

Chapitre VI

C++ avancé

- 1 Héritage multiple
- 2 Espaces de noms
- 3 Classes emboîtées
- 4 Modèle de classes

Chapitre VI

C++ avancé

- 1 Héritage multiple
- 2 Espaces de noms
- 3 Classes emboîtées
- 4 Modèle de classes

Héritage multiple

Une classe peut avoir **plusieurs** classes-mères.

Syntaxe

```
class CFille : public CMere1, public CMere2 ...  
{ ... }
```

Problème

Que se passe-t-il si les classes-mères définissent des attributs de même nom (éventuellement de types différents) ou des méthodes de même nom et de même signature ?

⇒ L'héritage multiple peut conduire à la définition de classes dont le comportement est peu intuitif.

⇒ Il est conseillé (à moins de très bien connaître C++) d'utiliser l'héritage multiple uniquement dans des cas où il n'y a pas de problèmes.

Héritage multiple

Quand l'utiliser ?

Principalement pour implémenter des **interfaces**.

- Une **interface** est une classe abstraite définissant **uniquement** des méthodes virtuelles pures (et un destructeur virtuel).
- Une classe peut **implémenter** plusieurs interfaces. Et être, en plus, une sous-classe d'une classe « classique ».
- En C++, on utilise l'héritage multiple pour signaler que la classe implémente des interfaces.

Héritage multiple

Exemple

Exemple

```
class IDebug
{
public:
    virtual ~IDebug();
    virtual unsigned int tailleOctets() const =0;
};

IDebug::~~IDebug()
{}
```

Héritage multiple

Exemple

Exemple

```
class IPile
{
public:
    virtual ~IPile();
    virtual void empiler(int i) =0;
    virtual bool vide() const =0;
    virtual void depiler() =0;
    virtual int sommet() const =0;
};

IPile::~~IPile()
{}
```

Héritage multiple

Exemple

Exemple

```
#include "idebug.h"
#include "ipile.h"
#include <vector>
class PileVector: public IDebug, public IPile
{
public:
    std::vector<int> m_contenu;
public:
    PileVector();
    ~PileVector();
    unsigned int tailleOctets() const;
    void empiler(int i);
    bool vide() const;
    void depiler();
    int sommet() const;
};
```

Héritage multiple

Exemple

Exemple

```
PileVector::PileVector() {}
PileVector::~~PileVector() {}
unsigned int PileVector::tailleOctets() const
{ return m_contenu.size() * sizeof(int); }
void PileVector::empiler(int i)
{ m_contenu.push_back(i); }
bool PileVector::vide() const
{ return m_contenu.empty(); }
void PileVector::depiler()
{ m_contenu.pop_back(); }
int PileVector::sommet() const
{ return m_contenu.back(); }
```

Héritage multiple

Exemple

- PileVector implémente les deux interfaces IDebug et IPile.
- **Toute instance de PileVector peut être passée comme paramètre à une fonction/méthode qui attend un IDebug ou un IPile.**

Il s'agit d'une solution sans problèmes : les super-classes n'ont pas de membres ayant le même nom.

Héritage multiple

Un cas à problème

Exemple

```
class A
{
public:
    int attribut;
    void meth();
};

class B
{
public:
    float attribut;
    int meth();
};

class C: public A, public B
{ };
```

Héritage multiple

Un cas à problème

Exemple

```
C c;
c.attribut = 0;
c.meth();
```

Erreur de compilation :

```
request for member 'attribut' is ambiguous
candidates are: float B::attribut    int A::attribut
request for member 'meth' is ambiguous
candidates are: int B::meth()        void A::meth()
```

Attention

La classe C a hérité de **deux** attributs nommés a et de **deux** méthodes nommées meth.

Héritage multiple

Un cas à problème

Il faut préciser **quel** membre doit être utilisé. . .

En préfixant son nom par le nom de la classe dont il est issu.

Exemple

```
c.A::attribut = 0;
c.B::meth();
```

Cela peut conduire à du code difficile à mettre au point.

Chapitre VI

C++ avancé

- 1 Héritage multiple
- 2 **Espaces de noms**
- 3 Classes emboîtées
- 4 Modèle de classes

Espaces de noms

- Les **espaces de noms** permettent de « ranger » des classes dans des ensembles.
- Chaque ensemble a un nom, et deux ensembles différents peuvent contenir une **classe de même nom**.
Le nom complet d'une classe est en fait *namespace::nomclasse*.
- Par exemple, l'espace de nom `std` peut contenir une classe `string`, et l'espace de nom `mabibliotheque` une classe `string`.
- Les espaces de noms sont habituellement utilisés lors du développement d'un code prévu pour être **réutilisable**, et importé dans d'autres projets (**bibliothèque** de classes)
Si le projet qui importe le code contient une classe de même nom, erreur de compilation.
Avec l'utilisation d'espaces de noms, pas d'ambiguïté.

