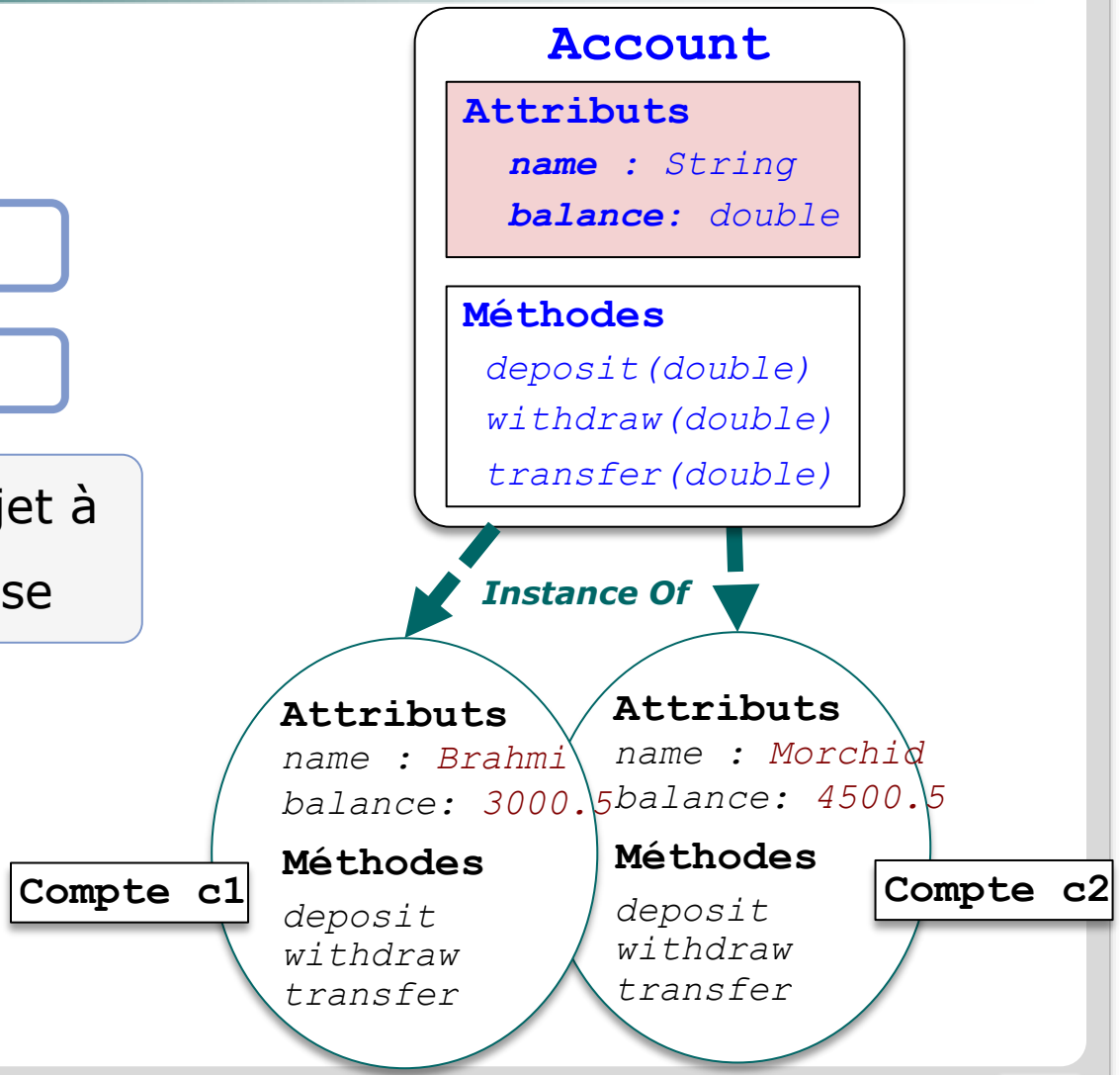
deposit(double)

withdraw(double)

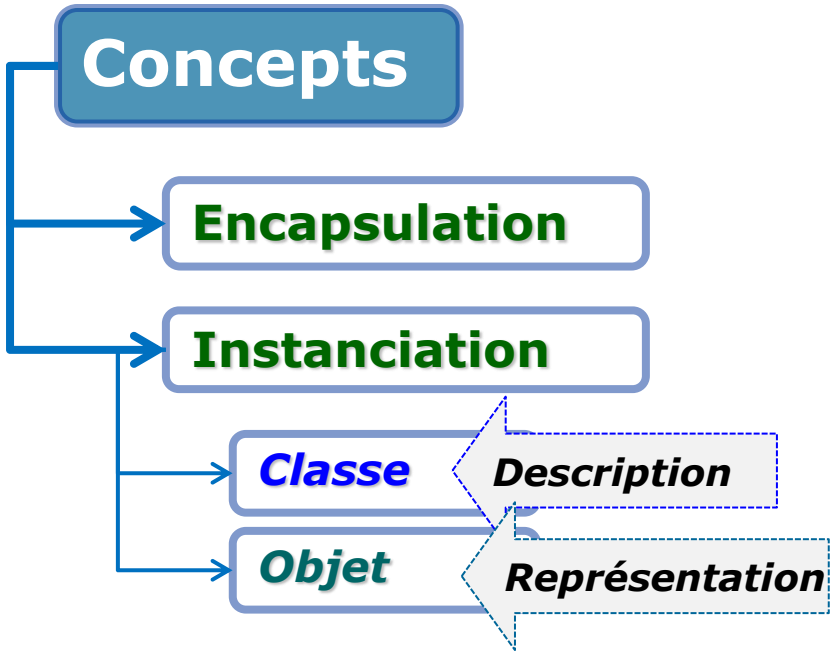
transfer(double)

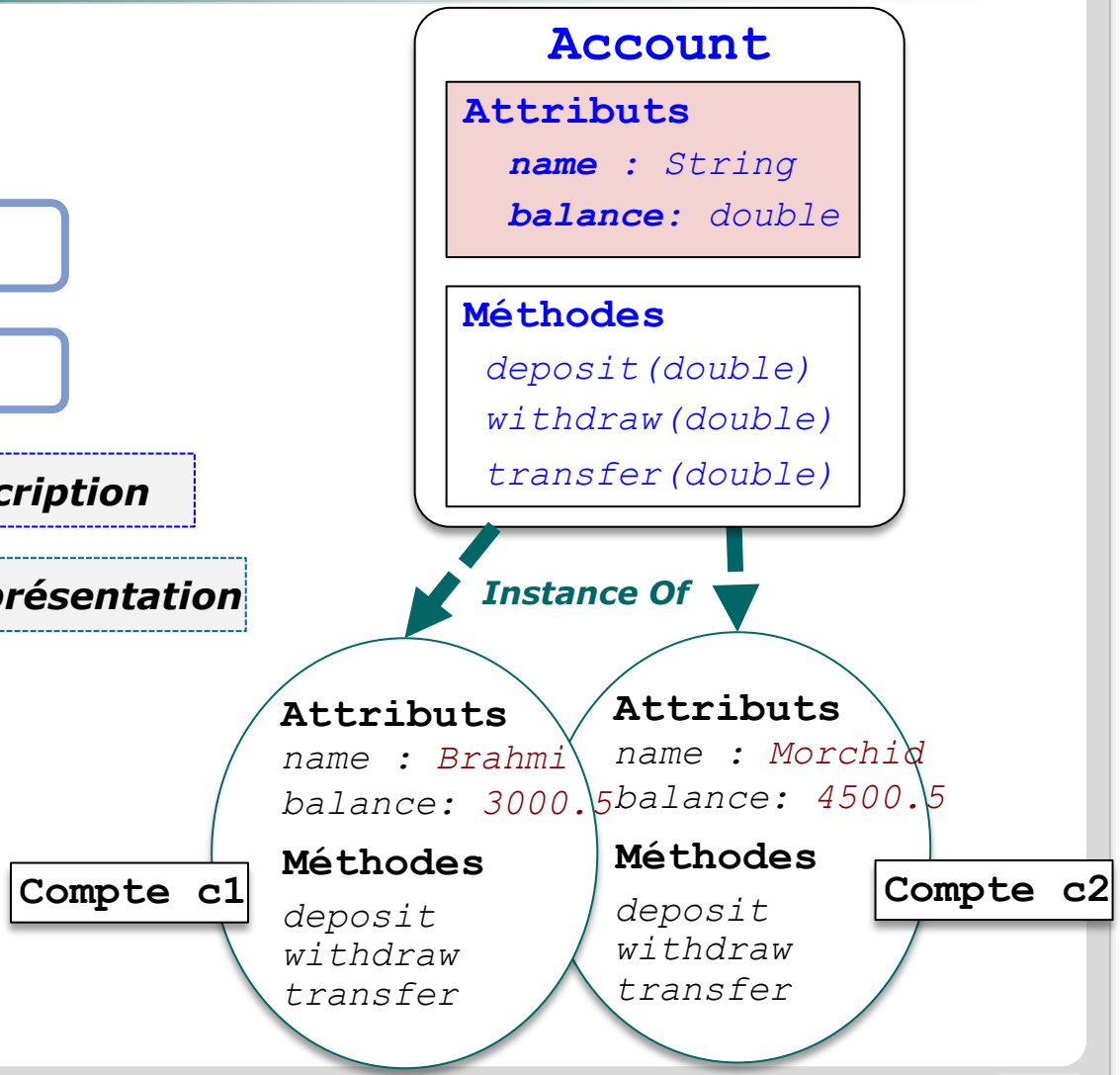
Paradigme Objet

- **Concepts**
  - **Encapsulation**
  - **Instanciation**
    - **Création** d'un objet à partir d'une classe



