

Support de cours

Cours:

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

Vidéo:

W14-02-1-protected-CPP-pt2

Concepts (extraits des sous-titres générés automatiquement) :

Méthode de la classe. Attribut d'un objet de type. Privé b. Droit d'accès. Exemple concret. Visibilité des membres d'une classe. Classes de sa descendance. Niveau d'énergie. Objet de type. Super-classes. Extension du niveau privé. Dehors du corps des sous-classes. Super-classe. Portée de ces sous-classes. Accès direct.



[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 : droit protégé

(Partie 2)

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

Jean-Cédric Chappelier, Jamila Sam et Vincent Lepetit

...

notes

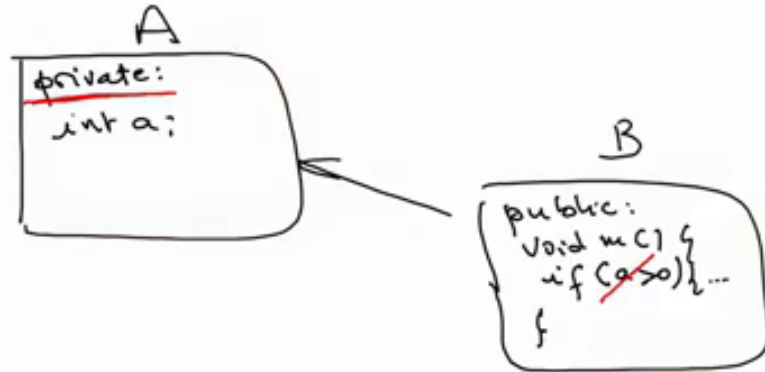
résumé

0m 0s



Jusqu'à maintenant, l'accès aux membres (attributs et méthodes) d'une classe pouvait être :

- ▶ soit **public** : visibilité totale à l'intérieur et à l'extérieur de la classe (mot-clé `public`)
- ▶ soit **privé** : visibilité uniquement à l'intérieur de la classe (mot-clé `private`)



mais il ne peut pas y accéder directement. Comme il peut sembler naturel

notes

résumé

0m 1s



Le niveau d'accès protégé correspond à une **extension du niveau privé** permettant l'accès aux sous-classes.

Exemple :

```
class Personnage {  
    // ...  
protected:  
    int energie;  
};  
  
class Guerrier : public Personnage {  
public:  
    // ...  
    void frapper(Personnage& le_pauvre) {  
        if (energie > 0) {  
            // frapper le perso  
        }  
    }  
};
```

qu'une méthode de la classe « B » puisse manipuler n'importe quel attribut d'un objet de type « B » et bien, il existe une troisième voie : le droit d'accès protégé. Le droit d'accès protégé permet à une sous-classe d'accéder à un membre hérité directement, il assure la visibilité des membres d'une classe dans toutes les classes de sa descendance et il se désigne par le mot clé « protected ». Concrètement, remplacer ici « private » par « protected » rend ceci possible, une sous-classe peut accéder à tous les membres protégés hérités des super-classes que ce soit des méthodes ou des attributs. Autre exemple concret, une sous-classe « Guerrier » hérite d'une super-classe « Personnage » et dispose ici de ce fait, d'un attribut caractérisant son niveau d'énergie. Si dans la classe « Guerrier », on souhaite maintenant écrire une méthode « frapper » permettant au guerrier de frapper un autre personnage et si l'on imagine que le guerrier ne peut frapper l'autre personnage que s'il dispose de suffisamment d'énergie alors, on est amené à utiliser l'attribut « energie », par exemple, comme ceci. L'accès direct dans une méthode de la classe « Guerrier » à un attribut hérité de plus haut, n'est ici possible que parce que cet attribut a été déclaré comme protégé. S'il avait été déclaré comme privé, cet accès direct n'aurait pas été possible. Que faudrait-il faire pour que le guerrier accède à son niveau d'énergie ? Ce qu'il est important de comprendre est que le niveau d'accès protégé n'est autre qu'une extension du niveau privé qui accorde des droits d'accès privilégiés aux membres protégés à toutes les sous-classes, mais nulle part ailleurs. En dehors du corps des sous-classes, le membre protégé est donc vu comme un membre privé.

