Centrale-Lyon - Programmation Orientée Objet - Cours Info 2A (2/4)

Programmation Orientée Objet en C++ (1ère partie)

- Les classes en C++
- Attributs et méthodes
- Masquage de l'implémentation
- Constructeurs et destructeurs
- Objets dynamiques
- Tableaux d'objets
- Réutilisation de classes

Version 2.2 – 8/11/2005

Rappel : Schéma de classe UML

- 3 schémas possibles selon le degré de détail :

Classe non-documentée :

Voiture

Classe documentée :

Voiture
marque couleur proprio
demarrer () tourner () accelerer () ralentir () arreter () changer_moteur ()

Classe détaillée :

Voiture
- marque : string - couleur : Couleur - proprio : string
+ demarrer () + tourner (in int) + accelerer (in double) : double + ralentir (in double) : double + arreter () # changer_moteur ()

Cours Info 2A (2/4) C++ (1ère partie) 2

Déclaration d'une classe C++

- Modèle de déclaration d'une classe :

```
class NomDeClasse : mode_heritage ClasseDeBase
{
private:
   déclarations attributs et méthodes privées
   ...
protected:
   déclarations attributs et méthodes protégées
   ...
public:
   déclarations attributs et méthodes publiques
   ...
};
```

Attention à ne pas oublier le « ; » final !

Cours Info 2A (2/4) C++ (1ère partie) 3

Attributs d'une classe (1/2)

- Type des attributs :
  - type de base
  - types construits
  - d'autres classes (Cf composition de classes)
- Déclarations des attributs :
  - Dans la classe (c'est aussi une définition)
- Bonne pratique :
  - Pour respecter le principe d'encapsulation les attributs sont généralement déclarés dans la section « private », voire « protected »
  - Pour chaque attribut, on prévoit, si nécessaire, pour pouvoir accéder aux attributs d'une instance :
    - ➔ une méthode de lecture
    - ➔ une méthode d'écriture

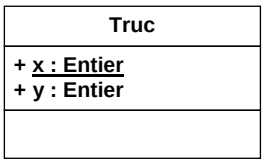
```
class Date
{
private:
   int jour, mois, an;
public:
   ...
};
```

Déclaration de 3 attributs privés

Cours Info 2A (2/4) C++ (1ère partie) 4

Attributs d'une classe (2/2)

- Attributs de classe
  - les attributs sont dupliqués dans les instances des classes
  - ***SAUF*** pour les attributs de classe qui sont communs à toutes les instances
  - Notation UML :




```
class Truc
{
public:
   static int x;        // attribut de classe
   int y;               // attribut d'instance
   ...
};

int main (void)
{
   Truc t;
   ...
   t.y = 7;
   Truc::x = 8;
   …
}
```

Accès à l'attribut d'instance y

Accès à l'attribut de classe x

  - :: est appelé opérateur d'accès
  - Usage « classique » : mise en œuvre d'un compteur d'instance

Cours Info 2A (2/4) C++ (1ère partie) 5

Définitions des méthodes (1/2)

- A l'extérieur de la classe (méthode "standard") :

```
class Date
{
private:
   int jour, mois, an;
public:
   int initialiser (int, int, int);
   void lire_date (int &, int &, int &);
   void afficher () const;
};

void Date::lire_date (int& j, int& m, int& a)
{   j = jour; m = mois; a = an;   }

void Date::afficher () const
{   cout << jour << "/" << mois << "/" ;
   cout << an << endl;   }
```

- A l'intérieur de la classe (méthode "inline")

```
class Date
{
private:
   int jour, mois, an;
public:
   int initialiser (int, int, int);

   void lire_date (int& j, int& m, int& a)
   {   j = jour; m = mois; a = an;  }

   void afficher () const
   {   cout << jour << "/" << mois << "/" ;
      cout << an << endl; }
};
```

Cours Info 2A (2/4) C++ (1ère partie) 6

Définitions des méthodes (2/2)

