

Support de cours

Cours:

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

Vidéo:

### W11-02-progclasses-CPP-pt3

Concepts (extraits des sous-titres générés automatiquement) :

**Déclaration des méthodes. Sein de la classe. Cas de notre exemple de la classe. Erreur de débutant. Cas de notre classe. Hauteur fois. Premières écritures de méthodes. Nom de la méthode. Liste des éventuels paramètres. Attributs de la classe. Type de retour. Entête de la méthode. Méthode surface. Seule différence. Définition de la méthode.**



[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/>

# Classes, objets, attributs et méthodes en C++ (Partie 3)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

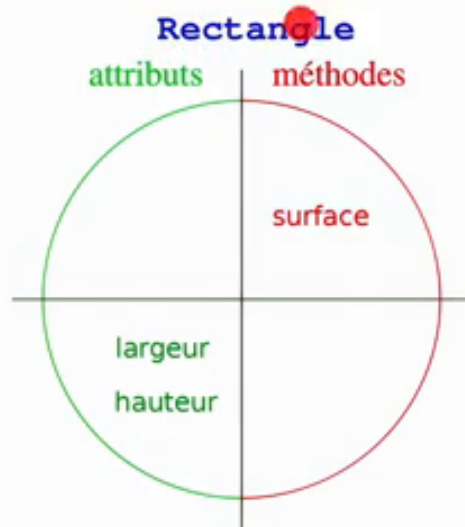
...

notes

résumé

0m 0s





Passons maintenant à la déclaration des méthodes. Les méthodes sont simplement des fonctions que l'on va mettre au sein de la classe.

notes

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

résumé

0m 1s



.....

.....

.....

.....

.....

## Déclaration des méthodes

La syntaxe de la définition des méthodes d'une classe est la syntaxe normale de définition des fonctions :

```
type_retour nom_methode (type_param1 nom_param1, ...)
{
    // corps de la méthode
    ...
}
```

mais elles sont simplement mises *dans* la classe elle-même.

Exemple : la méthode `surface()`  
de la classe `Rectangle` :

```
class Rectangle {
    //...
    double surface()
    {
        return hauteur * largeur;
    }
};
```

mais où sont passés les paramètres ?

```
double surface(double hauteur, double largeur)
{
    return hauteur * largeur;
}
```

Dans le cas de notre classe « Rectangle », on va s'intéresser ici à lui ajouter la méthode « surface ». Une méthode se déclare simplement comme une fonction avec le type de retour, le nom de la méthode puis entre parenthèses rondes, la liste des éventuels paramètres ; tout ceci qui est l'entête de la méthode étant suivi par la définition de la méthode, le corps de cette méthode entre accolades. La seule différence, c'est que les méthodes sont mises dans la classe elle-même. Dans le cas de notre exemple de la classe « Rectangle », la méthode `surface` s'ajouterait donc dans la classe « Rectangle », ici avec sa valeur de retour, `double`, -- une surface retourne un `double` --, son nom ; et puis ici elle n'a pas besoin de prendre de paramètres ; et elle retournerait la hauteur fois la largeur. La question qu'on pourrait se poser est « Mais où sont donc passés les paramètres ? » En effet, si on avait à écrire une fonction usuelle, non pas une méthode, mais une fonction au sens normal du terme, on aurait eu besoin de passer la hauteur et de passer la largeur comme arguments à la fonction de sorte que hauteur et largeur soient connues pour effectuer ce calcul. Comment se fait-il qu'ici dans la méthode de la classe « Rectangle », on n'ait rien eu besoin de passer ?

notes

résumé

0m 13s



Les attributs d'une classe constituent des variables *directement accessibles* dans toutes les méthodes de la classe (*i.e.* des « variables *globales à la classe* »).

On parle de « portée de classe ».



Il n'est donc **pas nécessaire de les passer comme arguments des méthodes.**

Par exemple, dans toutes les méthodes de la classe `Rectangle`, l'identificateur `hauteur` (resp. `largeur`) fait *a priori* référence à la valeur de l'attribut `hauteur` (resp. `largeur`) de l'instance concernée (par l'appel de la méthode en question)

Parce que tout simplement, cette hauteur et cette largeur sont des attributs de la classe. La méthode `surface` faisant partie de la classe « `Rectangle` », elle a accès aux attributs `hauteur` et `largeur` de la classe « `Rectangle` ».

notes

résumé

1m 25s



## Déclaration des méthodes

Les méthodes sont donc :

- ▶ des fonctions propres à la classe
- ▶ qui ont donc **accès** aux attributs de la classe

👉 Il ne faut donc **pas** passer les attributs comme arguments aux méthodes de la classe !

Exemple :

```
class Rectangle {
    //...
    double surface()
    {
        return hauteur * largeur;
    }
};
```

C'est ce que l'on appelle techniquement « une portée de classe », c'est-à-dire que tous les attributs d'une classe sont connus de l'entièreté de cette classe et en particulier de chacune de ces méthodes. Par exemple, toutes les méthodes de la classe « Rectangle » ont accès à la hauteur et à la largeur. Il n'est donc pas nécessaire de passer les attributs comme arguments aux méthodes. Pour résumer, une méthode, c'est simplement une fonction qui est propre à une classe, on va donc la mettre dans les classes, mais qui en plus, de ce fait, a accès aux différents attributs de la classe. Et donc il est très important, et c'est une erreur de débutant, des premières écritures de méthodes, il est très important de ne pas passer les attributs à une méthode de classe. Pour revenir encore une fois sur l'exemple, l'exemple est tout à fait correct ici, on a bien une méthode surface qui ne prend pas du tout ici de paramètres et qui peut accéder à la hauteur et à la largeur de « Rectangle » qui ont été déclarés comme précédemment dans la classe. comme précédemment dans la classe.

notes

résumé

1m 37s

