

GL / C++

Chapitre 3

Objet, Classe d'objets



1. Objet

- C'est une « partie de programme » composée d'**attributs** et de **méthodes**.
- **Attributs**
 - Variables locales de l'objet : valeurs caractérisant l'objet
- **Méthodes**
 - Fonctions ou des procédures que possède l'objet
 - Effectuent un traitement sur **les attributs de cet objet**
- **On parle aussi de Membres, Fonctions Membre de l'objet**

Exemple d'objet: l'objet Personne

■ Attributs

- ❑ le nom et l'adresse d'une personne

■ Méthodes :

- ❑ Initialisation des attributs à partir de valeurs données en paramètres
- ❑ Affichage des attributs de la personne à l'écran
- ❑ Changement de son adresse
- ❑ Comparaison du nom de la personne avec un nom donné

2. Classe d'objets

- **Type d'objets** ayant tous :
 - ❑ les mêmes attributs
 - ❑ les mêmes méthodes.
- Deux objets de même type :
 - ❑ différent seulement par les valeurs de leurs attributs,
 - ❑ ont les mêmes méthodes.
- Une classe a un identificateur (nom) et s'utilise comme un type ordinaire.

- **Exemple de classe d'objets**
 - ❑ Classe des objets représentant des personnes.
 - ❑ Deux personnes ont mêmes attributs et mêmes méthodes,
 - ❑ Elles diffèrent seulement par la valeur des attributs *nom* et *adresse*

3. Déclaration d'une classe d'objets

- La déclaration est composée de 2 parties :
 - ❑ **La spécification**
 - ❑ **L'implémentation.**
- **La Spécification** définit :
 - ❑ le nom de la classe
 - ❑ Les attributs de la classe
 - ❑ Les prototypes (en-tête) des méthodes de la classe.
- **L'Implémentation** définit :
 - ❑ **Les corps des méthodes** de la classe.

4. Organisation des programmes

- GL oblige, Spécification et implémentation doivent, de préférence, être faites dans des fichiers séparés ...
... mais ce n'est pas imposé par le langage !
- Fichiers de spécifications :
 - suffixe ".h".
- Fichiers d'implémentations :
 - suffixe : ".C", ".cc", ".cpp"

(on peut le changer par une directive de compilation)

5. Spécification : privée vs publique

- Dans une spécification, on distingue la section **privée** (*private*) et la section **publique** (*public*) d'une classe :
 - ❑ Les membres de la section **privée** ne sont visibles qu'à l'intérieur de la classe (par les méthodes de la classe donc)
 - ❑ Les membres de la section **publique** sont visibles par toutes les parties de programmes (par d'autres classes par exemple)
- Si rien n'est précisé quant à la nature publique ou privée d'un membre, **il est privé par défaut**
- On peut définir successivement plusieurs sections (publiques et/ou privées) dans une classe
- Il existe un troisième état, **protégé** (*protected*), qui n'a de sens que dans le contexte de l'héritage, nous en parlerons plus tard...

5. Privée/publique : règles de GL

- **Principe** très important pour la **fiabilité** des programmes :
 - ❑ **Les attributs devront toujours être déclarés privés**
 - ❑ **Les méthodes seront, elles, publiques ou privées selon les besoins**
- **Conséquence** : la classe devra fournir des méthodes pour consulter et/ou modifier les attributs ("*getters/setters*" ou "*accesseurs*") quand c'est nécessaire
- **Les méthodes qui modifient des attributs** doivent garantir la cohérence de l'objet (c-à-d la cohérence des valeurs données aux attributs)

5. Exemple de Spécification (Personne.h)

```
class Personne
{
    public: // membres publics accessibles aux "utilisateurs" de la classe
        Personne();
        // constructeur par défaut (donne des valeurs par défaut aux attributs)
        Personne(char nom[], char adresse[]);
        // constructeur donnant des valeurs des attributs
        void afficher();
        // affiche les attributs de la personne à l'écran
        void setAdresse(char adresse[]);
        // accessur (setter) qui permet de donner une valeur à l'attribut adresse
        int egal(char nom[]);
        // Retourne 1 si le nom de la personne est égal au paramètre, 0 sinon
//-----
    private: // membres privés de la classe
        char nom[30];
        char adresse[50];
        void verifLongChaine(char ch[], int lg);
        // méthode privée utilisée pour tester la cohérence
        // des paramètres des méthodes public déclarées ci-dessus
        // si (strlen(ch) > lg) affiche un message et arrête le programme
}; // Attention à ne pas oublier le point virgule
```

