Leçon 6 : La Programmation Orientée Objet 1 en C++

Université Virtuelle de Côte d'Ivoire

Table des matières

1- Généralité de la programmation orientée objet

I

Définition : 1.1- Principe

La programmation orientée objet (POO) consiste à définir des objets logiciels et à les faire interagir entre eux.

Syntaxe : 1.2- Notion de classe

Une classe déclare des propriétés communes à un ensemble d'objets. Elle apparaît comme un type ou un moule à partir duquel il sera possible de créer des objets.

Elle se caractérise par :

- Des variables appelées attributs (on parle aussi de variables membres) ;
- Des fonctions appelées méthodes (on parle aussi de fonctions membres).

Syntaxe :

```
class Nom_class {
// liste des attributs
//Liste des méthodes
};
```

Exemple :

```
class Rectangle
{
// liste des attributs
double longueur,largeur;
//Liste des méthodes
void perimetre();
void aire();
};
```

- *Accéder à une méthode ou attribut de la classe*

Variable.attribut ou *variable.methode()*

Exemple :

Rectangle P;

P.longueur = 10; *// Affecte la valeur 10 à l'attribut longueur de la classe rectangle*

P.perimetre() ; *//Appel la méthode périmètre de la classe Rectangle*

1.3- Notion d'objets

Un objet est une structure de données *(ses attributs c'est à dire des variables)* qui *définit son état* et une interface *(ses méthodes c'est à dire des fonctions)* qui *définit son comportement.*

Un objet est créé à partir d'un modèle appelé *classe*. Chaque objet créé à partir de cette classe est une *instance* de la classe en question.

Un objet possède une identité qui permet de distinguer un objet d'un autre objet (son nom, une adresse mémoire).

Exemple :

Rectangle val ; *// val est une instance (objet) de la classe Rectangle.*

1.4- Notion de visibilité

Le C++ permet de préciser le type d'accès des membres (attributs et méthodes) d'un objet. Cette opération s'effectue au sein des classes de ces objets. Ces types accès sont :

- *public* : les membres publics peuvent être utilisés dans et par n'importe quelle partie du programme.
- *private* : les membres privés d'une classe ne sont accessibles que par les objets de cette classe et non par ceux d'une autre classe.

Syntaxe :

class Nom_class {

private:

// Déclaration des attributs et méthodes privés

public:

// Déclaration des attributs et méthodes publics.

};

Exemple :

class Rectangle

{

private:

double longueur; *// attribut longueur est un membre privé à la classe rectangle donc elle ne peut pas être accessible en dehors de la classe*

public:

double largeur; *// attribut largeur est un membre privé à la classe rectangle donc elle peut être accessible de partout*

};

Exercice

II

[solution n°1 p.16]

Énoncé :

Écrire classe etudiant avec les attributs nom, prenom et aussi un tableau de 5 notes. Ces attributs ne doivent pas être accessible en dehors de la classe. Créer également les méthodes saisie, affichage et moyenne accessibles par tous.

Solution :

{

;

}

2- Création de classes et d'objets

III

Définition : 2.1- Construction d'objets

Pour créer des objets à partir d'une classe, il faut *un constructeur*, *un constructeur* est chargé d'initialiser un objet de la classe et *il est appelé automatiquement au moment de la création de l'objet*.

Un constructeur est une méthode qui porte toujours le même *nom que la classe*, Il existe quelques contraintes à savoir :

- Il peut avoir des paramètres, et des valeurs par défaut.
- Il peut y avoir plusieurs constructeurs pour une même classe.
- Il n'a jamais de type de retour.

********** *Déclaration du constructeur* **********

Syntaxe :

```
class Nom_classe
{
// liste des attributs
//Création du construteur
public:
Nom_classe(paramètre); // constructeur de la classe Nom_classe
};
```

Exemple :

```
class Rectangle
{
private:
// liste des attributs
double longueur,largeur;
public:
```

Rectangle(double longueur, double largeur); *// constructeur de la classe Rectangle*

};

**********Définir le constructeur **********

La définition du constructeur permettra d' initialiser tous les attributs de l'objet au moment de sa création.

Syntaxe :

Nom_classe: :Nom_constructeur(paramètre)

{

Instruction;

}

Exemple :

Rectangle::Rectangle(double longueur, double largeur)

{

// Initialiser tous les attributs de la classe

this->longueur = longueur; *// on affecte l'argument longueur à l'attribut longueur*

this->largeur = largeur; *// on affecte l'argument largeur à l'attribut largeur*

}

NB : Le mot clé <<*this*>> permet de désigner l'adresse de l'objet sur laquelle la fonction membre a été appelée. On peut parler d'auto-pointeur car l'objet s'autodésigne (sans connaître son propre nom).

