CHAPITRE 15

Héritage multiple

POLYTECHNIQUE MONTRÉAL

Notions importantes liées à l'héritage multiple:

- Ambiguïté potentielle des opérations,
- Perte du polymorphisme,
- Structure des objets construits à partir de classes définies par héritage multiple,
- Structure d'héritage multiple potentiellement défectueuse,
- Structure d'héritage multiple potentiellement valide.

POLYTECHNIQUE MONTRÉAL

Différents langages et l'héritage multiple

Langages supportant l'HM

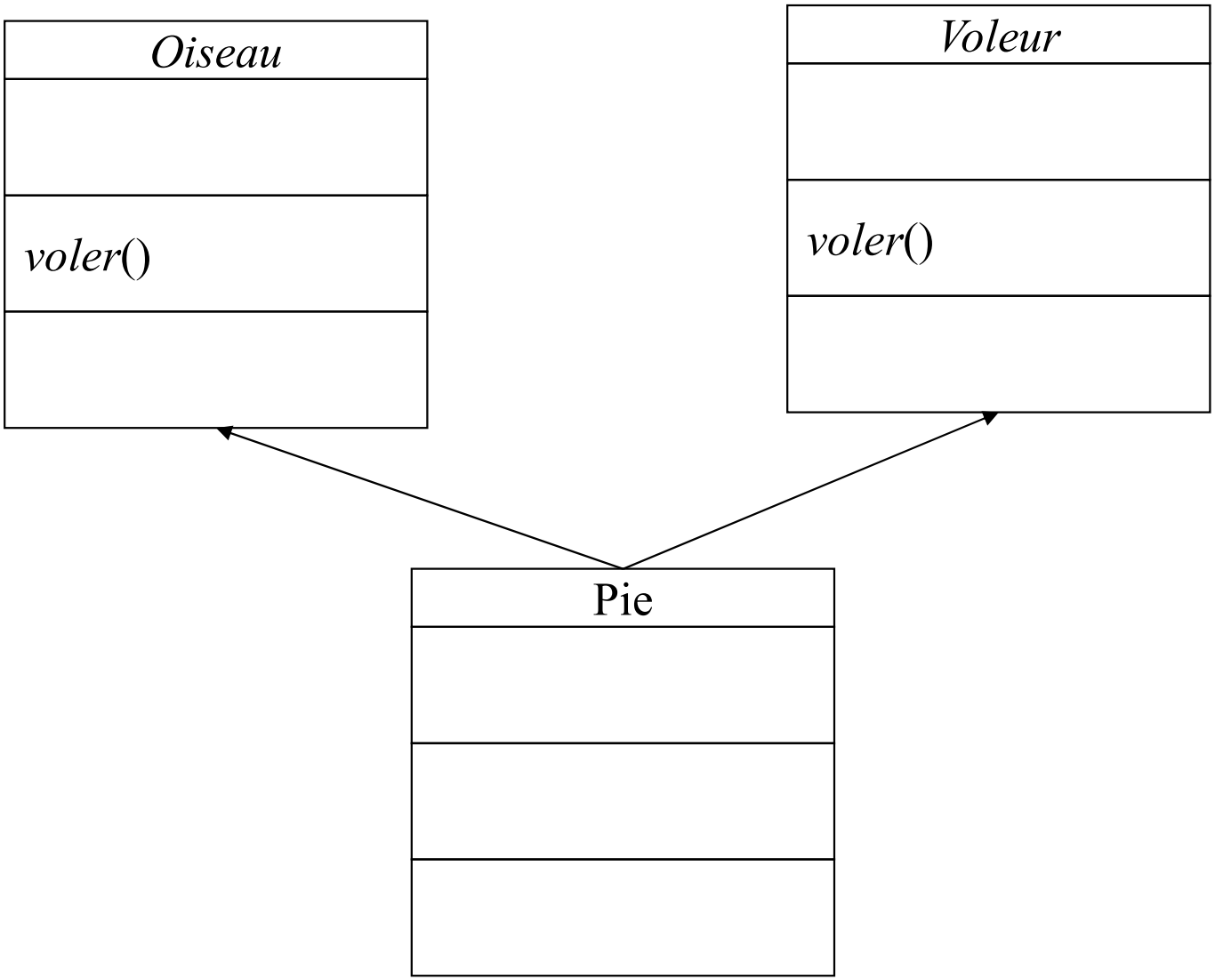
- C++
- Eiffel
- Common Lisp Object System (CLOS)
- Java (interfaces seulement)
- C# (interfaces seulement)
- Python

Langages ne supportant pas l'HM

- Smalltalk
- Objective C
- Object Pascal
- JavaScript

POLYTECHNIQUE MONTRÉAL

Ambiguïté des opérations



POLYTECHNIQUE MONTRÉAL

Ambiguïté des opérations

```
class Oiseau {
public:
  virtual void voler();
  ...
};
class Voleur {
public:
  virtual void voler();
  ...
};
```

```
class Pie :
  public Oiseau,
  public Voleur
{
public:
  ...
};
```

POLYTECHNIQUE MONTRÉAL

Ambiguïté des opérations

```
Pie* unePie = new Pie;


unePie->voler();        // erreur: appel ambigu
unePie->Oiseau::voler(); // Ok
unePie->Voleur::voler();  // Ok
```

Le fait de déclarer une des deux méthodes voler privée ne changerait rien à l'ambiguïté.