5. Conventions de Nommage

- Les identificateurs de classe commencent par une majuscule :
 - `Personne`, `EtudiantPerpetuel`, `ChienMechant`, ...
- Les identificateurs de membre(s), variables, procédures, fonctions, commencent par des minuscules :
 - `setNom`, `adressePostale`, `telPortable`, `unePersonne`, ...
- Si un identificateur est composé d'une concaténation de mots, on fait débiter chaque mot par une majuscule (notation CAMEL)

5. Constructeur

- Un constructeur est une méthode (procédure) particulière :
 - ❑ qui porte le **même nom que la classe**
 - ❑ qui n'a pas de type résultat (ni `void`, ni `int`, ni quoi que ce soit !)
 - ❑ qui, s'il existe, est appelé automatiquement à chaque création d'un objet de la classe
 - ❑ qui sert généralement à initialiser les attributs de l'objet créé
 - soit avec des valeurs passées au constructeur en paramètre
 - soit avec des valeurs constantes.
- Une classe peut avoir plusieurs constructeurs (surcharge)
- Un constructeur sans paramètres est appelé **constructeur par défaut**
- Si un ou plusieurs constructeurs sont définis, le compilateur doit pouvoir être capable d'en appeler un à chaque création (déclaration) d'objet
- Donc en cas de constructeurs multiples, il faut qu'il n'y ait pas d'ambiguïté concernant le constructeur qui sera appelé automatiquement.

5. Destructeur

- Une classe peut aussi **avoir un destructeur** (un seul)
 - ❑ C'est une procédure sans paramètres et sans type résultat
 - ❑ Il porte le nom de la classe précédé de ~
 - ❑ Il est appelé juste avant la destruction d'un objet
 - ❑ Il est en général chargé de libérer la mémoire allouée dynamiquement par l'objet

6. Déclaration et construction d'un objet

■ Déclaration sans construction ou avec construction par défaut

Personne p;

- Une telle déclaration est possible si :
 - il n'y a aucun constructeur dans la classe
 - ou s'il y a un constructeur par défaut (`Personne()`) qui est alors appelé. Dans ce cas, on aurait aussi pu écrire :

`Personne p = Personne ;`

■ Construction explicite dans la déclaration

`Personne p = Personne("dupond", "10 cours jean jaures");`

`Personne p("dupond", "10 cours jean jaures");`

- Ces deux déclarations sont équivalentes
- Dans les deux cas, c'est le constructeur
`Personne(char nom[], char adresse[]);`

qui est appelé.

Reconstruction (ré-initialisation) d'un objet

- On peut déclarer (et construire) un objet...

- `Personne p("dupond", "10 cours jean jaures");`

- ... et reconstruire ensuite cet objet

- `p = Personne("durand", "15 rue des fleurs");`

- C'est un peu comme lorsque l'on écrit :

```
int i = 1;           // déclaration et construction
i = 5;               // reconstruction
```

7. Instance d'une classe

- Tout objet dont le type est cette classe.

- **Exemple :**

```
Personne p1;
```

```
Personne p2("durand", "15 rue des fleurs");
```

- Les objets p1 et p2 sont deux instances de la classe personne.