Remarque :

- *Constructeur par défaut*

Si nous essayons de créer un objet sans lui fournir une longueur et une largeur, nous obtiendrons le message d'erreur suivant :

error: no matching function for call to 'Rectangle::Rectangle()'

Pour corriger cette erreur il faut créer un constructeur par défaut. *Son rôle est de créer une instance non initialisée quand aucun autre constructeur fourni n'est applicable.*

Syntaxe :

Nom_classe: :Nom_constructeur()// Sans paramètre !

{

Instruction;

}

Exemple :

Le constructeur par défaut de la classe Rectangle sera :

```
Rectangle::Rectangle()// Sans paramètre !
{
longueur = 0;
largeur = 0;
}
```

- *Appel du constructeur*

```
// Appel du constructeur par défaut de la classe Rectangle(utilise les valeurs par défaut pour x et y)
Rectangle rect0; // longueur =0 et largeur =0
// Appel du constructeur de la classe Rectangle
Rectangle rect1(10 , 6); // longueur =10 et largeur =6
Rectangle rect2(12 , 4.2);// longueur =12 et largeur =4.2
```

2.2- Allocation dynamique d'objet

Pour allouer dynamiquement un objet en C++, on utilisera l'opérateur *new*. Celui-ci renvoyant une adresse où est créé l'objet en question, il faudra un pointeur pour la conserver. Manipuler ce pointeur, reviendra à manipuler l'objet alloué dynamiquement.

Pour libérer la mémoire allouée dynamiquement en C++, on utilisera l'opérateur *delete*.

Syntaxe :

- *Allouer*

```
Nom_classe *nom_pointeur ;
nom_pointeur = new Nom_constructeur();
```

- *Libérer*

```
delete nom_pointeur;
```

Exemple :

- *Allouer*

```
Rectangle *p3 ; //Créer un pointeur sur objet de type rectangle
p3 = new Rectangle(); //allouer dynamiquement un objet de type Rectangle
```

- *Libérer*

```
delete p3;
```

- *Accéder à une méthode ou attribut de la classe*

Variable->attribut ou *variable->methode()*

Exemple :

```
Rectangle P;
```

P->longueur = 10; *// Affecte la valeur 10 à l'attribut longueur de la classe rectangle*

P->perimetre() ; *//Appel la méthode périmètre de la classe Rectangle*

2.3- Destruction d'objets

Le destructeur est la méthode membre appelée automatiquement lorsqu'une instance (objet) de classe cesse d'exister en mémoire. Son rôle est de libérer toutes les ressources qui ont été acquises lors de la construction (typiquement libérer la mémoire qui a été allouée dynamiquement par cet objet).

Un destructeur est une méthode qui porte toujours le même nom que la classe, précédé de "~".

Il existe quelques contraintes à savoir :

- Il ne possède aucun paramètre.
- Il n'y en a qu'un et un seul.
- Il n'a jamais de type de retour.

********** *Déclaration du Destructeur* **********

Syntaxe :

class Nom_classe

{

// liste des attributs

//Création du Destructeur

public:

~Nom_classe(); *// Destructeur de la classe Nom_classe*

};

Exemple :

class Rectangle

{

public:

~Rectangle(); *// Destructeur de la classe Rectangle*

};

