

Support de cours

Cours:

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

Vidéo:

### W14-01-heritageintro-CPP-pt2

Concepts (extraits des sous-titres générés automatiquement) :

**Super-classe. Attributs de ces classes. Relations d'héritage. Classe personnage. Syntaxe générale. Exemple des guerriers. Point de vue conceptuel. Exemple concret. Ensemble des méthodes. Niveau de la sous-classe. Définition usuelle de la classe. Enrichissement progressif de différentes classes. Héritage d'attributs. Introduction générale. Figure géométrique.**



[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 : concepts

## (Partie 2)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s





Nous avons vu qu'une sous-classe qui hérite d'une super-classe

notes

---

---

---

---

---

---

---

---

---

---

résumé

0m 1s



---

---

---

---

---

---

---

---

---

---

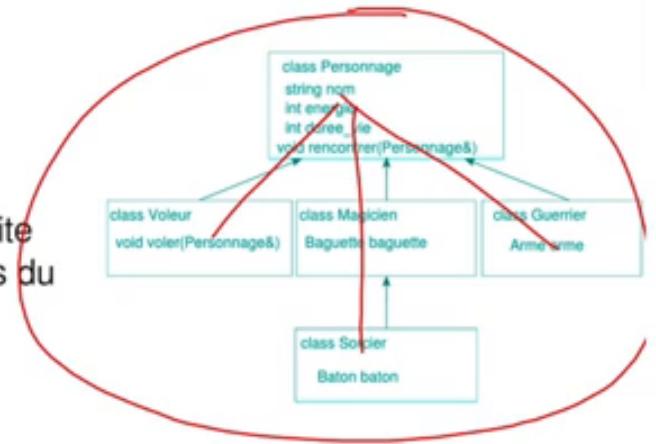
## Transitivité de l'héritage

Par transitivité, les instances d'une sous-classe possèdent :

- ▶ les attributs et méthodes (hors constructeurs/destructeur) de l'ensemble des classes parentes (super-classe, super-super-classe, etc.)

**Enrichissement par héritage :**

- ▶ crée un *réseau de dépendances* entre classes,
- ▶ ce réseau est organisé en une *structure arborescente* où chacun des nœuds hérite des propriétés de l'ensemble des nœuds du chemin remontant jusqu'à la racine.
- ▶ ce réseau de dépendances définit une **hiérarchie de classes**



héritait, c'est-à-dire recevait, possédait les attributs et les méthodes de cette super-classe sauf constructeur et destructeur. Cet aspect là est transitif, c'est-à-dire que si nous avons une super super-classe dont hérite la super-classe, alors, nous récupérerons au niveau de la sous-classe l'ensemble des méthodes et des attributs de ces classes. Plus concrètement, si j'ai une super super-classe « A » dont hérite une super-classe « B » et dont hérite une classe « C », nous allons récupérer au niveau de « C » les attributs et les méthodes de « B », mais également celles de « A » puisque « B » hérite de « A ». « B » récupère les attributs et les méthodes de « A » et donc dans « B » j'ai bien les attributs et les méthodes de « A » et donc par transitivité, je récupère au travers de « B » dans « C » également les attributs et les méthodes de « A » ; donc sur un exemple concret : si j'ai par exemple une classe Personnage dont hérite une classe « Magicien », et disons qu'un sorcier est une sorte de magicien donc la classe « Sorcier » hérite de la classe « Magicien ». A ce moment là dans la classe « Sorcier » nous aurons également un nom, une énergie, une durée de vie lesquels ont été donc hérités ainsi que la méthode « Rencontrer » bien sûr, lesquels ont été hérités dans le magicien et sont donc à nouveau hérités au niveau du sorcier et bien sûr le sorcier récupérera en plus un attribut « baguette » hérité du magicien. Nous avons donc comme ceci par héritage un enrichissement progressif de différentes classes ce qui fait que au final nous allons trouver, si on dessine toutes les relations d'héritage entre classes, un schéma arborescent, une structure comme

notes

résumé

0m 5s



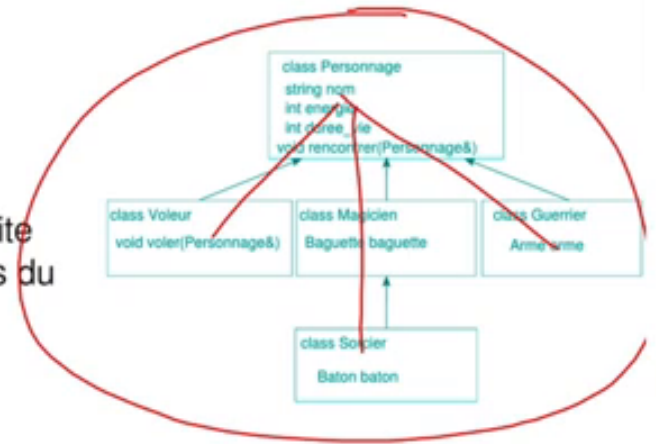
## Transitivité de l'héritage

Par transitivité, les instances d'une sous-classe possèdent :

- ▶ les attributs et méthodes (hors constructeurs/destructeur) de l'ensemble des classes parentes (super-classe, super-super-classe, etc.)

