

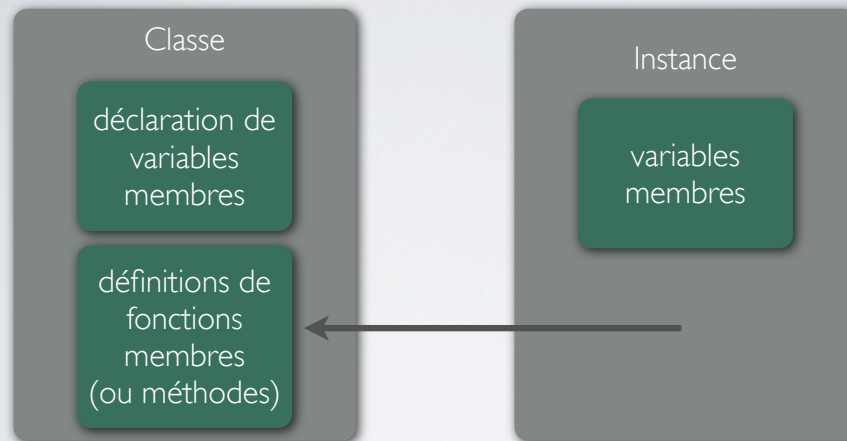
COURS DE C++

8 - Classes et Objets

PHILOSOPHIE PROGRAMMATION AVEC OBJETS

- Une donnée n'existe pas indépendamment des opérations que l'on veut effectuer sur celles-ci
- Il y a une relation naturelle entre l'ensemble des opérations que l'on veut effectuer sur une donnée d'un certain type et l'ensemble des données de ce type
- C'est la relation entre classes et objets

ANATOMIE D'UNE CLASSE



Toutes les instances d'une même classe ont les mêmes variables et fonctions membres, mais les valeurs des variables membres peuvent être différentes

ACCESSIBILITÉ DES MEMBRES

- On peut choisir d'exposer ou non certaines parties d'une classe
- Une fonction/variable membre qui est accessible depuis l'extérieur est dite publique
- Une fonction/variable membre qui n'est accessible que depuis la classe est dite privée
- Il existe une notion intermédiaires, variable/fonction protégée que l'on utilise dans un cadre bien spécifique, voir suite.

DÉFINITION D'UNE CLASSE

```
class NomDeLaClasse {
public:
  /* déclaration des membres
  publiques */
private:
  /* des membres privés */
protected:
  /* des membres protégés */
};
```

Attention au ;

EXEMPLE

```
class Foo {
public:
  int getValue() { return value; }
  void setValue(int v) { value = v; }
private:
  int value;
};
```

tout le monde
peut accéder à ces
fonctions

mais personne
ne peut accéder
directement à
la valeur

ACCÈS AUX MEMBRES

Depuis une fonction membre : on y accède directement.
Voir l'exemple précédent dans `getValue` et `setValue`

Depuis l'extérieur on accède au membre `x` de l'objet `o` avec
`o.x`

Si on a un pointeur `p` vers l'objet `o` on utilise `p->x`

EXEMPLE SUITE

```
class Foo {
public:
  int getValue() { return value; }
  void setValue(int v) { value = v; }
private:
  int value;
};
Foo f;
f.setValue(3);
cout << f.getValue();
```

mais
`cout << f.value;`
renvoie à la compilation :
error: 'int Foo::value' is private

INITIALISATION DES VARIABLES MEMBRES

Dans l'exemple précédent `value` n'a pas de valeur précise avant le premier appel à `setValue`

- On aimerait avoir une valeur par défaut

Idée : déclarer `int value = 2;`

- Erreur de syntaxe, ce n'est pas la syntaxe du C++
- A la place on écrit un constructeur qui va être une fonction membre appelée chaque fois qu'un objet est créé

CONSTRUCTEUR

Un constructeur pour la classe `Foo` est une fonction membre appelée également `Foo` et en général publique

Cas particulier : on la déclare comme une fonction qui ne renvoie rien (pas `void`, RIEN)

```
class Foo {
public:
    Foo() { value = 2; }
    int getValue() { return value; }
    void setValue(int v) { value = v; }
private:
    int value;
};
Foo f; // appelle implicitement f.Foo()
```

CONSTRUCTEUR A PARAMÈTRES

On peut très bien définir un constructeur

```
Foo(int v) { setValue(v); }
```

Mais alors il faut explicitement passer le paramètre à la définition :

```
Foo f(3);
```

- On peut avoir plusieurs constructeurs prenant des paramètres différents, le compilateur choisit le bon.

