Chapitre 2
La conception orientée objet

1. Approche procédurale et décomposition fonctionnelle

Avant d'énoncer les bases de la programmation objet nous allons revoir l'approche procédurale à l'aide d'un exemple concret d'organisation de code.

La programmation procédurale est un paradigme de programmation considérant les différents acteurs d'un système comme des objets pratiquement passifs qu'une procédure centrale utilisera pour une fonction donnée.

Prenons l'exemple de la distribution d'eau courante dans nos habitations et essayons d'en émuler le principe dans une application très simple. L'analyse procédurale (tout comme l'analyse objet d'ailleurs) met en évidence une liste d'objets qui sont :

– le robinet de l'évier ;
– le réservoir du château d'eau ;
– un capteur de niveau d'eau avec contacteur dans le réservoir ;
– la pompe d'alimentation puisant l'eau dans la rivière.

Le code du programme "procédural" consisterait à créer un ensemble de variables représentant les paramètres de chaque composant puis d'écrire une boucle de traitement de gestion centrale testant les valeurs lues et agissant en fonction du résultat des tests. On notera qu'il y a d'un côté les variables et de l'autre, les actions.

2. La transition vers l'approche objet

La programmation objet est un paradigme de programmation considérant les différents acteurs d'un système comme des objets actifs et en relation. L'approche objet est souvent très voisine de la réalité.

Dans notre exemple, l'utilisateur ouvre le robinet ; le robinet relâche la pression et l'eau s'écoule du réservoir à l'évier ; le capteur/flotteur du réservoir arrive à un niveau qui déclenche la pompe ; l'utilisateur referme le robinet ; alimenté par la pompe, le réservoir du château d'eau se remplit et le capteur/flotteur atteint un niveau qui arrête la pompe.

Dans cette approche, vous constatez que les objets interagissent ; il n'existe pas de traitement central définissant dynamiquement le fonctionnement des objets. Il y a eu, en amont, une analyse fonctionnelle qui a conduit à la création des différents objets, à leurs montages et à leurs mises en relations.

Le code du programme "objet" va suivre cette réalité en proposant autant d'objets que décrit précédemment mais en définissant entre ces objets des méthodes d'échange adéquates qui conduiront au fonctionnement escompté.

Remarque

Les concepts objets sont très proches de la réalité...

3. Les caractéristiques de la POO

3.1 L'objet, la classe et la référence

3.1.1 L'objet

L'objet est l'élément de base de la POO. L'objet est l'union :

– d'une liste de variables d'états,

– d'une liste de comportements,

– d'une identification.

Les variables d'états changent durant la vie de l'objet. Prenons le cas d'un lecteur de musiques numériques. À l'achat de l'appareil, les états de cet objet pourraient être :

– mémoire libre = **100%**

– taux de charge de la batterie = **faible**

– aspect extérieur = **neuf**

Au fur et à mesure de son utilisation, ses états vont être modifiés. Rapidement la mémoire libre va chuter, le taux de charge de la batterie variera et l'aspect extérieur va changer en fonction du soin apporté par l'utilisateur.

Les comportements de l'objet définissent ce que peut faire cet objet : Jouer la musique - Aller à la piste suivante - Aller à la piste précédente - Monter le son, etc. Une partie des comportements de l'objet est accessible depuis l'extérieur de cet objet : bouton Play, bouton Stop... et une autre partie est uniquement interne : lecture de la carte mémoire, décodage de la musique à partir du fichier. On parle "d'encapsulation" pour définir une limite entre les comportements accessibles de l'extérieur et les comportements internes.

L'identification d'un objet est une information, détachée de la liste des états, permettant de différencier l'objet de ses congénères (c'est-à-dire d'autres objets qui ont le même type que lui). L'identification peut être un numéro de référence ou une chaîne de caractères unique construite à la création de l'objet ; elle peut également être une adresse mémoire. En réalité, sa forme dépend totalement de la problématique associée.

L'objet POO peut être attaché à une entité bien réelle, comme pour notre lecteur numérique, mais il peut être également attaché à une entité totalement virtuelle comme un compte client, l'entrée d'un répertoire téléphonique... La finalité de l'objet informatique est de gérer l'entité physique ou de l'émuler.

Même si ce n'est pas dans sa nature de base, l'objet peut être rendu persistant c'est-à-dire que ses états peuvent être enregistrés sur un support mémorisant l'information de façon intemporelle. À partir de cet enregistrement, l'objet pourra être recréé quand le système le souhaitera.

3.1.2 La classe

La classe est le "moule" à partir duquel l'objet va être créé en mémoire. La classe contient les états et les comportements communs d'un même type et les valeurs de ces états seront contenues dans ses objets.

Les comptes courants d'une même banque contiennent tous les mêmes paramètres (coordonnées du détenteur, solde...) et ont tous les mêmes fonctions (créditer, débiter...). Ces définitions doivent être contenues dans une classe et, à chaque fois qu'un client ouvre un nouveau compte, cette classe servira de modèle à la création de l'objet compte.

