

Introduction à la POO illustrée par VB6

par [Xavier VLIEGHE](#)

Date de publication : 14/10/2005

Dernière mise à jour :

Manipulation d'un objet via un module de classe : le tutoriel a pour but de dessiner un simple rectangle en utilisant les classes, base de la POO.

Remerciements

- 1 - Public visé et contenu
- 2 - Pourquoi la POO ?
- 3 - Concepts de base
 - 3.1 - Objet
 - 3.2 - Classe
- 4 - Description de notre exemple, le module de classe clsRectangle
 - 4.1 - Propriété
 - 4.2 - Méthode
 - 4.3 - Evénements
 - a - Un événement, à quoi ça sert ?
 - b - Déclaration
 - c - Restriction
 - d - Déclenchement
- 5 - Le programme
- 6 - Pour aller plus loin
- 7 - Téléchargements

Remerciements

Merci à Jean-Serge pour m'avoir initié à la POO.

Merci à [khany](#), à mon parrain [ridan](#), à [Nigthfall](#) et à [bidou](#) pour leurs corrections et leurs conseils.

1 - Public visé et contenu

Les développeurs VB6 n'ayant pas eu l'occasion d'aborder la Programmation Orientée Objet (POO) via les modules de classe.

La liste des champs et des procédures de l'objet a été minimisée afin d'alléger le code, le but étant de présenter les concepts et non de faire du "volume".

2 - Pourquoi la POO ?

La POO est aujourd'hui plébiscitée par un grand nombre de développeurs !

En effet, ce type de programmation possède de nombreux avantages :

- Réutilisation du code, donc gain de temps
- Code mieux structuré, maintenance plus aisée

Ces concepts sont malheureusement assez peu mis en avant dans le langage VB6 (plutôt orienté programmation événementielle), nous allons voir qu'il est pourtant aisé de les utiliser.

3 - Concepts de base

Les notions inhérentes à la POO seront abordées ici de manière plutôt succincte. Pour ceux qui souhaitent creuser, je vous proposerai tout au long de la rédaction des liens afin d'approfondir les concepts concernés.

3.1 - Objet

Un **objet** est le modèle conceptuel d'une notion bien précise, appelée **Domaine**.

Un objet est caractérisé par :

son identité : C'est ce qui permet d'identifier un objet parmi d'autres. Dans le code, le nom de variable remplit cette fonction.

ses propriétés (ou attributs) : Les propriétés sont les données intrinsèques de l'objet que l'on souhaite gérer. En prenant l'exemple d'un objet Rectangle, il faudra au minimum gérer sa hauteur et sa largeur.

ses méthodes : Les méthodes sont des fonctions applicables à un objet.

Avec un objet Rectangle, il pourrait être utile d'implémenter diverses fonctions telles que :

- Calcul de son périmètre
- Calcul de la longueur de la diagonale,
- Calcul de la surface,
- etc.

3.2 - Classe

Une classe permet de définir les propriétés, les méthodes et les événements propres à un domaine.

En VB, le code correspondant est stocké dans un module d'un type particulier, le **module de classe** (extension **.cls**).

La valeur d'une propriété est gérée dans une variable privée du module de classe, et donc inaccessible hors du module.

En dehors du module, la propriété peut être :

- accessible en lecture, si une procédure **Public Property Get "NomPropriete"** a été définie,
- accessible en écriture si une procédure **Public Property Let "NomPropriete"** a été définie.

Ceci correspond, en 2 mots, au concept d'[encapsulation](#)

Voici ce que donne un exemple simple de la propriété **Hauteur** d'un objet **Rectangle**, accessible en lecture ET en écriture :

clsRectangle.cls (partie déclarative)

```
Private lHauteur As Long 'Hauteur actuelle
```

clsRectangle.cls (partie procédurale)

```
Public Property Let Hauteur(ByVal vData As Long)
    'Affectation d'une nouvelle valeur
    lHauteur = vData
End Property

Public Property Get Hauteur() As Long
    'Lecture de la valeur
    Hauteur = lHauteur
End Property
```

4 - Description de notre exemple, le module de classe clsRectangle

4.1 - Propriété

Les propriétés (**Property**) gérées sont les suivantes :

- **Hauteur** : Hauteur (en Twip) du rectangle,
- **Largeur** : Largeur (en Twip) du rectangle,
- **HauteurMin** : permet de mémoriser la valeur minimum qu'a prise la hauteur du rectangle au cours de sa vie,
- **LargeurMin** : permet de mémoriser la valeur minimum qu'a prise la largeur du rectangle au cours de sa vie,

Les 2 dernières propriétés (HauteurMin et LargeurMin) ne sont accessibles qu'en lecture, puisque leurs valeurs sont mises à jour lors du changement de valeur des 2 premières propriétés.

