POO : Visibilité des membres, pointeur 'this' et Constructeur de recopie

L'encapsulation:

L'encapsulation est une manière de définir une classe de telle sorte que ses attributs ne puissent pas être directement manipulés de l'extérieur de la classe, mais seulement indirectement par l'intermédiaire des méthodes. Un des avantages de cette approche est la possibilité de redéfinir la représentation interne des attributs, sans que cela affecte la manipulation externe d'un objet de cette classe.

Visibilité des membres d'une classe :

3 mots clés sont utilisés pour définir un niveau de visibilité donné à un membre de la classe : **private**, **public** et **protected**.

- Le mot clé **private** permet de rendre un membre privé. Les membres privés d'une classe ne sont accessibles que par les méthodes de la classe:
  - Les attributs privés d'une classe ne peuvent être utilisés que par les méthodes de cette classe.
  - Les méthodes privées d'une classe ne peuvent être appelées que par les méthodes de cette classe.
- Le mot clé **public** est appliqué à des membres (en principe seulement des méthodes) pour les rendre accessibles depuis l'extérieur de la classe :
  - Les attributs publiques d'une classe sont donc accessibles (en lecture ou en écriture) depuis n'importe quelle méthode.
  - Les méthodes publiques d'une classe peuvent être appelées par n'importe quelle méthode.
- Le mot clé **protected** offre un niveau de visibilité situé entre les 2 niveaux précédant. Des membres « **protected** » sont des membres inaccessibles en dehors de leur classe sauf dans des classes filles (la partie de l'héritage).

Par défaut, les attributs sont privés. Si vous ne précisez rien une classe C++ est donc encapsulée.

Accesseurs et mutateurs :

Parmi les méthodes pouvant assurer l'accès contrôlé aux propriétés qui sont habituellement privées, il est à définir des méthodes d'introduction et de récupération de données. Ces méthodes sont appelées des Accesseurs (getters) et Mutateurs (Setters). Ainsi, pour une donnée membre nommée « **x** », il est à prévoir les deux méthodes suivantes :

Appelée setter et permettant d'affecter la valeur du paramètre à la propriété « x »

```
void setX(type x);
```

Appelée getter et permettant de retourner la valeur de « x ».

```
Type getX();
```

Exemple :

```
#include <iostream>
#include <string>
class Voiture {
private:
    int  Compteur;
    string Immat;

public:
    Voiture(string im) {
        Compteur = 0; Immat = im;
    }
    bool ChangerPneu() {
        return (Compteur >= 10000);
    }

    int get_Compteur() {
        return Compteur;
    }
    void set_Compteur(int km) {
        Compteur = km;
    }
    string get_Immat() {
        return Immat;
    }
    void set_Immat(string im) {
        Immat = im;
    }
}
```

```
// exemple méthode Choix_afficher ----

void Choix_Afficher() {

      cout << "Immatriculation: " <<
LaVoiture->get_Immat();
      cout << "Kilometrage: " <<
LaVoiture->get_Compteur();
}
```

L'objet « this »

Il existe un objet dynamique (pointeur) prédéfini « this » pouvant être utilisé à l'intérieur d'une classe, permettant de référencer à l'aide de la notation « **this->…** » les membres (propriétés ou méthodes) de l'objet courant. L'utilisation de « this » permettra par exemple de distinguer entre une propriété et une variable locale ou un paramètre.

Exemple :

```
void Note::setValue(double value) {
if (value >= 0 && value <= 20) {
this->value = value;
 }
}
```

Constructeur par copie :

Un constructeur de copie est une fonction membre qui initialise un objet en utilisant un autre objet de la même classe. Un constructeur de copie a le prototype suivant:

```
ClassName(const ClassName& obj);
```

Ici, `obj` est la référence d'un objet utilisé pour initialiser un autre objet.

Exemple du constructeur de copie

```
/* Point.h */
#include<iostream>
using namespace std;

class Point
```

```
/* Point.cpp */
#include "Point.h"
#include <iostream>
using namespace std;
```

```
/* main.cpp */
#include "Point.h"
#include <iostream>
using namespace std;
int main()
```

```
{
private:
	int x, y;

public:
	Point(int, int);

	/* Constructeur de
copie */
	Point(const Point& p);
	void afficher();
};
```

```
	Point::Point(int x1,
int y1)
	{
		x = x1;
		y = y1;
	}

	/* Constructeur de
copie */
	Point::Point(const
Point& p)
	{
		x = p.x;
		y = p.y;
	}

	void Point::afficher()
	{
		cout << x << " "
<< y << endl;
	}
```

```
{
	Point obj1(3, 7);
// Constructeur paramétré
	Point obj2 = obj1;
// Constructeur de copie
	cout << "
Constructeur paramétré : ";
	obj1.afficher();
	cout << "
Constructeur de copie : ";
	obj2.afficher();
	return 0;
}
```