



[Accueil principal](#)

[Informatique](#) ▢

[Ordinateurs](#)

[Windows](#)

[macOS](#)

[Ubuntu](#)

[Imprimantes](#)

[Stockage](#)

[Périphériques](#)

[Composants](#)

[Technologies](#)

[Mobile](#) ▢

[Téléphones](#)

[Opérateurs & forfaits](#)

[Android](#)

[iOS](#)

[Batteries & chargeurs](#)

[Accessoires](#)

[Tablettes](#)

[Montres & capteurs](#)

[Applis & Sites](#) ▢

[Réseaux sociaux](#)

[Mail](#)

[Messagerie instantanée](#)

[Bureautique](#)

[Applications & logiciels](#)

[Transports & cartes](#)

[Services en ligne](#)

[Image & Son](#) ▢

[Streaming](#)

[TV & vidéo](#)

[Audio](#)

[Photo](#)

[Jeux vidéo](#)

[Maison](#) ▢


[Box & connexion Internet](#)

[Réseau & Wifi](#)

[Enceintes connectées](#)

[Caméras connectées](#)

[Objets connectés](#)

[Sécurité](#) 

[Virus](#)

[Piratage](#)

[Arnaque](#)

[Protection](#)

[Confidentialité](#)

[VPN](#)

[Sauvegarde](#)

[Téléchargement](#) 

[Audio](#)

[Bureautique](#)

[Développement](#)

[Graphisme](#)

[Internet](#)


[Jeux](#)

[Pilotes/Drivers](#)

[Sécurité](#)

[Système](#)

[Vidéo](#)

[Forum](#) 

[Bureautique](#)

[Développement](#)

[Internet](#)

[Jeux vidéo](#)

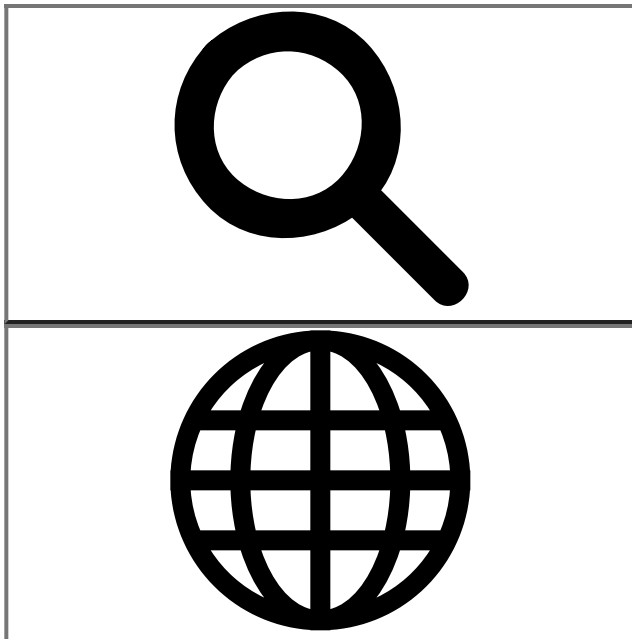
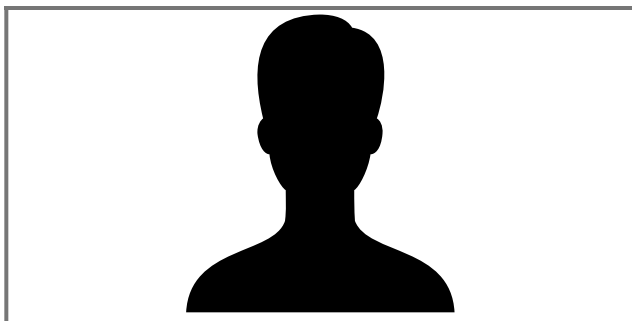
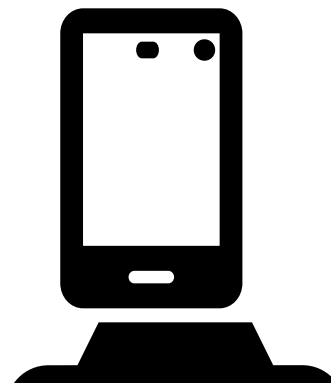
[Matériel](#)

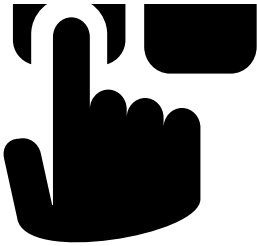
[Réseau](#)

[Vidéo/TV](#)

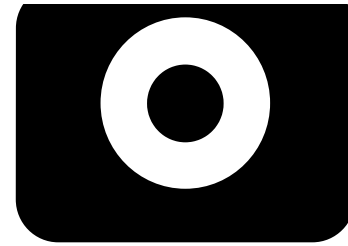
[Virus/Sécurité](#)



[Deutsch](#)[English](#)[Español](#)[Français](#)[Português](#)[Bahasa Indonesia](#)[Italiano](#)[Русский](#)[Polsky](#)[Nederlands](#)[हिंदी](#)[S'inscrire](#)[Connexion](#)[Informatique](#)[Mobile](#)



