Glossaire

Programmation Orientée Objet

amitié
: L'amitié en C++ permet à des fonctions, des méthodes ou des classes d'accéder à des membres privées d'une autre classe. Les amies se déclarent en faisant précéder la déclaration classique du mot clé `friend` à l'intérieur de la déclaration de la classe cible. L'amitié n'est pas transitive : les amis des amis ne sont pas des amis. Une classe A amie d'une classe B, elle-même amie d'une classe C, n'est pas amie de la classe C par défaut. L'amitié n'est pas héritée : mes amis ne sont pas les amis de mes enfants. Si une classe A est amie d'une classe B et que la classe C est une classe fille de la classe B, alors A n'est pas amie de la classe C par défaut.

attribut
: Un attribut est une variable membre d'un objet.

classe
: Une classe décrit des propriétés communes (caractéristiques, responsabilités, comportements) d'un ensemble d'objets. Elle apparaît comme un type (ou un « moule à objets ») à partir duquel il sera possible de créer des objets. Les éléments de cet ensemble sont les instances de la classe (les objets).

classe abstraite
: Une classe est dite abstraite lorsqu'on ne peut instancier d'objets à partir de cette classe. En C++, cela signifie qu'elle contient au moins une méthode déclarée virtuelle pure.

constructeur
: Lorsqu'on instancie un objet, le constructeur de la classe est appelé automatiquement au moment de la création. Un constructeur est chargé d'initialiser l'état d'un objet (donc ses membres). Un constructeur est une méthode (ou plusieurs) qui porte toujours le même nom que la classe. Il peut avoir des paramètres, et des valeurs par défaut. Il n'a jamais de type de retour.

constructeur de copie
: Le constructeur de copie permet d'initialiser un nouvel objet à partir d'un objet existant. Il est appelé dans la création d'un objet à partir d'un autre objet pris comme modèle, le passage en paramètre d'un objet par valeur à une fonction ou une méthode et le retour d'une fonction ou une méthode renvoyant un objet.

constructeur par défaut
: Un constructeur par défaut a pour rôle de créer une instance non initialisée quand aucun autre constructeur fourni n'est applicable. Il est donc conseillé de toujours écrire un constructeur par défaut.

destructeur
: Le destructeur est la méthode membre appelée automatiquement lorsqu'une instance (objet) de classe cesse d'exister en mémoire. Son rôle est de libérer toutes les ressources qui ont été acquises par cet objet. Un destructeur est une méthode qui porte toujours le même nom que la classe, précédé de "~". Il ne possède aucun paramètre. Il n'y en a qu'un et un seul. Il n'a jamais de type de retour.

encapsulation
: L'encapsulation est l'idée de protéger les variables contenues (les attributs) dans un objet et de ne proposer que des méthodes pour les manipuler. En respectant ce principe, tous les attributs d'une classe seront donc privées (`private`). L'objet est ainsi vu de l'extérieur comme une « boîte noire » possédant certaines propriétés et ayant un comportement spécifié. C'est le comportement d'un objet qui modifiera son état. On ajoutera souvent (mais pas obligatoirement) des méthodes

publiques pour établir un accès contrôlé aux attributs privés de la classe : un accesseur `get()` (*getter*) qui permet l'accès en lecture et un mutateur ou manipulateur `set()` (*setter*) qui permet l'accès en écriture.

exception

Les exceptions ont été ajoutées à la norme du C++ afin de faciliter la mise en œuvre de code robuste. Une exception est l'interruption de l'exécution du programme à la suite d'un événement particulier (= exceptionnel !) et le transfert du contrôle à des fonctions spéciales appelées gestionnaires. Le but des exceptions est de réaliser des traitements spécifiques aux événements qui en sont la cause. La gestion d'une exception est découpée en deux parties distinctes : le déclenchement avec l'instruction `throw` et le traitement (l'inspection et la capture) avec deux instructions inséparables `try` et `catch`.

héritage

L'héritage est un concept fondamental de la programmation orientée objet. Il se nomme ainsi car le principe est en quelque sorte le même que celui d'un arbre généalogique. Ce principe est fondé sur des classes « filles » qui héritent des caractéristiques des classses « mères ». L'héritage permet d'ajouter des propriétés à une classe existante pour en obtenir une nouvelle plus précise. Il permet donc la spécialisation ou la dérivation de types. En utilisant l'héritage, il est possible d'ajouter des caractéristiques (attributs) et/ou des comportements propres (méthodes), d'utiliser les caractéristiques héritées et de redéfinir les comportements (méthodes héritées).

instance

Une instance d'une classe est un objet. Instancier revient à créer des objets à partir de classes.

liste d'initialisation

La liste d'initialisation permet d'utiliser le constructeur de chaque donnée membre, et ainsi d'éviter une affectation après coup. La liste d'initialisation doit être utilisée pour certains objets qui ne peuvent pas être contruits par défaut : c'est le cas des références et des objets constants. La liste d'initialisation est aussi utilisée pour appeler le constructeur d'une classe parente en cas d'héritage. Elle suit la définition du constructeur après avoir ajouté le caractère '`:`'. Chaque initialisation est séparée par des virgules '`,`'.