CONSTRUCTEUR PAR DÉFAUT

- Pour pouvoir avoir un constructeur par défaut, on a deux possibilités

Soit on définit explicitement un constructeur qui ne prend aucun paramètre : `Foo() { setValue(3); }`

Soit on définit des valeurs par défaut pour un constructeur à paramètres : `Foo(int v = 3) { setValue(v); }`

- On ne peut pas appeler un constructeur depuis un constructeur !

On l'appelle avec `Foo f;` PAS AVEC `Foo f();` qui est une fonction nommée `f` renvoyant un `Foo`

APARTÉ : POURQUOI PEUT-ON DÉFINIR DES VARIABLES N'IMPORTE OÙ EN C++?

Si ce n'était pas le cas :

```
Foo f; // je ne sais pas quels paramètres passer
      //au constructeur à ce stade
...
x = uneFonction(); // maintenant j'ai récupéré la valeur
f = Foo(x); // j'appelle donc le "bon" constructeur
```

Problème : j'ai créé deux objets Foo au lieu d'un

A la place : `Foo f(x);`

TABLEAU D'OBJETS

- On peut tout à fait définir un tableau d'objets d'une même classe :

`Foo f[10];` définit un tableau de 10 objets de la classe Foo

- Le constructeur appelé est alors le constructeur par défaut !

Pour pouvoir faire autrement il faut utiliser la STL où définir explicitement le tableau :

```
Foo f[2] = { Foo(3), Foo(4) };
```

LISTES D'INITIALISATION

- En fait ce n'est pas une bonne pratique que d'écrire des constructeurs de cette forme :

```
Foo(int v) { value = v; } // Copie la valeur v avant de l'assigner
```

- A la place on utilise des listes d'initialisation :

```
Foo(int v) : value(v) { }
```

- Si on a plusieurs variables membres :

```
Foo(...) : v1(...), v2(...), ..., vn(...) { }
```

SÉPARATION DEF°/DECLR° DES FONCTIONS MEMBRES

- Dans la définition de la classe on peut remplacer la définition de la fonction membre par sa déclaration et la définir ultérieurement

Lorsqu'on la définit tout ce passe comme si on était dans un espace de nom associé à la classe (on ne peut pas faire un namespace Foo {...})

- Exemple :

```
class Foo {
public:
void f();
};
void Foo::f() { }
```

DÉFINITION EN PRATIQUE

Foo.h

```
#ifndef FOO_H
#define FOO_H
class Foo {
public:
    Foo();
    void f();
private:
    int bar;
};
#endif // FOO_H
```

Foo.cpp

```
#include "Foo.h"

Foo::Foo() {
    ...
}

void Foo::f() {
    ...
}
```

DÉFINITION AVEC ECLIPSE

- Voir Démo.

POINTEUR THIS

Dans une fonction membre on peut toujours récupérer un pointeur vers l'objet courant en écrivant `this`

COMPOSITION

- OOP 101 : Il faut non seulement décomposer son code, mais aussi décomposer ses structures de données
- Une classe doit être découpée en sous-classes lorsque cela fait du sens et déléguer
- Exemple : un Tank contient une Tourelle et des Chenilles si le tank tire c'est en déléguant l'opération à la tourelle, pareil quand il avance. La tourelle ne sait pas où se situe le tank, il y a séparation.
- On dit que la classe Tank est composée des classes Tourelle et Chenilles

DÉFINITION DE LA COMPOSITION

- On utilise des variables membres des classes correspondantes
- qu'on initialise avec des listes d'initialisation (voir cas problématique en démo)

COMPOSITION/AGRÉGATION

- La classe A peut contenir la classe B de plusieurs manières :
 - de manière interne : tout objet de la classe A contient physiquement un objet de la classe B, c'est la composition
 - de manière externe : tout objet de la classe A fait référence à un objet de la classe B existant en dehors, c'est l'agrégation
- On peut agréger par référence ou par pointeur (Dans le premier cas liste d'initialisation obligatoirement)

UN EXEMPLE CONCRET

- On veut faire un programme qui manipule des départements et des employés
- Un département aura un certain nombre d'employé fixé, par exemple 2
- Un employé aura un département d'appartenance
- Les employés peuvent changer de département
- On va réaliser deux classes Département et Employe