- Intérêt des méthodes (ou fonctions membres) "inline" :
  - les fonctions membres « classiques » sont accessibles par les instances via des pointeurs (gérés par la classe)
  - l'utilisation de fonctions inline réduit la lourdeur de l'appel ce qui améliore le temps d'exécution du programme.
- Inconvénient :
  - les fonctions membres inline sont dupliquées dans les instances
  - seules les petites fonctions, doivent être mises inline
- Finalement :
  - on doit chercher un compromis entre place mémoire et temps d'exécution
- En cas de doute :
  - on utilisera des méthodes « classiques »

Cours Info 2A (2/4) C++ (1ère partie) 7

L'opérateur d'accès "::"

- Désignation d'une méthode d'une classe particulière
- Résolution des cas de conflit de noms de variables :
  - Ambiguïté entre mois (attribut de la classe) et mois (argument du constructeur)
  - **Date::mois** désigne l'attribut mois de la classe Date
  - **mois** désigne le paramètre formel du constructeur
  - **::mois** désigne la variable globale mois

```
int mois;                 // variable globale
class Date
{
   int jour, mois, an;
public:
   Date (int, int, int);
   void lire_date (int &, int &, int &);
   void afficher () const;
};
Date::Date (intj, int mois, int a)
{
   jour = j;
   an = a;
   Date::mois = mois;
   ::mois = mois;
}
...
```

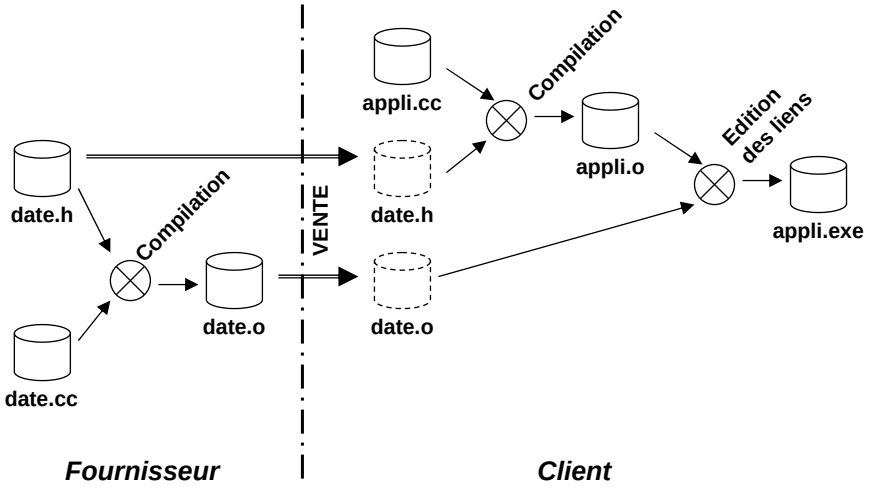
Cours Info 2A (2/4) C++ (1ère partie) 8

Masquage de l'implémentation

- Organisation en 2 fichiers :
  - un fichier "*classe*.h" (header) contenant la déclaration de la classe
    - ➔ c'est **l'interface de la classe**
    - ➔ ce fichier est accessible à un utilisateur qui a une vue externe de la classe
  - un fichier "*classe*.cc" contenant la définition des méthodes
    - ➔ c'est **l'implémentation de la classe**
    - ➔ ce fichier est compilé est transmis en binaire à l'utilisateur qui n'a pas accès aux mécanismes de réalisation
- Utilisation : fichier "*appli*.cc"
  - doit inclure le fichier "*classe*.h"
  - crée des instances de la classe
  - appelle les fonctions membres de la classe
  - doit être lié au fichier *classe*.obj résultant de la compilation de *classe*.cc

Cours Info 2A (2/4) C++ (1ère partie) 9

Schéma de production de programme



Cours Info 2A (2/4) C++ (1ère partie) 10

Exemple : date.h, date.cc et appli.cc

```
class Date
{
private:
   int jour, mois, an;
public:
   bool initialiser (int, int, int);
   void lire_date (int &, int &, int &);
   void afficher () const;
};
```