POLYTECHNIQUE MONTRÉAL

Perte du polymorphisme

Qualifier explicitement la fonction à appeler n'est pas une solution:

➔ Les fonctions virtuelles ne sont plus virtuelles

```
Class PieGrieche : public Pie {
 public:
   virtual void voler(); // lequel des voler est
                         // surchargé ??
   ...
 };
```

Perte du polymorphisme

Pie* unePie = new **PieGrieche**;

unePie->voler(); // erreur: toujours ambigu

unePie->Oiseau::voler(); // appelle Oiseau::voler

unePie->Voleur::voler(); // appelle Voleur::voler

Il n'y a pas de moyen simple d'appeler la nouvelle fonction.

Perte du polymorphisme

Mais il y a un autre problème:

➔ Les deux fonctions voler() sont virtuelles, comment faire pour les surcharger toutes les deux ?

Une modification au langage a failli être adoptée pour permettre de renommer des méthodes virtuelles.

Finalement une astuce a été trouvée:

➔ Introduire deux classes auxiliaires.

Introduction de classes auxiliaires

```
Class OiseauAux : public Oiseau
{
public:
  virtual void planer() = 0;

  virtual void voler()
  {
    planer();
  };

  ...
};
```

Introduction de classes auxiliaires

```
Class VoleurAux : public Voleur
{
public:
  virtual void derober() = 0;

  virtual void voler()
  {
    derober();
  };

  ...
};
```

POLYTECHNIQUE MONTRÉAL

Introduction de classes auxiliaires

```
Class Pie : public OiseauAux, public VoleurAux
{
public:
    virtual void planer();
    virtual void derober();
    ...
};
```

POLYTECHNIQUE MONTRÉAL

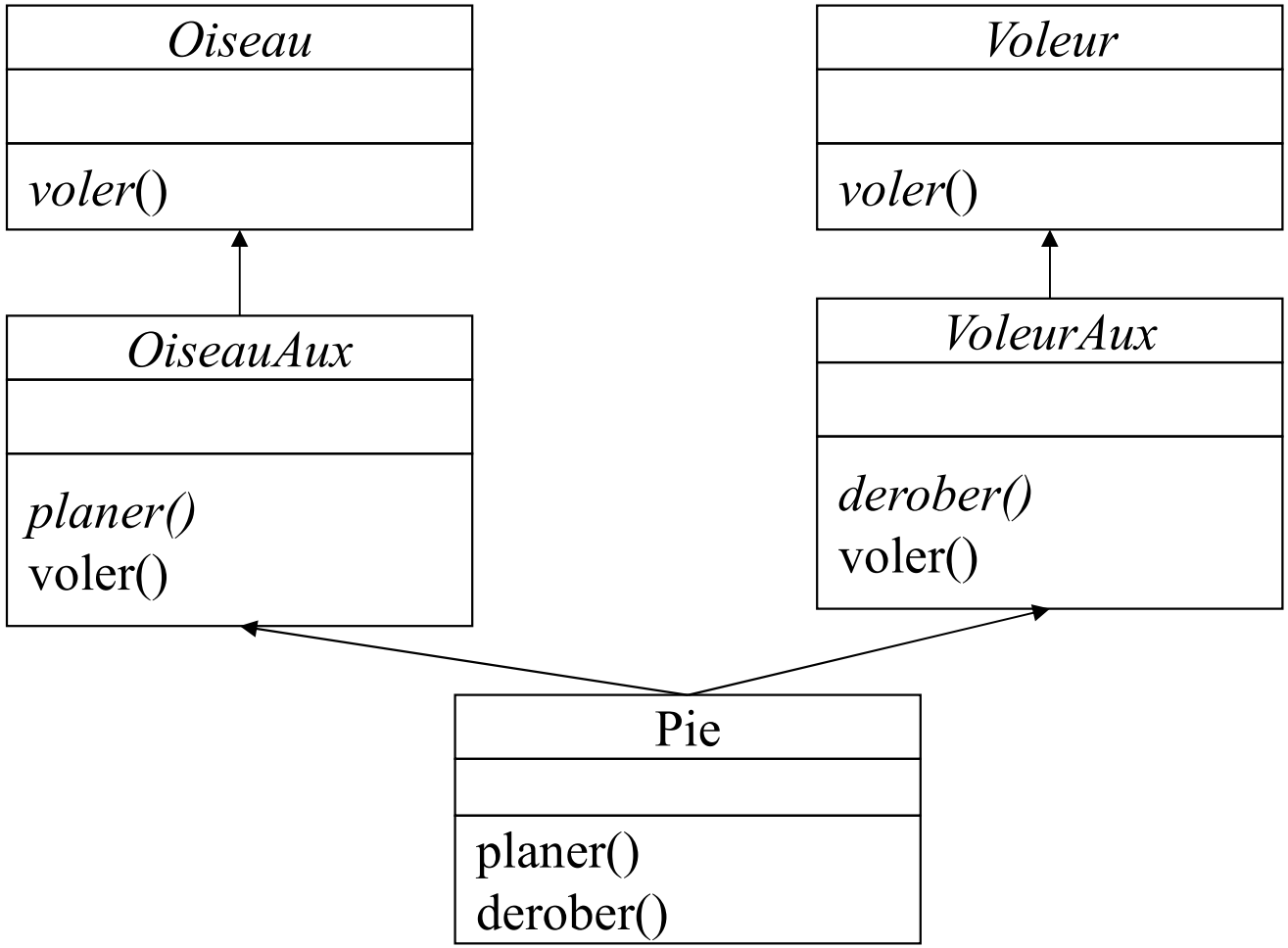
Introduction de classes auxiliaires

```
Pie* unePie = new Pie;
Oiseau* unOiseau = unePie;
Voleur* unVoleur = unePie;

unOiseau->voler(); // appelle la fonction
                   // OiseauAux::voler() qui appelle
                   // ensuite Pie::planer()

unVoleur->voler(); // appelle la fonction
                   // VoleurAux::voler() qui appelle
                   // ensuite Pie::derober()
```

