Cours2 - C++

Classes et objets en C++ : partie 1

J-F. Kamp

Janvier 2025

Classes en C++

Déclaration (fichier .h)

- Une classe doit d'abord être DECLAREE dans un fichier en-tête (ici fichier CCercle.h)

```
class CCercle {

    // D'abord les attributs d'instance
    private :              // si rien => private par défaut
        int m_x;
        int m_y;
        unsigned int m_r;

    // Ensuite les méthodes publiques (d'instance)
    public :
        void setRayon ( unsigned int newR );
        float getSurface ( );

    // Et les méthodes privées (d'instance)
    private :
        etc...
};  // Ne pas oublier le « ; »
```

Déclaration (fichier .h)

Et le constructeur ?

- Si rien n'est prévu, il existe un constructeur par défaut (comme en Java) qui initialise les attributs à N'IMPORTE QUOI
- Généralement, il possède des arguments pour initialiser les attributs à des valeurs précises

```
class CCercle {
    // D'abord les attributs d'instance
    private :
        int m_x;
        int m_y;
        unsigned int m_r;

    // Constructeur publique d'initialisation des attributs
    public :
        CCercle ( int x, int y, unsigned int r );
        CCercle ( );   // Constructeur sans arguments

    // Ensuite les méthodes publiques (d'instance)
    public :
        void setRayon ( unsigned int newR );
        float getSurface ( );
};
```

Définition (fichier .cpp)

Le code de toutes les méthodes (et constructeurs) doit être écrit dans un fichier séparé (CCercle.cpp)

```
// NE PAS OUBLIER d'inclure le fichier de déclaration
#include "CCercle.h"

// Code de toutes les méthodes

// Les constructeurs
CCercle :: CCercle ( int x, int y, unsigned int r ) {
    m_x = x;
    m_y = y;
    m_r = r;
}

CCercle :: CCercle ( ) { m_x = m_y = m_r = 0; }

// Les autres méthodes
void CCercle :: setRayon ( unsigned int newR ) {
    m_r = newR;
}

float CCercle :: getSurface ( ) {
    return ( 3.14 * m_r * m_r );
}
```

Destructeur

- Rappel : toute variable de type pointeur allouée dynamiquement avec l'opérateur new DOIT entrainer une opération delete de libération de cette mémoire

- Un objet peut très bien contenir une variable d'instance de type pointeur (un tableau par exemple)

  => comment libérer la mémoire allouée pour cette variable une fois que l'objet disparait ?

  Réponse : il faut écrire dans la classe le code de libération de cette mémoire à l'intérieur d'une méthode spécifique appelée DESTRUCTEUR

Destructeur (exemple)

Soit une classe de type CListe qui contient une liste de nombres sur laquelle on désire faire des opérations

```
// Déclaration à écrire dans le fichier "CListe.h"

class CListe {

    // Attributs d'instance
    private :
        int m_nbElem;
        int* m_pTabNbres;  // type POINTEUR !!

    // Constructeur
    public :
        CListe ( int nbElem );

    // Autre méthode publique
    public :
        int getMax ( );

    // DESTRUCTEUR
        ~CListe ( );
};
```

Destructeur (exemple)

```
#include "CListe.h"
#include <cstdlib>    // pour la fonction rand()

// Constructeur
CListe :: CListe ( int nbElem ) {
    m_nbElem = nbElem;

    // Allocation dynamique nécessitant DELETE
    m_pTabNbres = new int[m_nbElem];

    // Remplissage du tableau avec des entiers
    for ( int i = 0; i <  m_nbElem; i++ ) {
        m_pTabNbres[i] = rand();
    }
}
// Autres Méthodes
int CListe :: getMax ( ) { ... return... }

// Destructeur
CListe :: ~CListe ( ) {
    if (m_pTabNbres != NULL) {
        delete[] m_pTabNbres;
    }
}
```

Règles de style en C++

1. Une classe se nomme toujours **CMaClasse** (majuscule précédée de C). Les attributs de la classe et les signatures des méthodes sont écrits dans un premier fichier **CMaClasse.h** et le code des méthodes est écrit dans un second fichier séparé **CMaClasse.cpp** qui DOIT inclure la définition de la classe **CMaClasse.h** avec l'instruction **#include CMaClasse.h**

2. Un attribut de la classe s'appelle "donnée membre", son nom doit être précédé de **m_** et doit être suivi d'une minuscule **m_ maVar**

3. Les constructeurs et le destructeur portent le même nom que la classe

Règles de style en C++

4. Une variable de type pointeur commence toujours par **p** (minuscule), une donnée membre de type pointeur s'écrit **m_pMonPt**

5. NE JAMAIS laisser un pointeur avec une valeur indéterminée :
    - soit le mettre à NULL : **pVar = NULL;**
    - soit l'alloué : **pVar = new...**

6. Une méthode d'instance de la classe s'appelle "fonction membre" et commence par une minuscule **maFonction (...)**

Objets en C++

Instanciation automatique

- Comme en Java, l'instanciation d'un objet passe par l'appel d'un constructeur de la classe dont il est issu
- En C++, un objet peut être instancié dès sa déclaration SANS utiliser l'opérateur NEW

```
#include "CCercle.h"        // ne pas oublier l'inclusion !!

int main() {

    CCercle maVar ( 12, -25, 10 );

    cout << "Surface = " << maVar . getSurface();
}
```

- maVar a comme contenu un objet CCercle initialisé

- maVar ne contient PAS l'adresse de l'objet car maVar n'est PAS de type pointeur