Fichier "date.h"

```
#include "date.h"
int main ()
{
   Date hier ;
   hier.initialiser (4, 11, 1992)
   hier.afficher();
}
```

Fichier "appli.cc"

```
#include "date.h"

bool Date::initialiser (int j, int m, int a)
{
   if (j >= 1 && j <= 31) jour = j;
      else return false;
   if ( m >= 1 && m <= 12) mois = m;
      else return false;
   if (a > 0) an = a;
      else return false;
   return true;
}

void Date::lire_date (int& j, int& m, int& a)
{   j = jour; m = mois; a = an;  }

void Date::afficher () const
{   cout << jour << "/" << mois << "/" ;
   cout << an << endl;   }
```

Fichier "date.cc"

Cours Info 2A (2/4) C++ (1ère partie) 11

Constructeurs (1/2)

- Constructeur
  - méthode spéciale de même nom que la classe
  - sert à initialiser **les instances** lors de la définition
  - appelée automatiquement (ou explicitement) à la création de l'objet
- Exemple :
  - on remplace la méthode initialiser par un constructeur

```
class Date
{
private:
   int jour, mois, an;
public:
   Date (int, int, int);
   ...
   void afficher () const;
};
```

Fichier "date.h"

```
#include "date.h"
Date::Date (int j, int m, int a)
{
   jour = j;  mois = m;  an = a;
}
…
void Date::afficher () const
{   cout << jour << "/" << mois << "/" ;
   cout << an << endl;   }
```

Fichier "date.cc"

Cours Info 2A (2/4) C++ (1ère partie) 12

Constructeurs (2/2)

- Utilisation des constructeurs

```
#include "date.h"

int main ()
{
   Date aujourdhui = Date (15,9,2000);
   Date noel (25,12,2000);
   Date mon_anniversaire;
}
```

ERREUR : Appel du constructeur sans arguments !

Fichier "appli.cc"

- Constructeur par défaut :
  - si on n'écrit pas de constructeur dans la classe, le compilateur en fournit un sans argument : **constructeur par défaut**
  - si on écrit un constructeur dans la classe, le compilateur **ne fournit plus** un constructeur par défaut sans argument

Cours Info 2A (2/4) C++ (1ère partie) 13

Destructeurs

- Durée de vie d'une variable locale :

```
void f()
{
   int x;
   Date d;
   …
}
```

x est créé à cet endroit

d est créé à cet endroit

x et d sont détruits à cet endroit

NB: Une instance locale de classe a une durée de vie identique à celle d'une variable d'un type de base

- Destructeur
  - fonction spéciale, sans arguments, qui s'exécute avant la destruction d'un objet.
  - traitement des problèmes annexes liés à la disparition de l'objet (libération d'espace mémoire acquis dynamiquement par l'objet, fermeture des fichiers, etc.)
  - fonction ayant le nom de la classe précédé par tilde (~)

Cours Info 2A (2/4) C++ (1ère partie) 14

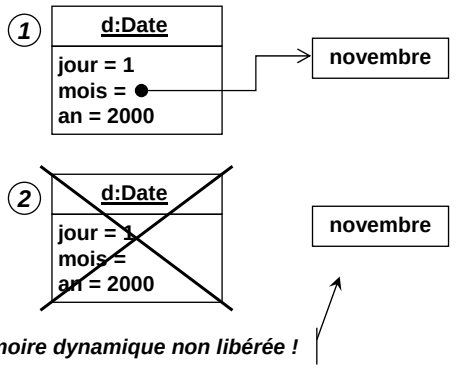
Exemple 1 : sans destructeur

```
class Date   // sans destructeur
{
   int jour, an;
   char * mois;  // chaine de caractères
public:
   Date (int, char *, int);
   ...
};

Date::Date (int j, char * m, int a)
{
   jour = j;
   an = a;
   mois = new char[strlen(m)+1];
   strcpy (mois, m);
}
```

NB: Utilisation de chaînes C classiques (char*)

```
int main ()
{
   Date d (1,"novembre", 2000); (1)
   ...
}  // d est détruit ici (2)
```



Mémoire dynamique non libérée !