POLYTECHNIQUE MONTRÉAL

Introduction de classes auxiliaires

Structure des classes permettant de renommer des méthodes dont le nom est en conflit dans deux classes de base en conservant le polymorphisme:



POLYTECHNIQUE MONTRÉAL

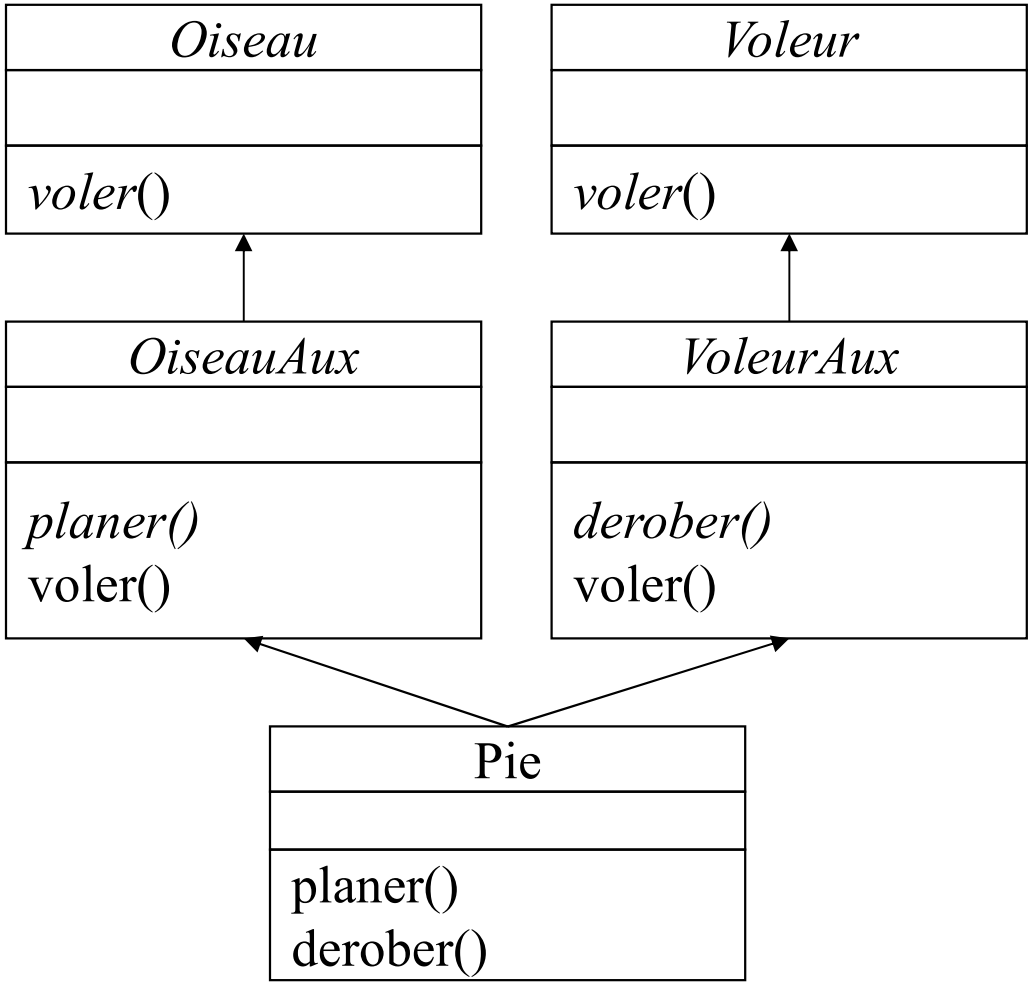
Perte du polymorphisme

Cette structure de classes permet de renommer les deux méthodes voler issues des classes Oiseau et Voleur dans la classe Pie.

Cette solution implique l'introduction de deux nouvelles classes auxiliaires qui ne sont ni des abstractions du domaine du problème, ni des abstractions du domaine de la solution: ce sont des dispositifs d'implémentation.

