

Support de cours

Cours:

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

Vidéo:

### W11-03-publicprivate-CPP-pt1

Concepts (extraits des sous-titres générés automatiquement) :

**Séquence vidéo précédente. Méthodes d'une classe. Parties visibles. Extérieur de la classe. Tour des aspects. Droits d'accès. Aspect de ce diagramme. Instances d'autres classes. Instances de la classe. Mots-clé. Objet de cette séquence. Concepts généraux de la programmation. Méthodes de la classe. Parties cachées. Classe.**



[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/>



public: **et** private:  
(Partie 1)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s





Dans la séquence vidéo précédente,

notes

---

---

---

---

---

---

---

---

---

---

résumé

0m 1s



---

---

---

---

---

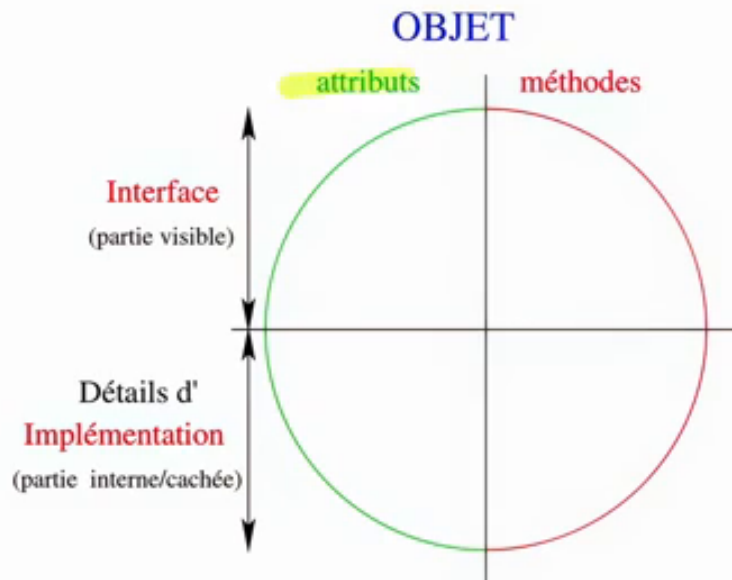
---

---

---

---

---



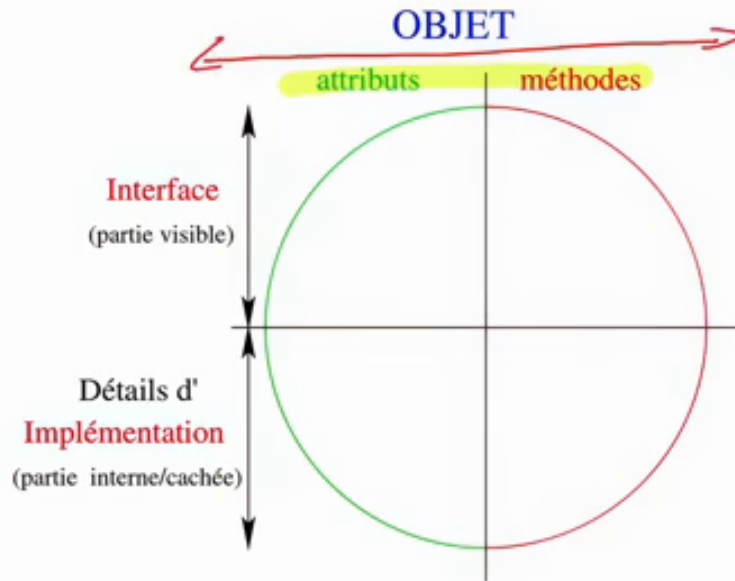
nous avons vu comment déclarer concrètement une classe, comment déclarer concrètement des instances en C++ nous avons vu comment déclarer et utiliser à la fois les attributs et les méthodes. Mais nous n'avons pas fait le tour des aspects « encapsulation » et « abstraction », puisque pour l'instant, nous avons vu uniquement les attributs et les méthodes,

notes

résumé

0m 5s





donc si l'on veut, cet axe-là de l'aspect de ce diagramme,

notes

résumé

0m 25s





mais on n'a pas encore vu comment séparer en parties visibles, ce qu'on appelle l'interface, et en parties cachées, les détails d'implémentation. Donc on n'a pas encore vu cet axe-ci de ce diagramme, ce qui fait l'objet de cette séquence.

