

Support de cours

Cours:

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

Vidéo:

W13-01-static-CPP-pt1

Concepts (extraits des sous-titres générés automatiquement) :

Instance de la classe. Intérieur d'une classe. Petit exemple du rectangle. Zone mémoire accessible. Objet de type rectangle. Instance de rectangle. Variable globale. Attribut de la classe rectangle. Séquences précédentes. Objet de la séquence. Attribut de classe. Zone mémoire. Mauvaise solution. Ensemble des instances. Seul moyen.



[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/>

Variables et méthodes de classe

(Partie 1)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s





Dans les séquences précédentes, nous avons vu que les attributs

notes

résumé

0m 1s

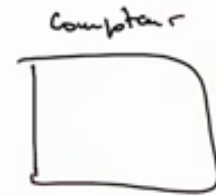


Revenons à notre exemple du *comptage des instances* :

```
int compteur(0); // Hmm....

class Rectangle {
    // constructeur par défaut
    Rectangle(): hauteur(0.0), largeur(0.0)
    { ++compteur; }

    // destructeur
    ~Rectangle() { --compteur; }
    //...
```



Oh horreur !.. Nous avons utilisé une *variable globale* !

C'est une **très mauvaise solution** ! (contraire au principe d'encapsulation, effets de bord, mauvaise modularisation)

définis à l'intérieur d'une classe représentent des informations qui sont spécifiques à chaque instance de la classe. Si l'on reprend notre petit exemple du rectangle, chaque instance de rectangle, chaque objet de type Rectangle va avoir sa propre largeur, sa propre hauteur, des informations qui lui sont spécifiques et qui la caractérisent. Que se passe-t-il cependant si plusieurs instances d'une même classe ont besoin d'accéder à une même information, une information commune ? C'est l'objet de la séquence qui suit. Pour illustrer le problème qui nous intéresse, revenons à notre petit exemple du comptage d'instances que nous avons déjà eu l'occasion d'étudier, dans nos compléments sur la construction/destruction d'objet. Il s'agissait donc de comptabiliser l'ensemble des instances existant à un moment donné dans un programme, et il fallait pour cela disposer d'une zone mémoire accessible par toutes les instances, qui était incrémentée à chaque création d'instance et décrétementée à chaque destruction. Au moment où nous avons abordé ce problème pour la première fois, le seul moyen dont nous disposions pour pouvoir définir cette zone mémoire, qui serait accessible par toutes les instances pour être incrémentée et décrétementée, était de passer par une variable globale. Nous avons pris le soin, à l'époque, d'assortir la déclaration de notre variable globale d'un petit : " Hmm... " dubitatif. En effet, utiliser une variable globale est quelque chose de typiquement néfaste dans un programme, il s'agit d'une très mauvaise solution. Une variable globale, telle que celle-ci, n'est attachée à aucune entité du programme. Ceci est donc contraire au principe d'une bonne encapsulation et traduit une mauvaise modularisation. De plus, cela peut induire des effets de bord tout à fait néfastes. En effet, une telle variable est accessible par différents endroits du programme, de façon tout à fait incontrôlée. Il s'agit donc de quelque chose à

notes

résumé

0m 5s

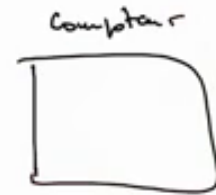


Revenons à notre exemple du *comptage des instances* :

```
int compteur(0); // Hmm....

class Rectangle {
    // constructeur par défaut
    Rectangle(): hauteur(0.0), largeur(0.0)
    { ++compteur; }

    // destructeur
    ~Rectangle() { --compteur; }
    //...
```



Oh horreur !.. Nous avons utilisé une *variable globale* !