POLYTECHNIQUE MONTRÉAL

Structure en mémoire des objets construits par héritage multiple



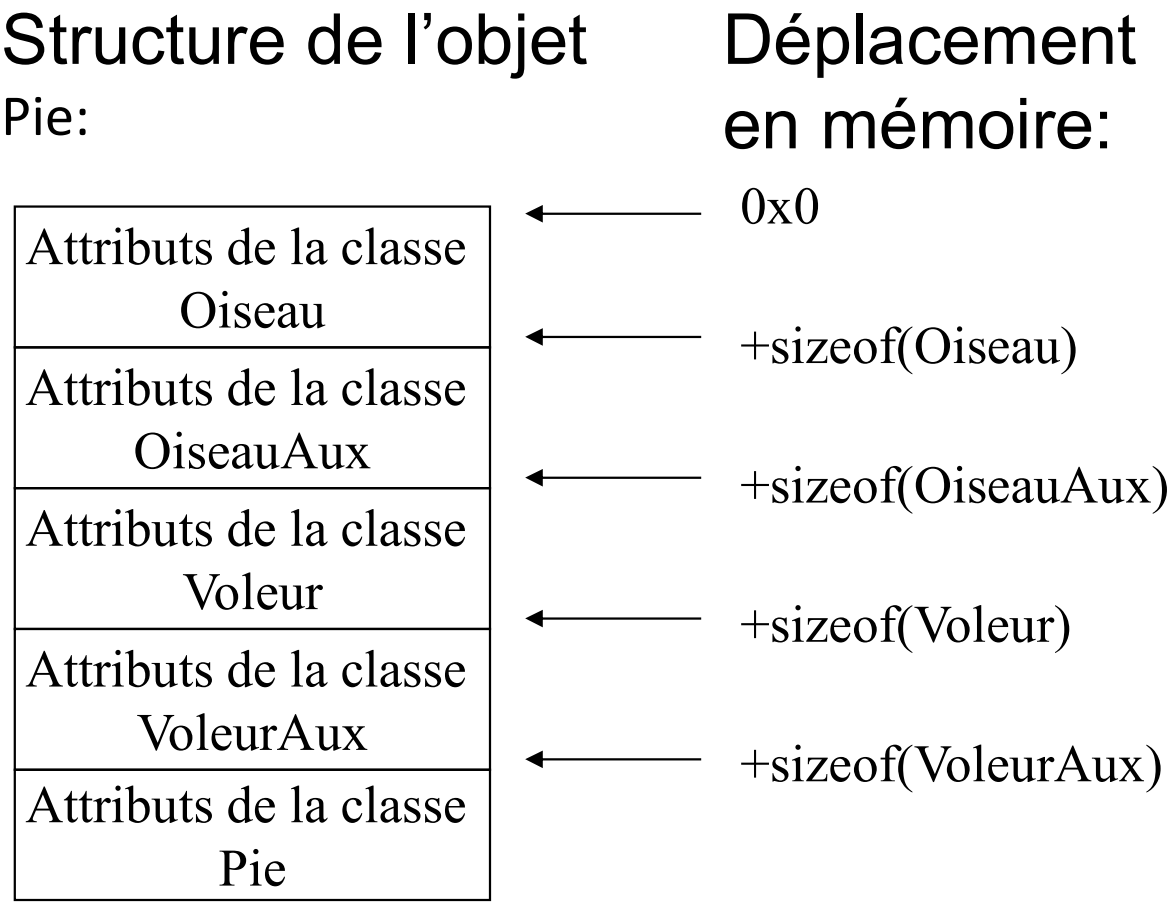
Structure de l'objet Pie:

Attributs de la classe Oiseau
Attributs de la classe OiseauAux
Attributs de la classe Voleur
Attributs de la classe VoleurAux
Attributs de la classe Pie

POLYTECHNIQUE MONTRÉAL

Structure en mémoire des objets construits par héritage multiple

Notes sur l'adresse d'un objet construit en utilisant l'HM



POLYTECHNIQUE MONTRÉAL

Structure en mémoire des objets construits par héritage multiple

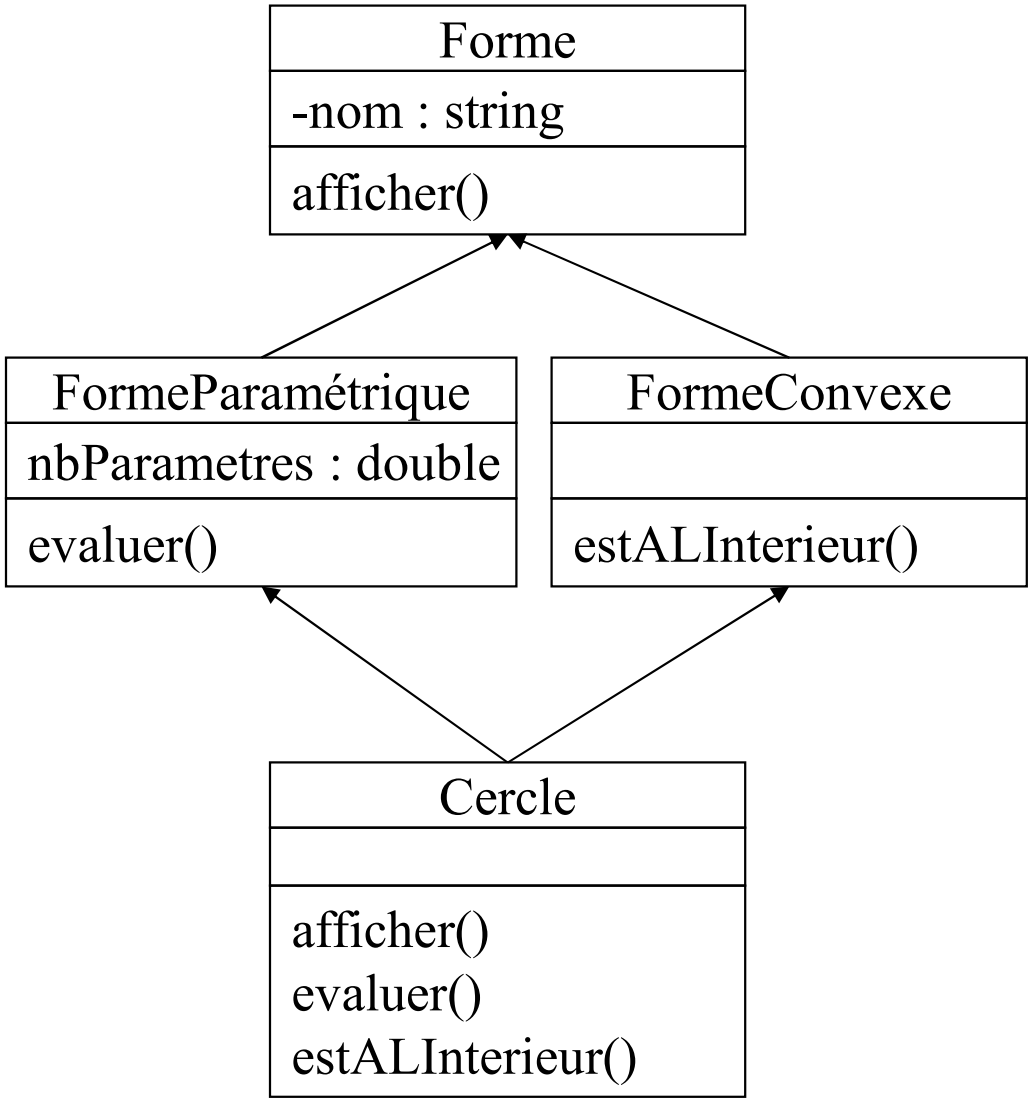
Un pointeur sur un objet Pie n'aura pas la même valeur s'il est converti en pointeurs des différents types utilisés comme classes de base:

