

Support de cours

Cours:

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

Vidéo:

### W14-03-1-heritageconstr-CPP-pt1

Concepts (extraits des sous-titres générés automatiquement) :

**Moyen d'un constructeur. Constructeurs de la classe. Constructeurs. Instanciation d'une sous-classe. Constructeur de la classe. Constructeur des sous-classes. Point de vue général. Appel explicite. Partie liste d'initialisation. Ensemble des attributs de la classe. Séquence vidéo. Constructeur de la sous-classe. Travers de l'appel de ce constructeur. Classe. En-tête du constructeur de la sous-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/>

# Héritage : constructeurs (1)

## (Partie 1)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



# Constructeurs et héritage



Lors de l'instanciation d'une sous-classe, il faut initialiser :

- ▶ les attributs *propres à la sous-classe*
- ▶ les attributs *hérités des super-classes*

Rectangle r (3.4, 4.5);  
Rectangle::Rectangle (...)

## MAIS...

...il ne doit pas être à la charge du concepteur des sous-classes de réaliser lui-même l'*initialisation des attributs hérités*

L'accès à ces attributs pourrait notamment être interdit ! (*private*)

L'initialisation des attributs hérités doit donc se faire au niveau des classes où ils sont explicitement définis.

**Solution** : l'initialisation des attributs hérités doit se faire en **invquant les constructeurs des super-classes**.

Dans cette séquence vidéo, nous allons nous intéresser aux conséquences de l'héritage sur les constructeurs c'est-à-dire l'initialisation et les destructeurs. Nous avons vu dans une séquence vidéo précédente sur les constructeurs que lors de l'instanciation d'une sous-classe, il nous fallait initialiser les attributs. C'est ce que nous faisons par exemple, quand on déclare une instance « r » de la classe « Rectangle » en passant par exemple la largeur et la hauteur. Ceci était fait au moyen d'un constructeur qui était en charge d'initialiser les attributs. Mais si la classe « Rectangle » hérite d'une classe « FigureGeometrique » alors la classe « Rectangle » reçoit l'ensemble des attributs de la classe « FigureGeometrique ».

## notes

## résumé

0m 1s



## Constructeurs et héritage : appel explicite

L'invocation du constructeur de la super-classe se fait au **début de la section d'appel aux constructeurs des attributs**.

Syntaxe :

```
SousClasse(liste de paramètres)
: SuperClasse(Arguments),
  attribut1(valeur1),
  ...
  attributN(valeurN)
{
    // corps du constructeur
}
```



Lorsque la super-classe admet un constructeur par défaut, l'invocation explicite de ce constructeur dans la sous-classe n'est pas obligatoire

☛ le compilateur se charge de réaliser l'invocation du constructeur par défaut

Donc, les constructeurs de la classe « Rectangle » ont pour charge d'initialiser les attributs de la classe « Rectangle », y compris les attributs hérités de la super-classe « FigureGeometrique ». Cependant, ce ne doit pas être le concepteur de la classe « Rectangle » qui doit lui-même initialiser les attributs de la classe « FigureGeometrique ». Il serait peut-être même bien incapable de le faire si la super-classe avait des attributs privés ; il ne pourrait pas y accéder. Et donc, comment le concepteur de la sous-classe « Rectangle » peut-il initialiser les attributs de la super-classe « FigureGeometrique » ? Pour cela, il doit faire appel dans le constructeur de « Rectangle » au constructeur de la classe « FigureGeometrique ». L'initialisation des attributs hérités doit se faire en appelant les constructeurs des super-classes dans le constructeur des sous-classes. Voyons comment tout ceci s'écrit en C++, d'abord d'un point de vue général puis ensuite en l'illustrant sur un exemple. L'appel au constructeur de la super-classe

notes

résumé

0m 49s



## Constructeurs et héritage : exemple 1

Si la classe parente n'admet pas de constructeur par défaut, l'**invocation explicite** d'un de ses constructeurs **est obligatoire** dans les constructeurs de la sous-classe

☛ La sous-classe doit admettre *au moins un constructeur explicite*.

