

Support de cours

Cours:

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

Vidéo:

### W11-03-publicprivate-CPP-pt2

Concepts (extraits des sous-titres générés automatiquement) :

**Valeur de la hauteur. Plupart des méthodes. Hauteur de mon rectangle. Valeur d'un attribut. Hauteur d'un rectangle. Privé tous. Valeur de l'attribut hauteur. Extérieur de la classe. Valeur du même type. Largeur du rectangle. Hauteur du rectangle. Hauteur de l'instance courante. Attributs d'une classe. Hauteur de rect1. Bonnes pratiques.**



[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 2)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



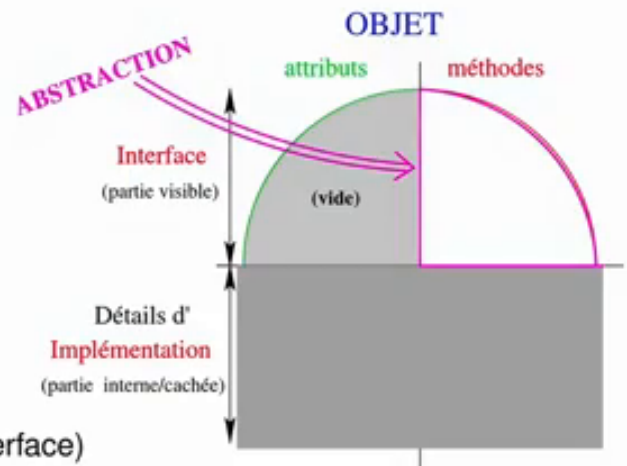
## Encapsulation et interface (2)

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:
    // ...
};
```

Dans la plupart des cas :

- ▶ **Privé :**
  - ▶ Tous les attributs
  - ▶ La plupart des méthodes
- ▶ **Public :**
  - ▶ Quelques méthodes bien choisies (interface)



Laissez-moi maintenant vous rappeler les bonnes pratiques qui ont été présentées dans la partie très générale sur les principes de l'orientée objet,

notes

résumé

0m 1s



## « Accesseurs » et « manipulateurs »

Tous les attributs sont privés ?

- Et si on a besoin de les utiliser depuis l'extérieur de la classe ?!

Par exemple, comment « manipuler » la largeur et la hauteur d'un rectangle ?

*Rectangle rect;*

*...*

*cout << ~~rect~~ . ~~hauteur~~ << endl;*

*N'*

qui vise donc, que l'on mettra en privé tous les attributs et la plupart des méthodes qui vont servir en interne et qu'on mettra dans la partie « public » uniquement quelques méthodes bien choisies, ce que l'on appelle l'interface. Il est vraiment important de n'avoir absolument aucun attribut ici dans la partie « public ». Ceci-dit, vous me direz, mais si tous les attributs sont en privé, comment faire si on a besoin de les utiliser ? Par exemple, si je voulais changer la hauteur de mon rectangle -- ceci en soi est discutable, est-ce qu'on veut vraiment changer la hauteur d'un rectangle ? Mais soit. -- Supposons que l'on veuille donc accéder à la hauteur du rectangle, ou ne serait-ce que la connaître, comment faire pour connaître la hauteur ou la largeur du rectangle, s'ils sont en privé, puisque je ne peux pas y accéder depuis l'extérieur de la classe ? Je ne peux par exemple pas faire quelque chose comme ceci ; déclarer un nouveau « Rectangle » et vouloir afficher sa hauteur. Je ne peux pas faire cela

notes

résumé

0m 12s





parce que l'attribut est en privé.

notes

---

---

---

---

---

---

---

---

---

---

résumé

1m 13s



---

---

---

---

---

## « Accesseurs » et « manipulateurs »

Tous les attributs sont privés ?

- Et si on a besoin de les utiliser depuis l'extérieur de la classe ?!

Par exemple, comment « manipuler » la largeur et la hauteur d'un rectangle ?

*Rectangle rect;*

*...*

*cout << ~~rect.hauteur~~ << endl;*

*NON!*

Pour cela, on va mettre dans l'interface les méthodes que nous estimons nécessaires pour manipuler les attributs en modification ou en consultation, donc par exemple, une méthode pour accéder à la hauteur, ou une méthode pour modifier la largeur.

notes

résumé

1m 17s





Par exemple, ici on voudrait une méthode qui permet d'accéder à la valeur de la hauteur et on écrirait quelque chose comme ça, au travers d'une méthode « get » hauteur ». J'insiste sur le fait que cette partie de la conception est extrêmement importante, il ne faut pas systématiquement mettre des méthodes qui permettent de modifier ni même de lire tous les attributs d'une classe, mais il faut bien réfléchir à quels sont les attributs que l'on veut offrir au travers d'une méthode, soit en modification soit en accès, en lecture,

notes

résumé

1m 32s



## « Accesseurs » et « manipulateurs »

Si le programmeur *le juge utile*, il **inclut les méthodes publiques nécessaires** ...

### 1. Accesseurs (« méthodes get » ou « getters ») :

- Consultation (i.e. « prédicat »)
- Retour de la valeur d'une variable d'instance précise

```
double getHauteur() const { return hauteur; }
double getLargeur() const { return largeur; }
```

### 2. Manipulateurs (« méthodes set » ou « setters ») :

- Modification (i.e. « action »)
- Affectation de l'argument à une variable d'instance précise