➔ L'objet change de place en mémoire selon le type du pointeur qui lui fait référence.

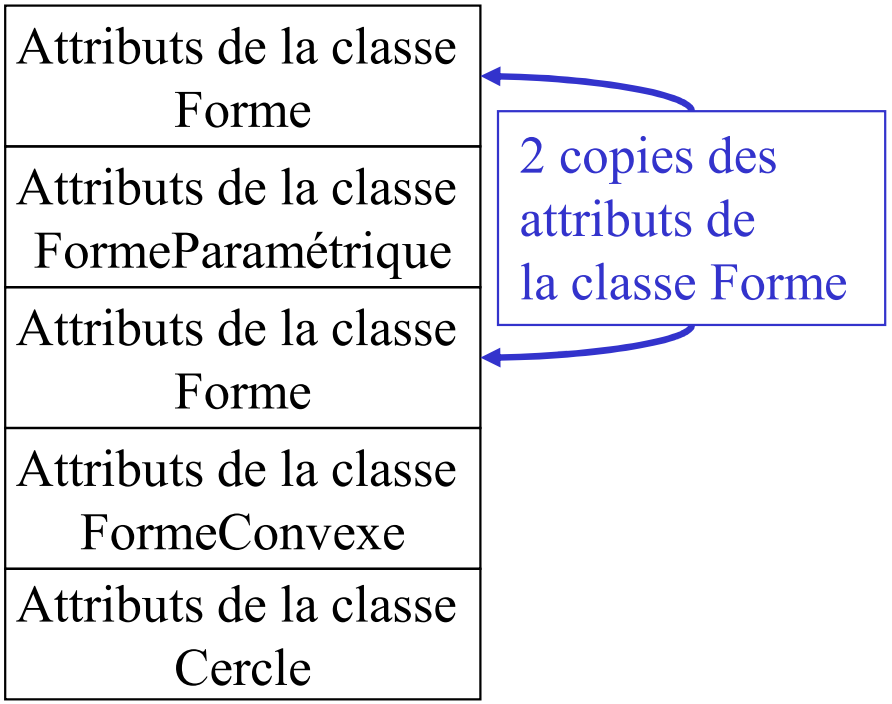
```
Pie* unePie = new Pie;

Oiseau* unOiseau = unePie;

Voleur* unVoleur = unePie;

if( (void*)unOiseau != (void*)unVoleur ) // les types doivent être comparables.

  // ce n'est pas le même objet ? Oups!…
```

POLYTECHNIQUE MONTRÉAL

Structure en mémoire des objets construits par héritage multiple



Structure de l'objet Cercle (héritage non virtuel):