- maVar est détruit automatiquement à la sortie du main(), il y a donc appel de son destructeur

Instanciation : constructeur sans arguments

```
class CCercle {
    // D'abord les attributs d'instance
    private :
        int m_x;
        int m_y;
        unsigned int m_r;

    // Constructeurs
    public :
        CCercle ( int x, int y, unsigned int r );
        CCercle ( );   // constructeur SANS arguments
    etc...
};
```

```
#include "CCercle.h"      // ne pas oublier l'inclusion !!

int main() {
    // Appel du constructeur sans arguments.
    // L'objet maVar est bel et bien construit dès
    // sa déclaration.
    CCercle maVar;       // et PAS maVar() !!

    cout << "Surface = " << maVar.getSurface();
}
```

Instanciation dynamique

- Comme en Java, un objet peut être instancié dynamiquement à un moment dans l'exécution (ou dès sa déclaration) avec l'opérateur NEW.
  => la variable est alors FORCEMENT de type pointeur et contiendra L'ADRESSE de l'objet créé

```
#include "CCercle.h"      // ne pas oublier l'inclusion !!

int main() {
    // Déclaration d'une variable de type « pointeur
    // sur un objet de type CCercle »
    CCercle* pMaVar = NULL; // Notez * et NULL !!

    // Instanciation et renvoie d'une adresse valide
    pMaVar = new CCercle ( 12, -25, 10 );

    // Appel d'une méthode, notez le symbole ->
    cout << "Surface = " << pMaVar -> getSurface();

    // Destruction manuelle obligatoire
    // Appel du destructeur de CCercle
    delete pMaVar;
}
```

Instanciation dynamique

- pMaVar étant de type pointeur, avant l'appel du constructeur avec new, son contenu est une adresse INDEFINIE OU initialisée à NULL

- new :
  - permet la réservation d'un emplacement mémoire pour l'objet (comme en Java)
  - renvoie une adresse valide qui est mémorisée dans pMaVar (comme en Java)
  - OBLIGERA le programmeur à libérer (avec l'opérateur delete) la mémoire occupée
  - entraînera l'appel du destructeur de l'objet avant que celui-ci ne disparaisse

- En C++, l'accès aux méthodes à partir de l'adresse de l'objet se fait en utilisant le symbole « -> »

Compilation et exécution

Avant de pouvoir utiliser la classe CCercle dans un lanceur ou une autre classe, il faut la compiler

```
// Déclaration à écrire dans un fichier "CCercle.h"
#ifndef CCERCLE_H
#define CCERCLE_H

#include "AllIncludes.h"      // Les .h de l'API C++
                              // Voir + loin
class CCercle {
    // D'abord les attributs d'instance
    private :
        int m_x;
        int m_y;
        unsigned int m_r;

    // Constructeur publique d'initialisation des attributs
    public :
        CCercle ( int x, int y, unsigned int r );

    // Ensuite les méthodes publiques (d'instance)
    public :
        void setRayon ( unsigned int newR );
        float getSurface ( );
};
#endif
```

Compilation et exécution

Le code de la classe est écrit dans un fichier séparé

CCercle.cpp

```
// NE PAS OUBLIER d'inclure le fichier de déclaration
#include "CCercle.h"

// Code de toutes les méthodes
// !! NOTEZ l'utilisation du this –> de type POINTEUR

// Le constructeur
CCercle :: CCercle ( int x, int y, unsigned int r ) {
    this –> m_x = x;
    this –> m_y = y;
    this –> m_r = r;
}

// Les autres méthodes
void CCercle :: setRayon ( unsigned int newR ) {
    this –> m_r = newR;
}

float CCercle :: getSurface ( ) {
    return ( 3.14 * (this –> m_r) * (this –> m_r) );
}
```

Compilation et exécution

- La classe est ensuite compilée (son code binaire fera partie du code binaire final de l'application)
- Compilation à la ligne de commande

  > g++ -Wall CCercle.cpp -o CCercle.o -c

  => si il n'y a pas d'erreurs de compilation dans la classe CCercle.cpp, un fichier binaire CCercle.o est produit

```
// A écrire dans le fichier "Lanceur.cpp"
#include "CCercle.h"

int main() {
    CCercle* pMaVar = NULL;

    pMaVar = new CCercle ( 12, -25, 10 );

    cout << "Surface = " << pMaVar –> getSurface();

    delete pMaVar;
}
```

Compilation et exécution

```
// Déclaration à écrire dans un fichier "AllIncludes.h"
// L'API C++ et l'espace de nommage (obligatoire)
// A inclure dans "CCercle.h"
#ifndef ALLINCLUDES_H
#define ALLINCLUDES_H

#include <iostream>
#include <cstdlib>
#include <string.h>

using namespace std;

#endif
```

```
// Fichier "Lanceur.cpp"
#include "CCercle.h"      // ne pas oublier l'inclusion !!

int main() {
    CCercle* pMaVar = NULL;

    pMaVar = new CCercle ( 12, -25, 10 );

    cout << "Surface = " << pMaVar –> getSurface();

    delete pMaVar;
}
```

Compilation et exécution

- Compilation à la ligne de commande de l'application finale (notez l'utilisation de CCercle.o)

  > g++ -Wall Lanceur.cpp CCercle.o -o Lanceur.bin

  C'est ce que l'on appelle « l'édition de liens ».

- Exécution de l'application finale

  > Lanceur.bin