

Support de cours

Cours:

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

Vidéo:

### W11-01-intropoo-CPP-pt3

Concepts (extraits des sous-titres générés automatiquement) :

**Programmeur utilisateur. Travers de l'interface. Programmeur concepteur. Petit exemple. Règles fondamentales. Intérêts fondamentaux. Interface d'utilisation. Existence d'un nouveau type. Détails d'implémentation. Surface d'un rectangle de largeur. Attributs de type double. Méthodes visibles. Bons programmes. Bonnes règles de programmation. En-tête des 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/>

# Introduction

## (Partie 3)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

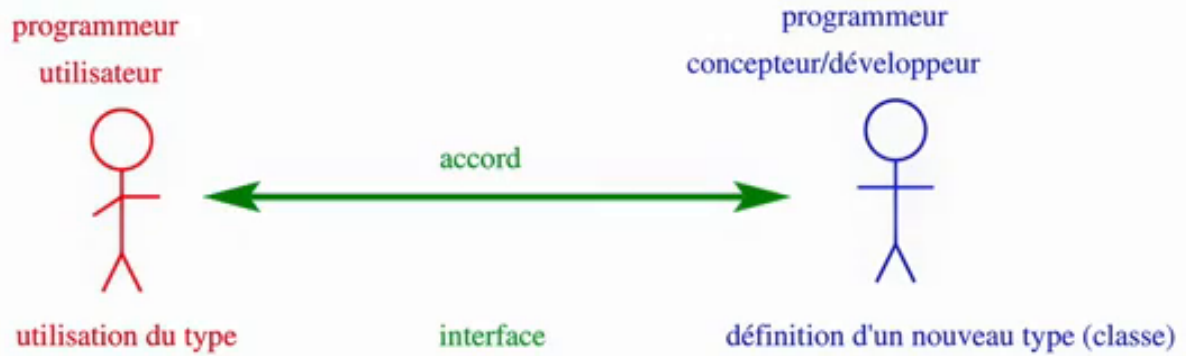
...

notes

résumé

0m 0s





Donc concrètement le programmeur concepteur

notes

résumé

0m 1s



programmeur  
utilisateur



utilisation du type

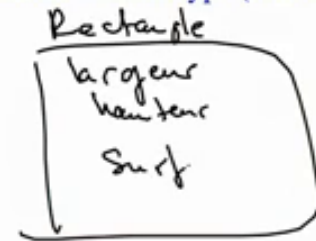
accord

programmeur  
concepteur/développeur



définition d'un nouveau type (classe)

interface



va donc décider de l'existence d'un nouveau type, et va devoir se préoccuper de tous les détails d'implémentation.