Espaces de noms

Syntaxe

- Déclaration d'une classe (dans un .h)

```
namespace nomespace
{
    class nomclasse { ... };
}
```

- Définition d'une classe (dans un .cpp)

```
namespace nomespace
{
    nomclasse::nomclasse() { ... };
}
```

Espaces de noms

Syntaxe

- Utilisation d'une classe
nomespace::*nomclasse*
- Recherche des classes dans un espace de noms (import)
using namespace *nomespace* ;
À **éviter** dans un fichier .h
- Import d'un élément d'un espace
using *nomespace*::*nomelement*
(exemple : using std::cout;)

Chapitre VI

C++ avancé

- 1 Héritage multiple
- 2 Espaces de noms
- 3 Classes emboîtées
- 4 Modèle de classes

Classes emboîtées

- Il est possible de définir des classes (ou types : struct, union, enum) **à l'intérieur** d'une classe.
- La visibilité s'applique à ces types.
- Cela est utilisé pour définir une classe qui n'a de sens qu'« à l'intérieur » d'une autre classe.

Exemple. Un graphe (orienté) est composé de sommets et d'arcs.

Classes emboîtées

Exemple

```

class Graphe
{ public:
  class Sommet
  { private:
    std::string m_etiquette;
  public:
    Sommet(std::string const & et);
    std::string const & etiquette() const; };
  class Arc
  { public:
    Sommet * m_origine;
    Sommet * m_extremite; };
  private:
    std::vector<Sommet *> m_sommets;
    std::vector<Arc *> m_arcs;
  public:
    ~Graphe();
    void ajouter(Sommet * s);
    void ajouter(Arc * s);
    std::vector<Sommet *> const & sommets() const;
};

```

Classes emboîtées

Exemple

```

Graphe::Sommet::Sommet(string const & et)
  :m_etiquette(et)
{}

string const & Graphe::Sommet::etiquette() const
{ return m_etiquette; }

Graphe::~~Graphe()
{
  for (vector<Sommet *>::iterator i=m_sommets.begin();
    i!=m_sommets.end(); i++)
    delete *i;
  for (vector<Arc *>::iterator i=m_arcs.begin();
    i!=m_arcs.end(); i++)
    delete *i;
}

```

Classes emboîtées

Exemple

```
void Graphe::ajouter(Sommet * s)
{ m_sommets.push_back(s); }

void Graphe::ajouter(Arc * s)
{ m_arcs.push_back(s); }

vector<Graphe::Sommet *> const & Graphe::sommets() const
{ return m_sommets; };

int main()
{
    Graphe g;
    g.ajouter(new Graphe::Sommet("1"));
    ...
}
```

Chapitre VI

C++ avancé

- 1 Héritage multiple
- 2 Espaces de noms
- 3 Classes emboîtées
- 4 **Modèle de classes**

Modèle de classes

- Un **modèle de classes** (ou **patron** de classes) permet de construire plusieurs classes qui diffèrent par l'utilisation d'un type de données.
- `std::vector` est un modèle qui permet de définir les classes `std::vector<int>`, `std::vector<std::string>`, ...

Attention

Un modèle de classes n'est pas une classe.

Il ne peut donc pas être instancié ou utilisé comme un type.

Exemple (erroné)

```
std::vector v;
v.push_back(???) ;
```

Les conteneurs de la STL sont des modèles de classes. Il est possible de définir ses propres modèles.

Modèle de classes

Exemple (pile.h)

```
template <typename t>
class Pile
{
private:
    std::vector<t> m_contenu;
public:
    bool vide() const { return m_contenu.empty(); };
    void empiler(t v) { m_contenu.push_back(v); };
    void depiler()    { m_contenu.pop_back(); };
    t sommet() const;
};

template <typename t>
t Pile<t>::sommet() const
{
    t s = m_contenu.back();
    return s;
};
```

Modèle de classes

Exemple

```
#include "pile.h"
#include <string>

using namespace std;
int main()
{
    Pile<int> pint; Pile<string> pstring;
    pint.empiler(2);
    pstring.empiler("essai");
    ...
}
```

Modèle de classes

- Les modèles de classes sont souvent utilisés pour définir des classes conteneurs.
- Un modèle peut avoir **plusieurs** arguments.
- La définition des méthodes est habituellement donnée dans le fichier `.h`
 - **dans** la déclaration de la classe.
 - **après** la déclaration de la classe.
 - dans un **fichier d'implémentation** du modèle, inclus dans le fichier `.h`.