

Support de cours

Cours:

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

Vidéo:

### W13-01-static-CPP-pt3

Concepts (extraits des sous-titres générés automatiquement) :

**Attribut de classe. Usage des méthodes static. Attribut static. Intérieur d'une classe. Méthode static. Méthode d'instance. Méthode de classe. Genre de tournure. Methode1 de la classe a.. Travers du nom de la classe. Simples fonctions usuelles. Création d'une instance. Objet de type rectangle. Consigne générale. Méthodes.**



[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 3)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



## Méthodes de classe

Similairement, si on ajoute `static` à une méthode :

- ▶ on peut accéder aussi à la méthode *sans* objet, à partir du nom de la classe et de l'opérateur de résolution de portée « `::` »

```
class A {
public:
    static void methode1() { cout << "Méthode 1" << endl; }
    void methode2() { cout << "Méthode 2" << endl; }
};

int main () {
    A::methode1(); // OK
    A::methode2(); // ERREUR
    A x;
    x.methode1(); // OK
    x.methode2(); // OK
}
```

*n.surface()*

Vous savez donc maintenant ce qu'est un attribut de classe, un attribut static. Est-il possible de faire la même chose avec des méthodes ? La réponse est oui, nous le présentons ici par souci d'exhaustivité, même si l'usage des méthodes static est relativement peu courant et est peu recommandé en programmation orientée objet. Une méthode static est donc simplement une méthode définie à l'intérieur d'une classe mais dont la déclaration serait précédée du mot réservé static. Pour invoquer une méthode d'instance, c'est-à-dire les méthodes avec lesquelles nous avons l'habitude de travailler jusqu'ici, nous sommes dans l'obligation d'avoir au préalable créé un objet et invoqué la méthode sur cet objet. Par exemple le calcul d'un surface de rectangle nécessite d'avoir créé un objet de type Rectangle et d'invoquer le calcul de surface sur le rectangle. Une méthode de classe peut par contre parfaitement être invoquée sans qu'il y ai eu au préalable de création d'un quelconque objet de la classe. Par exemple, ici, si dans une classe A, je dispose d'une méthode static, methode1, je peux appeler cette méthode par le biais de l'opérateur de résolution de portée sans avoir au préalable créé un quelconque objet de type A. On dit donc simplement ici que l'on appelle la methode1 de la classe A. Ce genre de tournure n'est évidemment pas licite pour des méthodes non static. Par exemple, ici, dans la classe A, nous avons une seconde méthode, methode2, qui n'est pas déclarée comme static. Si nous essayons de l'invoquer uniquement au travers du nom de la classe, sans passer par la création d'une instance, nous aurons une erreur. Pour rappeler methode2 qui est non static, il faut impérativement avoir créé un objet de type A, sur lequel invoquer methode2. C'est ce que nous faisons également ici, pour calculer la surface d'un rectangle r donné. Vous noterez enfin au passage

notes

résumé

0m 1s



Similairement, si on ajoute `static` à une méthode :

- ▶ on peut accéder aussi à la méthode *sans* objet, à partir du nom de la classe et de l'opérateur de résolution de portée « `::` »

```
class A {  
public:  
    static void methode1() { cout << "Méthode 1" << endl; }  
    void methode2() { cout << "Méthode 2" << endl; }  
};  
  
int main () {  
    A::methode1(); // OK  
    A::methode2(); // ERREUR  
    A x;  
    x.methode1(); // OK  
    x.methode2(); // OK  
}
```

*n.surface()*

qu'une méthode static peut, elle, être invoquée soit au travers du nom de la classe, indépendamment de la création de toute instance, ce que nous avons fait ici, mais qu'elle peut également être invoquée au travers d'une instance, si une instance existe. Donc ce genre de tournure est licite, même s'il est peu courant et peu recommandé. Enfin, ce genre de tournure d'accès à un membre static via une instance est aussi possible pour les attributs, sous réserve bien sûr, de l'accessibilité de l'attribut. Le fait qu'une méthode de classe soit invoquable indépendamment de la création de toute instance

notes

résumé

## Restrictions sur les méthodes de classe

Puisqu'une méthode de classe peut être appelée sans objet :

- ▶ elles n'ont pas le droit d'utiliser de méthode ni d'attribut d'instance (y compris `this`)
- ▶ elles ne peuvent accéder **seulement** **qu'à** d'autres méthodes ou **attributs de classe**

☞ Ce sont simplement des *fonctions usuelles* mises dans une classe.

Le recours à des méthodes de classe ne se justifie que dans des situations **très** particulières :

- ▶ affichage spécifique d'attributs de classe
- ▶ manipulation d'attributs de classe non constants et `private`

☞ Préférez toujours les fonctions usuelles et évitez absolument la prolifération de `static` !

induit un certain nombre de restrictions sur ce que l'on peut faire à l'intérieur d'une telle méthode. En effet, puisqu'une méthode de classe ne peut pas présupposer de l'existence d'un objet courant sur lequel elle s'applique, elle ne pourra pas à son tour utiliser de méthode ou d'attribut qui présuppose l'existence d'un objet. Donc elle ne pourra pas utiliser de méthode ou d'attribut d'instance. Ceci veut dire, en clair, qu'une méthode de classe ne peut accéder qu'à d'autres méthodes ou attributs de classe. Une méthode static ne peut utiliser que des membres static à son tour. On peut donc parfaitement voir ces méthodes de classe comme étant de simples fonctions usuelles qui ont été placées dans une classe. Dans les langages comme C++ où il existe une couche non-objet où l'on peut utiliser des fonctions usuelles sans passer par la création de classes, le recours à des méthodes de classe ne se justifie que dans des situations très particulières. On peut imaginer comme exemple, la nécessité de créer une méthode de classe pour afficher un certain nombre de valeurs d'attributs de classe. On peut aussi imaginer qu'il soit nécessaire d'en définir pour manipuler des attributs de classe, des attributs static qui soient non-constants et qui seraient privés, donc on aurait besoin de définir de telles méthodes. La consigne générale, c'est de toujours préférer les fonctions usuelles en C++, et non pas les méthodes de classe. Et puis surtout, d'éviter absolument la prolifération de static dans un programme. Pourquoi ? Puisqu'un membre static peut être invoqué indépendamment de la création de tout objet, cela casse toute l'approche orientée objet. toute l'approche orientée objet.

notes

résumé

2m 25s