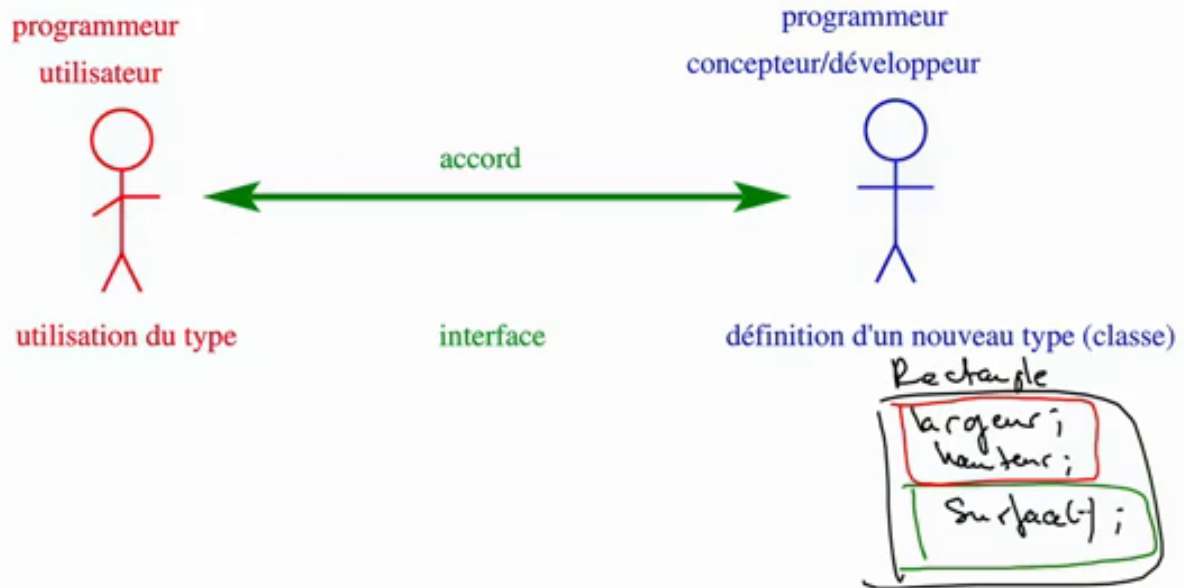
notes

résumé

0m 5s



## Les « 3 facettes » d'une classe



Il va devoir décider de ce qui est visible pour le monde extérieur, de ce qui est utilisable et de ce qui ne l'est pas. Le programmeur utilisateur, de son côté, est

notes

résumé

0m 19s



programmeur  
utilisateur



utilisation du type

Rectangle rectr... ;  
rect.

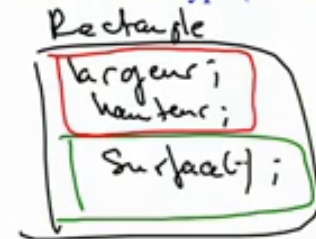
accord

interface

programmeur  
concepteur/développeur



définition d'un nouveau type (classe)



client du nouveau type de donnée, il va pouvoir l'utiliser. Mais va pouvoir l'utiliser uniquement, au travers de l'interface.

notes

résumé

0m 37s



programmeur  
utilisateur



utilisation du type

Rectangle rect... ;  
rect. su.

accord

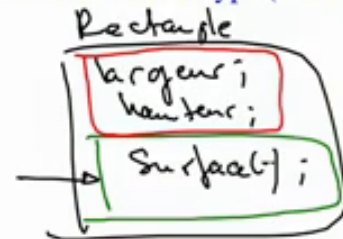


interface

programmeur  
concepteur/développeur



définition d'un nouveau type (classe)



C'est à dire des méthodes visibles, typiquement.

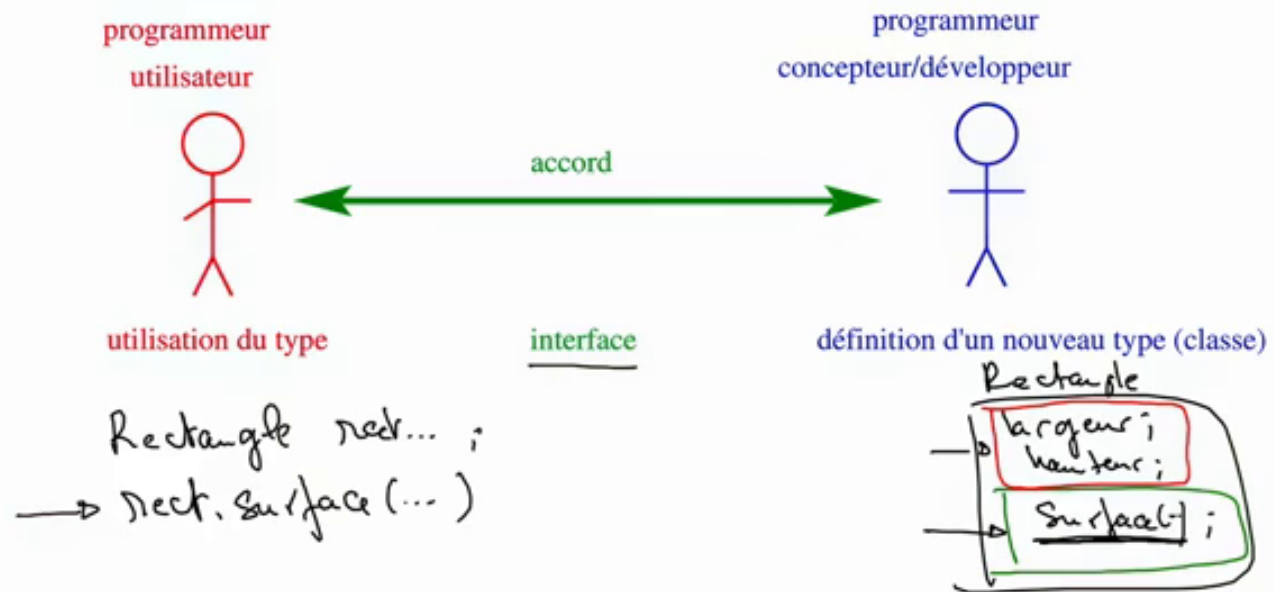
notes

résumé

0m 47s



## Les « 3 facettes » d'une classe



Et n'aura pas accès aux détails internes. L'interface d'utilisation est, typiquement, ce qui va permettre d'établir le lien entre le concepteur développeur et l'utilisateur. Et de façon très concrète, cette interface va pouvoir être entièrement décrite par l'en-tête des méthodes qui sont offertes au programmeur utilisateur.