C'est une **très mauvaise solution** ! (contraire au principe d'encapsulation, effets de bord, mauvaise modularisation)

éviter à tout prix.

notes

résumé

La solution à ce problème consiste à utiliser un **attribut de classe** :

```
class Rectangle {  
private:  
    double hauteur, largeur;  
    static int compteur;  
    //...  
};
```

Alternativement, définir le compteur comme un attribut de la classe rectangle n'est clairement pas non plus une bonne solution, parce que cela signifierait que chaque instance de rectangle a son propre compteur, et ce n'est pas ce que nous souhaitons. Nous souhaitons avoir une zone commune accessible par toutes les instances. Donc cette solution n'est pas une bonne solution non plus. La solution à ce problème consiste à définir l'information que l'on souhaite commune à toutes les instances d'une classe, comme étant un attribut de classe. Contrairement aux attributs relatifs aux instances, qui étaient ceux que l'on avait l'habitude de définir jusqu'ici.

notes

résumé

1m 49s

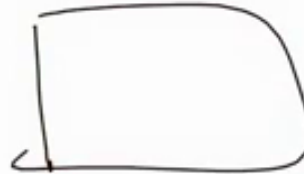


La solution à ce problème consiste à utiliser un **attribut de classe** :

```
class Rectangle {
private:
    double hauteur, largeur;
    static int compteur;
    //...
};
```



Rectangle r1;
Rectangle r2;



Un attribut de classe est donc un attribut qui est défini à l'intérieur de la classe, mais est assorti du mot réservé static. Typiquement, si on imagine que dans un programme co-existent plusieurs instances de Rectangle, ici deux pour simplifier, la situation en mémoire serait la suivante : L'objet r1 est un objet qui aurait ses propres valeurs spécifiques pour ses attributs d'instance, à savoir la hauteur et la largeur. Il en va de même pour l'objet r2,

notes

résumé

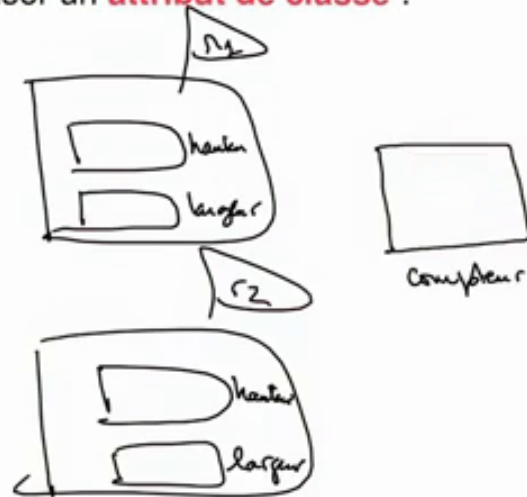
2m 25s



La solution à ce problème consiste à utiliser un **attribut de classe** :

```
class Rectangle {
private:
    double hauteur, largeur;
    static int compteur;
    //...
};
```

Rectangle r1;
Rectangle r2;



qui aurait ses propres valeurs pour la largeur et la hauteur. Par contre l'attribut de classe, compteur, est une zone mémoire unique qui est accessible par toutes les instances, mais qui n'existe en mémoire qu'une seule fois. Notez que les règles de visibilité pour un attribut static sont les mêmes que pour les attributs non static. Cela va se traduire par le fait que cette zone mémoire

notes

résumé

3m 1s



La solution à ce problème consiste à utiliser un **attribut de classe** :

```
class Rectangle {  
private:  
    double hauteur, largeur;  
    static int compteur;  
    //...  
};
```

- ▶ La déclaration d'un attribut de classe est précédée du mot clé **static**
- ▶ Un attribut de classe est **partagé par toutes les instances** de la même classe (on parle aussi d'« *attribut statique* »)
- ▶ Il existe même lorsqu'*aucune* instance de la classe n'est déclarée
- ▶ Un attribut de la classe peut être **privé** ou **public**

est accessible sans qu'aucune instance ne soit créée, uniquement au travers du nom de la classe. Très concrètement, si dans un programme j'ai déclaré une classe Rectangle comme ceci, avec un attribut de classe, je peux accéder à cet attribut de classe sans avoir créé aucune instance de Rectangle au préalable. Ceci va se faire au moyen d'une notation particulière, qui est la suivante : J'accède à la variable de la classe Rectangle, qui s'appelle compteur, au travers de l'opérateur de résolution de portée. Pour résumer, définir un attribut de classe revient à définir un attribut mais précéder sa déclaration du mot réservé static. Nous avons vu qu'un attribut de classe était partagé par toutes les instances. Il s'agit d'une zone mémoire unique partagée par toutes les instances. Et cette zone mémoire existe même lorsque nous n'avons créé aucune instance de la classe. Nous avons vu que nous pouvions y accéder par le biais de cette notation. Pour terminer, un attribut de classe peut, comme un attribut d'instance être privé ou public. être privé ou public.

notes

résumé

3m 37s