```
void setHauteur(double h) { hauteur = h; }
void setLargeur(double l) { largeur = l; }
```

depuis l'extérieur. De telles méthodes sont ce qu'on appelle des « accesseurs », pour accéder aux attributs, et des « manipulateurs », pour les modifier. Les accesseurs, aussi appelés « méthodes get » ou « getters » sont donc des « prédicats », puisqu'ils ne vont pas changer les attributs, on mettra donc le mot-clé « const » derrière leur entête, et permettent de retourner la valeur d'un attribut, donc par exemple, si on veut retourner la valeur de l'attribut hauteur, hauteur étant de type « double », on va retourner ici une valeur du même type on va retourner la hauteur, donc un « double getHauteur » la méthode est donc un prédicat, donc on va écrire ici « const » et on écrira simplement « return hauteur; ». Je vous rappelle que toutes les méthodes ont accès à tous les attributs, donc ici c'est bien la hauteur de l'instance courante. De la même façon, on peut ici déclarer un accesseur pour l'attribut largeur. On a aussi des manipulateurs, appelés aussi « méthodes set » ou « setters », qui permettent de modifier cette fois-ci, et ce sont donc des actions, de modifier les attributs en leur affectant une valeur. Donc pour ça, pour affecter une valeur, il vaut bien qu'on reçoive une valeur de l'extérieur, c'est cette valeur que l'on va mettre dans l'instance, et donc on va recevoir ici un paramètre qui est la valeur à mettre

### notes

### résumé

2m 1s





## Notre programme (4/4)

```
#include <iostream>
using namespace std;

class Rectangle {
public:
    double surface() const
    { return hauteur * largeur; }

    double getHauteur() const
    { return hauteur; }
    double getLargeur() const
    { return largeur; }

    void setHauteur(double h)
    { hauteur = h; }
    void setLargeur(double l)
    { largeur = l; }
```

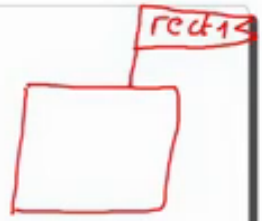
```
private:
    double hauteur;
    double largeur;
};

int main()
{
    Rectangle rect1;

    rect1.setHauteur(3.0);
    rect1.setLargeur(4.0);

    cout << "hauteur : "
         << rect1.getHauteur()
         << endl;

    return 0;
}
```



et on ne retournera cette fois-ci rien, ce n'est pas le but de retourner une valeur, mais de mettre une valeur; et la valeur qu'on l'on a reçue, donc par exemple, ici, le paramètre « h », on va la recopier dans « setHauteur », on la recopiera dans hauteur, et le paramètre « l » ici dans « setLargeur », on le recopiera dans l'attribut largeur. Regardons donc un exemple complet, toujours avec notre classe « Rectangle », dans la partie interface publique on aura donc déclaré la méthode surface, puis on aura ajouté un accesseur à la hauteur, qui est un prédicat const ici, un accesseur à la largeur, qui va retourner la largeur, et puis ici, deux modifieurs comme présentés précédemment, qui vont recevoir respectivement une valeur pour la hauteur et une valeur pour la largeur. Et puis dans la partie privée de notre classe « Rectangle », qui se termine ici, nous aurons donc les deux attributs hauteur et largeur. Comment on utilise ça ? Dans le « main », on va déclarer ici une instance « rect1 » de la classe « Rectangle », et on va utiliser la méthode « setHauteur » pour affecter la hauteur de rect1 à la valeur 3.0 ce qui aura pour effet de faire la chose suivante : on a une instance rect1 de la classe « Rectangle », et ici on appelle « setHauteur » qui va donc affecter une valeur à la hauteur de rect1. C'est donc bien la hauteur de rect1 qui va recevoir la valeur 3.0 Puis on va appeler la méthode « setLargeur » avec la valeur 4.0 donc ici on va placer 4.0 dans la largeur, mais la largeur de rect1. C'est donc bien la largeur de rect1 qui recevra la valeur 4.0, puis ensuite, par exemple, on pourrait vouloir afficher la hauteur, et on appelle ici la méthode « getHauteur »

### notes

### résumé

3m 13s



## Notre programme (4/4)

```
#include <iostream>
using namespace std;

class Rectangle {
public:
    double surface() const
    { return hauteur * largeur; }

    double getHauteur() const
    { return hauteur; }
    double getLargeur() const
    { return largeur; }

    void setHauteur(double h)
    { hauteur = h; }
    void setLargeur(double l)
    { largeur = l; }
```

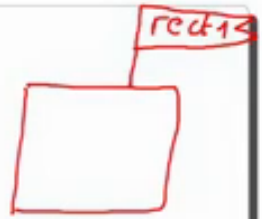
```
private:
    double hauteur;
    double largeur;
};

int main()
{
    Rectangle rect1;

    rect1.setHauteur(3.0);
    rect1.setLargeur(4.0);

    cout << "hauteur : "
         << rect1.getHauteur()
         << endl;

    return 0;
}
```



» ici donc, qui va retourner la hauteur, mais la hauteur de rect1, et donc cet appel ici qui va retourner la valeur 3.0. la valeur 3.0.

notes

résumé