notes

résumé

0m 50s





1. L'intérêt de regrouper les traitements et les données conceptuellement reliées est de permettre une *meilleure visibilité* et une meilleure cohérence au programme, d'offrir une plus grande modularité.

```
double largeur(3.0);  
double hauteur(4.0);  
  
cout << "Surface : "  
      << surface(largeur, hauteur)  
      << endl;
```

```
Rectangle rect(3.0, 4.0);  
cout << "Surface : "  
      << rect.surface()  
      << endl;
```

Rectangle  
double largeur  
double hauteur

Alors un des intérêts fondamentaux de l'orientée objet est d'assurer une meilleure visibilité, une meilleure cohérence au programme. Vous avez ici sous les yeux deux programmes qui font sensiblement la même chose, qui calculent la surface d'un rectangle de largeur 3 et de hauteur 4. Nous anticipons un tout petit peu sur la suite des séquences à venir. A savoir que l'on peut initialiser la largeur et la hauteur d'un rectangle selon cette syntaxe là. Mais c'est surtout pour vous donner un aperçu de ce que donnerait au final le programme en terme de lisibilité. Donc si l'on compare les deux approches, on se rend compte que ici l'on est en train de travailler avec des données de très bas niveau. Ici nous travaillons avec un rectangle, ce qui est beaucoup plus informatif, quant aux intentions de programmation. En lisant ce programme, on sait tout de suite que l'on est en train de travailler avec un rectangle. Ici le lien entre les données et le traitement ne se fait que par le biais du passage des arguments. Ce qui est indirect. Alors qu'ici le lien peut se faire de façon explicite. On appelle un calcul de surface sur un rectangle donné. De plus, de par le fait qu'en orientée objet, on ne peut manipuler l'objet, qu'au travers, des méthodes des fonctionnalités de l'interface d'utilisation. Cela permet de donner un cadre d'utilisation qui est beaucoup plus rigoureux. Par exemple ici, comme on le montrait tout à l'heure, rien ne m'oblige, dans cette approche, à avoir des valeurs cohérentes, pour la largeur et la hauteur. Donc la largeur et la hauteur qui appartiendraient à un même rectangle. Ici, cette erreur n'est plus possible, puisqu'on manipule un rectangle particulier et donc forcément les valeurs qui y sont liées seront cohérentes. De plus, de par le

notes

résumé

1m 13s



1. L'intérêt de regrouper les traitements et les données conceptuellement reliées est de permettre une *meilleure visibilité* et une meilleure cohérence au programme, d'offrir une plus grande modularité.

```
double largeur(3.0);  
double hauteur(4.0);  
  
cout << "Surface : "  
      << surface(largeur, hauteur)  
      << endl;
```

```
Rectangle rect(3.0, 4.0);  
cout << "Surface : "  
      << rect.surface()  
      << endl;
```

Rectangle  
double largeur  
double hauteur

même fait, que l'utilisateur est contraint de passer par l'interface d'utilisation pour manipuler l'objet, il n'est pas impacté par des modifications d'implémentation internes. Comme évoqué précédemment, si on décide plutôt que de modéliser un rectangle comme étant

notes

résumé

2. L'intérêt de séparer les niveaux *interne* et *externe* est de donner un **cadre** plus **rigoureux** à l'utilisation des objets utilisés dans un programme

Les objets ne peuvent être utilisés qu'au travers de leurs interfaces (niveau externe)  
et donc les éventuelles **modifications** de la structure interne restent **invisibles** à l'extérieur

(même idée que la séparation prototype/définition d'une fonction)

**Règle : les attributs d'un objet ne doivent pas être accessibles depuis l'extérieur, mais uniquement par des méthodes.**