POLYTECHNIQUE MONTRÉAL

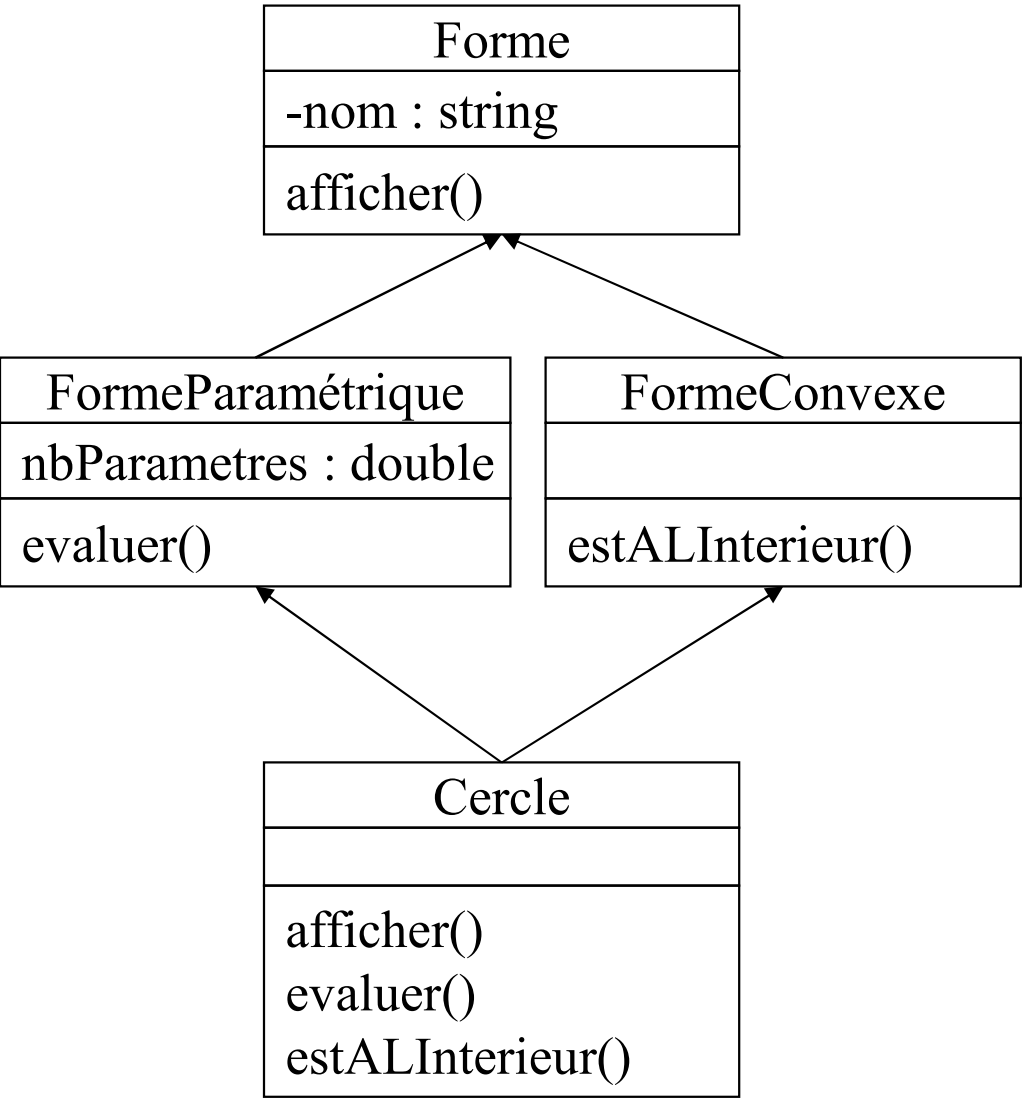
Structure en mémoire des objets construits par héritage multiple

Déclarations des classes utilisées pour l'héritage non-virtuel:

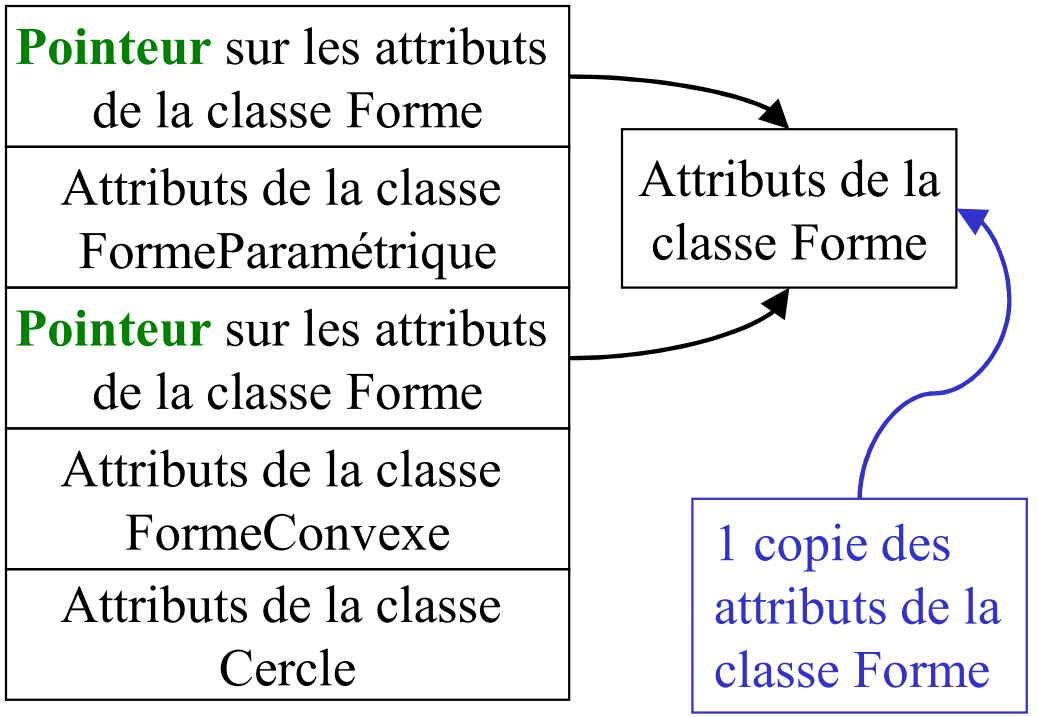
```
class Forme {…};

class FormeParametrique : public Forme {…};

class FormeConvexe : public Forme {…};

class Cercle : public FormeParametrique,
               public FormeConvexe {…};
```

POLYTECHNIQUE MONTRÉAL

Structure en mémoire des objets construits par héritage multiple



Structure de l'objet Cercle (héritage virtuel):



POLYTECHNIQUE MONTRÉAL

Structure en mémoire des objets construits par héritage multiple

Déclarations des classes utilisées pour l'héritage virtuel:

```
class Forme {…};

class FormeParametrique : virtual public Forme {…};

class FormeConvexe : virtual public Forme {…};

class Cercle : public FormeParametrique,
               public FormeConvexe
{…};
```

POLYTECHNIQUE MONTRÉAL

Structure en mémoire des objets construits par héritage multiple

Comment décider à priori si FormeParametrique et FormeConvexe doivent hériter de façon virtuelle ou non-virtuelle de la classe de base Forme ?