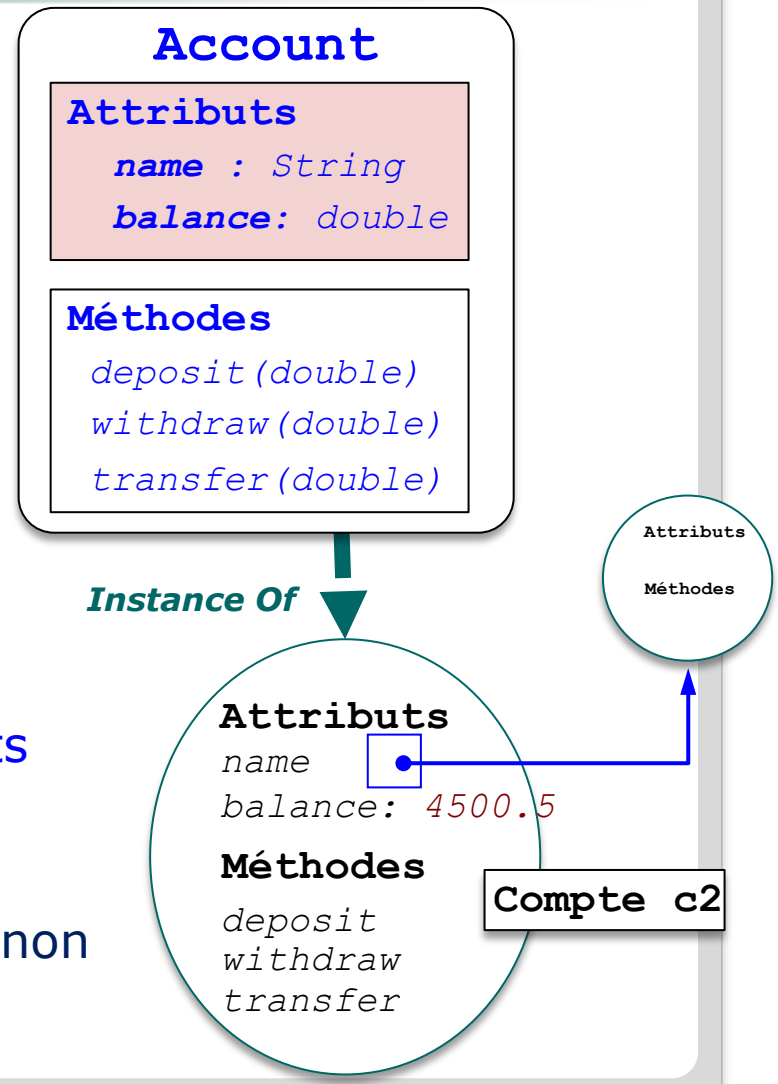
Paradigme Objet





Paradigme Objet

- **Concepts**
  - **Encapsulation**
  - **Instanciation**
    - **Création** d'un objet à partir d'une classe



- Notion de **constructeur** d'objets (dépend du **langage**)
- **Destructeur** : automatique ou non

Paradigme Objet

- **Concepts**
  - Encapsulation
  - Instanciation
  - Héritage

Héritage

➔ Exemple : Comptes bancaires

- Codes des classes se répètent

→ Plutôt réutiliser **(code & expérience)**

CurrentAccount
number name balance ***creditLimit***
+ deposit(double) + withdraw(double) + transfer(double) + bankStatement() **+ getCreditLimit()**

SavingsAccount
number name balance ***interestRate***
+ deposit(double) + withdraw(double) + transfer(double) + bankStatement() **+ addInterest()**

