

Support de cours

Cours:

Introduction à la programmation orientée objet (en C++)

Vidéo:

W12-01-constrintro-CPP-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Attributs hauteur. Méthode spécifique. Seul coup tous. Façon suivante. Bonne façon. Partie de l'interface. Mauvaise solution. Largeur de cette instance. Programmeur utilisateur de la classe. Instances de ces classes. Point de vue de l'encapsulation. Variable auxiliaire. Séquences vidéos précédentes. Programmeur développeur de la classe. Attributs.



[vers la recherche de séquences vidéo](#)

(dans Introduction à la programmation orientée objet (en C++).)



[vers la vidéo](#)

Center for Digital Education. Plus de matériel de soutien pédagogique ici :

<https://www.epfl.ch/education/educational-initiatives/cede/educational-technologies-gallery/boocs-en/>

Constructeurs (introduction)

(Partie 1)

Introduction à la programmation orientée objet (en C++)

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



Dans les vidéos précédentes, nous avons vu comment déclarer des classes et des objets.

On peut par exemple déclarer une instance de la classe `Rectangle` :

```
Rectangle rect;
```

Une fois que l'on a fait cette déclaration, comment faire pour *initialiser les attributs* de `rect` ?

Dans les séquences vidéos précédentes, vous avez appris à déclarer des classes et des instances de ces classes, c'est-à-dire des objets. Par exemple, si vous voulez déclarer une instance « rect » de la classe « Rectangle », vous écrivez simplement comme ceci : « Rectangle rect ; » La question qu'on peut se poser maintenant c'est comment donner des valeurs à ces attributs, voire même comment initialiser, par exemple,

notes

résumé

0m 1s



Première solution : **affecter individuellement une valeur** à chaque attribut

```
Rectangle rect;  
double lu;  
cout << "Quelle hauteur? "; cin >> lu;  
rect.setHauteur(lu);  
cout << "Quelle largeur? "; cin >> lu;  
rect.setLargeur(lu);
```

les attributs hauteur et largeur de cette instance « rect » de la classe « Rectangle ». Pour donner des valeurs, on pourrait utiliser, comme on a vu, les manipulateurs « setHauteur », « setLargeur », c'est-à-dire affecter individuellement une valeur à chaque attribut comme par exemple ici, où on déclare l'instance « rect » de la classe « Rectangle », on utilise une variable auxiliaire « lu » de type « double » pour pouvoir lire donc, ici, par exemple, la hauteur, on affiche ici un message à l'utilisateur, on récupère depuis « cin » une valeur « lu » dans la variable « lu » et puis donc, avec ce manipulateur « setHauteur », on affecte la hauteur

notes

résumé

0m 25s



Initialisation des attributs

Première solution : **affecter individuellement une valeur** à chaque attribut

```
Rectangle rect;
double lu;
cout << "Quelle hauteur? "; cin >> lu;
rect.setHauteur(lu);
cout << "Quelle largeur? "; cin >> lu;
rect.setLargeur(lu);
```

Ceci est une **mauvaise solution** dans le cas général :

- ▶ elle implique que tous les **attributs** fassent **partie de l'interface** (**public**) ou soient assortis d'un manipulateur
 - ☞ casse l'encapsulation
- ▶ oblige le programmeur-utilisateur de la classe à initialiser explicitement tous les attributs
 - ☞ risque d'oubli

et puis de même, on demande la largeur, on la lit dans « lu » et on l'appelle manipulateur « setLargeur » pour affecter donc la largeur de l'instance rect. Mais ceci est une très mauvaise solution parce que ça suppose que tous les attributs, puisqu'on doit bien initialiser tous les attributs, fassent soit partie de l'interface, c'est-à-dire soit en public, ce qui est vraiment très mauvais du point de vue de l'encapsulation,

notes

résumé

1m 1s





notes

résumé

1m 25s



Initialisation des attributs

Première solution : **affecter individuellement une valeur** à chaque attribut

```
Rectangle rect;
double lu;
cout << "Quelle hauteur? "; cin >> lu;
rect.setHauteur(lu);
cout << "Quelle largeur? "; cin >> lu;
rect.setLargeur(lu);
```