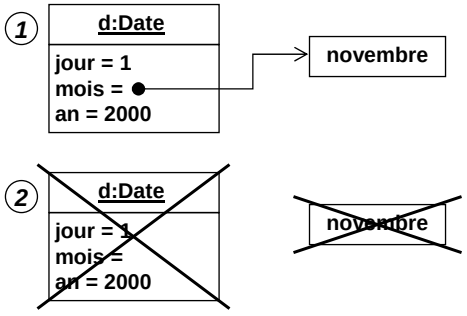
Exemple 2 : avec destructeur

```
class Date   // avec destructeur
{
   int jour, an;
   char * mois;  // chaine de caractères
public:
   Date (int, char *, int);
   ~Date ();
   ...
};

Date::Date (int j, char * m, int a)
{
   … // idem ci-avant
}

Date::~Date ()
{   delete [] mois;  }
...
```

```
int main ()
{
   Date d (1,"novembre", 2000); (1)
   ...
}  // d est détruit ici (2)
```



- **un destructeur est nécessaire si l'objet alloue de la mémoire dynamique**

Surcharge des constructeurs

```
class Date
{
   int jour, an;
   char * mois;  // chaine de caractères
public:
   Date (int, char *, int);  // J-M-A
   Date (char *);     // chaine
   Date (int);        // jour
   Date ();           // date courante
   ...
};
Date::Date (int j, char * m, int a)
{   // initialisation par J-M-A }
Date::Date (char * chdate)
{   // initialisation par une chaîne }
Date::Date (int j)
{   // initialisation par le jour }
Date::Date ()
{   // utilisation de la date système }
```

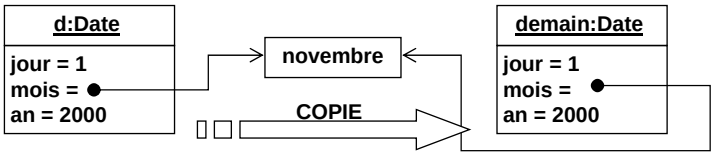
```
int main ()
{
   Date aujourdui (13);
   Date noel ("25 decembre 2000");
   Date maintenant;
   Date demain (14, "decembre", 2000);
   ...
}
```

- On peut redéfinir autant de fois que nécessaire les constructeurs d'une classe

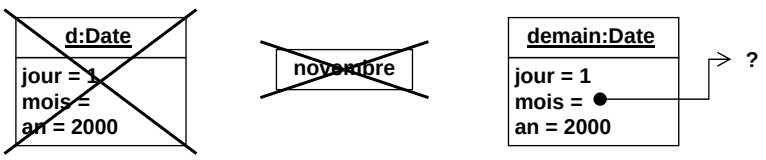
Construction par copie d'objet

- demain est une copie bit par bit de d : d et demain utilisent la même zone pour la chaîne "novembre"

```
Date d (1, "novembre", 2000);
Date demain = d;
...
```



- Pb: si d est détruit, la chaîne "novembre" est libérée par le destructeur et demain pointe sur une zone indéterminée

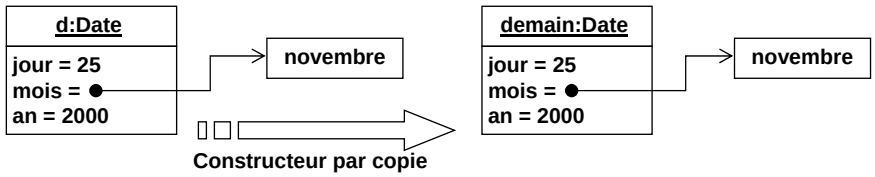


Constructeur par copie

```
class Date
{
   int jour, an;
   char * mois;
public:
   Date (int, char *, int);
   ...
   Date (const Date &);
};
```

```
Date::Date (const Date & d)
{
   jour = d.jour;
   an = d.an;
   mois = new char [strlen(d.mois)+1];
   strcpy (mois, d.mois);
}
```

```
Date d ("25 novembre 2000");
Date demain = d;          // appel constructeur par copie
```