4.2 - Méthode

Notre module de classe possède une méthode (ici une **Function**, mais cela peut également être une **Sub**) :

getPerimetre : renvoie la valeur du périmètre du rectangle,

Il est bien entendu possible d'en implémenter d'autres, comme cela a été spécifié dans le paragraphe 3.1.

clsRectangle.cls (partie procédurale)

```
Public Function getPerimetre() As Long
    getPerimetre = 2 * (lHauteur + lLargeur)
End Function
```

4.3 - Événements

Deux événements (**Event**) sont également présents :

- **HauteurChangee** : déclenché si la hauteur du rectangle est modifiée,
- **LargeurChangee** : déclenché si la largeur du rectangle est modifiée.

a - Un événement, à quoi ça sert ?

De la même manière que pour n'importe quel contrôle de votre application, vous pouvez déclarer des événements

sur vos objets.

Les 2 événements décrits ci-dessus servent respectivement à intercepter la modification de la hauteur et de la largeur du rectangle.

b - Déclaration

La déclaration se fait dans le module de classe, après celle des variables et avant toute procédure :

- Un événement est déclaré dans un module de classe*
- Un événement est toujours Public,

```
clsRectangle.cls (partie déclarative)
Public Event HauteurChangee()
```

* : les Form sont aussi des objets qu'il est possible de "manipuler" en tant que tel.

Il est par exemple possible de déclarer un événement dans une Form si celle-ci est instanciée à l'aide d'une classe !

c - Restriction

Il est possible de gérer des paramètres pour vos événements, il faut donc savoir que :

- les arguments facultatifs (**Optional**) ne sont pas autorisés,
- les arguments de type **paramArray** ne sont pas autorisés.

d - Déclenchement

Lors de la modification de la hauteur ou de la largeur du rectangle, il faut donc indiquer que l'événement doit être déclenché.

Considérons la hauteur : Lors de la modification de celle-ci (Property **Let**), le déclenchement en question est fait grâce au mot clé **RaiseEvent**.

```
clsRectangle.cls (Public Property Let Hauteur)
'Déclenchement de l'événement
RaiseEvent HauteurChangee
```

Capture du déclenchement : dans la feuille de saisie dans laquelle l'objet rect1 a été déclaré, il faut :

- utiliser le mot clé **WithEvents** lors de la déclaration de l'instance de la classe,

frmSaisie (partie déclarative)

```
Dim WithEvents rect1 As clsRectangle
```

- implémenter la procédure suivante :

frmSaisie (partie déclarative)

```
Private Sub rect1_HauteurChangee()  
    . . .  
End Sub
```

Cette procédure sera alors exécutée en cas de modification de la propriété **Hauteur** de l'objet **rect1**

Le séparateur entre le nom de l'objet et celui de l'événement est un **underscore**, comme pour chaque événement se rapportant à un contrôle.

5 - Le programme

L'interface se compose de 2 feuilles, une pour la saisie des caractéristiques du rectangle, une pour l'affichage.

Voici à quoi ressemble l'interface de saisie :



Un objet est créé dès le chargement de la feuille.

Il est donc possible de saisir la hauteur et la largeur du rectangle via les 2 premières zones. La validité de la saisie est contrôlée dans l'événement `Validate` de chacune de ces zones, et les propriétés de l'objet sont mises à jour immédiatement.

La case à cocher "Prise en compte immédiate" permet de dessiner le rectangle sans cliquer sur le bouton `Afficher`. Cette fonctionnalité sert uniquement à illustrer la capture d'un événement.

Le bouton "Afficher" charge la feuille servant à l'affichage, qui se présente ainsi :



Le rectangle est affiché via un simple contrôle Shape.

Les valeurs des propriétés et le périmètre du rectangle sont indiqués en dessous.

6 - Pour aller plus loin

VB6 est néanmoins assez limité, car il ne prend pas en compte certaines notions avancées de la POO.

En particulier, l'**Héritage**, qui permet à partir d'une classe "généraliste" (que l'on appellera super-classe), de définir des classes plus "spécialisées".

Ces classes filles héritent automatiquement de tous les attributs et méthodes de la super-classe, et il est bien entendu possible de leur en ajouter de nouveaux.

Il est par contre possible d'utiliser le concept de **Polymorphisme**.

Ce concept permet de définir, pour diverses classes, des méthodes de même nom.

L'intérêt de ce mécanisme est de pouvoir faire appel ensuite à une méthode sans se "soucier" de l'origine de l'objet manipulé.

Exemple :

Class1.cls

```
Public Sub Hello()  
    MsgBox "message n° 1"  
End Sub
```

Class2.cls

```
Implements Class1  
  
Public Sub Hello()  
    MsgBox "message n° 2"  
End Sub
```

Form1.frm

```
Dim c1 As Class1, c2 As Class1  
  
Set c1 = New Class1  
Set c2 = New Class2  
  
c1.Hello    'affiche "message n° 1"  
c2.Hello    'affiche "message n° 2"
```

Implements sert à décrire ce qu'une classe permet de faire.

Dans l'exemple précédent, tout ce qui est public dans Class1 doit également être déclaré dans Class2.

Il est donc possible de créer des classes qui décriront chacune un comportement spécifique, et de gérer les objets correspondants via une interface commune.

Généralement on ne met aucun code dans Class1.

Cet exemple est tiré du [forum VB](#).

7 - Téléchargements

Le programme : [TUTO_VB6_POO.zip](#)

Le tutoriel au format PDF :