### Enrichissement par héritage :

- ▶ crée un *réseau de dépendances* entre classes,
- ▶ ce réseau est organisé en une *structure arborescente* où chacun des nœuds hérite des propriétés de l'ensemble des nœuds du chemin remontant jusqu'à la racine.
- ▶ ce réseau de dépendances définit une **hiérarchie de classes**



ça d'arbres où on va voir donc toutes les dépendances entre les classes, toutes les relations « est-un » et les relations donc d'héritage d'attributs, de méthodes et aussi de type

### notes

### résumé

## Sous-classe, Super-classes

Une **super-classe** :

- ▶ est une classe « parente »
- ▶ déclare les variables/méthodes communes
- ▶ peut avoir plusieurs sous-classes

Une **sous-classe** est :

- ▶ une classe « enfant »
- ▶ étend **une (ou plusieurs)** super-classe(s)
- ▶ hérite des **attributs**, des **méthodes** et du **type** de la super-classe

Une variable/méthode héritée peut s'utiliser comme si elle était déclarée dans la sous-classe au lieu de la super-classe (en fonction des droits d'accès, voir plus loin)

☛ On évite ainsi la **duplication de code** **EST-UN**

qui va nous donner ce que l'on appelle donc une hiérarchie de classe avec les classes les plus générales en haut, « Personnage » et les classes les plus spécialisées, les plus enrichies, en bas. Pour résumer, la relation d'héritage permet d'éviter de la duplication de code en nous permettant de représenter dans notre conception des programmes la relation « est-un » on va concevoir des super-classes qui seront plus générales, qu'on va appeler aussi les classes parente qui vont regrouper les aspects plus généraux et dont hériteront ce qu'on appelle des sous-classes aussi des classes enfant qui récupéreront l'ensemble les attributs, des méthodes et aussi le type de l'ensemble des super-classes dont elles dépendent. Dans un premier temps, on va faire hériter les sous-classes que d'une super-classe puis, on verra dans quelques semaines que si l'on veut on peut faire hériter

notes

résumé

2m 1s





des sous-classes de plusieurs super-classes ce que l'on appelle l'héritage multiple.

notes

---

---

---

---

---

---

---

---

---

---

résumé

2m 49s



---

---

---

---

---

---

---

---

---

---

## Passons à la pratique...

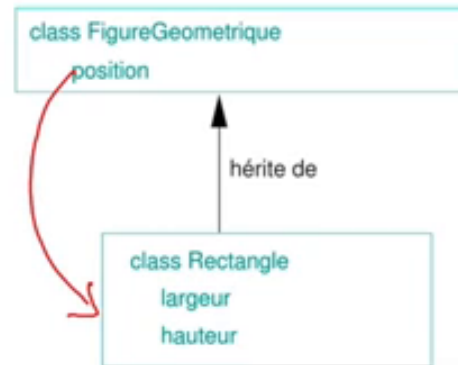
Définition d'une sous-classe en C++ :

Syntaxe :

```
class NomSousClasse : public NomSuperClasse
{
    /* Déclaration des attributs et méthodes
       spécifiques à la sous-classe */
};
```

Exemple :

```
class Rectangle public FigureGeometrique
{
    //...
private:
    double largeur; double hauteur;
};
```



Voilà tout ceci vous a donc présenté d'un point de vue conceptuel ce qu'est l'héritage, d'un point de vue peut être un peu théorique, mais comment en pratique faisons-nous pour faire hériter une sous-classe d'une super-classe ? Commençons par regarder un exemple concret : supposons que l'on veuille définir une classe « Rectangle » qui a une largeur et qui a une hauteur et qui est une figure géométrique lesquelles figures géométriques ont une position. On aurait donc le diagramme d'héritage suivant : une super-classe « FigureGeometrique » avec un attribut « position » et une sous-classe « Rectangle » qui hérite de « FigureGeometrique », un rectangle est une figure géométrique, et qui aurait donc des attributs supplémentaires, elle hériterait bien sûr l'attribut « position » mais elle aurait des attributs supplémentaires que sont une largeur et une hauteur. Comment écrivons-nous ceci donc en C++ ? La syntaxe générale pour déclarer une sous-classe, pour faire hériter une classe d'une super-classe, est simplement d'ajouter « : public », le nom de la super-classe, derrière la déclaration de la classe, derrière l'entête de la classe, derrière la première ligne « class », nom de la sous-classe ; donc par exemple pour notre « Rectangle » qui hérite de « FigureGeometrique », nous avons la définition usuelle de la classe « Rectangle »

notes

résumé

2m 55s





```
class Personnage {
    // ...
};
// ...
class Guerrier : public Personnage {
public:
    // constructeurs, etc.
private:
    Arme arme;
};
```

mais on rajoute derrière « class Rectangle » « : public FigureGeometrique ». Cela fait que la classe « Rectangle » hérite de la classe « FigureGeometrique » un rectangle est une figure géométrique et récupère l'ensemble de ses attributs et de ses méthodes, mis à part le constructeur et destructeur. La seule chose à ajouter donc à la définition usuelle de notre classe « Rectangle », pour qu'elle hérite de la classe « FigureGeometrique », c'est simplement cette portion là de code. De la même façon, si je reviens à notre exemple des guerriers où un guerrier est un personnage on a donc une classe « Personnage » qui serait définie, qu'obtiendrait tous les attributs dont nous avons déjà parlé

### notes

### résumé

4m 13s





notes

## résumé

4m 53s