Exemple :

```
class FigureGeometrique {
protected:    Position position;
public:
    FigureGeometrique(double x, double y) : position(x, y) {}
    // ...
};

class Rectangle : public FigureGeometrique {
protected:    double largeur; double hauteur;
public:
    Rectangle(double x, double y, double l, double h)
        : FigureGeometrique(x,y), largeur(l), hauteur(h) {}
    // ...
};
```

depuis le constructeur de la sous-classe, se fait dans la partie liste d'initialisation. Cette partie, qui commence par deux points qu'on appelle aussi parfois « section deux points » qui est juste entre l'en-tête de la sous-classe et sa définition. Concrètement, juste après l'en-tête du constructeur de la sous-classe, dans la section deux points, vous commencerez par faire un appel au constructeur de la super-classe que vous souhaitez appeler donc vous placerez les arguments dont vous avez besoin et bien sûr, le constructeur de la super-classe a le même nom que cette super-classe. Ensuite, avec une virgule, exactement comme on initialisait les différents attributs, vous aurez donc la suite de la liste d'initialisation qui initialisera les attributs. Voyons donc un exemple concret, avec ici la classe « Rectangle » qui hérite de la classe « FigureGeometrique » ; on a défini une classe « FigureGeometrique » qui, je vous rappelle, a un attribut « position » qu'on aura mis ici en « protected » et puis qui par exemple a un constructeur qui reçoit deux coordonnées « x » et « y » pour pouvoir initialiser la position. Donc ici, dans la liste d'initialisation du constructeur « FigureGeometrique », on a l'initialisation de son attribut « position ». Maintenant, regardons le constructeur de la sous-classe « Rectangle », le constructeur de « Rectangle » va prendre comme d'habitude deux paramètres « l » et « h » pour initialiser sa largeur et sa hauteur, comme on l'avait fait avant l'héritage, mais il prendra en plus deux paramètres « x » et « y », on a décidé ici de les mettre en premier mais ce n'est pas nécessaire, pour pouvoir initialiser la position de la « FigureGeometrique » au travers de l'appel de ce constructeur de « FigureGeometrique » qui prend deux paramètres. Donc, le constructeur de « Rectangle » aura ses quatre paramètres puis dans la section liste d'initialisation,

notes

résumé

1m 49s



## Constructeurs et héritage : exemple 1

Si la classe parente n'admet pas de constructeur par défaut, l'**invocation explicite** d'un de ses constructeurs **est obligatoire** dans les constructeurs de la sous-classe

☞ La sous-classe doit admettre *au moins un constructeur explicite*.

Exemple :

```
class FigureGeometrique {
protected:    Position position;
public:
    FigureGeometrique(double x, double y) : position(x, y) {}
    // ...
};

class Rectangle : public FigureGeometrique {
protected:    double largeur; double hauteur;
public:
    Rectangle(double x, double y, double l, double h)
        : FigureGeometrique(x,y), largeur(l), hauteur(h) {}
    // ...
};
```

nan

notes

résumé

## Constructeurs et héritage : appel explicite

L'invocation du constructeur de la super-classe se fait au **début de la section d'appel aux constructeurs des attributs**.

Syntaxe :

```
SousClasse(liste de paramètres)
: SuperClasse(Arguments),
  attribut1(valeur1),
  ...
  attributN(valeurN)
{
    // corps du constructeur
}
```

Lorsque la super-classe admet un constructeur par défaut, l'invocation explicite de ce constructeur dans la sous-classe n'est pas obligatoire

☞ le compilateur se charge de réaliser l'invocation du constructeur par défaut

un appel au constructeur « FigureGeometrique », ici on a choisi l'appel à deux paramètres, puis derrière, avec une virgule, la liste d'initialisation de ses propres attributs. Ici, nous avons fait un appel explicite à un constructeur de la classe « FigureGeometrique ».

notes

résumé

3m 37s





## Constructeurs et héritage : exemple 1

Si la classe parente n'admet pas de constructeur par défaut, l'**invocation explicite** d'un de ses constructeurs **est obligatoire** dans les constructeurs de la sous-classe

☛ La sous-classe doit admettre *au moins un constructeur explicite*.