membre statique

Un membre statique est déclaré avec l'attribut `static` et il sera partagé par tous les objets de la même classe. Il existe même lorsque aucun objet de cette classe n'a été créé. Un membre statique peut être un attribut ou une méthode. Un attribut statique doit être initialisé explicitement, à l'extérieur de la classe (même s'il est privé), en utilisant l'opérateur de résolution de portée (::) pour spécifier sa classe. En général, son initialisation se fait dans le fichier `.cpp` de définition de la classe. On peut accéder directement à un membre statiquer en le nom de la classe suivi de l'opérateur de résolution de portée :: et du nom du membre.

méthode

Une méthode est une fonction membre d'un objet.

méthode constante

On déclare et on définit une méthode constante en ajoutant le modificateur `const`. Une méthode constante est tout simplement une méthode qui ne modifie aucun des attributs de l'objet.

méthode inline

C++ présente une amélioration des macros du langage C avec les fonctions (et donc les méthodes) *inline*. Elles ont le comportement des fonctions (vérification des arguments et de la valeur de retour) et elles sont substituées dans le code après vérification. Une méthode est dite *inline* lorsque le corps de la méthode est définie directement dans la déclaration de la classe ou lorsqu'on définit une méthode, on ajoute au début de sa définition le mot-clé `inline`. Les fonctions *inline* (ou les macros) ont l'avantage d'être plus rapide qu'une fonction normale mais les inconvénients de produire un programme binaire plus volumineux et de compliquer la compilation séparée.

méthode virtuelle
Une méthode virtuelle est une méthode définie dans une classe qui est destinée à être redéfinie dans les classes qui en héritent. En C++, on utilisera le mot clé `virtual` dans la déclaration de la méthode. Les fonctions virtuelles permettent d'exprimer des différences de comportement entre des classes de la même famille. Ces différences sont ce qui engendre un comportement polymorphe.

méthode virtuelle pure
Une méthode virtuelle ne possédant qu'une déclaration à l'intérieur d'une classe, sans être définie dans cette classe est dite abstraite ou virtuelle pure. Dans ce cas, la classe est dite abstraite car il sera impossible techniquement d'instancier un objet à partir de cette classe. En C++, la syntaxe spéciale `= 0` est utilisée dans la déclaration de la méthode.

objet
Concrètement, un objet est une structure de données (ses attributs c.-à-d. des variables) qui définit son état et une interface (ses méthodes c.-à-d. des fonctions) qui définit son comportement. Un objet est créé à partir d'un modèle appelé classe. Chaque objet créé à partir de cette classe est une instance de la classe en question. Un objet posséde une identité qui permet de distinguer un objet d'un autre objet (son nom, une adresse mémoire).

objet constant
On déclare un objet constant avec le modificateur `const`. On ne pourra pas modifier l'état (ses attributs) d'un objet constant. On ne peut appliquer que des méthodes constantes sur un objet constant.

paramètre par défaut
Le langage C++ offre la possibilité d'avoir des valeurs par défaut pour les paramètres d'une fonction (ou d'une méthode), qui peuvent alors être sous-entendus au moment de l'appel. Il suffit de faire suivre la déclaration du paramètre de l'opérateur `=` et d'une valeur.

polymorphisme
Le polymorphisme est un moyen de manipuler des objets hétéroclites de la même manière, pourvu qu'ils disposent d'une interface commune. Un objet polymorphe est donc un objet susceptible de prendre plusieurs formes pendant l'exécution. Le polymorphisme représente la capacité du système à choisir dynamiquement la méthode qui correspond au type de l'objet en cours de manipulation. Le polymorphisme est implémenté en C++ avec les méthodes virtuelles (`virtual`) et l'héritage.

programmation orientée objet
La programmation orientée objet (POO) consiste à définir des objets logiciels et à les faire interagir entre eux. Il existe actuellement deux grandes catégories de langages à objets : les langages à classes (C++, C#, Java, Python, ...) et les langages à prototypes (JavaScript).

redéfinition (*overriding*)
Une redéfinition (*overriding*) permet de fournir une nouvelle définition d'une méthode d'une classe ascendante pour la remplacer dans le cas d'un héritage. Elle doit avoir une signature rigoureusement identique à la méthode parente. La signature d'une méthode (ou d'une fonction) comprend son nom et sa liste de paramètres. Le type de retour ne fait pas partie de la signature.

surcharge (*overloading*)
Une surcharge (ou surdéfinition) permet d'utiliser plusieurs méthodes qui portent le même nom au sein d'une même classe avec des signatures différentes. La signature d'une méthode (ou d'une fonction) comprend son nom et sa liste de paramètres. Le type de retour ne fait pas partie de la signature. Pour surcharger une méthode, il suffit que le type et/ou le nombre de paramètres soit différent.

template
En C++, la fonctionnalité template fournit un moyen de réutiliser du code source. Les *templates*

permettent d'écrire des fonctions et des classes en paramétrant le type de certains de leurs constituants (type des paramètres ou type de retour pour une fonction, type des éléments pour une classe collection par exemple). Les *templates* permettent d'écrire du code générique, c'est-à-dire qui peut servir pour une famille de fonctions ou de classes qui ne diffèrent que par la valeur de ces paramètres.

visibilité

La notion de visibilité indique qui peut avoir accès à un membre. La visibilité est précisée au sein des classes (et non des objets) et elle peut être : `public` (`+`) quand toutes les autres classes ont accès, `protected` (`#`) quand seules la classe elle-même et les classes filles (héritage) ont accès et `private` (`-`) quand seule la classe elle-même a accès.