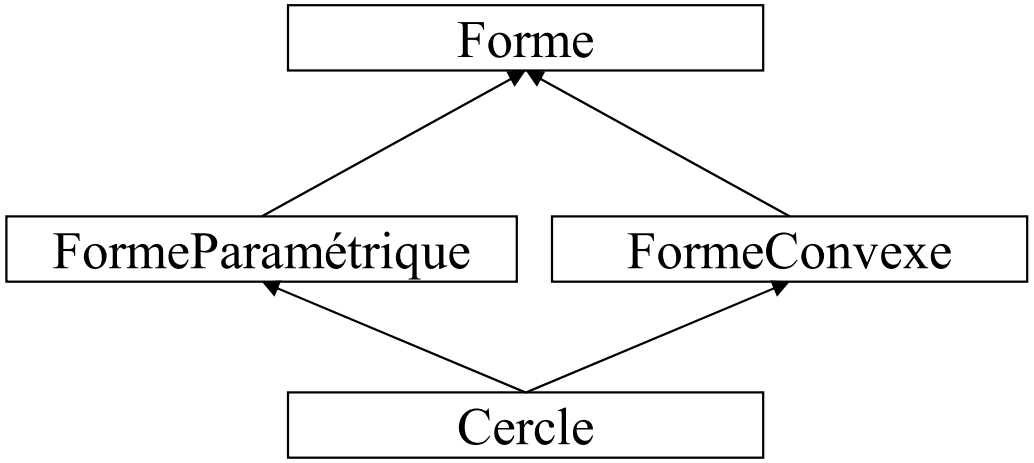
- ➔ En pratique, on ne peut pas décider: si on ne sait pas que la classe Cercle va apparaître dans la hiérarchie de classe, on ne peut pas prévoir que les classes FormeParametrique et FormeConvexe doivent hériter de façon virtuelle de la classe Forme.
- ➔ Lors de la conception d'une bibliothèque de classes, il n'est pratiquement pas justifiable d'imposer a priori les coûts supplémentaires lié à l'héritage virtuel au cas où l'héritage multiple serait utilisé.

POLYTECHNIQUE MONTRÉAL

Structure d'héritage multiple potentiellement défectueuse

Les structures d'héritage en losange devraient être considérées avec beaucoup de précautions:

- Uniquement dans la mesure où toutes les classes sont accessibles et peuvent être recompilées,
- En se souvenant que des règles particulières gouvernent la construction d'objets dont certaines classes de base sont virtuelles.



Structure d'héritage multiple potentiellement valide: notion d'interface

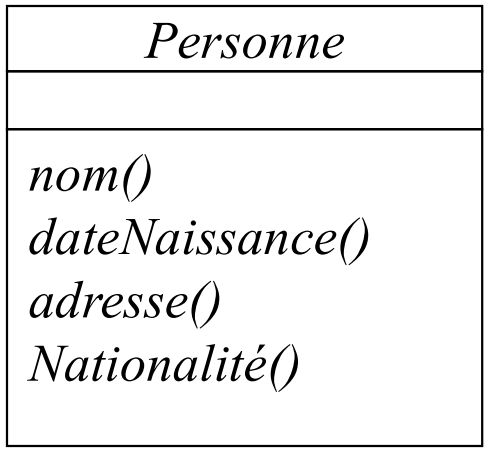
La définition d'une classe de base abstraite, ou interface, procure plusieurs avantages au plan de la conception d'une hiérarchie de classes:

- Une classe de base abstraite (CBA) est une classe contenant uniquement des fonctions virtuelles pures,
- Une CBA fournit un mécanisme efficace pour découpler les aspects d'implantation des aspects d'interface d'une classe,
- Les CBAs peuvent être combinées efficacement à l'aide du mécanisme de l'héritage multiple pour construire de façon sécuritaire des «composantes» logicielles.

POLYTECHNIQUE MONTRÉAL

Structure d'héritage multiple potentiellement valide: CBA

Une CBA ne définit qu'un ensemble de comportements auxquels les sous-classes s'engagent à se conformer.




```
class Personne {
 public:
   virtual ~Personne();
   virtual const string& nom() = 0;
   virtual const string& dateNaissance() = 0;
   virtual const string& adresse() = 0;
   virtual const string& nationalité() = 0;
 };
```

Structure d'héritage multiple potentiellement valide: CBA

Les clients de la classe Personne doivent être programmés en termes de pointeur ou de référence à des Personnes puisque seuls des objets de sous-classes de Personne peuvent être construits.

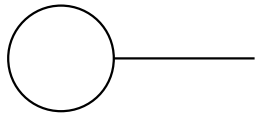
Les clients de la classe Personne doivent créer des objets appartenant à des sous-classes concrètes de la classe Personne.

POLYTECHNIQUE MONTRÉAL

Modélisation UML

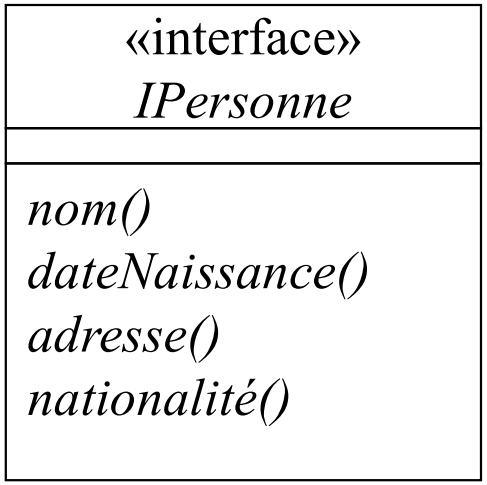
Les classes de base abstraites peuvent être modélisées en UML à l'aide d'une Interface.