Exemple :

```
class FigureGeometrique {
protected:    Position position;
public:
    FigureGeometrique(double x, double y) : position(x, y) {}
    // ...
};

class Rectangle : public FigureGeometrique {
protected:    double largeur; double hauteur;
public:
    Rectangle(double x, double y, double l, double h)
        : FigureGeometrique(x,y), largeur(l), hauteur(h) {}
    // ...
};
```

Bien entendu, si la classe a un constructeur par défaut, il n'est pas nécessaire de faire une invocation explicite de ce constructeur. Un constructeur par défaut, je vous le rappelle, c'est un constructeur qui n'a pas besoin d'arguments, c'est le constructeur qui est sans arguments, et dans ce cas-là, il n'est pas du tout nécessaire de rajouter explicitement l'appel au constructeur, c'est le compilateur qui va se charger d'invoquer automatiquement le constructeur par défaut. Mais on aura de toute façon un appel au constructeur, ce sera l'appel au constructeur par défaut. Par contre, si la classe n'a pas de constructeur par défaut, alors il faut évidemment invoquer explicitement un de ces constructeurs, sinon le compilateur ne saura pas quoi faire,

notes

résumé

3m 56s





## Constructeurs et héritage : exemple 2

Autre exemple (qui ne fait pas la même chose) :

```
class FigureGeometrique {
protected:   Position position;
public:
    /* Note : le constructeur par défaut par défaut de FigureGeometrique
     *        appelle le constructeur par défaut de Position.
     */
    // ...
};

class Rectangle : public FigureGeometrique {
protected:   double largeur; double hauteur;
public:
    Rectangle(double l, double h)
        : largeur(l), hauteur(h)
    {}
    // ...
};
```

*Rectangle r(*

il veut absolument appeler un constructeur de la super-classe, si cette super-classe n'a pas de constructeur par défaut, il faut absolument utiliser un constructeur explicite qui doit être présent, et donc faire un appel explicite dans la section deux points, dans la liste de l'initialisation du constructeur de la sous-classe. Bien sûr, la sous-classe elle-même doit avoir un constructeur explicite, il faut bien un endroit où écrire cet appel explicite au constructeur de la super-classe. Si maintenant notre « FigureGeometrique » avait un constructeur par défaut, il n'était donc pas nécessaire de faire cet appel explicite à un constructeur, et on aurait plus simplement pu ne pas l'écrire. Voici un autre exemple, qui est différent du précédent, où la classe « FigureGeometrique » a un constructeur par défaut, par défaut. Regardons ce qui se passe dans ce cas, ici nous avons le constructeur de la sous-classe « Rectangle » qui, dans sa liste d'initialisation, ne fait pas explicitement appel aux constructeurs de la super-classe dont elle hérite, « FigureGeometrique », ceci est possible parce que la classe « FigureGeometrique » a un constructeur par défaut, on suppose ici qu'on n'a pas écrit le constructeur et donc elle a le constructeur par défaut, fourni par défaut, par le compilateur. Donc, il y a bien ici, au début du constructeur de la sous-classe « Rectangle », un appel au constructeur par défaut de la super-classe « FigureGeometrique ». A noter que nous avons ici deux fois un appel à un constructeur par défaut, puisque ici la classe « FigureGeometrique », elle, a une position, comme attribut, et comme on n'a pas explicitement écrit le constructeur, c'est le constructeur par défaut, qui est fourni par défaut, par le compilateur, lequel va appeler les constructeurs par défaut de chacun des attributs. Donc ici, le constructeur par défaut, par défaut, appelle le constructeur par défaut de « Position ». Pour résumer, l'appel au constructeur

notes

résumé

4m 37s



## Constructeurs et héritage : exemple 2

Autre exemple (qui ne fait pas la même chose) :

```
class FigureGeometrique {
protected:   Position position;
public:
    /* Note : le constructeur par défaut par défaut de FigureGeometrique
     *        appelle le constructeur par défaut de Position.
     */
    // ...
};

class Rectangle : public FigureGeometrique {
protected:   double largeur; double hauteur;
public:
    Rectangle(double l, double h)
        : largeur(l), hauteur(h)
    {}
    // ...
};
```

*Rectangle r(*

de « Rectangle », par exemple, si je faisais une déclaration comme ceci, « Rectangle r(3.0, 4.0); » appellera le constructeur de « Rectangle » ici en passant 3 et 4 comme paramètres, et commencera par appeler le constructeur par défaut, fourni par défaut, par le compilateur de la classe « FigureGeometrique », lequel appellera le constructeur par défaut de « Position », je suppose ici que « Position » est une classe, et qu'il a donc soit un constructeur par défaut, par défaut, soit un constructeur par défaut explicitement écrit. On aura donc en premier une construction par défaut de position qui terminera la construction par défaut de « FigureGeometrique » et qui permettra ensuite au constructeur de « Rectangle » de continuer à initialiser ses autres attributs. autres attributs.

notes

résumé