17

Salade de sémantique

S'il est un problème auquel il faut être beaucoup plus attentif en C++ qu'avec d'autres langages orientés objet, c'est sans aucun doute celui de la sémantique que l'on donne aux différentes classes que l'on va créer. Il est en effet possible de classer l'ensemble des classes dans deux grandes catégories : les classes ayant *sémantique de valeur* d'une part et les classes ayant *sémantique d'entité* de l'autre.

Les classes ayant *sémantique de valeur* sont des classes pour lesquelles il est possible, à un instant T donné, d'avoir plusieurs instances en mémoire présentant des valeurs tout à fait identiques. C'est le cas, par exemple, de classes comme `couleur`, `point`, `réservoir à essence`, `adresse`, etc. Bref, tout ce qui est susceptible d'exister en plusieurs exemplaires scrupuleusement identiques sans qu'il soit nécessaire d'assurer une identité unique à chaque exemplaire. Ces classes sont :

- généralement (mais pas obligatoirement) constantes : si vous modifiez ne serait-ce qu'une des valeurs qui les composent, vous obtenez un objet totalement différent ;
- copiables : vous pouvez souhaiter peindre votre salon dans la même couleur que celui de votre voisin ;
- affectables : vous pouvez décider de repeindre votre salon dans une couleur différente de sa couleur d'origine ;
- comparables, au minimum par égalité : en comparant les valeurs qui composent deux instances distinctes, vous pourrez déterminer si elles sont égales. Dans certains cas, vous pourrez aussi envisager de les comparer avec les opérateurs `>`, `<`, `>=` et `<=` ;
- peu enclines à intervenir dans une relation d'héritage, ni en tant que classe de base, ni en tant que classe dérivée.

Les classes ayant *sémantique d'entité* sont, quant à elles, des classes dont chaque instance doit impérativement pouvoir être identifiée de manière unique et non ambiguë.

Bien sûr, vous me direz que, de toute façon, toutes les variables peuvent l'être ne serait-ce que parce qu'elles se trouvent à des adresses mémoire différentes !

Ici, je parle plutôt du fait qu'il y a sans doute une (ou plusieurs) donnée(s) qui vous donnera la certitude de manipuler une instance bien particulière à l'exclusion de n'importe quelle autre et ce, même si elle est noyée dans la masse d'autres instances de la même classe. C'est le cas de classes comme `personne`, `véhicule`, ou encore `compte bancaire`. Bref, toutes les classes pour lesquelles chaque instance doit être scrupuleusement unique pour avoir la certitude que si vous en modifiez une donnée, la modification s'applique effectivement à cette instance particulière.

La manière dont on est susceptible de s'assurer l'unicité référentielle importe peu en soi. Ce qui importe, c'est que l'on ait la garantie que le dépôt qui est destiné à notre compte courant n'arrive pas sur le compte d'une autre personne.

Ces classes sont :

- généralement modifiables, du moins pour tout ce qui n'intervient pas dans la définition de ce qui permet de l'identifier de manière unique. Vous pourrez faire varier le solde d'un compte en banque, mais en revanche vous ne pourrez pas modifier le numéro qui permet de l'identifier ;
- non copiables : vous n'apprécieriez pas trop que quelqu'un ait un compte en banque portant le même numéro que le vôtre et que votre salaire soit versé... sur le compte de cette personne ;
- non affectable, pour la même raison ;
- non comparable : vous pouvez envisager de comparer certaines données auxquelles la classe donne accès (comme le solde ou le numéro d'identification), mais il n'y a aucune raison d'en arriver à comparer la classe dans son intégralité ;
- des candidats idéaux à intervenir dans une relation d'héritage, que ce soit comme classe de base ou comme classe dérivée.

En fait, nous pourrions plutôt dire que, dès qu'une classe intervient dans une hiérarchie d'héritage, il y a de très fortes chances pour qu'elle ait une sémantique d'entité.

Cette distinction est particulièrement importante car elle pose les bases de ce que vous pourrez envisager de faire avec les différentes classes que vous pourrez créer et des services que vous serez en droit d'en attendre.

Nous disposons donc de recettes toutes faites qui nous évitent de perdre du temps à tenter de faire des choses qui ne marcheront jamais correctement.