Héritage

1. Abstraire

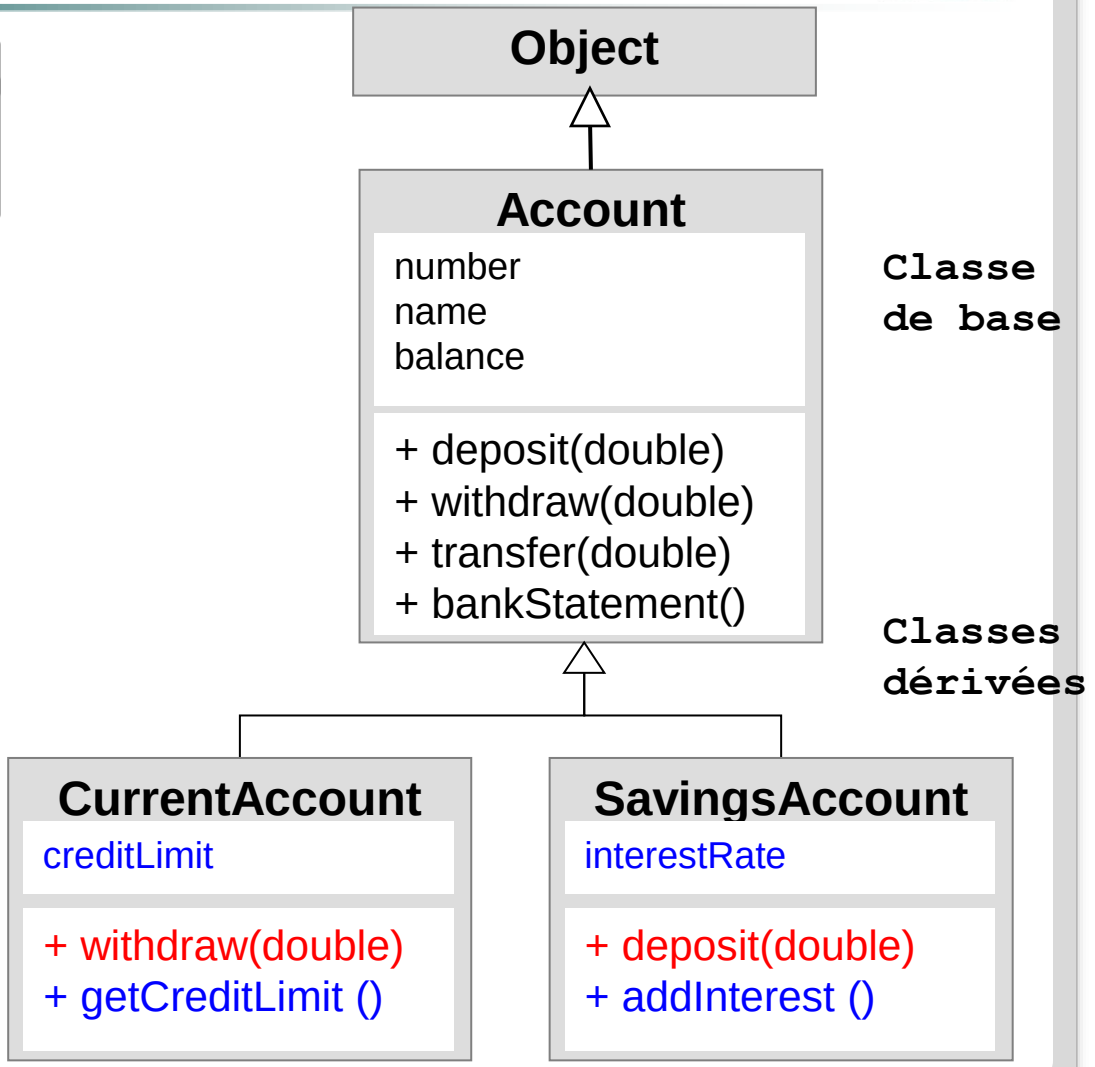
- Propriétés générales

2. Spécialiser

- Enrichissement
- Substitution

Graphe d'héritage

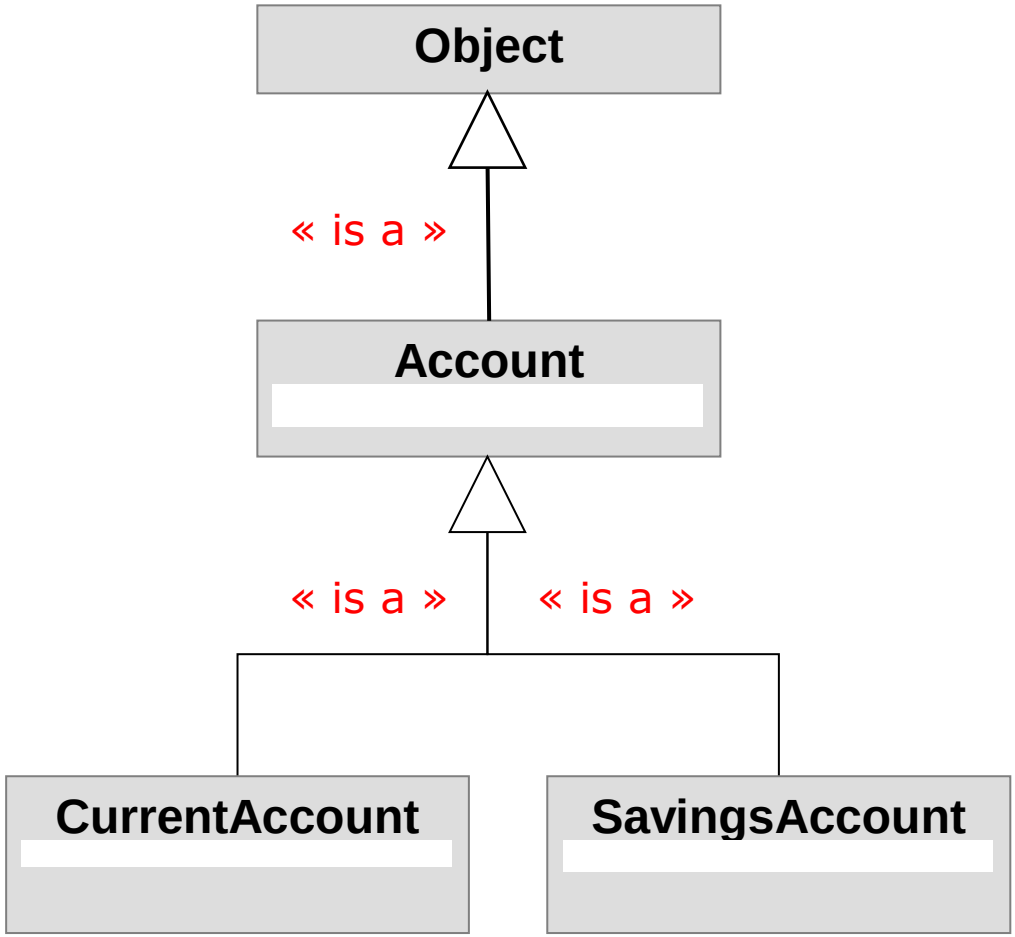
Toute classe hérite directement ou indirectement de **Object**



Héritage

Sémantique

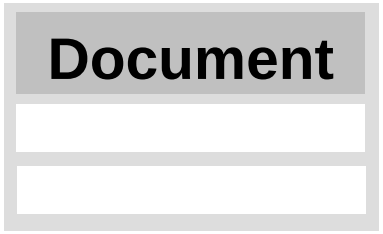
« is a »

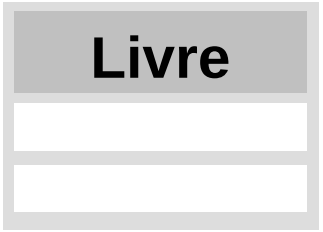


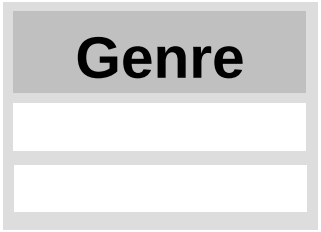
Héritage

Sémantique

« is a »



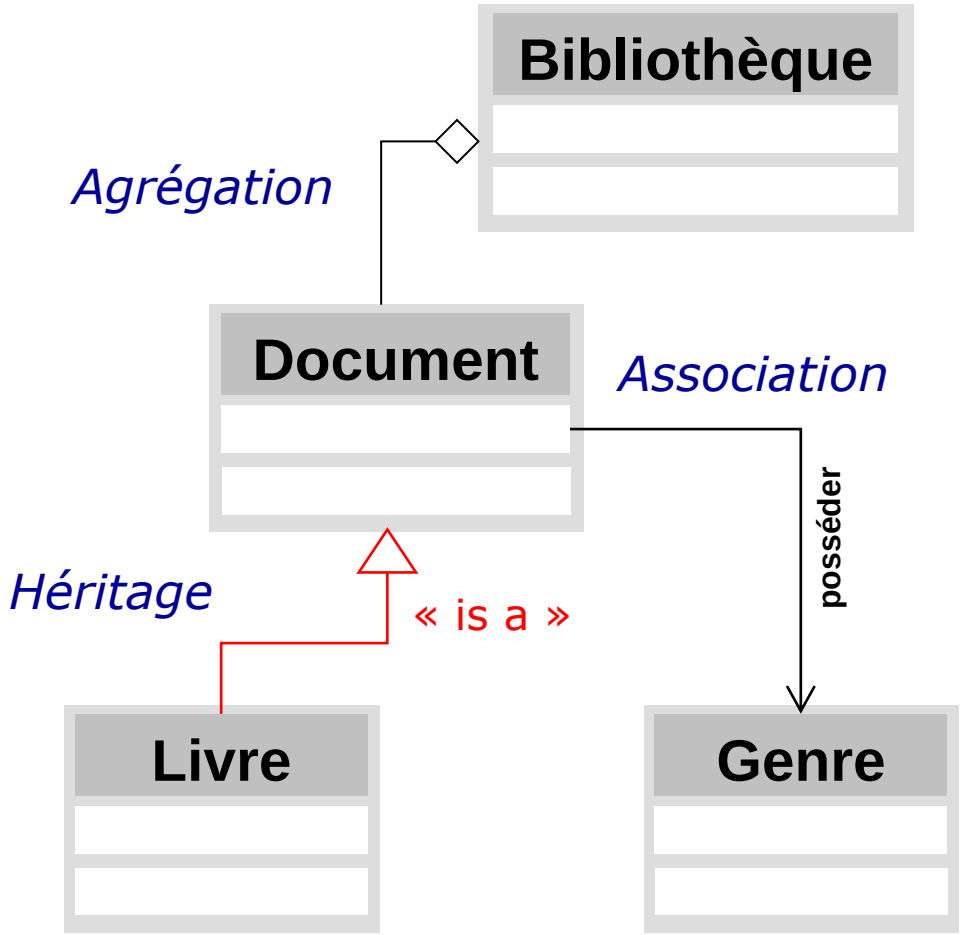




Héritage

Sémantique

« is a »



ENSIAS

Héritage

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

Sémantique

« is a »

Héritage

Sémantique

« is a »

Exercices d'examen

Parmi les pairs suivants lesquelles représentent une relation « **is-a** » ?

.....

A	*Oiseau/Nid*	**C**	*Oiseau/Vol*
B	*Canard / Oiseau*	**D**	*Bec / Oiseau*

Exercices d'examen

Parmi les pairs suivants lesquelles représentent une relation « **is-a** » ?

.....**B**.....