notes

résumé

0m 5s



Le niveau d'accès protégé correspond à une extension du niveau privé permettant l'accès aux sous-classes... **mais uniquement dans leur portée** (de sous-classe), et non pas dans la portée de la super-classe

```
class B {
    private:
        int b;
        m(B autre) {
            this b
            autre.b
        }
};
```

Par exemple, si dans un programme principal... je déclare un objet de type « Guerrier », et que je tente d'accéder à son niveau d'énergie par un accès de ce genre et bien ceci n'est pas licite, en dehors des méthodes des sous-classes, l'attribut est perçu comme privé. Plus précisément, un membre protégé va être accessible directement dans les sous-classes, mais attention, uniquement dans leur portée, dans la portée de ces sous-classes. Commençons par rappeler brièvement ce que signifie la portée d'une classe sur un exemple simple, indépendamment de l'héritage. Supposons que l'on ait une classe « B » dans laquelle nous aurions défini un attribut, par exemple, « privé b ». Nous savons que l'attribut b est accessible dans toute la portée de « B », c'est-à-dire dans toute méthode de la classe « B ». Plus précisément, C++ met en oeuvre ce que l'on appelle la portée de classe ce qui signifie concrètement que cet attribut « b » va être accessible dans la portée de « B » via tout objet de type « B ». Que ce soit l'objet pointé par « this » ou tout autre instance de « B ». Ce qui veut dire concrètement ici que dans une méthode de la classe « B », on a le droit d'accéder à l'attribut « b », lorsqu'on écrit quelque chose comme ceci, implicitement on est en train de travailler avec l'attribut « b » de l'instance courante. C'est tout à fait équivalent à écrire quelque chose comme ceci. On a aussi le droit d'accéder à l'attribut « b », via tout autre instance de la classe « B ». C'est-à-dire ici, par exemple, si la méthode « m » prenait en paramètre un autre objet de type « B », on aurait aussi le droit d'accéder à l'attribut « b » via cet autre objet

notes

résumé

2m 1s



Le niveau d'accès protégé correspond à une extension du niveau privé permettant l'accès aux sous-classes... **mais uniquement dans leur portée** (de sous-classe), et non pas dans la portée de la super-classe

```
class B {  
    private:  
        int bi  
        m(B autre) {  
            this → b  
            autre.b  
        }  
};
```

this
autre B

de type « B ».

notes

résumé

Accès protégé : portée (2)

```
class A {
    // ...
protected: int a;
private:   int prive;
};

class B: public A {
public:
    // ...
    void f(B autreB, A autreA, int x) {
        a      = x; // OK A::a est protected => accès possible
        prive = x; // Erreur : A::prive est private

        a += autreB.prive; // Erreur (même raison)
        a += autreB.a      ; // OK : dans la même portée (B::)

        a += autreA.a      ; // INTERDIT ! : this n'est pas de la même
                               // portée que autreA
    }
};
```

→ ... Personnage {
protected :
 int energie;
 ...
};

En clair, dans la classe « B », on a le droit d'accéder à cet attribut via tout objet de type « B », que ce soit « this » ou tout autre « B ». Lorsque je dis ici que l'attribut « b » est de portée « B », je signifie qu'il est accessible dans la classe « B » uniquement via tout objet de type « B ». Voyons maintenant sur un exemple un peu plus complexe ce que signifie le fait que le droit d'accès, protégé cette fois, permet l'accès aux sous-classes « uniquement dans leur propre portée ». Ici, nous avons donc une sous-classe « b » qui hérite d'une super-classe « A » laquelle possède un attribut protégé « a » et un attribut privé. L'attribut protégé « a » est accessible dans la sous-classe « B » mais uniquement dans sa portée. C'est-à-dire à l'intérieur de la classe « B » via tout objet de type « B » que ce soit « this » ou un autre « b ». L'attribut privé n'est accessible que dans la portée de A, il ne peut être accéder à l'intérieur de « B ». Illustrons ceci au travers de cette méthode « f » de la classe « B » qui prend en paramètre un objet de type « B », un objet de type A et enfin, un entier. Lorsqu'on écrit ceci on accède à l'attribut protégé A dans la portée de « B » puisque nous sommes à l'intérieur de « B » et que nous accédons au A de l'instance courante. Nous sommes bel et bien dans la portée de « B » et ceci est donc possible. Ici, étant donné qu'il n'est pas possible dans la sous-classe d'accéder un membre privé de la super-classe, cette ligne donnerait lieu à une erreur. L'accès ici a un