Ceci est une **mauvaise solution** dans le cas général :

- ▶ elle implique que tous les **attributs** fassent **partie de l'interface** (**public**) ou soient **assortis d'un manipulateur**
 - ☞ casse l'encapsulation
- ▶ oblige le **programmeur-utilisateur** de la classe à **initialiser** explicitement tous les attributs
 - ☞ risque d'oubli

Le but de l'encapsulation étant justement de séparer clairement l'interface et l'implémentation. Si on avait pour chaque attribut un manipulateur dans l'interface, on traduirait beaucoup trop dans l'interface les choix d'implémentation et c'est dans ce sens là que ça casse l'encapsulation. De plus, au niveau conception, cela implique que ce soit le programmeur utilisateur de la classe qui soit obligé d'initialiser tous les attributs,

notes

résumé

1m 39s





qui doivent penser à initialiser tous les attributs et on a là, un risque d'oubli, un défaut majeur dans la conception. Normalement, c'est plutôt le programmeur développeur de la classe qui devrait avoir la charge d'initialiser les attributs ou en tout cas d'offrir une possibilité d'initialisation à quelques attributs bien choisis par ce programmeur développeur de la classe, ce n'est pas au programmeur utilisateur de la classe

[illegible]

Deuxième solution : définir une **méthode dédiée à l'initialisation** des attributs

```
class Rectangle {  
public:  
    void init(double h, double L)  
    {  
        hauteur = h;  
        largeur = L;  
    }  
    ...  
private:  
    double hauteur;  
    double largeur;  
};
```

de penser à tous les attributs possibles qui doivent être initialisés. Une autre solution, comme d'habitude, consiste à créer une méthode spécifique pour la tâche que l'on veut faire. Ici, c'est initialiser les attributs, donc créer une méthode spécifique pour initialiser les attributs. Par exemple, on pourrait comme ceci ajouter une méthode disons « init » qui initialiserait les deux attributs hauteur et largeur de notre rectangle et pour pouvoir le faire, elle devrait recevoir une valeur, ici, « h » pour mettre dans la hauteur et une valeur, ici, « L » pour mettre dans la largeur. Cette méthode « init » serait un peu comme un « set » à la fois hauteur

notes

résumé

2m 25s



Deuxième solution : définir une **méthode dédiée à l'initialisation** des attributs

```
class Rectangle {  
public:  
    void init(double h, double L)  
    {  
        hauteur = h;  
        largeur = L;  
    }  
    ...  
private:  
    double hauteur;  
    double largeur;  
};
```

*Rectangle rect;
rect.init(3.0, 4.0);*

et à la fois largeur de « set » de tous les attributs pour initialiser d'un seul coup tous les attributs qui ont besoin d'être initialisés. On l'utiliserait de la façon suivante : en déclarant un « Rectangle rect ». On ferait appel à la méthode « init » en lui passant deux valeurs par exemple 3.0 pour la hauteur et 4.0 pour la largeur.

notes

résumé

3m 1s





Cette façon de faire est certainement une très bonne façon,

notes

résumé

3m 25s



Les constructeurs

Un constructeur est une méthode :

- ▶ invoquée *automatiquement* lors de la déclaration d'un objet
- ▶ chargée d'effectuer toutes les opérations requises en « début de vie » de l'objet (dont *l'initialisation des attributs*)

Syntaxe de base :

```
NomClasse(liste_paramètres)
{
    /* initialisation des attributs
       en utilisant liste_paramètres */
}
```



c'est tellement une bonne façon, qu'elle est déjà disponible en tant que tel en C++. C'est ce que l'on appelle les constructeurs. Les constructeurs sont ces méthodes particulières qui vont avoir la responsabilité d'initialiser la classe. Un constructeur est donc une méthode qui est appelée automatiquement lors de la déclaration de chaque objet, au début de vie de chaque instance. Son rôle est donc justement de faire ce qui doit être fait au début de la vie d'un objet. Comme par exemple, initialiser ses attributs. initialiser ses attributs.

notes

résumé

3m 26s