A	*Oiseau/Nid*	**C**	*Oiseau/Vol*
B	*Canard / Oiseau*	**D**	*Bec / Oiseau*

Héritage simple

Définition

Toute classe hérite d'**une seule** super-classe

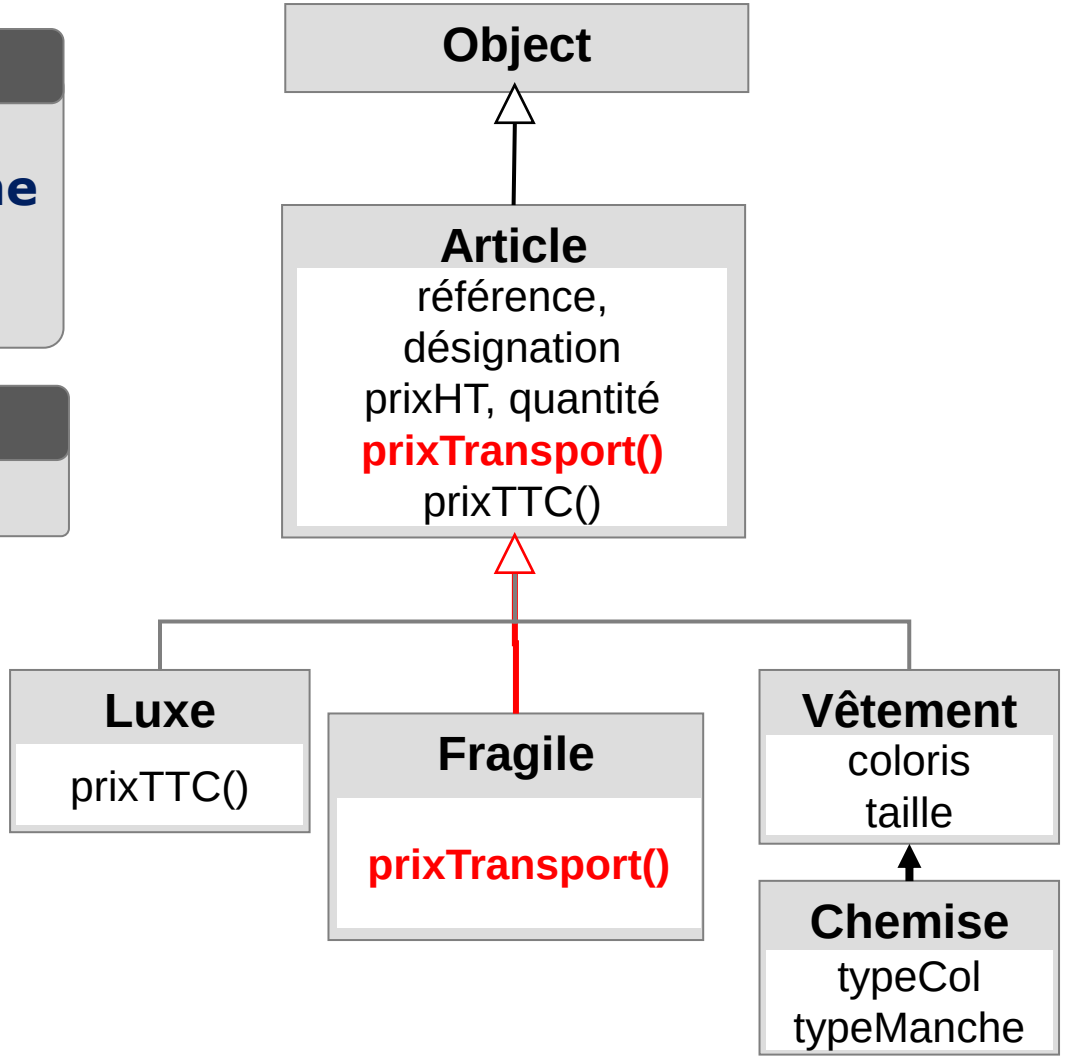
Graphe d'héritage

- Arbre

Choix d'une méthode :

prixTransport() ?

Evident



Héritage multiple

Définition

Une classe peut hériter de **plusieurs** classes

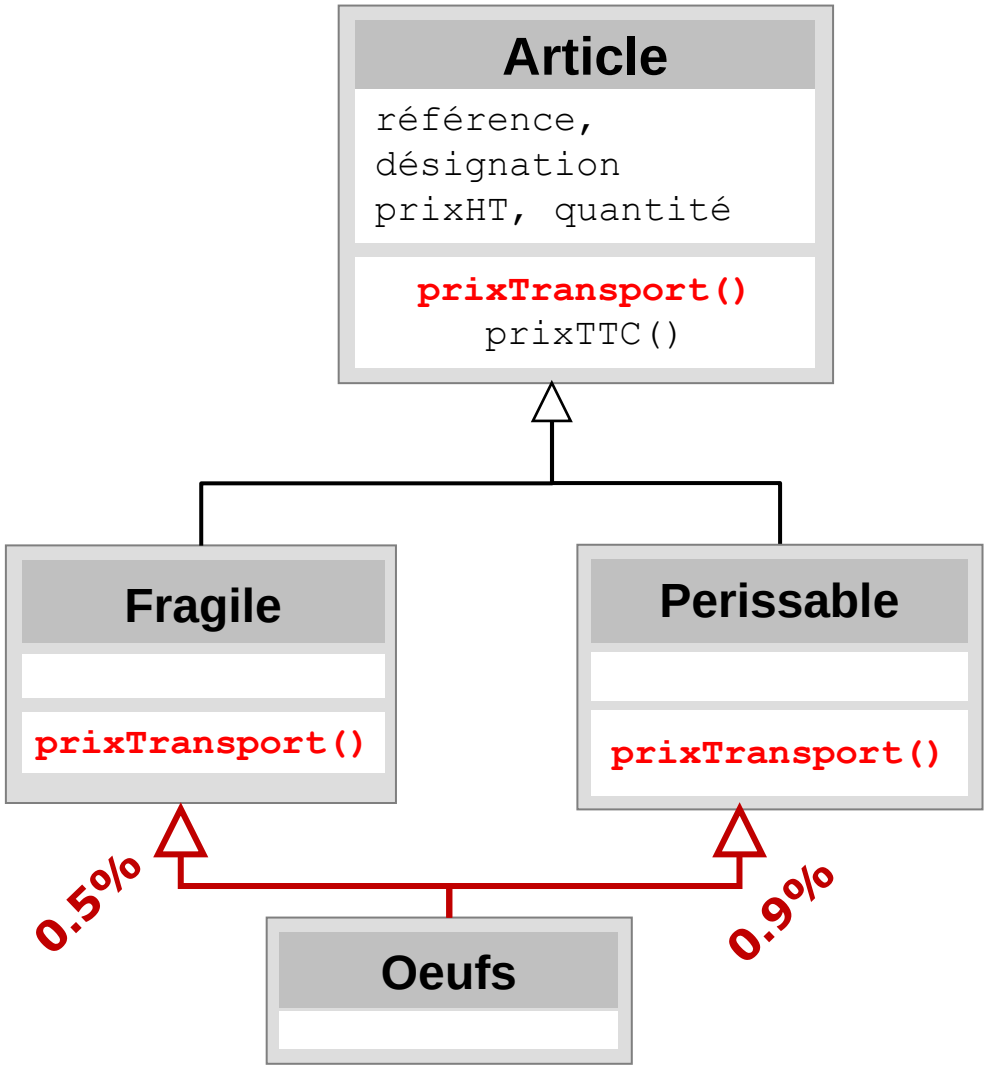
Graphe d'héritage

- Graphe

Prix de transport des œufs ?

↓

Conflit



Héritage multiple

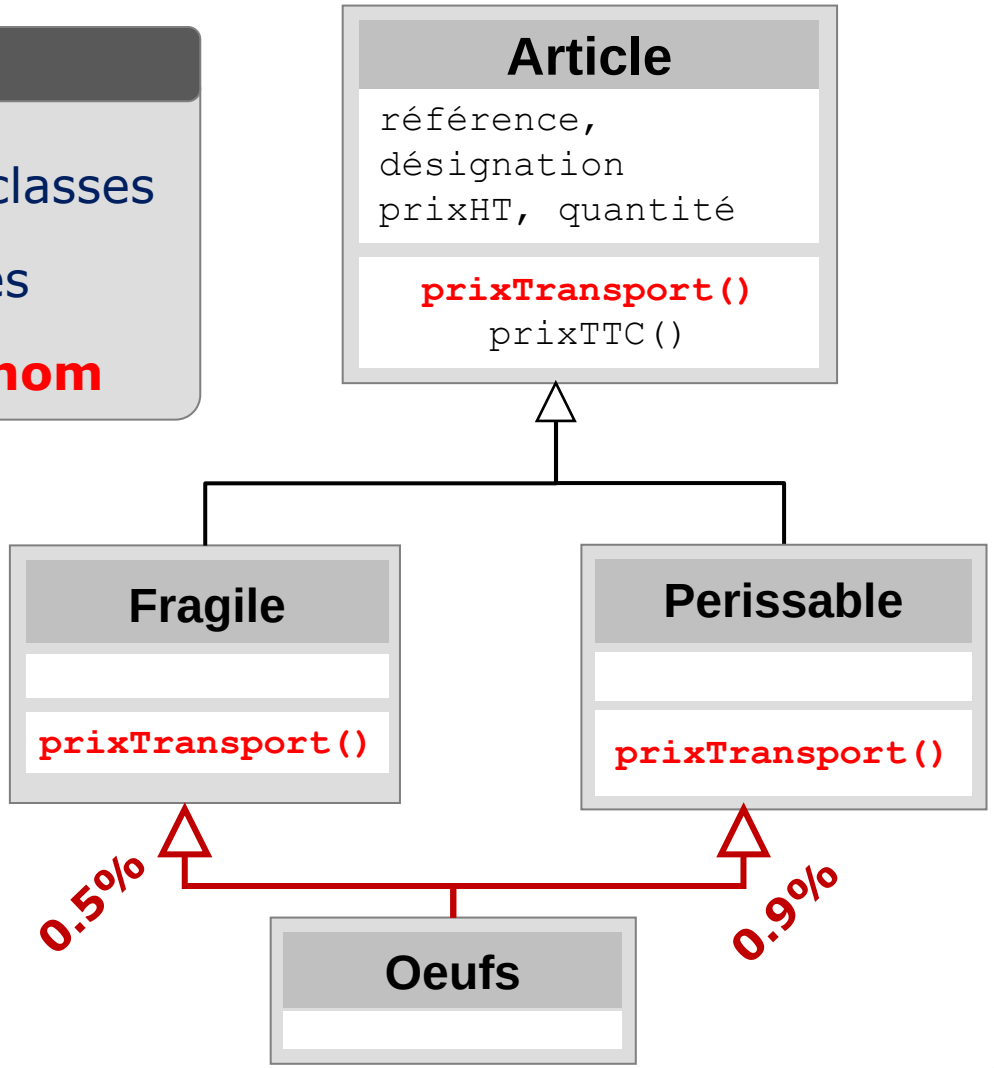
1. Conflit de nommage

Au moins deux des super classes d'une classe définissent des propriétés avec le **même nom**

Prix de transport des œufs ?