Cas d'appel d'un constructeur par copie

- Création d'une instance copie d'une autre :

```
Date hier = maintenant;
```

- Passage de paramètre par valeur à une fonction :

```
int f(Date d);
{
   …
   …
   ...
}
```

```
int main (void)
{
   Date hier (1, "janvier", 1970);
   …
   y = f(hier);   // appel du constructeur
}
```

- Retour d'une valeur d'une fonction :

```
Date g(int x, int y, int z);
{
   Date d;
   …
   return d;          // appel du constructeur par copie
}
```

Données dynamiques

- Rappel:
  - les données dynamiques sont créées pendant l'exécution à partir du tas (ou mémoire dynamique)
  - les données dynamiques sont manipulées à travers des pointeurs
  - En C++, 2 opérateurs :
    - ➔ new pour allouer de la mémoire
    - ➔ delete pour libérer de la mémoire

```
char *p, *q;          // p, q 2 pointeurs
struct machin
{
   int x;
   double y;
} * r;                // r, pt sur struct machin
...
p = new char;         // alloc d'un char
q = new char[10];     // alloc d'une chaine
r = new machin;       // alloc d'un struct machin
…
*p = 'a';             // accès à l'élement pointé
                      // par p
(*r).x = 2;           // accès aux membres de
r->y = 3.14;          // la struct pointée par r
…
delete p;             // libération du tas
delete[] q;
delete r;
```

Cours Info 2A (2/4) C++ (1ère partie) 21

Durée de vie des objets

- Objets déterminés à la compilation :
  - les objets déclarés comme des variables sont totalement déterminés par le programmeur lors de la compilation, leur création ne dépend pas d'un résultat intermédiaire
  - un objet déterminé à la compilation a une durée de vie analogue à celle d'une variable d'un type de base
  - un objet local à un bloc disparaît au franchissement de « } » et son destructeur est appelé à ce moment
- Objets dynamiques
  - les objets, comme les variables des types de base peuvent être alloués et libérés pendant l'exécution
  - les objets dynamiques sont manipulés à travers des pointeurs
  - les opérateurs new et delete s'appliquent aux objets instances de classes définies par l'utilisateur

Cours Info 2A (2/4) C++ (1ère partie) 22

Exemple d'objets dynamiques

- Commentaires :
  - s1 est une instance de Stock
  - ptr1 est un pointeur sur un objet Stock
  - ptr2 est un pointeur sur un objet Stock
  - ptr1 pointe sur un objet dynamique créé avec un constructeur vide
  - ptr2 pointe sur une objet dynamique créé avec le constructeur par copie

```
class Stock
{
    string nom;
    int q;
public:
    Stock ()  { nom = ""; q = 0; }
    Stock (Stock & s);
    Stock (string p, int i) { nom = p; q = i; }
};
…
int main (void)
{
    Stock s1 ("tomates", 5000);
    Stock *ptr1, *ptr2;
    ptr1 = new Stock();
    ptr2 = new Stock(s1);
    …
}
```

Cours Info 2A (2/4) C++ (1ère partie) 23

Attribut caché "this"

- this est un attribut caché non défini par le programmeur
- this est un pointeur sur l'instance en cours de la classe
- une méthode peut utiliser l'attribut this

```
class Truc
{
    int x;
    float y;
public:
    void afficher () const;
  …
};

void Truc::afficher () const
{
    cout << this->x << " " << this->y << endl;
}
```

NB: ici l'utilisation de this n'apporte rien !

On peut écrire :
cout << x << " " << y << endl;

Cours Info 2A (2/4) C++ (1ère partie) 24

Tableau d'objets

- Définition d'un tableau

```
Date t[10];
Date *ptr = new Date[20]
```

- Appel de constructeurs :

```
Date t[2] = { Date (15,9,2000), Date (25,12,2000) };
Date u[3] = { Date (15,9,2000), Date (25,12,2000) };
```

NB: le 2ème exemple ne marche que s'il existe aussi un constructeur sans arguments sinon il y a erreur !

