

Support de cours

Cours:

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

Vidéo:

W11-01-intropoo-CPP-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Exemple du rectangle. Haut niveau. Nouveaux types de données. Entité des données. Représentation du rectangle. Processus d'abstraction. Niveau de l'utilisation. Programmeur utilisateur. Niveau externe. Notion d'encapsulation. Notion d'abstraction. Détail d'implémentation interne. Types de données. Interface d'utilisation. Fonctionnalité du calcul de surface.



[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 2)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



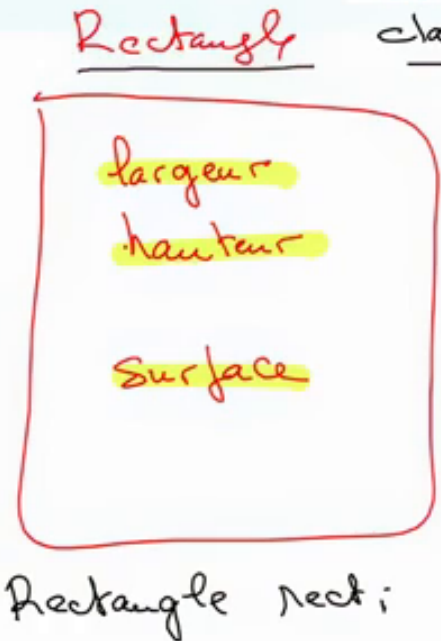
class

Principe d'encapsulation :

regrouper dans le même objet informatique («concept»), les données et les traitements qui lui sont *spécifiques* :

- ▶ **attributs** : les données incluses dans un objet
- ▶ **méthodes** : les fonctions (= traitements) définies dans un objet

Les objets sont définis par leurs attributs et leurs méthodes.



Commençons par la notion d'encapsulation. Ce que l'on entend par encapsulation, c'est le fait de pouvoir regrouper dans une seule et même entité des données et des traitements qui agissent sur ces données. Typiquement, dans l'exemple du rectangle, nous allons regrouper dans une seule et même entité, la largeur et la hauteur qui caractérisent la représentation du rectangle et la fonctionnalité du calcul de surface. Donc en terme de jargon, on va parler pour les données, de la notion d'attributs, et on va parler, pour les fonctionnalités, de la notion de méthode. Donc en programmation orientée objet, on va être dans la possibilité de définir des nouveaux types de données, au travers, nous allons le voir un peu plus tard, de la notion de classes. Ces types de données peuvent être utilisés pour travailler concrètement avec des données

notes

résumé

0m 1s



Notion d'abstraction

Pour être véritablement intéressant, un objet doit permettre un certain degré d'**abstraction**.

Le processus d'abstraction consiste à identifier pour un ensemble d'éléments :

- ▶ des caractéristiques communes à tous les éléments
 - ▶ des mécanismes communs à tous les éléments
- ☞ description **générique** de l'ensemble considéré :
se focaliser sur l'essentiel, cacher les détails.

de type plus abstrait, le rectangle. Ces données vont être des objets qui vont cohabiter dans le programme et interagir dans le programme. Un programme orienté objet va donc typiquement travailler avec des objets qui sont caractérisés par leurs attributs et leurs méthodes.

notes

résumé

1m 1s



Notion d'abstraction

Pour être véritablement intéressant, un objet doit permettre un certain degré d'**abstraction**.

Le processus d'abstraction consiste à identifier pour un ensemble d'éléments :

- ▶ des caractéristiques communes à tous les éléments
- ▶ des mécanismes communs à tous les éléments

description **générique** de l'ensemble considéré :
se focaliser sur l'essentiel, cacher les détails.

[double largeur₁ (2.5);
double hauteur₁ (4.0);

[double largeur₂ (4.5);
double hauteur₂ (5.0);

surface (largeur₂, hauteur₂);
surface (largeur₂, hauteur₂);