8. Exemple d'utilisation d'une classe d'objets

- Il faut inclure le fichier .h de cette classe.

```
#include "Personne.h"
int main()
{
    Personne p("dupond", "10 cours jean jaures");

    p.afficher();
    if (p.egal("dupond"))
        p.setAdresse("15 rue des fleurs");
    return 0;
}
```

9. Envoi de message à un objet

- `p.setAdresse("15 rue des fleurs");`

est un **envoi d'un message à l'objet p**

- Ce message demande à l'objet p d'utiliser sa méthode `setAdresse` pour **modifier son adresse**
- Un envoi de message à un objet est composé :
 - ❑ de **l'objet** auquel est envoyé le message (**p**)
 - ❑ d'un point (**•**)
 - ❑ du **nom de la méthode** à utiliser (`setAdresse`)
 - ❑ et de **valeurs pour les paramètres** (`"15 rue des fleurs"`)

10. Paramètre implicite d'une méthode

- `p.setAdresse("15 rue des fleurs");`
range la chaîne "15 rue des fleurs" dans l'attribut adresse de l'objet **p**.
- `setAdresse` a donc réellement **2 paramètres** :
 - l'objet `p` auquel est envoyé le message :
 - paramètre particulier
 - spécifié avec une notation particulière (**`p.setAdresse`**)
 - "15 rue des fleurs", paramètre classique.
- L'objet qui reçoit le message n'apparaît pas dans la liste des paramètres de la méthode : C'est un **paramètre implicite** de la méthode

11. Accès à l'objet qui reçoit le message - Pointeur **this**

- Dans l'implémentation d'une méthode, on dispose d'une **variable prédéfinie** pour désigner l'objet qui reçoit le message, variable qui est un pointeur sur cet objet.
- Cette variable s'appelle **this**.
- Ainsi, lors de l'envoi du message

```
p.setAdresse("15 rue des fleurs");
```

this est, dans la méthode `p.setAdresse`, un pointeur sur `p`.
- Dans une méthode de la classe `Personne`, les attributs de l'objet qui reçoit le message sont donc désignés par :
`this->nom` et `this->adresse`.
- On peut aussi écrire simplement `nom` et `adresse`.

11. Accès à l'objet qui reçoit le message - Pointeur **this**

- On a besoin du pointeur **this** :
 - soit pour désigner l'objet tout entier
 - soit lorsqu'un paramètre d'une méthode a le même nom qu'un attribut de l'objet
- Dans les autres cas le pointeur **this** est facultatif.

12. Implémentation de la classe **personne**

- **Noms des méthodes préfixés avec**

`Personne::`

- **Si plusieurs classes ont une méthode de même nom, le préfixe permet de les distinguer.**

12. Implémentation (Personne.cc)

```
#include "Personne.h"
#include <iostream>
#include <string.h>
#include <stdlib.h>
using namespace std;

//-----
Personne::Personne()    // préfixe Personne::
{
    nom[0] = adresse[0] = '\0'; // chaînes vides
}

//-----
Personne::Personne(char nom[], char adresse[])
{
    verifLongChaine(nom, 29);
    verifLongChaine(adresse, 49);
    strcpy(this->nom, nom);    // this pour l'attribut
    strcpy(this->adresse, adresse);
}
```

12. Implémentation – suite (Personne.cc)

```
void Personne::setAdresse(char adresse[])
{
    verifLongChaine(adresse, 49);
    strcpy(this->adresse, adresse);
}
//-----
int Personne::egal(char nom[])
{
    return strcmp(this->nom, nom) == 0;
}
//-----
void Personne::verifLongChaine(char ch[], int lg)
{
    if (strlen(ch) > lg) {
        cout << "chaîne trop longue" << ch << endl;
        exit(0);
    }
}
//-----
void Personne::afficher()
{
    cout << "Nom : " << nom << " - Adresse : " << adresse << endl ;
}
```

13. Représentation en mémoire

- Considérons les objets suivants :

```
Personne p1("dupond", "10 rue des fleurs");
```

```
Personne p2("durand", "25 rue des fleurs");
```

- Représentation mémoire de ces 2 objets et de leur classe :

classe Personne

```
Personne();  
Personne(char nom[], char adresse[]);  
void afficher();  
void setAdresse(char adresse[]);  
int egal(char nom[]);
```

Personne p1

nom "dupond"

adresse "10 rue des fleurs"

Personne p2

nom "durand"

adresse "25 rue des fleurs"

13. Représentation en mémoire

- A chaque classe correspond une zone de mémoire contenant les méthodes
- A chaque instance d'une classe (objet) correspond une zone de mémoire contenant les valeurs des attributs pour cet instance.
- Pour toutes les instances d'une même classe, les méthodes ne sont présentes qu'une seule fois en mémoire.

