Chapitre 5
Programmation objet

1. Introduction

Avec Visual Basic .NET, la notion d'objet est devenue omniprésente et nécessite un minimum d'apprentissage. Nous allons donc voir dans un premier temps le principe de la programmation objet et le vocabulaire associé, puis nous verrons comment mettre cela en application avec Visual Basic.

Dans un langage procédural classique, le fonctionnement d'une application est réglé par une succession d'appels aux différentes procédures et fonctions disponibles dans le code. Il n'y a aucune liaison entre les données et les procédures qui les manipulent. Dans un langage objet, on va au contraire essayer de regrouper au maximum les données et le code pour les manipuler. Les classes sont la représentation symbolique des objets. Elles décrivent les champs, propriétés, méthodes et événements de la même manière qu'un plan d'architecte décrit les différentes parties d'un bâtiment.

Poursuivons notre analogie entre une classe et un plan de bâtiment. Nous savons qu'il est possible de construire plusieurs bâtiments à partir du même plan. De la même manière, plusieurs objets peuvent être construits à partir de la même classe. Une classe peut donc être utilisée pour créer autant d'instances que nécessaire.

Sur un plan de bâtiment, certaines zones peuvent avoir un accès limité à certaines personnes. De la même façon, dans une classe, certains éléments peuvent avoir un accès restreint. C'est le principe d'encapsulation.

Les termes classe et objet sont souvent confondus mais il s'agit, en fait, d'éléments bien distincts. Une classe représente la structure d'un élément alors que l'objet est un exemplaire créé sur le modèle de cette structure. La modification d'un élément dans un objet ne change absolument pas les autres objets créés à partir du même modèle (classe). Dans notre exemple de plan de bâtiment, l'ajout d'une nouvelle pièce à un bâtiment existant ne change en rien les autres bâtiments construits suivant le même plan. Par contre, la modification du plan (de la classe) entraîne des modifications pour tous les nouveaux bâtiments (tous les nouveaux objets).

Les classes sont constituées de champs, propriétés, méthodes et événements. Les champs et les propriétés représentent les informations contenues dans les objets. Les champs sont considérés comme des variables et il est possible de lire leur contenu ou de leur affecter une valeur directement. Par exemple, si vous avez une classe représentant un client, vous pouvez enregistrer son nom dans un champ.

Les propriétés se manipulent de la même façon que les champs, mais sont mises en œuvre à partir de procédures de propriété `Get` et `Set`. Ceci autorise plus de contrôle sur la façon dont les valeurs sont lues ou affectées et permet de valider les données avant leur utilisation.

Les méthodes représentent les actions qu'un objet peut effectuer. Elles sont mises en œuvre par la création de procédures ou fonctions dans une classe.

Les événements sont des informations qu'un objet reçoit ou transmet depuis ou vers un autre objet ou application. Les événements permettent aux objets d'exécuter des actions lorsqu'une situation particulière se produit. Comme Windows est un système d'exploitation événementiel, les événements peuvent provenir d'autres objets, du système ou des actions de l'utilisateur sur la souris et le clavier.

Ceci n'est qu'une facette de la programmation orientée objet. Trois autres éléments sont également fondamentaux :

– L'encapsulation.
– L'héritage.
– Le polymorphisme.

L'encapsulation est la capacité permettant de créer et de contrôler l'accès à un groupe d'éléments. Les classes fournissent le moyen le plus fiable d'assurer l'encapsulation. Si nous prenons l'exemple d'un compte bancaire, dans une programmation classique, il nous faudrait de nombreuses variables et procédures ou fonctions pour manipuler les informations. La situation serait encore plus complexe si nous devions gérer simultanément plusieurs comptes bancaires. Il faudrait alors travailler avec des tableaux et jongler avec les index.