↓

Conflit



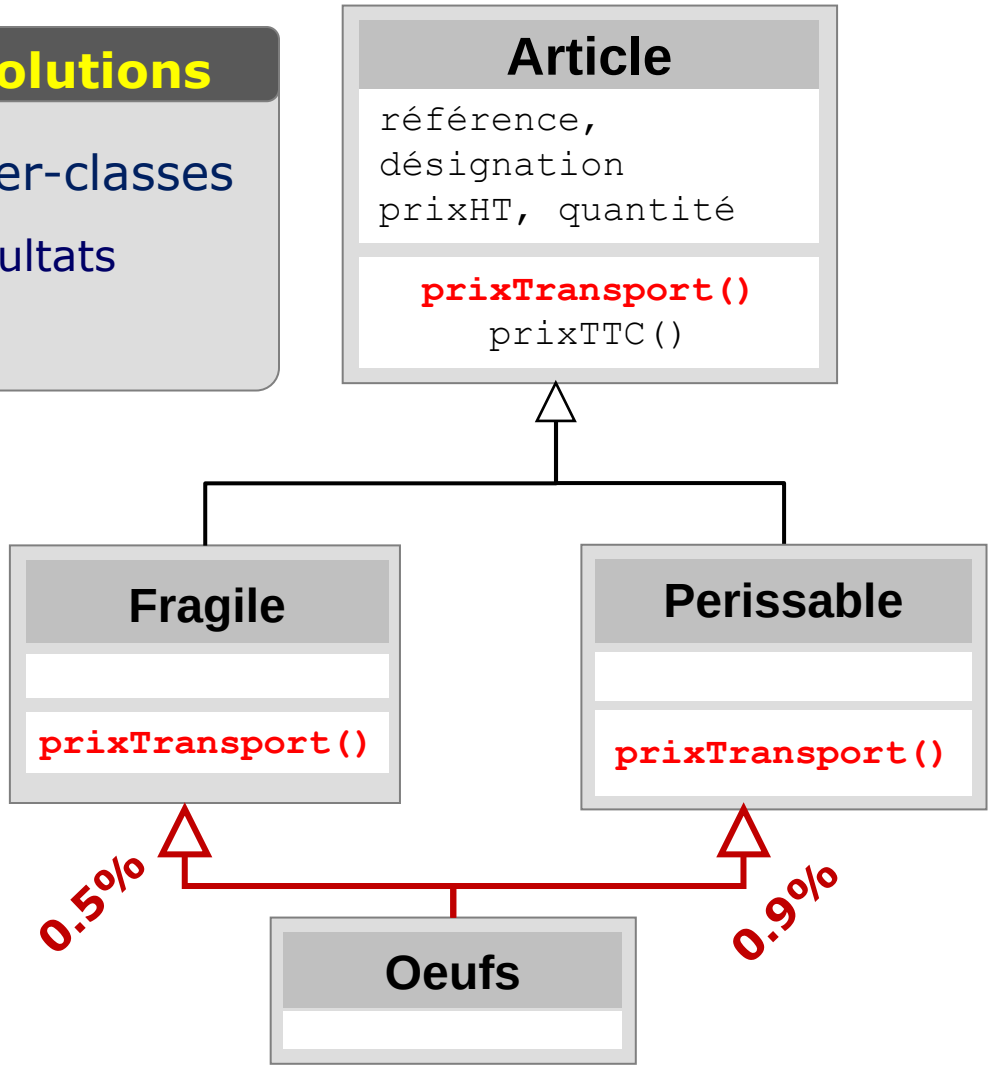
Héritage multiple

1. Conflit de nommage-solutions

a. Fournir un ordre des super-classes

Pas très pratique : les « résultats dépendent de ce choix »

```
class Oeufs :
  public Fragile,
  public Perissable
```

