

Support de cours

Cours:

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

Vidéo:

### W15-05-polymcoll1-CPP-pt2

Concepts (extraits des sous-titres générés automatiquement) :

**Boucle for. For auto référence. Pointeurs uniques. Variable quidam. Façon suivante. Nouveau voleur. Tableau dynamique. For auto. Unique pointers. Référence constante. Innocente classe jeu. Petite subtilité. Première étape. Case mémoire. Fait possible.**



[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/>

# Collections hétérogènes

## (Partie 2)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

résumé

0m 0s



## Exemple complet : classes

Comment l'utiliser ?

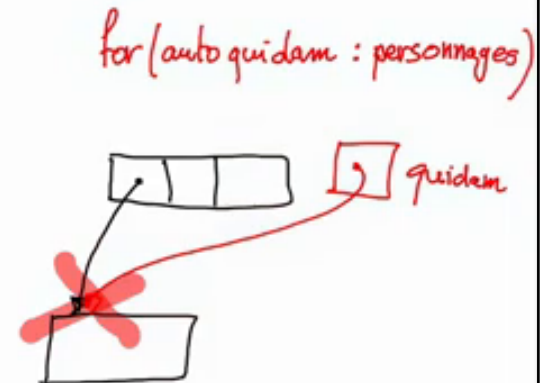
Le plus simple, comme dans la séquence vidéo précédente :

```
Jeu jeu;
jeu.ajouter_personnage(new Guerrier(...));
```

On aurait donc :

```
class Jeu {
public:
    void afficher() const;
    void ajouter_personnage(Personnage*);

private:
    vector<unique_ptr<Personnage>> personnages;
};
```



En effet, notre tableau dynamique « personnages » est un tableau dynamique de « unique\_ptr », c'est-à-dire des pointeurs uniques. On ne peut pas avoir plusieurs pointeurs sur la case mémoire. En mémoire, on a quelque chose comme ceci : un tableau dynamique de « unique\_ptr » qui pointent sur des « Personnages ». Quand on écrit la « boucle for » comme ceci, il se trouve que la variable quidam va être une autre variable qui va, tour à tour, valoir chacune des valeurs du tableau « personnage ». Donc « quidam » est bien un pointeur qui veut pointer au même endroit, que chacun, tour à tour, des différents pointeurs sur « Personnages », ici stockés dans ce tableau. Donc par exemple, la première étape, ce serait de dire quidam égal ce pointeur. Donc égalité de pointeurs, ça veut dire qui veut pointer au même endroit. Or, ceci n'est pas possible du fait que l'on a ici des unique\_ptr.

notes

résumé

0m 1s



```
void Jeu::afficher() const {
    for (auto const& quidam : personnages) {
        quidam->afficher();
    }
}
```

notes

1m 13s



```
Jeu jeu;  
jeu.ajouter_personnage(new Guerrier(...));  
jeu.ajouter_personnage(new Magicien(...));  
jeu.ajouter_personnage(new Voleur(...));  
jeu.ajouter_personnage(new Guerrier(...));  
// ...  
jeu.afficher();
```

Donc une référence constante sur des unique\_ptr. Voilà donc une petite subtilité sur les unique\_ptr. Voyons maintenant comment nous allons utiliser notre classe « Jeu ». Donc on déclarerait ici un jeu et puis on ajouterait des personnages en appelant la méthode ajouter\_personnage et en créant dynamiquement avec new un nouveau « Guerrier », ici avec des paramètres pour le constructeur de « Guerrier ».

### notes

---

---

---

---

---

---

---

---

---

---

### résumé

1m 49s



---

---

---

---

---



On ajoute ensuite un nouveau « Magicien », un nouveau voleur, peut-être encore un nouveau « Guerrier », etc. Quand on souhaite afficher le Jeu, on appellerait tout simplement `jeu.afficher`. Mais cette innocente classe `jeu` comporte cependant quand même un danger potentiel. cependant quand même un danger potentiel.

notes

résumé

2m 13s

