

La sémantique d'entité

Jusque maintenant, on a vu qu'une seule sémantique, la sémantique de valeur. Pourquoi ? Principalement parce que la bibliothèque standard contient majoritairement des classes à sémantique de valeur. Les classes à sémantiques d'entités seront plus utilisées dans le code métier.

Critère d'unicité

A quoi correspond la sémantique d'entité ? Aux objets comme on le conçoit habituellement. Prenons par exemple une table et deux chaises. Chaque objet est *unique*, deux chaises ont beau être identiques, ce sont bien deux objets différents. Quoi que l'on fasse avec les chaises (sauf les détruire...), les chaises restent individualisables. C'est le principal critère de la sémantique d'entité : l'unicité.

A contrario, la sémantique de valeur ne reconnaît pas cette unicité. Par exemple, si on considère la valeur "5", que l'on entre directement cette valeur dans le code, que cette valeur soit le résultat d'un calcul ou le retour d'une fonction, la valeur est toujours la même, il n'est pas possible de distinguer la valeur "5" provenant de l'une ou l'autre moyen de la créer.

```
const int i = 5;  
const int j = 3 + 2;  
const int k = get_5();
```

Pour prendre un exemple issu de l'univers informatique, prenons par exemple les fenêtres d'un système. Chaque fenêtre est individualisable, en général pour leur position et leur dimensions. Même si deux fenêtres semblent identiques en tous points, il reste possible de les déplacer séparément, d'en fermer une seule, etc. Ce sont bien des entités, au sens "objet unique".

En termes de code, cela signifie que l'on va pouvoir écrire :

```
auto w1 = create_widget("Windows 1");  
auto w2 = create_widget("Windows 2");
```

Conséquences sur l'interface des classes

Fondamentalement, la syntaxe pour créer une classe à sémantique d'entité est la même que pour la sémantique de valeur. Ce qui va distinguer l'une ou l'autre sémantique sera les fonctions que l'on va créer ou non.

Pour rappel, pour définir une classe, il suffit d'utiliser le mot clé `class` ou `struct`, suivi du nom de la classe et de sa définition. Les règles pour le nom des classes sont identiques à celles des noms de variables et fonctions (n'importe quelle lettre majuscule ou minuscule, le caractère souligné et les chiffres sauf en première position).

```
class MyClass {  
};
```

La définition de la classe est donnée dans un bloc, entre crochets. Il ne faut pas oublier le point-virgule à la fin de la définition de la classe.

Accessibilité des membres

La différence entre `class` et `struct` est que dans le premier cas, les membres ne sont pas accessibles depuis l'extérieur de la classe, dans le second cas, ils le sont. Pour changer l'accessibilité, il faut utiliser les mots-clés `public` et `private`.

En pratique, il est classique d'indiquer systématiquement l'accessibilité.

```
class MyClass {  
private:  
    int un_variable_privée {};
```

```
public:  
    int un_variable_public {};  
};
```

Chapitre précédent	Sommaire principal	Chapitre suivant
---------------------------	---------------------------	-------------------------

[Cours, C++](#)