14. Programme utilisant la classe **Personne**

```
#include "Personne.h"
#define NMAX_PERSONNES 100
//-----
void afficher(Personne tab[], int nb_elem)
{
    for (int i=0; i<nbElem; i++)    tab[i].afficher();
}
//-----
void ajouter( Personne p,           // Une personne
              Personne tab[],      // adresse d'un tableau
              short & nbElem)      // Nbre d'élém. du tableau
{
    if ( nbElem != NMAX_PERSONNES )
        tab[nbElem++]=p;
}
//-----
int main()
{
    Personne tab[100];             // tableau de Personne
    short nbElem=0;

    ajouter(Personne("dupond", "10 rue des fleurs"), tab, nbElem);
    ajouter(Personne("durand", "25 rue des fleurs"), tab, nbElem);
    afficher(tab, nbElem);
    return 0;
}
```

Remarques

- La classe `Personne` s'utilise comme un type de donnée ordinaire (`int`, `char`, ...).
- Le programme précédent correspond encore à une approche procédurale
- Dans une approche objet, on écrirait plutôt une classe **TableauDePersonnes** :
 - ❑ dont chaque instance représente un tableau de personnes
 - ❑ qui fournit des méthodes `ajouter` et `afficher`

15. Classe tab_personnes – Spécification (TableauDePersonnes.h)

```
#include "Personne.h"
class TableauDePersonnes {

    private:
        const int NMAX_PERSONNES = 100;

        Personne tab[NMAX_PERSONNES];

        int nbElem;

    public:
        TableauDePersonnes(); // constructeur par défaut
                               // crée un tableau vide (nb_elem=0)
        void ajouter(Personne p);

        void afficher();
};
```

Utilisation (exemple.cc)

```
#include "TableauDePersonnes.h"
```

```
int main()
```

```
{
```

```
    TableauDePersonnes tabPers;
```

```
    tabPers.ajouter(Personne("dupond", "10 rue des fleurs"));
```

```
    tabPers.ajouter(Personne("durand", "25 rue des fleurs"));
```

```
    tabPers.afficher();
```

```
    return 0 ;
```

```
}
```

Remarques

- Programme principal **plus simple** et **plus rapide**.
- Dans les appels des méthodes plus besoin de donner :
 - ❑ le tableau de personnes
 - ❑ ni le nombre d'éléments.
- On a regroupé tout cela dans les objets du type `TableauDePersonnes`.
- A chaque appel de méthode, l'objet qui reçoit le message suffit pour tout passer à la méthode.
- L'appel est donc plus simple et plus rapide.

Implémentation (TableauDePersonnes.cc)

```
#include « TableauDePersonnes.h»
```

```
TableauDePersonnes::TableauDePersonnes()
```

```
{
```

```
    nbElem = 0;
```

```
}
```

```
//-----
```

```
void TableauDePersonnes::ajouter(Personne p)
```

```
{
```

```
    if ( (nbElem) < NMAX_PERSONNES )
```

```
        tab[nbElem++] = p;
```

```
}
```

```
//-----
```

```
void TableauDePersonnes::afficher()
```

```
{
```

```
    for (int i=0; i<nbElem; i++) tab[i].afficher();
```

```
}
```

Remarques :

- Corps des méthodes souvent plus simples.
- Les méthodes ont moins de paramètres car on utilise les attributs (variables locales de l'objet qui sont globales aux méthodes).

Exercice

- Ecrire la spécification et l'implémentation d'une classe **Point** permettant de manipuler un point du plan. Prévoir :
 - Un constructeur permettant d'initialiser les coordonnées du point
 - Des méthodes permettant de consulter les coordonnées (fonctions d'accès)
 - Une méthode permettant de déplacer le point par une translation (dx,dy)
 - Une méthode permettant d'afficher à l'écran les coordonnées
- Ecrire un programme principal qui déclare un point, l'affiche, le déplace et l'affiche à nouveau
- La déclaration suivante serait-elle correcte ? Pourquoi ?
Point p;