notes

résumé

4m 1s



Accès protégé : portée (2)

```
class A {
    // ...
protected: int a;
private:   int prive;
};

class B: public A {
public:
    // ...
    void f(B autreB, A autreA, int x) {
        a      = x; // OK A::a est protected => accès possible
        prive = x; // Erreur : A::prive est private

        a += autreB.prive; // Erreur (même raison)
        a += autreB.a      ; // OK : dans la même portée (B::)

        a += autreA.a      ; // INTERDIT ! : this n'est pas de la même
                               // portée que autreA
    }
};
```

→ ... Personnage {
protected :
 int energie;
 ...
};

membre privé de la super-classe via tout objet de type « B » n'est pas possible dans « B », que ce soit via « this », comme c'était le cas ici, ou tout autre « B ». Ici encore, la tentative d'accès à un membre privé donne lieu à une erreur... cette ligne là. Comme nous avons vu que les membres protégés sont accessibles dans toute la portée de « B », c'est-à-dire à l'intérieur de « B » via tout objet de type « B » alors, cet accès est possible autant que celui-ci. Tous les objets via lesquels on accède à A sont dans la portée de « B ». Cet accès est possible. Enfin, dernier cas, ici on essaye d'accéder aux membres protégés A mais nous ne sommes pas dans la portée de « B ». Pourquoi ? Nous sommes certes bel et bien à l'intérieur de « B » dans une méthode de « B », mais nous essayons d'accéder à cet attribut via un objet qui n'est pas de type « B ». Nous essayons d'y accéder via un objet de type A. Du coup, nous essayons d'accéder à cet attribut protégé en dehors de la portée de « B » et ceci est interdit. Pour prendre un exemple un peu plus concret, imaginons qu'à la place de notre classe « A », nous ayons une classe « Personnage » qui aurait un attribut protégé donnant le niveau d'énergie du personnage. Le rôle de la classe « B » serait joué par la sous-classe « Guerrier » qui étant la classe « Personnage » et en guise d'équivalent de la méthode « f », nous aurions dans la classe « Guerrier » une méthode « frapper » où le guerrier va frapper un autre personnage. Si l'on veut faire en sorte que dans la

notes

résumé

Accès protégé : portée (2)

```

class A {
    // ...
protected: int a;
private:   int prive;
};

class B: public A {
public:
    // ...
    void f(B autreB, A autreA, int x) {
        a      = x; // OK A::a est protected => accès possible
        prive = x; // Erreur : A::prive est private

        a += autreB.prive; // Erreur (même raison)
        a += autreB.a      ; // OK : dans la même portée (B::)

        a += autreA.a      ; // INTERDIT ! : this n'est pas de la même
                               // portée que autreA
    }
};

```

→ ... Personnage {
protected :
int energie;
...
};

méthode « frapper » le fait que le guerrier frappe le personnage diminue l'énergie de ce dernier, on peut imaginer, par exemple, l'écrire comme ceci. Or, ceci ne fonctionnera pas, cet accès n'est pas possible en vertu de la même règle que celle qui est appliquée ici : l'objet « p », qui a le même rôle que « autreA » dans l'autre exemple, est hors de portée, n'est pas dans la même portée que l'objet « this ». que l'objet « this ».

notes

résumé