

Support de cours

Cours:

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

Vidéo:

W14-02-2-masquage-CPP-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Masquage de méthode. Objet de type. Méthode générale. Guerrier de quelques bonnes manières. Hiérarchie de classes. Termes de jargon. Séquence suivante. Défaut toutes. Méthode spécialisée. Méthode. Règles de résolution de portée. Super-classe. Syntaxe particulière. Éventuelles sous-classes. Exemple.



[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 : masquage

(Partie 2)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

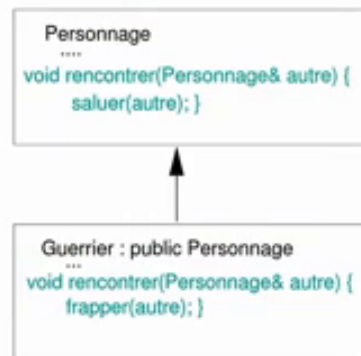
...

notes

résumé

0m 0s





La méthode `rencontrer` de `Guerrier` **masque** celle de `Personnage`

- ▶ Un objet de type `Guerrier` n'utilisera donc **jamais** la méthode `rencontrer` de la classe `Personnage`
- ▶ Vocabulaire OO :
 - ▶ Méthode héritée = méthode générale, *méthode par défaut*
 - ▶ Méthode qui masque la méthode héritée = *méthode spécialisée*

Pour résumer, le masquage de méthode dans une hiérarchie de classes, consiste à définir dans une sous-classe une méthode portant le même nom qu'une méthode déjà présente dans la super-classe. En termes de jargon, on dira que la méthode héritée est la méthode générale, celle dont pourront bénéficier par défaut toutes les éventuelles sous-classes qui ne la masquent pas. La méthode masquant celle héritée est la méthode spécialisée qui va donc répondre spécifiquement aux besoins de la sous-classe dans laquelle elle est présente. Comme évoqué précédemment, sur un objet de type « Guerrier », c'est toujours la méthode spécialisée qui va primer ici. Donc, si l'on définit un objet de type « Guerrier », et qu'on invoque sur cet objet la méthode « rencontrer », ça n'est jamais la méthode « rencontrer » de « Personnage » qui est invoquée, mais c'est la méthode spécialisée. On ne fait ici qu'appliquer les règles de résolution de portée usuelles, on résout toujours à la portée la plus proche. Un objet de type « Guerrier » dispose en fait de deux méthodes « rencontrer » : sa méthode spécialisée, et la méthode dont il a hérité de plus haut.

notes

résumé

0m 1s



- ▶ Il est parfois souhaitable d'accéder à une méthode/un attribut masqué(e)
- ▶ Exemple :
 - ▶ Le `Guerrier` commence par rencontrer le personnage comme le fait n'importe quel personnage (il le salue) avant de le frapper !
- ▶ Code désiré :
 1. Personnage non-`Guerrier` :
 - ▶ Méthode générale (`rencontrer` de `Personnage`)
 2. Personnage `Guerrier` :
 - ▶ Méthode spécialisée (`rencontrer` de `Guerrier`)
 - ▶ Appel à la méthode générale depuis la méthode spécialisée

Il est parfois utile, dans certaines situations, de quand même invoquer la méthode « rencontrer » héritée, même si l'on a spécialisé cette méthode plus bas dans la hiérarchie. Comment procéder dans ce cas ? Par exemple, imaginons que l'on veuille doter notre guerrier de quelques bonnes manières, et que l'on souhaite qu'il salue le personnage qu'il rencontre avant de le frapper, ou l'inverse. Cela signifie que le guerrier commence par rencontrer le personnage comme le ferait n'importe quel autre personnage de la hiérarchie, c'est à dire qu'il le salue, avant de lui appliquer des traitements un peu plus spécifiques. Le code qui permet à un personnage d'en saluer un autre

notes

résumé

1m 13s



Pour accéder aux attributs/méthodes masqué(e)s de la

- ▶ on utilise **l'opérateur de résolution de portée**
- ▶ Syntaxe : *NomClasse::méthode* ou *attribut*
- ▶ Exemple :

```
class Guerrier : public Personnage {  
    //...  
    void rencontrer (Personnage& perso) {  
        Personnage::rencontrer(perso); // salutation d'usage !!  
        frapper(perso);  
    }  
};
```

est déjà présent dans la méthode « rencontrer » générale, c'est-à-dire la méthode « rencontrer » de Personnage. Il est inutile de dupliquer ce code dans la méthode spécifique « rencontrer » de « Guerrier ». Ce qu'il nous faudrait faire donc, est que dans la méthode spécialisée, il soit possible d'appeler la méthode générale déjà définie plus haut, puis d'appliquer des actions spécifiques. Pour accéder au membre masqué d'un objet donné, on utilise une syntaxe particulière qui fait appel à l'opérateur de résolution de portée (::). Par exemple, si dans la méthode « rencontrer » définie de façon spécialisée dans la sous-classe « Guerrier », je veux invoquer la méthode « rencontrer » définie de façon plus générale dans la classe « Personnage », eh bien, dans la méthode « rencontrer » ici je vais dire : « J'appelle la méthode « rencontrer » de la classe « Personnage » » et je fais le lien entre le nom de la méthode et le nom de la classe à laquelle cette méthode appartient avec l'opérateur de résolution de portée. Donc ici, la méthode spécialisée « rencontrer » appelle la méthode « rencontrer », la méthode plus générale, en utilisant cette notation particulière. Ensuite, elle enchaîne un certain nombre d'actions un peu plus spécifiques.

notes

résumé

1m 49s





Ceci est très utile, car cela permet d'éviter de dupliquer ici toutes les actions qui sont entreprises dans la méthode plus générale. Nous avons donc ici, dans cet exemple, grâce à l'accès à une méthode « masquer », permis à notre guerrier de devenir un petit peu plus poli en saluant son adversaire avant de lui porter quelques coups, et sans dupliquer aucune ligne de code. C'est donc terminé pour cette séquence. Dans la séquence suivante, nous nous intéresserons à la construction et à la destruction dans le cadre de l'héritage. le cadre de l'héritage.

notes

résumé

3m 1s