notes

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

résumé

0m 29s



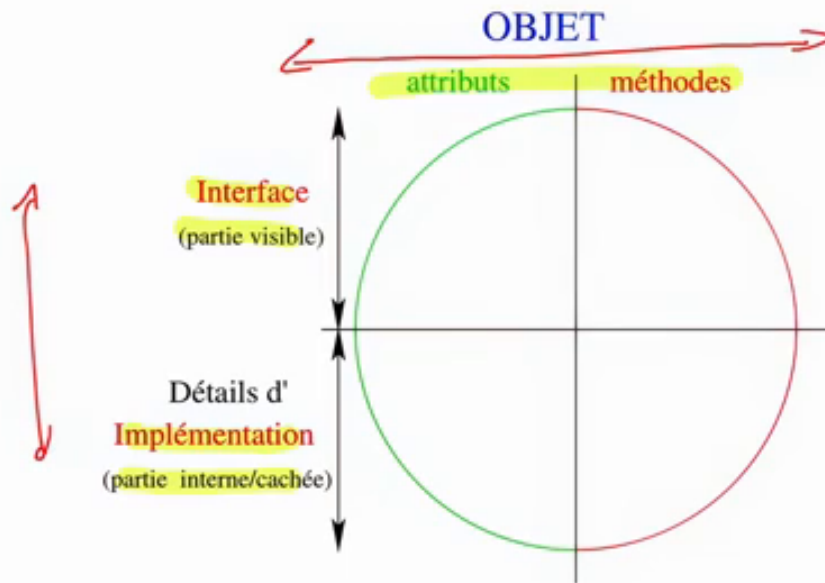
.....

.....

.....

.....

.....



Comme nous avons vu dans la vidéo sur les concepts généraux de la programmation orientée objets, il n'est pas bon que tous les attributs et toutes les méthodes d'une classe soient accessibles à tout le monde ; il vaut même mieux, d'ailleurs, limiter l'accès à quelques méthodes bien choisies. Concrètement, pour la classe « Rectangle »

notes

résumé

0m 45s



## Encapsulation et interface

Tout ce qu'il n'est pas nécessaire de connaître à l'extérieur d'un objet devrait être dans le corps de l'objet et identifié par le mot clé `private` :

```
class Rectangle {
    double surface() const;
private:
    double hauteur;
    double largeur;
};
```

Attribut d'instance **privée** = inaccessible depuis l'extérieur de la classe.  
C'est également *valable pour les méthodes*.

Erreur de compilation si référence à un(e) attribut/méthode d'instance privée :

`Rectangle.cc:16: 'double Rectangle::hauteur' is private`

**Note :** Si aucun droit d'accès n'est précisé, c'est `private` par défaut.

nous avons décidé de limiter l'interface à la méthode « surface » et de mettre les attributs « largeur » et « hauteur » dans la partie cachée, dans les détails d'implémentation. Comment traduire cela concrètement en C++ ? Pour cela, on va utiliser deux mots-clé, qui sont les mots-clé « public » et « private ». Commençons par « private », qui va déclarer la partie détails d'implémentation, la partie qui va rester cachée, la partie qui est privée. Donc ici, on va dire que les attributs hauteur et largeur sont « private ». Une fois qu'on a mis le mot-clé « private » suivi des deux points,

notes

résumé

1m 1s







toutes les variables, les méthodes, qui suivent ce mot-clé « private » sont toutes inaccessibles depuis l'extérieur de la classe, c'est-à-dire, restent accessibles uniquement aux méthodes de la classe en question.

notes

---

---

---

---

---

---

---

---

---

---

résumé

1m 37s



---

---

---

---

---

---

---

---

---

---

## Encapsulation et interface

Tout ce qu'il n'est pas nécessaire de connaître à l'extérieur d'un objet devrait être dans le corps de l'objet et identifié par le mot clé `private` :

```
class Rectangle {
    double surface() const;
    private:
    double hauteur;
    double largeur;
};
```

*Rectangle rect;  
rect.hauteur = 3.6;*

Attribut d'instance **privée** = inaccessible depuis l'extérieur de la classe.  
C'est également valable pour les méthodes.