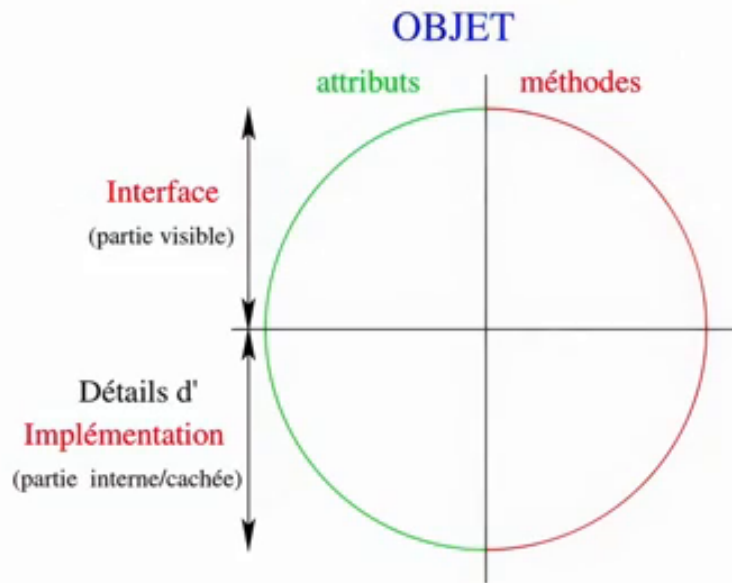
un objet doté de 2 attributs de type double hauteur et largeur. on passe plutôt à une représentation sous la forme d'un tableau à deux entrées, qui seraient les dimensions. Alors, si le programmeur concepteur de la classe Rectangle, a bien fait son travail, et donc, a adapté en conséquence le calcul de surface. Celui qui utilise le calcul de surface n'est pas impacté. Nous venons de voir sur ce petit exemple l'intérêt qui découle de séparer le niveau interne du niveau externe. On obtient donc un cadre d'utilisation plus rigoureux. Et les modifications de la structure interne restent invisibles à l'extérieur.

notes

résumé

3m 1s





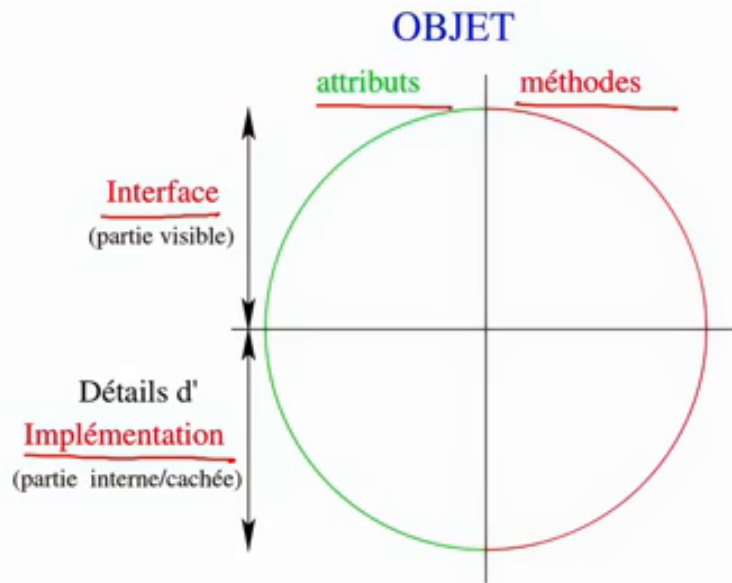
Donc une des règles fondamentales, qu'on a déjà vue dans l'introduction, les attributs, de par le fait qu'il faut faire des choix quant à leur modélisation, des choix techniques, des choix d'implémentation, sont considérés comme des détails d'implémentation de façon systématique dans tous bons programmes orientés objet.

notes

résumé

3m 37s





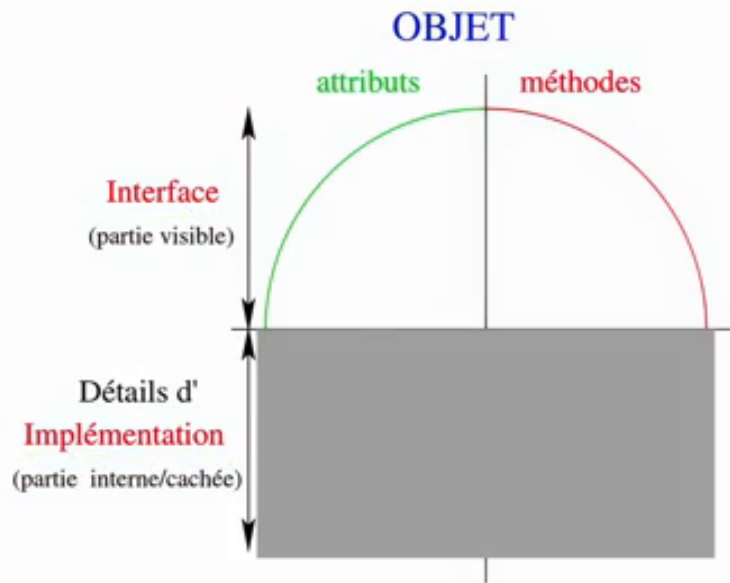
Pour résumer, lorsque l'on définit un nouveau type d'objet au travers d'une classe, on va être amené à définir des attributs caractéristiques de la classe ainsi que les méthodes qui lui sont spécifiques, et on va devoir se préoccuper de définir concrètement, ce qui est visible : l'interface d'utilisation, et ce qui ne l'est pas : les détails d'implémentation.