2.4- Application

Écrire un programme qui permet de calculer et afficher le périmètre et l'aire d'un rectangle en utilisant les classes.

```
1 //Cas 1 : Allocation non dynamique
2 #include<iostream>
3 #include<string>
4 using namespace std;
```

```
5  class Rectangle
6  {
7  private :
8  // liste des attributs
9  double longueur,largeur;
10
11 public:
12 Rectangle(); // constructeur de la classe Rectangle par défaut
13 Rectangle(double longueur, double largeur); // constructeur de la classe
   Rectangle
14 ~Rectangle(); // Destructeur de la classe Rectangle
15 void aire(); // méthode de calcul de l'aire
16 void Perimetre();// méthode de calcul du périmètre
17 };
18 Rectangle::Rectangle(double longueur, double largeur)
19 {
20 // Initialiser tous les attributs de la classe
21 this->longueur = longueur; // on affecte l'argument longueur à l'attribut
   longueur
22 this->largeur = largeur; // on affecte l'argument largeur à l'attribut
   largeur
23 }
24 // constructeur par défaut
25 Rectangle::Rectangle()
26 {
27 // Initialiser tous les attributs de la classe
28 longueur = 0; // on affecte 0 à l'attribut longueur
29 largeur = 0; // on affecte 0 à l'attribut largeur~
30 }
31 // Destructeur
32 Rectangle::~Rectangle()
33 {
34     /* Rien à mettre ici car on ne fait pas d'allocation dynamique
35     dans la classe Rectangle. Le destructeur est donc inutile  */
36
37 }
38 // méthode aire
39 void Rectangle::aire()
40 { double air;
41 air = longueur * largeur;
42 cout<<"Aire = "<<air<<endl;
43 }
44 // méthode périmètre
45 void Rectangle::Perimetre()
46 { double peri;
47 peri = (longueur + largeur)*2;
48 cout<<"Périmètre  = "<<peri<<endl;
49 }
50
51
52
53 main()
54 {
55 Rectangle p3(25,20); //Créer une variable sur objet de type rectangle
56
57 p3.aire(); // Appel de la methode aire()
58 p3.Perimetre(); // Appel de la methode Perimetre()
59
60
```

```
61 }
62
63  //Cas 2 : Allocation dynamique
64 #include<iostream>
65 #include<string>
66 using namespace std;
67 class Rectangle
68 {
69 private :
70 // liste des attributs
71 double longueur,largeur;
72
73 public:
74 Rectangle(); // constructeur de la classe Rectangle par défaut
75 Rectangle(double longueur, double largeur); // constructeur de la classe
   Rectangle
76 ~Rectangle(); // Destructeur de la classe Rectangle
77 void aire(); // méthode de calcul de l'aire
78 void Perimetre();// méthode de calcul du périmètre
79 };
80 Rectangle::Rectangle(double longueur, double largeur)
81 {
82 // Initialiser tous les attributs de la classe
83 this->longueur = longueur; // on affecte l'argument longueur à l'attribut
   longueur
84 this->largeur = largeur; // on affecte l'argument largeur à l'attribut
   largeur
85 }
86 // constructeur par défaut
87 Rectangle::Rectangle()
88 {
89 // Initialiser tous les attributs de la classe
90 longueur = 0; // on affecte 0 à l'attribut longueur
91 largeur = 0; // on affecte 0 à l'attribut largeur~
92 }
93 // Destructeur
94 Rectangle::~Rectangle()
95 {
96     /* Rien à mettre ici car on ne fait pas d'allocation dynamique
97     dans la classe Rectangle. Le destructeur est donc inutile  */
98
99 }
100 // méthode aire
101 void Rectangle::aire()
102 { double air;
103 air = longueur * largeur;
104 cout<<"Aire = "<<air<<endl;
105 }
106 // méthode périmètre
107 void Rectangle::Perimetre()
108 { double peri;
109 peri = (longueur + largeur)*2;
110 cout<<"Périmètre  = "<<peri<<endl;
111 }
112
113
114
115 main()
116 {
```

```
117 Rectangle *p3; //Créer un pointeur sur objet de type rectangle
118 p3= new Rectangle(25,20); //allouer dynamiquement un objet de type
    Rectangle
119 p3->aire(); // Appel de la methode aire()
120 p3->Perimetre(); // Appel de la methode Perimetre()
121 delete p3; //Libérer  la mémoire
122
123 }
124
```

Exercice

IV

[solution n°2 p.16]

Énoncé :

Soit la classe etudiant, créer les constructeurs, ensuite écrire leur code.

Solution :

class etudiant

{

private:

string nom,prenom; double notes[5];

public:

(, , ,);

void saisie();

void affichage();

float moyenne();

};

(, , ,)

{

int i;

= nom;

= prenom;

(; i< ;)

{

= notes[i] ;

}

}

```
{
int i;
 = "";
= "";
for ( ; i <  ;  )
{
=0 ;
}
}
```

Solutions des exercices

> **Solution** n°1

Exercice p. 6

Énoncé :

Écrire classe etudiant avec les attributs nom, prenom et aussi un tableau de 5 notes. Ces attributs ne doivent pas être accessible en dehors de la classe. Créer également les méthodes saisie, affichage et moyenne accessibles par tous.

Solution :

```
class etudiant
{
private:
string nom,prenom;
double notes[5];
public:
void saisie();
void affichage();
float moyenne();
}
```

> **Solution** n°2

Exercice p. 14

Énoncé :

Soit la classe etudiant, créer les constructeurs, ensuite écrire leur code.

Solution :

```
class etudiant
{
private:
string nom,prenom; double notes[5];
public:
etudiant();
etudiant(string nom, string prenom,double notes[5]);
void saisie();
void affichage();
float moyenne();
};
etudiant::etudiant(string nom, string prenom,double notes[5])
{
int i;
this->nom = nom;
this->prenom= prenom;
for (i=0; i< 5 ; i++)
{
this->notes[i] = notes[i] ;
}
}
etudiant::etudiant()
{
int i;
this->nom = "";
this->prenom= "";
```

```
for (i=0 ; i < 5 ; i++)
{
notes[i]=0 ;
}
}
```