- Deux notations possibles pour une interface:
  - Notation sous forme de « ballon »



IPersonne

  - Notations sous forme de classe stéréotypée



POLYTECHNIQUE MONTRÉAL

Héritage privé

Alors que l'héritage public d'une classe par une autre signifie « l'objet de la classe dérivée est un objet de la classe de base », la signification d'héritage privé est complètement différente. L'héritage privé d'une classe par une autre signifie: « la classe dérivée est implémentée en termes de la classe de base ».

Alors qu'un pointeur à un objet d'une sous-classe publique d'une classe de base peut être converti automatiquement en un pointeur à un objet de la classe de base, une telle conversion est interdite dans le cas de l'héritage privé.

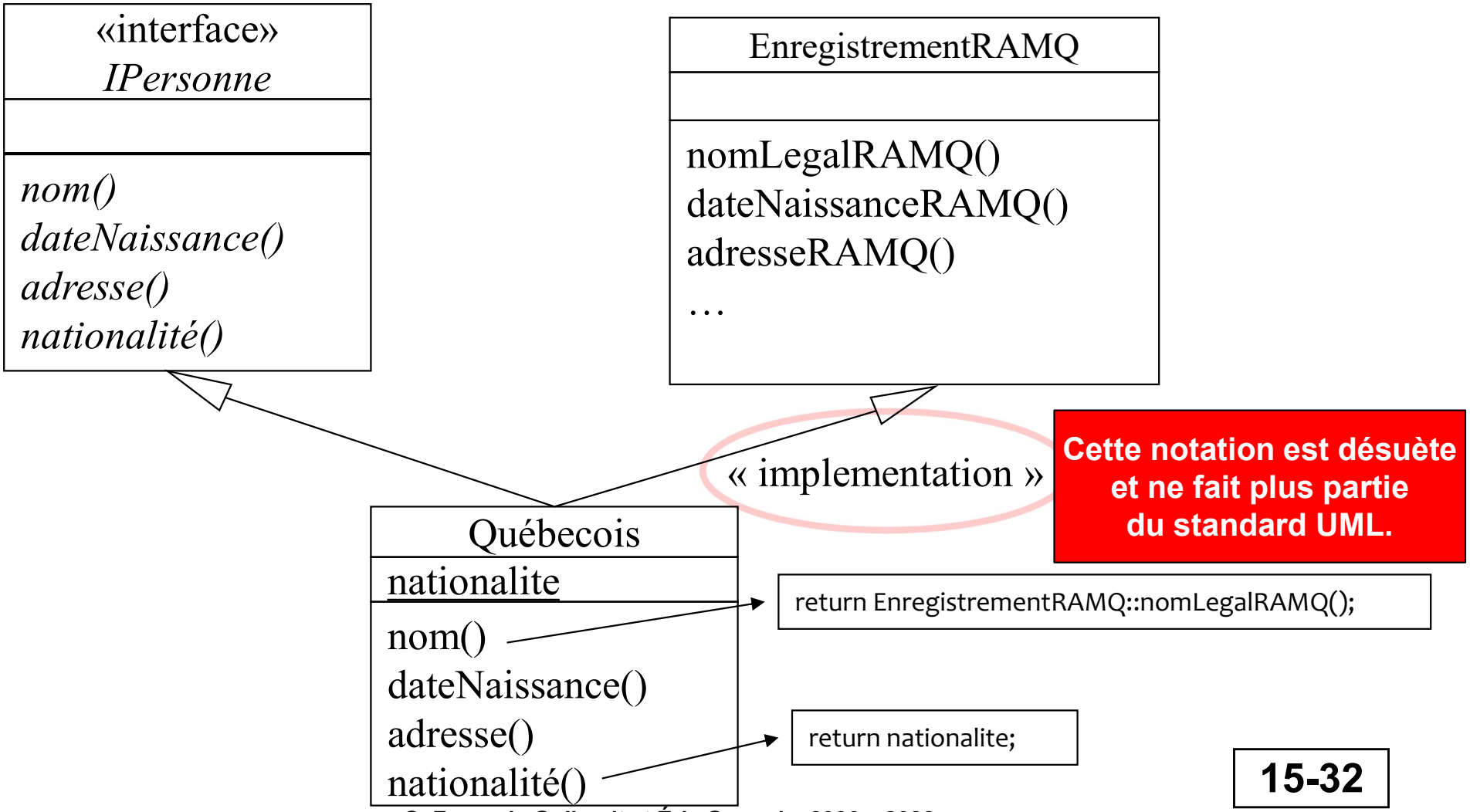
Héritage privé

L'héritage privé est purement une technique d'implémentation et non pas une technique de conception.

L'héritage privé peut cependant être combiné efficacement à l'héritage public pour concevoir des classes présentant une interface provenant d'une classe donnée, et une implantation provenant d'une autre classe.

POLYTECHNIQUE MONTRÉAL

Structure d'héritage multiple potentiellement valide : interface et implantation



Structure d'héritage multiple potentiellement valide : interface et implantation

Pourquoi utiliser l'héritage multiple ?

- Dériver de la classe permet d'accéder directement aux attributs et aux méthodes de EnregistrementRAMQ, entre autres aux méthodes protégées,
- Dériver de la classe permet de surcharger certaines fonctions virtuelles incluses dans EnregistrementRAMQ
- Si l'interface publique suffit, on devrait plutôt utiliser une forme d'association, par exemple l'agrégation.