Initiation aux concepts objets en C++

Du C au C++ - Découverte du paradigme objet

Sommaire

Sommaire (détail)

- III. Classes et objets
  A. Vocabulaire (rappels)
  B. Définition et structure d'une classe
  C. Constructeurs
  D. Destructeurs
  E. Accesseurs
  F. Méthodes
  G. Pointeur *this*

Objectifs

- Comprendre la notion de classe et d'objet
- Comprendre le principe d'encapsulation
- Savoir créer une classe complète

Vocabulaire (rappel)

- Classe = définition d'un nouveau type
  - Exemple : classe « voiture »
- Objet = « instance d'une classe »
  - Exemple : objet « mercedes »
- Méthode = fonction ou procédure
  - Exemple : accesseurs ou mutateurs

Structure d'une classe (1/4)

```
class Car
{
    private:
        // Attributs (members)
        int m_color;
        int m_maxspeed;
        int m_speed = 0; // allowed in C++11
        std::string m_model;

    public:
        // Constructors and destructor
        Car();
        Car(Car const& copy);
        Car(int _color, int _maxspeed, std::string _model);
        ~Car();

        // Methods (functions and procedures)
        void accelerate();
        void brake();

        // Getters and setters
        int getColor() const;
        void setColor(int col);
        int getMaxSpeed() const;
        std::string getModel() const;
};
```

Structure d'une classe (2/4)

- Déclaration d'une classe :
  - Utilisation du mot-clé « **class** »
  - Première lettre de la classe en **MAJUSCULE**
  - **Ne pas oublier le « ; » après l'accolade fermante !**

- Partie « privée »
  - Utilisation du mot-clé « **private** »
  - **Ne pas oublier le « : »**

- Partie « publique »
  - Utilisation du mot-clé « **public** »
  - **Ne pas oublier le « : »**

Structure d'une classe (3/4)

- Partie « privée »
  - *Accessible uniquement à l'intérieur de la classe*

- Partie « privée » - Déclaration des attributs
  - **Préfixer les attributs par « m_ »**
  - **Respect du principe d'encapsulation !!**

Structure d'une classe (4/4)

- Partie « publique »
  - *Accessible à l'intérieur et à l'extérieur de la classe*
  - Déclaration des constructeurs
  - Déclaration du destructeur
  - Déclaration des méthodes
  - Déclaration des accesseurs

Constructeurs (1/6)

- **Rôle : Créer l'objet et initialiser ses attributs**

- 3 types de constructeur en C++
  - Constructeur par défaut
  - Constructeur par copie
  - Constructeur surchargé

- Exemples :

```
Car();
Car(Car const& copy);
Car(int _color, int _maxspeed, std::string _model);
```

Constructeurs (2/6)

- Constructeur par défaut
  - Ne prend aucun paramètre
  - Ne retourne aucune valeur
  - Initialise les paramètres « par défaut »

- Exemple :

```
// Default constructor
Car::Car()
    : m_color(0), m_maxspeed(0), m_model("")
{
    // Nothing to do here !
}
```

Constructeurs (3/6)

- Constructeur par copie
  - Prend un seul paramètre (pointeur ou référence constante) sur un objet du type de la classe
  - Ne retourne aucune valeur
  - Initialise les paramètres par copie
  - **Indispensable si la classe à des pointeurs**

- Exemple :

```
// Copy constructor
Car::Car(Car const& copy)
{
    m_color = copy.m_color;
    m_maxspeed = copy.m_maxspeed;
    m_model = copy.m_model;
}
```

Constructeurs (4/6)

- Constructeur surchargé
  - Prend au minimum un paramètre
  - Ne retourne aucune valeur
  - Initialise les paramètres avec des valeurs particulières

- Exemple :

```
// Overload constructor
Car::Car(int _color, int _maxspeed, std::string _model)
    : m_color(_color), m_maxspeed(_maxspeed), m_model(_model)
{
    // Nothing to do here !
}
```

Constructeurs (5/6)

- Deux manières d'initialiser les attributs
  - Par la méthode « directe »
  - Par la liste d'initialisation

- Exemple avec la méthode directe :

```
// Default constructor
Car::Car()
{
    m_color = 0;
    m_maxspeed = 0;
    m_model = "";
}
```

Constructeurs (6/6)

- Exemple avec la liste d'initialisation

```
// Default constructor
Car::Car()
    : m_color(0), m_maxspeed(0), m_model("")
{
    // Nothing to do here !
}
```

- Méthode conseillée car plus efficace

Destructeur (1/2)

- **Rôle : Détruire l'objet et libérer la mémoire**

- Un seul type de destructeur !
  - Ne prend aucun paramètre
  - Ne retourne aucune valeur