Le présentoir de lecteurs de musiques numériques propose un même modèle en différentes couleurs, avec des tailles mémoire modulables, etc. Chaque appareil du présentoir est un objet qui a été fabriqué à partir des informations d'une seule classe. À la réalisation, les attributs de l'appareil ont été choisis en fonction de critères esthétiques et commerciaux.

Une classe peut contenir beaucoup d'attributs. Ils peuvent être de type primitif – des entiers, des caractères... – mais également de type plus complexe. En effet, une classe peut contenir une ou plusieurs classes d'autres types. On parle alors de composition ou encore de "couplage fort". La destruction de la classe principale entraîne, évidemment, la destruction des classes qu'elle contient. Par exemple, si une classe *hôtel* contient une liste de *chambres*, la destruction de l'*hôtel* entraîne la destruction des *chambres*.

Une classe peut faire "référence" à une autre classe ; dans ce cas, le couplage est dit "faible" et les objets peuvent vivre indépendamment. Par exemple, votre PC est relié à votre imprimante. Si votre PC rend l'âme, votre imprimante fonctionnera certainement avec le futur PC ! On parle d'association.

En plus de ses attributs, la classe contient également une série de "comportements", c'est-à-dire une série de méthodes avec signatures et code attaché. Ces méthodes sont directement "copiées" dans les objets et utilisées telles quelles.

On déclare la classe et son contenu dans un même fichier source à l'aide d'une syntaxe que l'on étudiera en détail. Les développeurs C++ apprécieront le fait qu'il n'existe plus, d'un côté, une partie définitions et, de l'autre, une partie implémentations. En effet, en C#, le fichier programme (d'extension .CS) contient les définitions de tous les états avec éventuellement une valeur "de départ" et les définitions et implémentations de tous les comportements. On verra qu'il existe en C# une solution permettant de définir une classe dans plusieurs fichiers afin que des intervenants puissent travailler en parallèle sans jamais effacer le travail de l'autre.

3.1.3 La référence

Les objets sont construits à partir de la classe, par un processus appelé l'instanciation, et donc, tout objet est l'instance d'une classe. Chaque instance commence à un emplacement mémoire unique. Cet emplacement mémoire connu sous le nom de *pointeur* par les développeurs C et C++ devient une *référence* pour les développeurs C# et Java.

Quand le développeur a besoin d'un objet pendant un traitement, il doit :

– déclarer et nommer une variable du type de la classe à utiliser ;

– instancier l'objet et enregistrer sa référence dans cette variable.

Une fois cette instanciation réalisée, le programme accédera aux propriétés et aux méthodes de l'objet par la variable contenant sa référence. Chaque instance est unique. Par contre, plusieurs variables peuvent "pointer" sur une même instance. C'est d'ailleurs quand plus aucune variable ne pointe sur une instance donnée que le ramasse-miettes enregistre cette instance comme devant être détruite.

3.2 L'encapsulation

L'encapsulation consiste à créer une sorte de boîte noire contenant en interne un mécanisme protégé et en externe un ensemble de commandes qui vont permettre de le manipuler. Ce jeu de commandes est fait de telle sorte qu'il sera impossible d'altérer le mécanisme protégé en cas de mauvaise utilisation. La boîte noire sera si opaque qu'il sera impossible à l'utilisateur d'intervenir en direct sur le mécanisme.

Vous l'aurez compris, la boîte noire n'est autre qu'un objet avec des méthodes publiques de "haut niveau" contrôlant avec rigueur les paramètres passés avant de les utiliser dans un traitement. En respectant le principe de l'encapsulation, vous ferez en sorte que jamais l'utilisateur de l'objet ne puisse accéder "en direct" à vos données. Grâce à ce contrôle, votre objet sera correctement utilisé, donc plus fiable, et sa mise au point sera plus facile. Dans le monde Java, les méthodes permettant d'accéder en lecture aux données sont des accesseurs et celles permettant d'accéder en écriture sont des mutateurs. Nous verrons que C# propose une solution très élégante, les propriétés, qui permet de garder le côté pratique de l'accès direct aux données de l'objet tout en respectant les principes de l'encapsulation.

3.3 L'héritage

Une autre notion importante de la POO concerne l'héritage. Pour l'expliquer, imaginons que nous devions construire un système de gestion des différents types de collaborateurs d'une société. Une analyse rapide met en évidence une liste de propriétés communes à tous les postes. En effet, que le collaborateur soit ouvrier, cadre, intérimaire ou encore directeur, il a toujours un nom, un prénom et un numéro de sécurité sociale... On appelle cela la généralisation ; elle consiste à factoriser les éléments communs d'un ensemble de classes dans une classe plus générale appelée superclasse en Java et plutôt "classe de base" en C# et C++. La classe qui hérite de la superclasse est appelée sous-classe ou classe héritière.