- Appel de destructeurs :
  - un tableau local est entièrement détruit à la sortie du bloc
  - un tableau global est détruit à la fin du programme
  - un tableau dynamique doit être libéré par le programmeur à l'aide de **delete[]**
- NB: il est aussi possible d'utiliser la STL et le type Vector :
  - Vector<Date> …

Cours Info 2A (2/4) C++ (1ère partie) 25

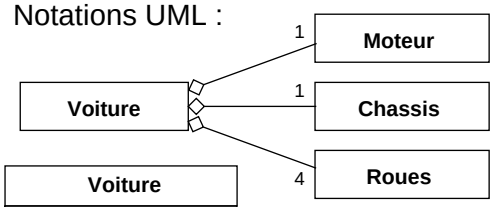
Réutilisation de classes

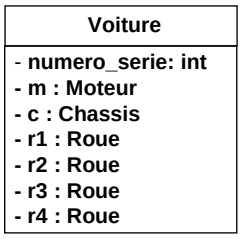
- Problème de la réutilisation
  - Quand une classe est testée et marche, elle marchera toujours, on peut donc la réutiliser
  - une classe qui marche est généralement compilée ce qui garantit qu'on ne la modifiera pas, ce qui pourrait perturber les applications actuelles utilisatrices de cette classe
  - une classe compilée ***est fermée*** car on ne peut pas remettre en cause son code pour prendre en compte de nouvelles applications
  - une classe peut servir à la construction de nouvelles classes, en ce sens elle ***est ouverte*** ; 2 mécanismes principaux :
    - ➔ la dérivation ou l'héritage
    - ➔ la composition de classe
  - la programmation par objets permet d'avoir du logiciel à la fois ***ouvert et fermé***

Cours Info 2A (2/4) C++ (1ère partie) 26

Réutilisation de classes par composition

- En C++ :
  - les attributs peuvent être:
    - ➔ des variables de type de base
    - ➔ des instances d'autres classes
- Notations UML :





NB: il existe une 3ème notation: Cf. polycopié, § A.2.5

```
class Moteur
{
   ...
};
class Chassis
{
   ...
};
class Roue
{
   ...
};
class Voiture
{
   int numero_serie;
   Moteur m;
   Chassis c;
   Roue r1, r2, r3, r4;
  ...
}
```

Cours Info 2A (2/4) C++ (1ère partie) 27

Constructeurs et destructeur d'une classe composée

- Constructeurs
  - Le constructeur de la classe composée appelle **tous** les constructeurs des membres instances de classes composants avant d'effectuer son propre traitement (appel des constructeurs sans argument qui doivent donc exister !)
  - Sinon, on peut appeler explicitement les constructeurs :
    - ➔ après l'en-tête du constructeur de la classe composée
    - ➔ introduit par ':'
    - ➔ identifié par le nom de l'attribut
  - les attributs correspondants à des types de base disposent de constructeurs par défaut ce qui permet de les initialiser comme les instances d'autres classes
- Destructeurs
  - les destructeurs de membres composants sont appelés automatiquement par le destructeur de la classe composée

Cours Info 2A (2/4) C++ (1ère partie) 28

Exemple

- Reprise de l'exemple de la voiture

```
class Moteur
{
   int puissance
public:
   Moteur (int p)  { puissance = p; }
   ...
};

class Chassis
{
   int longueur, largeur;
public:
   Chassis (int lo, int la)
            { longueur = lo; largeur = la; }
   …
};
```

```
class Roue
{
   int taille;
public:
   Roue (int t) { taille = t; }
  …
};

class Voiture
{
   int numero_serie;
   Moteur m;
   Chassis c;
   Roue r1, r2, r3, r4;
public:
   Voiture (int n, int p, int lo, int la, int t) :
            m (p), c (lo,la), r1 (t), r2 (t),
            r3 (t), r4 (t), numero_serie (n)
        {  …   }
   ...
}
```

Cours Info 2A (2/4) C++ (1ère partie) 29