- **Très utile pour libérer les pointeurs ☺**

Destructeur (2/2)

- Exemple :

```
// Destructor
Car::~Car()
{
    // Nothing to do here in this example
    // Do not forget to free memory here :)
}
```

Accesseurs (1/4)

- Deux types d'accesseurs
  - En lecture (« getter » en anglais)
  - En écriture (« setter » en anglais)

- Accesseurs en lecture ou « getter »
  - Permet de lire la valeur d'un attribut
  - Retourne **TOUJOURS** une valeur (fonction)
  - Préfixer le nom de la fonction par « get »
  - **Respect du principe d'encapsulation !!**

Accesseurs (2/4)

- Exemple d'accesseurs en lecture :

```
// Getter to read "m_color" attribut
int Car::getColor() const
{
    return m_color;
}

// Getter to read "m_maxspeed" attribut
int Car::getMaxSpeed() const
{
    return m_maxspeed;
}

// Getter to read "m_model" attribut
std::string Car::getModel() const
{
    return m_model;
}
```

Accesseurs (3/4)

- Accesseurs en écriture ou « setter »
  - Permet de modifier la valeur d'un attribut
  - **NE** retourne **PAS** de valeur (dans 98% des cas)
  - Préfixer le nom de la fonction par « set »
  - Permet de faire des tests sur les données
  - **Respect du principe d'encapsulation !!**

Accesseurs (4/4)

- Exemple d'accesseur en écriture :

```
// Setter to modify "m_color" attribut
void Car::setColor(int col)
{
    // Perform a check before modifying data...
    if (col > 0 && col < 5)
    {
        m_color = col;
    }
}
```

Méthodes (1/4)

- Deux types de méthode (comme en C)
  - Les fonctions : retourne une valeur
  - Les procédures : ne retourne rien *(void)*

- Les méthodes :
  - Permet d'interagir avec les données internes de la classe
  - Commencent par une **MINUSCULE**
  - Doivent utiliser les accesseurs (en théorie ☺)

Méthodes (2/4)

- Exemple « incorrect » :

```
void Car::accelerate()
{
    if (m_speed + 10 <= m_maxspeed)
    {
        m_speed += 10; // increase speed
        std::cout << "Current speed = " << m_speed << std::endl;
    }
    else
    {
        std::cout << "Max speed (" << m_maxspeed << ") reached !! ";
        std::cout << "Cannot accelerate..." << std::endl;
    }
}
```

Méthodes (3/4)

- Exemple « correct » :

```
// Good version
void Car::accelerate()
{
    // Use accessors and not directly members
    if (getSpeed() + 10 <= getMaxSpeed())
    {
        setSpeed(getSpeed() + 10); // increase speed
        std::cout << "Current speed = " << getSpeed() << std::endl;
    }
    else
    {
        std::cout << "Max speed (" << getMaxSpeed() << ") reached !! ";
        std::cout << "Cannot accelerate..." << std::endl;
    }
}
```

Méthodes (4/4)

- Autre exemple « incorrect » :

```
void Car::brake()
{
    if (m_speed - 15 >= 0)
    {
        m_speed -= 15; // decrease speed
        std::cout << "Current speed = " << m_speed << std::endl;
    }
    else
    {
        std::cout << "Speed cannot be negative !" << std::endl;
    }
}
```

Le pointeur *this* (1/2)

- Pointeur transparent pour le programmeur
- Pointeur passé automatiquement
  - À chaque fonction membre
  - Au(x) constructeur(s)
  - Au destructeur
- *this* représente un pointeur sur l'objet
- **this* représente donc l'objet en lui-même

Le pointeur *this* (2/2)

- Utilité :
  - Si une fonction doit renvoyer l'objet en lui-même
  - Pour la surcharge d'opérateur
  - Pour la comparaison d'objet

- Exemple :

```
// Get object thanks to this pointer
Car Car::getObject() const
{
    return *this;
}
```

Pour résumer…

- Classe : c'est comme un nouveau type (une grosse structure avec plein de fonctions…)

- Objet : celui qui va utiliser la classe (c'est une instance d'une classe)

- **Respect du principe d'encapsulation :**
  - **Les attributs sont TOUJOURS privés**
  - **On utilise les accesseurs (« getter » et « setter »)**

Les bonnes pratiques…

A. Première lettre en majuscule pour le nom des classes
B. Attributs préfixés par « m_ »
C. Variables statiques préfixées par « s_»
D. Pointeurs préfixés par « p »
E. Première lettre en minuscule pour les méthodes
F. Mettre des noms clairs et courts pour les variables
G. Libérer la mémoire des pointeurs dans le destructeur
H. Fermer les ressources dans le destructeur (fichiers…)
I. …

Prochaine séance…

- Conteneurs : vecteurs, piles et files