EPFL

Parlons maintenant de la notion d'abstraction. Supposons que je souhaite écrire un programme qui manipule plusieurs rectangles, pas uniquement un seul. Dans une approche procédurale, j'aurais typiquement l'occasion de déclarer autant de largeurs et de hauteurs que j'ai de rectangles. Ici je le fais pour le premier rectangle. Et je devrai évidemment faire exactement la même chose pour le second rectangle. Ce qui est relativement fastidieux. Si je souhaite maintenant calculer la surface de ces deux rectangles, je vais devoir appeler la fonction surface en lui fournissant, comme arguments, la largeur et la hauteur de chacun des rectangles de façon appropriée. Je fais un appel à la fonction surface en définissant à chaque fois les bons arguments. Donc il faut être précautionneux, ceci est source d'erreur. Imaginez par exemple, que je me trompe, et qu'au lieu de passer au premier appel de surface non pas hauteur₁ mais hauteur₂, je perd la cohérence de mes données. je ne suis plus en train de travailler avec les données d'un même rectangle. Le processus d'abstraction est le processus au terme duquel je réalise que je manipule des données d'un plus haut niveau, la notion de rectangle, à laquelle je peux associer une description générique. Ici, tous les rectangles sont caractérisés par une largeur et une hauteur auxquelles on peut attacher un même calcul de surface. Du coup, je décide de travailler avec la notion plus abstraite de rectangle, plutôt que de travailler avec une description intime de la représentation des rectangles, ce qui nous permet de nous focaliser sur l'essentiel et de cacher les détails.

notes

résumé

1m 22s



Notion d'abstraction

Pour être véritablement intéressant, un objet doit permettre un certain degré d'**abstraction**.

Le processus d'abstraction consiste à identifier pour un ensemble d'éléments :

- ▶ des caractéristiques communes à tous les éléments
- ▶ des mécanismes communs à tous les éléments

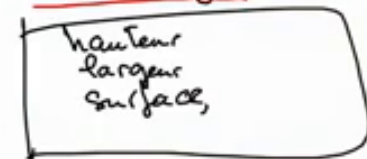
description **générique** de l'ensemble considéré :
se focaliser sur l'essentiel, cacher les détails.

$\left\{ \begin{array}{l} \text{double hauteur}_1 = 20; \\ \text{double largeur}_1 = 10; \end{array} \right\}$ rect₁

$\left\{ \begin{array}{l} \text{double hauteur}_2 = 4.5; \\ \text{double largeur}_2 = 2.5; \end{array} \right\}$ rect₂

$\left\{ \begin{array}{l} \text{Surface (hauteur}_1, \text{largeur}_2); \text{ rect}_1.\text{Surface}() \\ \text{Surface (hauteur}_2, \text{largeur}_2); \end{array} \right\}$

Rectangle



L'encapsulation me permet de regrouper en une seule et même entité, des données, comme une largeur et une hauteur, et des traitements, comme un calcul de surface. C'est l'outil qui me permet d'aboutir à une vision plus abstraite, des données que je manipule. Du coup je vais permettre à mon programme, de désormais, pouvoir travailler avec des entités plus abstraites, je vais travailler avec un premier rectangle et un deuxième rectangle, tous les deux de type Rectangle dans le programme. Je vais invoquer des calculs de surface sur ces rectangles. On va voir qu'en orientée objet, j'anticipe un petit peu sur la suite, je calcule ici la surface du premier rectangle. On va voir par la suite comment tout ceci s'exprime de façon beaucoup plus précise. Du coup, je permet à mon programme de se focaliser sur l'essentiel. Donc je ne suis plus en train de me préoccuper du fait qu'un rectangle est constitué d'une largeur et d'une hauteur,

notes

résumé

3m 13s



L'interface d'une voiture

- ▶ Volant, accélérateur, pédale de frein, etc.
- ▶ Tout ce qu'il faut savoir pour la conduire (mais pas la réparer ! ni comprendre comment ça marche)
- ▶ L'interface ne change pas, même si l'on change de moteur...
...et même si on change de voiture (dans une certaine mesure) :
abstraction de la notion de voiture (en tant qu'« objet à conduire »)

je me focalise sur les aspects essentiels. Je travaille avec des rectangles, je fais un calcul de surface sur un rectangle. Au niveau de l'utilisation, je ne vois désormais du rectangle que ce qu'on appelle en jargon orientée objet, son interface d'utilisation. C'est à dire, les fonctionnalités qui sont offertes pour la manipulation du rectangle, comme ici, le calcul de surface. Faisons une analogie avec un objet de la vie courante. Lorsque vous conduisez votre voiture, vous avez typiquement besoin d'en connaître que l'interface d'utilisation. Concrètement, eh bien, vous avez besoin d'avoir un volant, un accélérateur, une pédale de frein et à nul moment il ne nous est utile de savoir comment le moteur est construit. Pour conduire votre voiture, vous n'avez besoin d'en connaître que l'interface d'utilisation. C'est le même principe en programmation orientée objet. Pour utiliser un nouveau type d'objet, une nouvelle classe, vous n'avez besoin d'en connaître que l'interface d'utilisation, qui est prévue par le programmeur de la classe. Le reste est du détail d'implémentation interne qu'il n'est pas utile de connaître.

notes

résumé

4m 13s



Abstraction et Encapsulation

En plus du regroupement des données et des traitements relatifs à une entité, l'encapsulation permet en effet de définir **deux niveaux** de perception des objets :