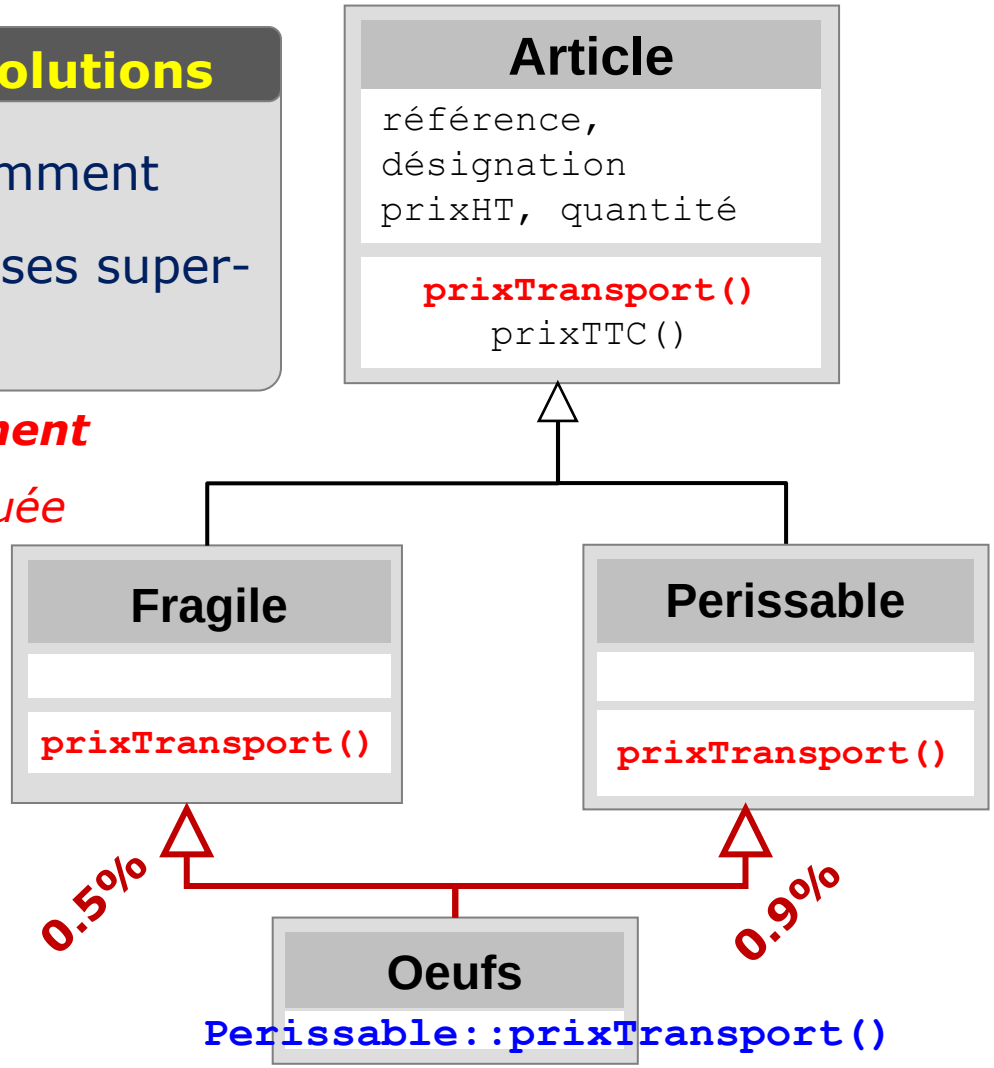


Héritage multiple

1. Conflit de nommage-solutions

b. La sous-classe définit comment utiliser les propriétés de ses super-classes

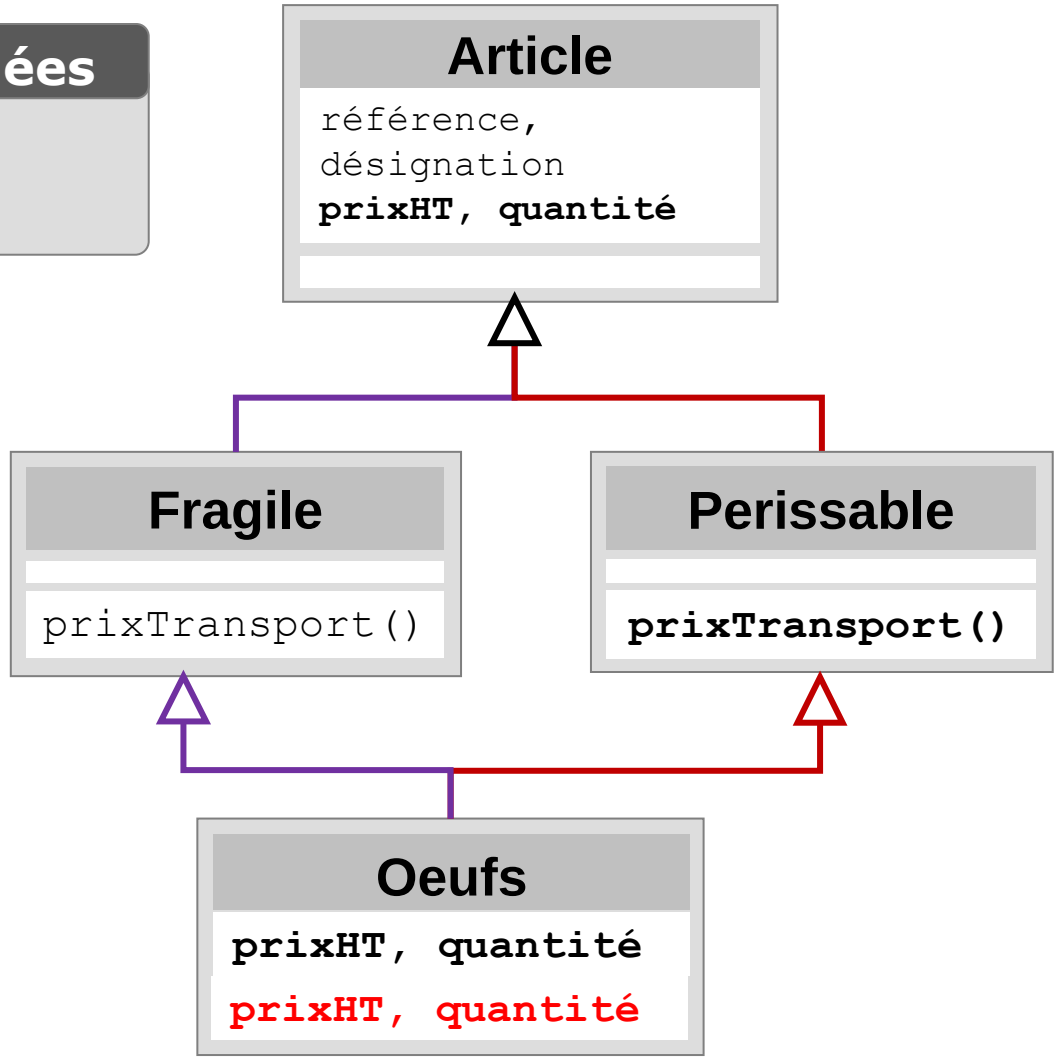
*La sous-classe doit **explicitement redéfinir la propriété** impliquée dans le conflit de noms*



Héritage multiple

2. Redondance de données

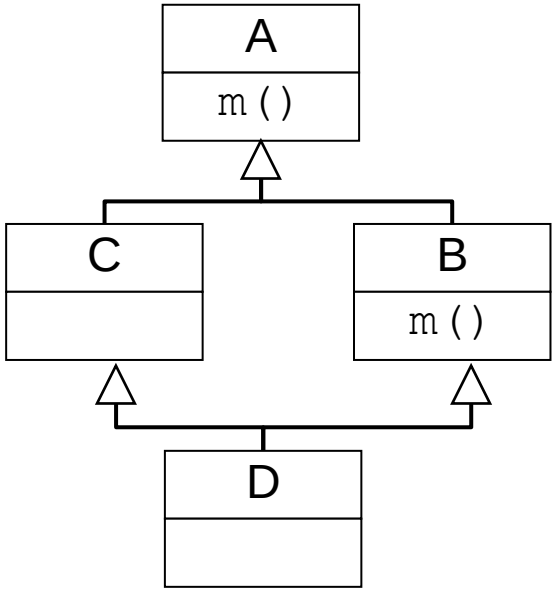
Héritage en diamant



Exercices d'examen

Dans le graphe d'héritage suivant

A	*Il y'a un conflit d'héritage*	**C**	*La méthode m() est héritée dans C*
B	*La méthode m() est héritée dans B*	**D**	*Il y'a un problème de redondance de données*

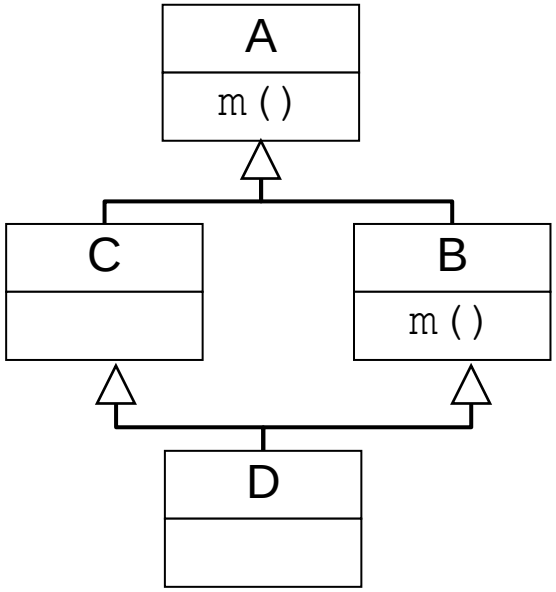


Exercices d'examen

Dans le graphe d'héritage suivant

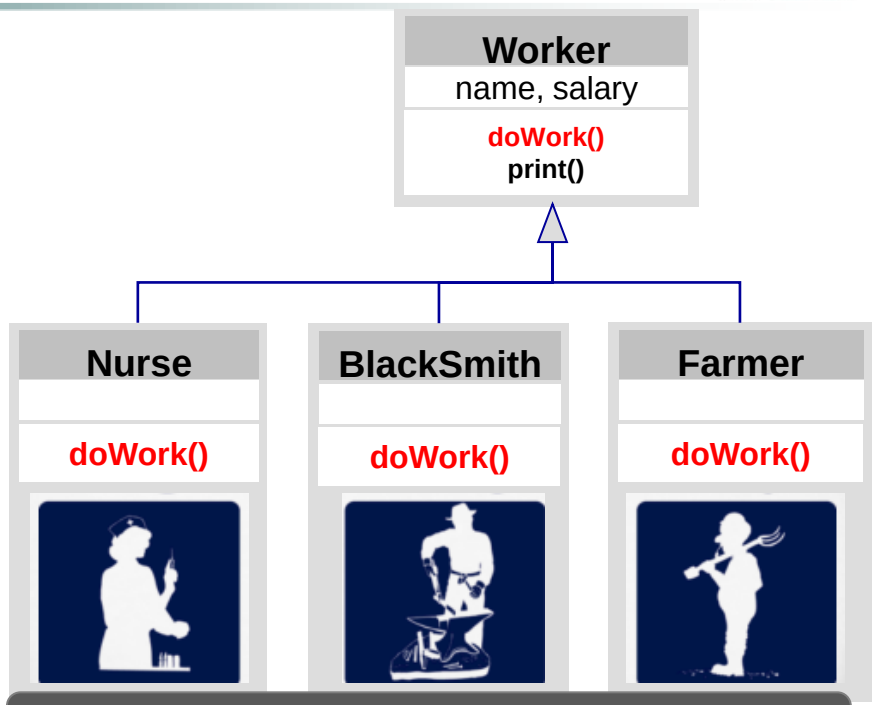
A C D

A	*Il y'a un conflit d'héritage*	C	*La méthode m() est héritée dans C*
B	*La méthode m() est héritée dans B*	D	*Il y'a un problème de redondance de données*



Polymorphisme

- Concepts
  - Encapsulation
  - Instanciation
  - Héritage
  - **Polymorphisme**
    - ***Liaison dynamique***