notes

résumé

3m 54s





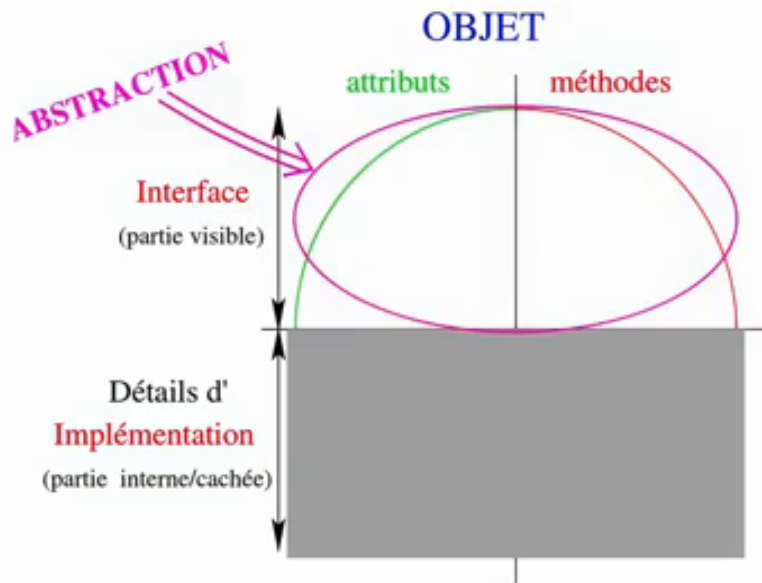
Donc une fois que l'on aura décidé de ce qui est à cacher,

notes

résumé

4m 13s





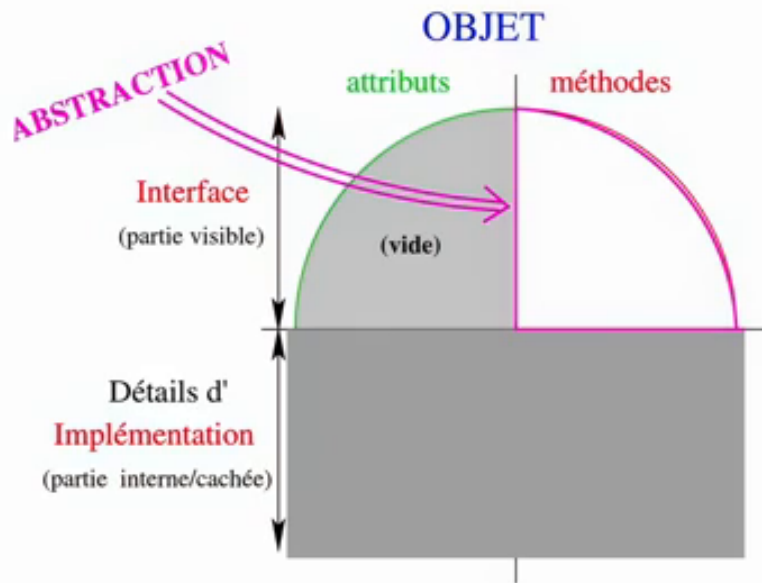
on va aboutir à une représentation de l'objet pour l'utilisateur externe qui se résume à l'interface d'utilisation. Donc l'utilisateur externe, n'aura de vision de cet objet,

notes

résumé

4m 19s





une vision abstraite qui est matérialisée par l'interface d'utilisation. Je ne vois d'un rectangle que son calcul de surface. Et si je veux suivre les bonnes règles de programmation orientée objet, je vais considérer que les attributs sont également des détails d'implémentation. Donc l'interface va se limiter à un nombre de méthodes bien choisies. L'utilisateur externe n'aura de vision de l'objet que cette interface d'utilisation. que cette interface d'utilisation.

notes

résumé

4m 29s