L'encapsulation permet de regrouper les informations et le code les manipulant dans une classe. Si vous devez travailler avec plusieurs comptes bancaires simultanément, vous aurez alors plusieurs instances de la même classe, limitant ainsi le risque d'erreurs. L'encapsulation assure également un contrôle sur l'utilisation des données et des procédures ou fonctions. Vous pouvez utiliser les modificateurs d'accès, tels que `Private` ou `Protected`, pour restreindre l'accès à certaines méthodes, propriétés ou champs. Une règle fondamentale de l'encapsulation stipule que les données d'une classe ne doivent être manipulées que par le code de la classe (procédures de propriétés ou méthodes). Cette technique est parfois appelée dissimulation de données. Elle assure la sécurité de fonctionnement de votre code en masquant les détails internes de la classe et en évitant ainsi qu'ils ne soient utilisés de manière inappropriée. Elle autorise aussi la modification d'une partie du code sans perturber le fonctionnement du reste de l'application.

L'héritage permet la création d'une nouvelle classe, basée sur une classe existante. La classe servant de modèle pour la création d'une autre classe est appelée classe de base. La classe ainsi créée hérite des champs, propriétés, méthodes et événements de la classe de base. La nouvelle classe peut être personnalisée en y ajoutant des champs, propriétés, méthodes et événements. Les classes créées à partir d'une classe de base sont appelées classes dérivées. Vous pouvez donc définir une classe de base et la réutiliser plusieurs fois pour créer des classes dérivées.

Le polymorphisme est une autre notion importante de la programmation orientée objet. Par son intermédiaire, il est possible d'utiliser plusieurs classes de manière interchangeable même si ces classes implémentent leurs propriétés et méthodes de manière différente. Ces propriétés et méthodes sont utilisables par le même nom, indépendamment de la classe à partir de laquelle l'objet a été construit.

Trois autres concepts sont également associés au polymorphisme. La surcharge, la substitution et le masquage de membres permettent la définition de membres d'une classe portant le même nom. Il existe cependant quelques petites distinctions entre ces trois techniques.

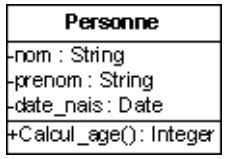
La surcharge est utilisée pour concevoir des propriétés ou des méthodes portant le même nom mais ayant un nombre de paramètres différents ou des types de paramètres différents.

La substitution permet la redéfinition de méthodes ou propriétés héritées d'une classe de base. Les membres substitués peuvent accepter le même nombre et type de paramètres que la méthode ou propriété de la classe de base.

Le masquage sert à remplacer localement, dans une classe, un membre d'une classe. N'importe quel type de membre peut masquer un autre membre. Par exemple, une propriété peut masquer une méthode héritée. Le masquage se fait uniquement grâce au nom. Les membres masqués ne sont pas héritables.

2. Mise en œuvre avec Visual Basic

Dans le reste de ce chapitre, nous allons travailler sur la classe `Personne` dont la représentation UML (*Unified Modeling Language*) est disponible ci-dessous.



UML est un langage graphique dédié à la représentation des concepts de programmation orienté objet. Pour plus d'informations sur ce langage, vous pouvez consulter l'ouvrage UML 2 - Initiation, exemples et exercices corrigés, dans la même collection.

2.1 Création d'une classe

La création d'une classe passe par la déclaration de la classe elle-même et de tous les éléments la constituant.

2.1.1 Déclaration de la classe

La déclaration d'une classe se fait en utilisant les mots clés `Class` et `End Class`. Entre ces deux mots clés, on trouve des déclarations de variables qui seront les champs de la classe et des procédures qui seront les méthodes de la classe.

La syntaxe générale de définition d'une classe est donc la suivante :

```
[ Public | Private | Protected | Friend | Protected Friend ]
[ MustInherit | NotInheritable ]
Class nom de la classe
[ Inherits nom de la classe de base]
[ Implements nom de l'interface]

End Class
```

De nombreux mots clés sont disponibles pour la personnalisation d'une classe. Au moment de sa déclaration, on peut spécifier la visibilité de la classe. Les mots clés suivants sont disponibles :

`Public`

La classe pourra être utilisée dans tout votre projet mais aussi dans d'autres projets.