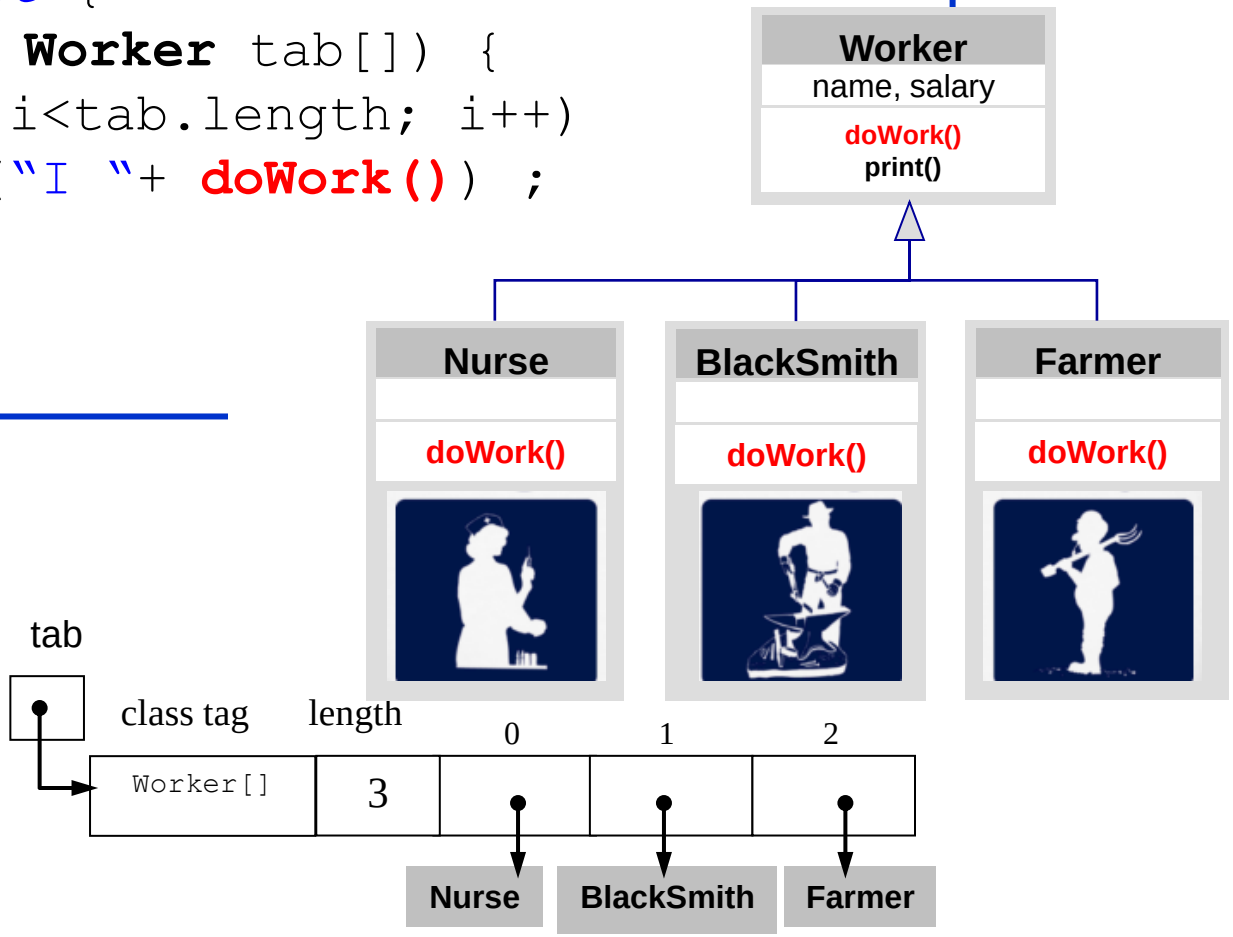


Définition

- Capacité pour une entité de prendre plusieurs formes
- *Même **nom de méthode** mais codes différents*

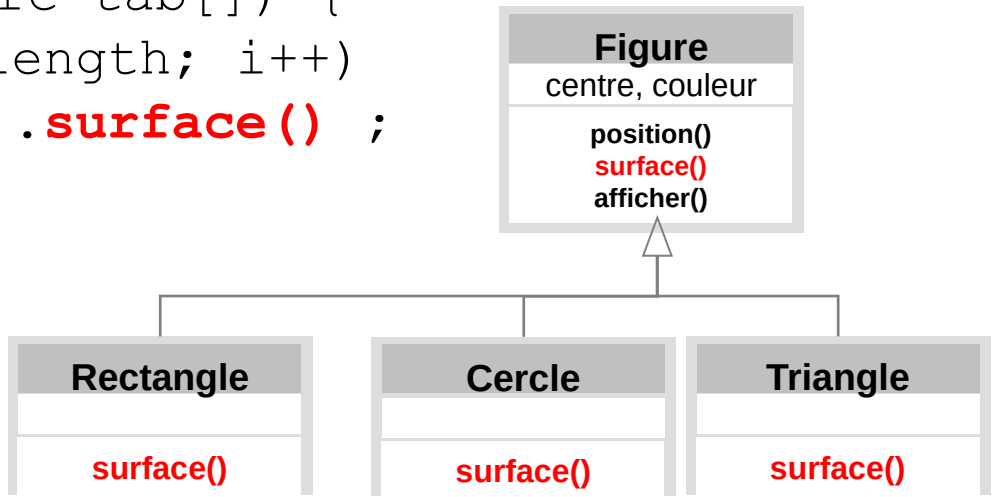
Polymorphisme

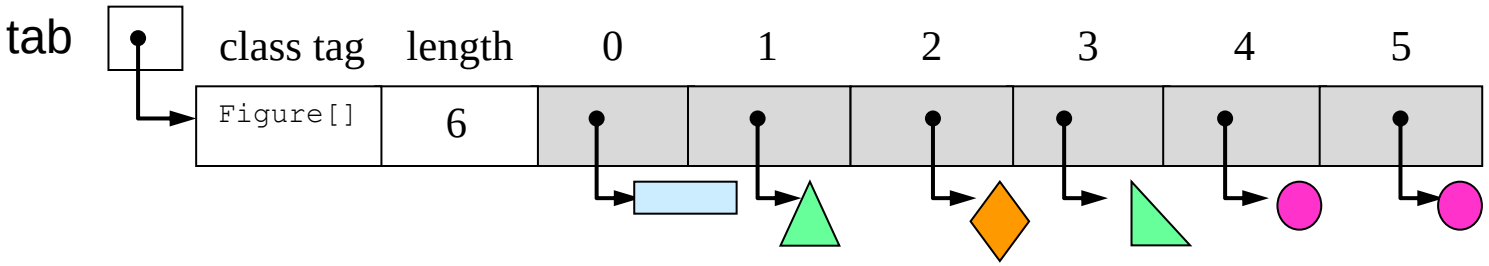
```
class WorkerTest {
   void doWork( Worker tab[]) {
        for(i=0; i<tab.length; i++)
            print("I "+ doWork()) ;
        return ;
   }
}
```



Polymorphisme

```
class SurfaceTest {
   float surface( Figure tab[]) {
       for(i=0; i<tab.length; i++)
           s = s+ tab[i].surface() ;
       return s;
   }
}
```





Polymorphisme

Méthode héritée

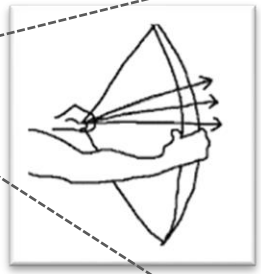
- **méthode1()**
- Réutilisée

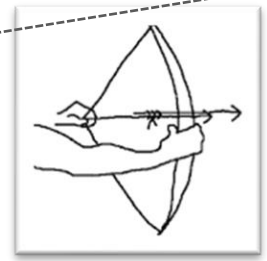
Méthode Redéfinie

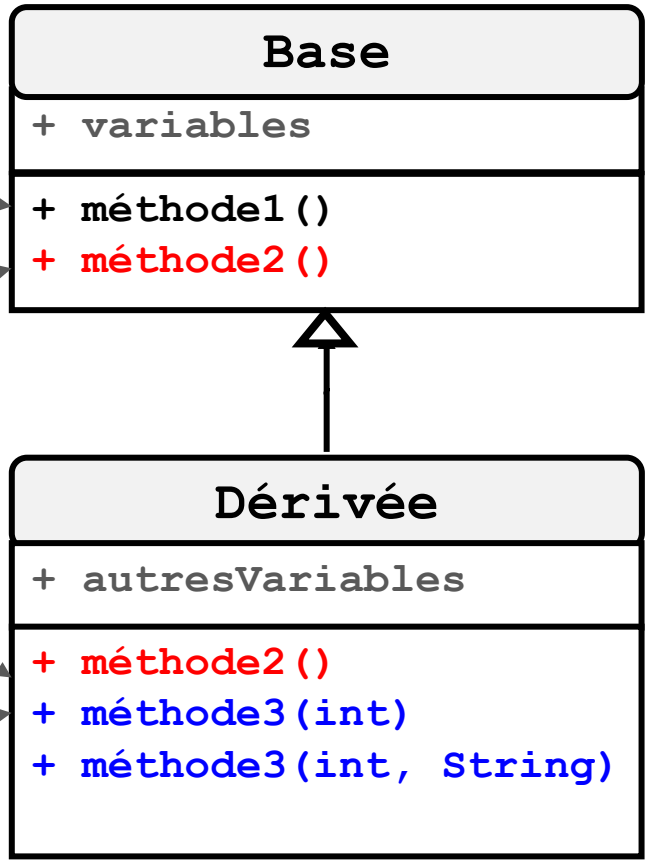
- **méthode2()**
- Polymorphisme d'héritage
- *Overloading*