Il y a donc deux niveaux de perception d'un objet :

notes

résumé

5m 13s



Abstraction et Encapsulation

En plus du regroupement des données et des traitements relatifs à une entité, l'encapsulation permet en effet de définir **deux niveaux** de perception des objets :

- ▶ **niveau externe** : partie « visible » (par les programmeurs-utilisateurs) :
 - ▶ **l'interface** : *entête* de quelques méthodes bien choisies
 - ▶ résultat du processus d'*abstraction*
- ▶ **niveau interne** : (détails d')**implémentation**
 - ▶ **corps** :
 - ▶ méthodes et attributs accessibles uniquement depuis l'intérieur de l'objet (ou d'objets similaires)
 - ▶ définition de toutes les méthodes de l'objet



un niveau externe qui est utile au programmeur utilisateur, celui qui utilise la notion de rectangle. dans un programme, il existe désormais un type Rectangle. Je peux déclarer des variables de type Rectangle, les initialiser de façon appropriée. Et ensuite, ce qui m'intéresse en tant que programmeur-utilisateur, c'est les fonctionnalités utiles, le calcul de surface. Donc ce niveau externe est le niveau qui intéresse le programmeur utilisateur celui qui utilise le type Rectangle. Second niveau : le niveau interne, celui qui a programmé le nouveau type, le type Rectangle, a du se préoccuper de tous les détails d'implémentation. A savoir, un rectangle, c'est une hauteur et une largeur. Il a du définir comment se fait concrètement le calcul de surface. Donc ceci constitue la partie implémentation. C'est un niveau interne qui n'est pas forcément utile à celui qui utilise le rectangle. Donc en programmation orientée objet,

notes

résumé

5m 15s



L'interface d'une voiture

- ▶ Volant, accélérateur, pédale de frein, etc.
- ▶ Tout ce qu'il faut savoir pour la conduire (mais pas la réparer ! ni comprendre comment ça marche)
- ▶ L'interface ne change pas, même si l'on change de moteur...
...et même si on change de voiture (dans une certaine mesure) :
abstraction de la notion de voiture (en tant qu'« objet à conduire »)

nous aurons, non seulement, la possibilité de regrouper en une seule et même entité des données et des traitements. On va pouvoir également définir des niveaux de visibilité. Celui qui programme le nouveau type Rectangle, concrètement une classe Rectangle, va pouvoir dire : « Ceci est un détail d'implémentation qui ne sera pas accessible à l'utilisateur externe. » Ceci est une fonctionnalité que l'on souhaite offrir à l'utilisateur externe et donc qui va être accessible à cet utilisateur. Tout ce qui est accessible à l'utilisateur, donc visible, constitue ce que l'on appelle, l'interface d'utilisation de la classe du type en question. Ici, dans notre classe Rectangle, dans notre nouveau type Rectangle nous avons défini comme interface d'utilisation le calcul de surface. Le reste constitue des détails d'implémentation, qui ne sont pas accessibles au programmeur qui utilise le type rectangle.

notes

résumé

6m 25s



Il y a donc deux facettes à l'encapsulation :

1. regroupement de tout ce qui caractérise l'objet : données (attributs) **et** traitements (méthodes)
2. isolement et dissimulation des détails d'implémentation

Interface = ce que le programmeur-utilisateur (hors de l'objet) peut utiliser

- ☞ Concentration sur les attributs et les méthodes concernant l'objet (*abstraction*)

Et nous avons ici une clef fondamentale nous permettant d'atteindre l'objectif d'une meilleure robustesse de nos programmes face aux changements. Typiquement, lorsque l'on change de voiture, même si la technologie du moteur a changé l'interface globalement, reste la même. Vous pouvez continuer, vous savez toujours conduire votre voiture, même si les détails internes de mise en œuvre de votre voiture ont changé entre temps. Donc vous ne voyez de votre voiture, quelque chose d'abstrait. Vous ne voyez de votre voiture, concrètement, que ce qui vous permet de la conduire, à savoir, le volant, l'accélérateur, la pédale de frein. Pour résumer, encapsuler, c'est regrouper en une seule et même entité les données et les traitements qui la caractérisent. C'est aussi dissimuler les détails d'implémentation. L'interface d'utilisation que l'on définit en utilisant justement cet outillage de l'encapsulation, est ce qui me permet d'aboutir à une abstraction, une vision abstraite des objets, que, en tant qu'utilisateur externe, je ne vois plus qu'en tant qu'objets manipulables au travers de leur interface d'utilisation. au travers de leur interface d'utilisation.

notes

résumé

7m 25s