`Friend`

L'accès à la classe est limité au projet dans lequel elle est définie.

`Private`
La classe ne peut être utilisée que dans le module dans lequel elle est définie.

`Protected`
La classe ne peut être utilisée que dans une sous-classe de celle dans laquelle elle est définie. Ce mot clé ne peut être utilisé que pour une classe déclarée dans une autre classe.

`Protected Friend`
Identique à l'union des portées `protected` et `friend`.

Vous pouvez également indiquer comment votre classe va se comporter vis-à-vis de l'héritage. Deux options sont possibles :

`MustInherit`
Indique que la classe sert de classe de base dans une relation d'héritage. Vous ne pourrez pas créer d'instances de cette classe. En général, dans ce genre de classe, seules les déclarations des méthodes sont définies, il faudra dans les classes dérivées écrire le contenu de ces méthodes.

`NotInheritable`
Cette classe sera la dernière de la hiérarchie. Il ne sera donc pas possible d'utiliser cette classe comme classe de base d'une autre classe.

Pour indiquer que votre classe récupère les caractéristiques d'une autre classe par une relation d'héritage, vous devez utiliser le mot clé `Inherits` suivi du nom de la classe de base. Vous pouvez également implémenter dans votre classe une ou plusieurs interfaces. Ces deux notions seront vues en détail plus loin dans ce chapitre.

Le début de la déclaration de notre classe `Personne` est donc le suivant :

```
Public Class Personne
   Dim  nom As String
   Dim prenom As String
   Dim date_naiss As Date
End Class
```

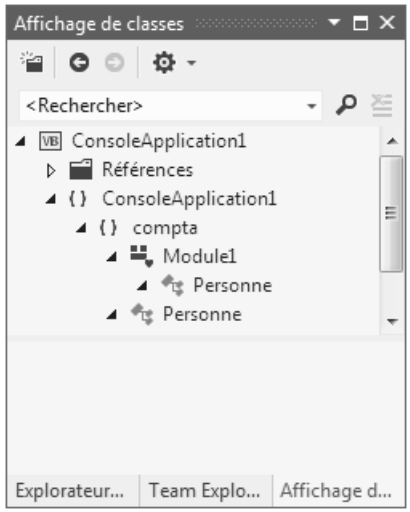
2.1.2 Classe partielle

La définition d'une classe peut être répartie sur plusieurs déclarations en utilisant le mot clé `Partial`. Cette technique autorise la définition de la classe dans plusieurs fichiers sources. Elle est très utilisée dans Visual Studio pour permettre la personnalisation de classes générées automatiquement. Le code généré est en général placé dans un fichier nommé `.designer.vb` qui ne doit, en principe, pas être modifié directement. Lors de la compilation, le compilateur regroupe toutes les définitions partielles pour obtenir le code source de la classe. Les différentes parties de la définition d'une classe doivent par contre être dans le même projet et faire partie du même namespace.

Il y a un autre petit piège à éviter également. Regardons le code suivant :

```
Namespace Compta
    Partial Public Class Personne
        Public Sub procedure1()
        End Sub
    End Class
    Module Module1
        Partial Public Class Personne
            Public Sub Procedure2()
            End Sub
        End Class
    End Module
End Namespace
```

Au premier abord rien d'illégal puisque le compilateur génère le code correctement. Par contre, il n'a pas la même vision des choses que nous. Regardons ce que nous présente l'affichage de classes.



Deux classes `Personne` sont disponibles. Le compilateur a en fait considéré que nos deux définitions de classe ne font pas partie du même namespace. En effet l'une d'entre elles est située dans le module, et le nom de celui-ci est rajouté automatiquement comme préfixe à la classe.