Méthode Surchargée

- **méthode3()**
- Polymorphisme paramétrique
- *Overriding*







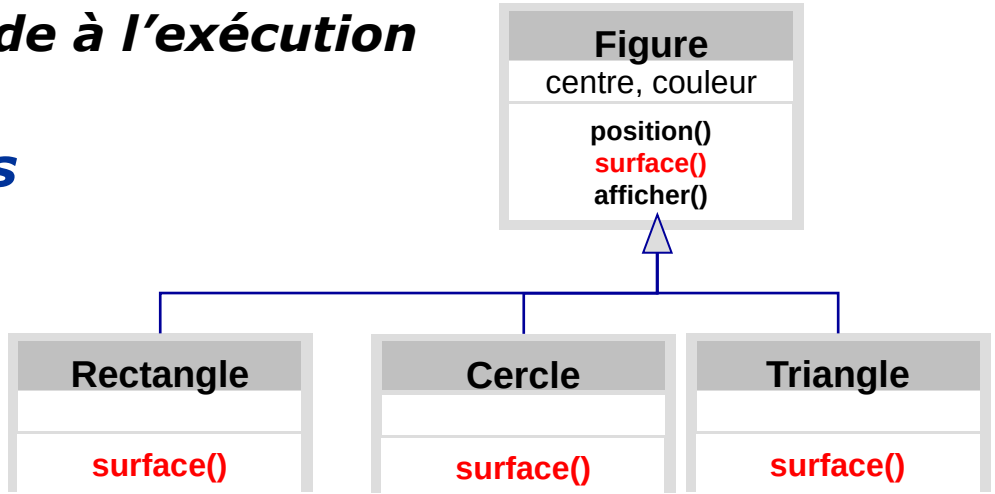
Liaison dynamique

➔ **Mise en œuvre du polymorphisme**

Code lié à la méthode à l'exécution

➔ **Méthodes virtuelles**

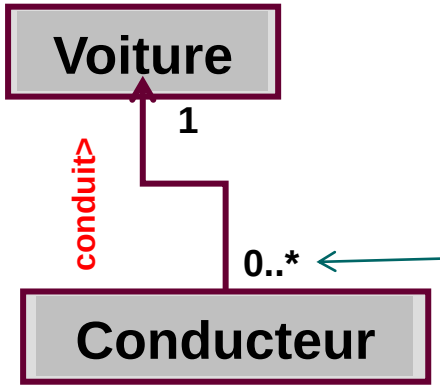
C++, Eiffel, Java



Autres relations

Association uses >

➔ Les objets sont sémantiquement liés



Cardinalité

Association

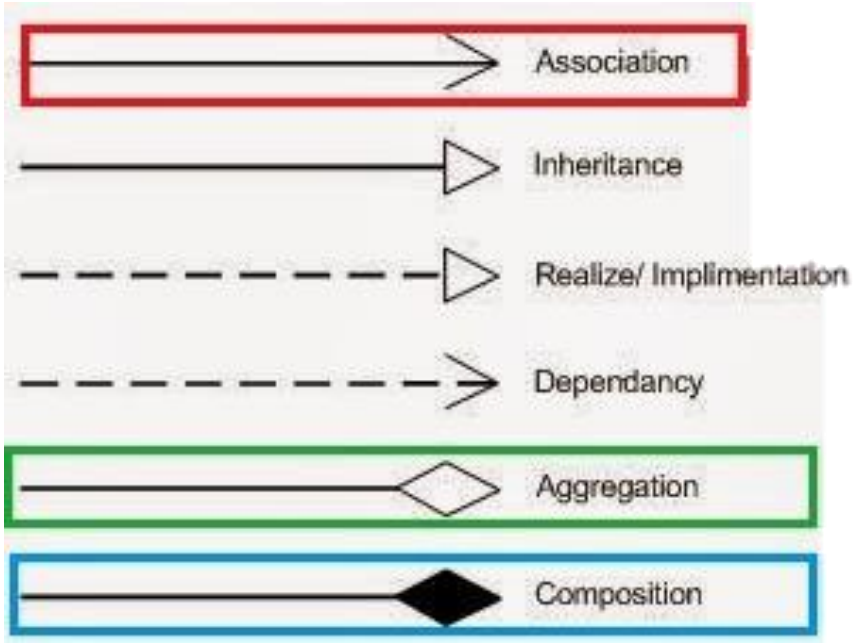
Pour une Voiture

- **elle peut avoir zéro ou plusieurs Conducteurs (concurrents) associés**

Autres relations

Agrégation **Composition** **«is part of»**

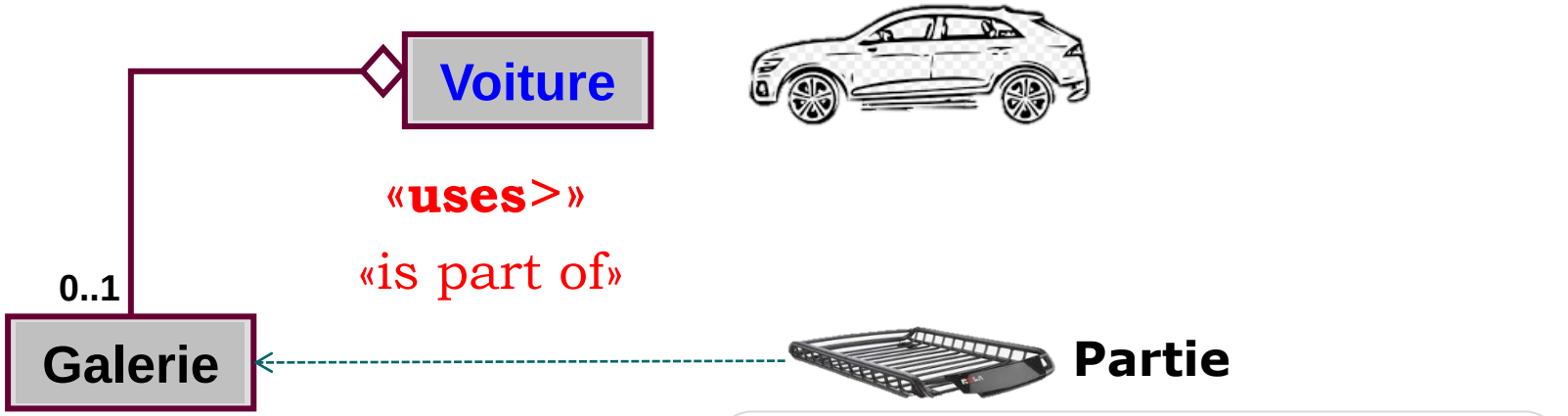
UML Notations



Autres relations

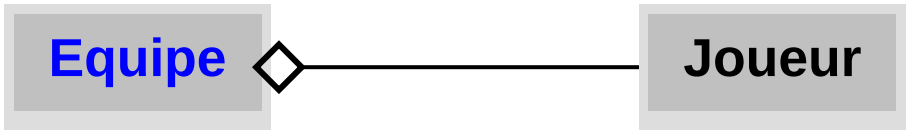
Agrégation

➔ Les cycles de vie sont indépendants



Agrégation

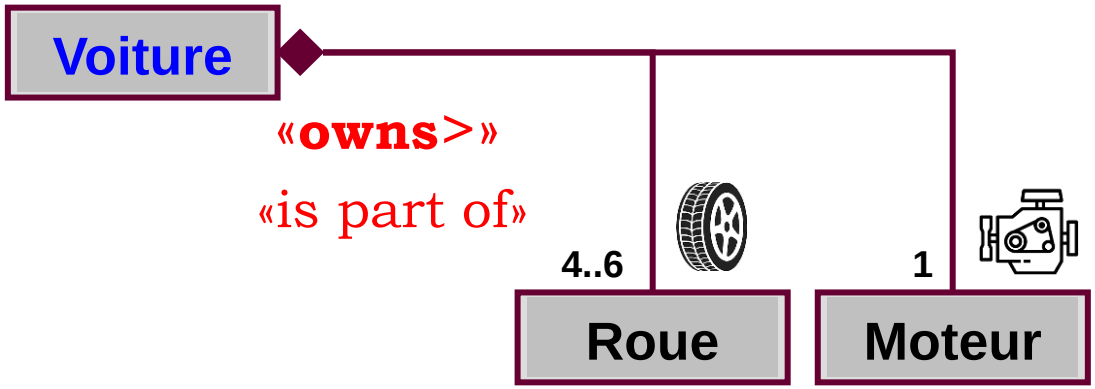
- Peut exister sans l'ensemble
- N'est pas créée par l'ensemble



Autres relations

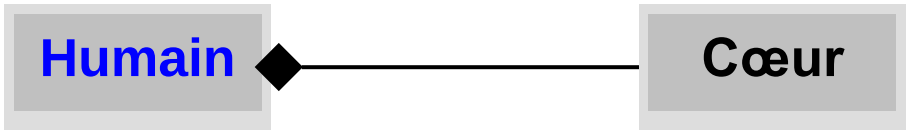
Composition

➔ **Agrégation mais avec relation plus forte**



– **Le Moteur dure la vie de la Voiture**

(Ne peut pas fonctionner individuellement lorsque la voiture est détruite)



ENSIAS

Paradigme Objet

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

Langages OO

Langage Statique

- **Typage** = fort
- **Liaison statique** = oui
- **Liaison dynamique** = méthodes virtuelles
- **Usage** = applications finies

Langage Dynamique

- **Typage** = faible
- **Liaison statique** = non
- **Liaison dynamique** = oui
- **Usage** = prototypage, simulation graphisme

Simula 1960 — **Eiffel** 1980 — **C++** 1983 — **Java** 1995 — **SmallTalk** 1980 — **CLOS** 1980 — **LOOPS** 1983


Programmation
Orientée Objet
2023-2024