Erreur de compilation si référence à un(e) attribut/méthode d'instance privée :

`Rectangle.cc:16: 'double Rectangle::hauteur' is private`

**Note :** Si aucun droit d'accès n'est précisé, c'est `private` par défaut.

Notez que l'on peut aussi mettre des méthodes dans la partie privée. Ce seront simplement des méthodes que seules les instances de la classe pourront appeler, et pas les instances d'autres classes ni appelables depuis le « main ». « Inaccessible depuis l'extérieur de la classe », cela veut simplement dire que par exemple, si dans le « main » ou dans une autre classe, on déclare comme ceci une instance « rect » de la classe « Rectangle » et que l'on cherche par exemple à accéder directement à la hauteur alors vous aurez le message que vous essayez d'accéder à, par exemple, hauteur et que c'est privé. Donc vous aurez un message du compilateur : vous ne pouvez pas le faire, cette partie « is private ». Par défaut, si vous ne spécifiez rien du tout, donc par exemple, si je n'avais pas mis cette classe-là, comme on l'a fait jusqu'à maintenant, donc par défaut, tout ce qui est déclaré dans la classe est « private ». En toute rigueur, sur l'exemple que l'on a ici,

notes

résumé

1m 51s



À l'inverse, l'interface, qui est accessible de l'extérieur, se déclare avec le mot-clé `public`

```
class Rectangle {  
public:  
    double surface() const;  
private:  
    // ...  
};
```

comme on n'a rien déclaré ici, la méthode « surface » est elle aussi « private ». Ceci-dit, je vous recommande de systématiquement expliciter les droits d'accès et donc expliciter, mettre « private » pour la partie « private » et donc mettre le mot-clé qui va permettre que l'on puisse accéder à la méthode surface de l'extérieur, qui est le mot-clé « public ». Donc par exemple, ici, on veut que la méthode surface soit accessible à tous, soit visible de l'extérieur, fasse partie de l'interface, et donc, on rajoute le mot-clé « public » suivi des deux points.

### notes

### résumé

2m 49s



À l'inverse, l'interface, qui est accessible de l'extérieur, se déclare avec le mot-clé `public`

```
class Rectangle {  
public:  
    double surface() const;  
private:  
    // ...  
};
```

*Rectangle rect;*

Comme le mot-clé « `private` », tout ce qui est depuis le mot-clé « `public` » jusqu'à un autre mot « `private` » est dans la zone de l'interface, est accessible au public, est accessible de l'extérieur de la classe. Concrètement, si je reprends l'exemple précédent où on avait dans le « `main` » déclaré une instance « `rect` » de la classe « `Rectangle` »,

notes

résumé

3m 25s



## Encapsulation et interface (2)

À l'inverse, l'interface, qui est accessible de l'extérieur, se déclare avec le mot-clé `public`

```
class Rectangle {
public:
    double surface() const;
private:
    // ...
};
```

*Rectangle rect;*  
*g = rect.surface();*

je pourrais dans ce « main », avec une variable par exemple, z « double » qui aurait déjà été déclarée au préalable, appeler la méthode surface... de la classe « Rectangle » ,

notes

résumé

3m 47s





puisque cette méthode surface est en « public », elle est accessible depuis n'importe où et en particulier depuis le « main ». Si par contre, on avait oublié ce mot « public » ici, alors par défaut le droit d'accès est un droit privé et on ne pourrait pas appeler la méthode surface à cet endroit-là. C'est exactement pour ça que dans la vidéo précédente, nous avons dit que l'exemple que nous vous donnions n'était pas tout à fait exact, comme nous n'avions pas précisé de droits d'accès, tout était en privé, donc nous n'avions pas le droit d'accéder aux attributs et aux méthodes depuis le « main ». depuis le « main ».

notes

résumé

4m 1s