Applis & Sites



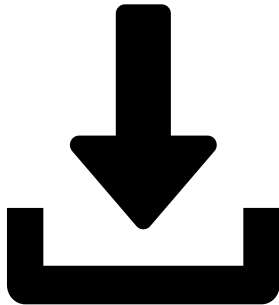
Image



Maison



& Son



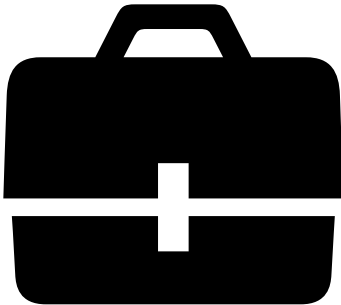
Téléchargement

Securité



Forum

En ce moment Attestation couvre-feu 19 hRayon 10 kmFlash PlayerBox InternetForfaits 4G et 5GSmartphones Android



Emploi

Fiches pratiques Développeurs / DBA Langages C++

C++ - Accesseurs (getters) et mutateurs (setters)

1. La protection des données membres
2. La notion d'accesseur
3. La notion de mutateur

4. A lire aussi: Getter c++

La protection des données membres

L'un des aspects les plus essentiels du concept « orienté objet » est l'encapsulation, qui consiste à définir des étiquettes pour les données membres et les fonctions membres afin de préciser si celles-ci sont accessibles à partir d'autres classes ou non...

De cette manière, des données membres portant l'étiquette *private* ne peuvent pas être manipulées directement par les fonctions membres des autres classes. Ainsi, pour pouvoir manipuler ces données membres, le créateur de la classe (vous en l'occurrence) doit prévoir des fonctions membres spéciales portant l'étiquette *public*, permettant de manipuler ces données.

- Les fonctions membres permettant d'accéder aux données membres sont appelées **accesseurs**, parfois *getter* (appellation d'origine anglophone)
- Les fonctions membres permettant de modifier les données membres sont appelées **mutateurs**, parfois *setter* (appellation d'origine anglophone)

La notion d'accesseur

Un accesseur est une fonction membre permettant de récupérer le contenu d'une donnée membre protégée. Un accesseur, pour accomplir sa fonction :

- doit avoir comme type de retour le type de la variable à renvoyer
- ne doit pas nécessairement posséder d'arguments

Une convention de nommage veut que l'on fasse commencer de façon préférentielle le nom de l'accesseur par le préfixe *Get*, afin de faire ressortir sa fonction première.

La syntaxe d'un accesseur réduit à sa plus simple expression ressemble donc à ceci :

```
class MaClasse{  
private : TypeDeMaVariable MaVariable; public : TypeDeMaVariable  
GetMaVariable();};TypeDeMaVariable MaClasse::GetMaVariable(){ return MaVariable;}
```

Sur l'exemple précédent, l'accesseur minimal de la donnée membre *age* pourrait être le suivant :

```
class Toto{  
private : int age; public : int GetAge();};int Toto::GetAge(){ return age;}
```

La notion de mutateur

Un mutateur est une fonction membre permettant de modifier le contenu d'une donnée membre protégée. Un mutateur, pour accomplir sa fonction :

- doit avoir comme paramètre la valeur à assigner à la donnée membre. Le paramètre doit donc être du type de la donnée membre
- ne doit pas nécessairement renvoyer de valeur (il possède dans sa plus simple expression le type *void*)

Une convention de nommage veut que l'on fasse commencer de façon préférentielle le nom du mutateur par le préfixe *Set*. La syntaxe d'un mutateur réduit à sa plus simple expression ressemble donc à ceci :

```
class MaClasse{
private : TypeDeMaVariable MaVariable; public : void
SetMaVariable(TypeDeMaVariable);};MaClasse::SetMaVariable(TypeDeMaVariable MaValeur){
MaVariable = MaValeur;}
```

Sur l'exemple précédent, le mutateur minimal de la donnée membre *age* pourrait être le suivant :

```
class Toto{
private : int _age; public : void SetAge(int);};void Toto::SetAge(int age){ _age =
age;}
```

L'intérêt principal d'un tel mécanisme est le contrôle de la validité des données membres qu'il procure. En effet, il est possible (et même conseillé) de tester la valeur que l'on assigne à une donnée membre, c'est-à-dire que l'on effectue un test de validité de la valeur de l'argument avant de l'affecter à la donnée membre. Le mutateur ci-dessus peut par exemple vérifier si l'âge de Toto est correct (on considérera pour cela que Toto ne peut pas vivre plus de 200 ans... c'est avec des hypothèses telles que celle-ci que peuvent apparaître des bogues... ah les progrès de la génétique !).

```
class Toto{
private : int _age; public : int SetAge(int);};int Toto::SetAge(int age){ if (age <
200) { _age = age; return 1; } else return 0;}
```

Cet article est régulièrement mis à jour par des [experts](#) sous la direction de [Jean-François Pillou](#), fondateur de CommentCaMarche.

