Vk

Zeste de savoir

Les structures et les classes en VBA

28 avril 2023

Table des matières

Introduction

La programmation orientée objet est un paradigme de programmation , c'est-à-dire une façon de concevoir les programmes, dans lequel des objets sont définis et interagissent entre eux.

Les objets ont un type et peuvent avoir des propriétés et des méthodes. Vous en avez déjà sans doute utilisé en VBA sans vous en rendre compte: les feuilles, les plages ou encore les tableaux (`ListObject`) en sont! En effet, une feuille possède notamment une propriété `Name` pour son nom et une méthode `Activate` pour la rendre active par exemple.

Au cours de ce billet, nous allons voir comment définir nos propres types de données en VBA.

C'est parti!

> *i* Pour ce billet, nous nous placerons dans l'environnement Microsoft Office.

1. Les structures

Les structures, aussi appelés types personnalisés, sont une façon simple d'ajouter de nouveaux types de données se composant de la même façon dans nos programmes.

Par exemple, nous pourrions avoir besoin d'un type *Client* qui aurait un nom, un prénom, une date de naissance et un nombre de commandes passées.

1.1. Définition

Pour définir un type personnalisé, nous utilisons l'instruction `Type`, en indiquant un niveau de visibilité, un nom et des variables internes.

```
Public Type TypeClient
    sNom As String
    sPrenom As String
    dDateNaissance As Date
    iNombreCommandes As Integer
End Type
```

> **!** En fonction de la visibilité (Private/Public), les structures ne peuvent pas être définies dans tous les types de fichier (UseForm, module de classe, feuille, ...)

1.2. Utilisation

Notre type étant défini, nous sommes maintenant en mesure de l'utiliser.

Tout d'abord, nous devons déclarer une variable du type personnalisé voulu de la sorte:

```
Dim tClient As TypeClient ' Déclaration
```

Ensuite, nous pouvons accéder à une valeur pour la lire ou la modifier, avec la notation `.`, comme ceci:

```
tClient.sNom = "Dupont"
Debug.Print (tClient.sNom) ' Dupont
```

Avec un peu d'imagination, nous pouvons construire des procédures ou des fonctions utilisant nos structures:

```
Public Sub InitialiseClient(ByRef tClient As TypeClient, ByVal
    sNom As String, ByVal sPrenom As String, ByVal dDateNaissance
    As Date, ByVal iNombreCommandes As Integer)
    With tClient
        .sNom = sNom
        .sPrenom = sPrenom
        .dDateNaissance = dDateNaissance
        .iNombreCommandes = iNombreCommandes
```

```vba
    End With
End Sub

Private Function ClientToString(ByRef tClient As TypeClient) As String
    Dim sResultat As String
    sResultat = "Client : " & tClient.sNom & " " & tClient.sPrenom & " " & tClient.dDateNaissance
    ClientToString = sResultat
End Function

Dim tClient2 As TypeClient
InitialiseClient tClient2, "Dupont", "Albert", "15/05/1965", 8
Debug.Print (ClientToString(tClient2)) ' Client : Dupont Albert 15/05/1965
```

i Il est tout à fait possible d'imbriquer des types. Par exemple, un type *Client* pourrait faire appel à une structure *Adresse* pour son adresse de livraison.

Au cours de cette section, nous avons comment nous servir des types personnalisés en VBA.

2. Les classes

Les classes sont un moyen plus complet de définir des types de données.

Il est alors possible d'encapsuler (c'est-à-dire donner une visibilité) différemment les éléments de la classe ou encore d'appliquer directement des méthodes à un objet instancié.

2.1. Création

Pour créer une classe, nous devons ajouter un module de classe au projet. Cela se fait en allant dans le menu "Insertion" et en cliquant sur "Module de classe" ou depuis la barre des raccourcis comme illustré ci-dessous:

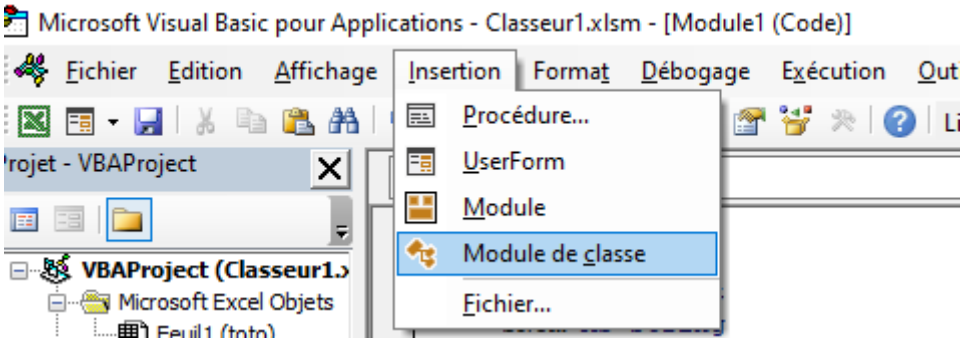

FIGURE 2.1. – Insertion module de classe

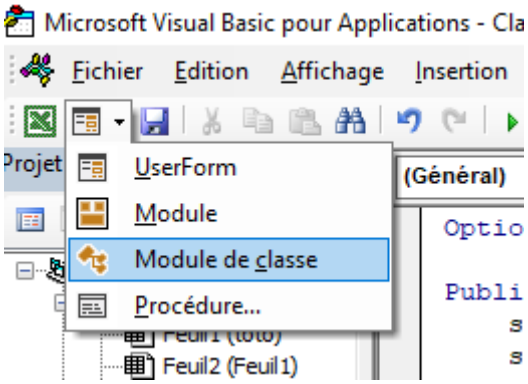

FIGURE 2.2. – Raccourci insertion module de classe

Nous pouvons changer le nom de la classe et son accessibilité depuis les propriétés (raccourci F4 pour afficher la fenêtre "Propriétés").

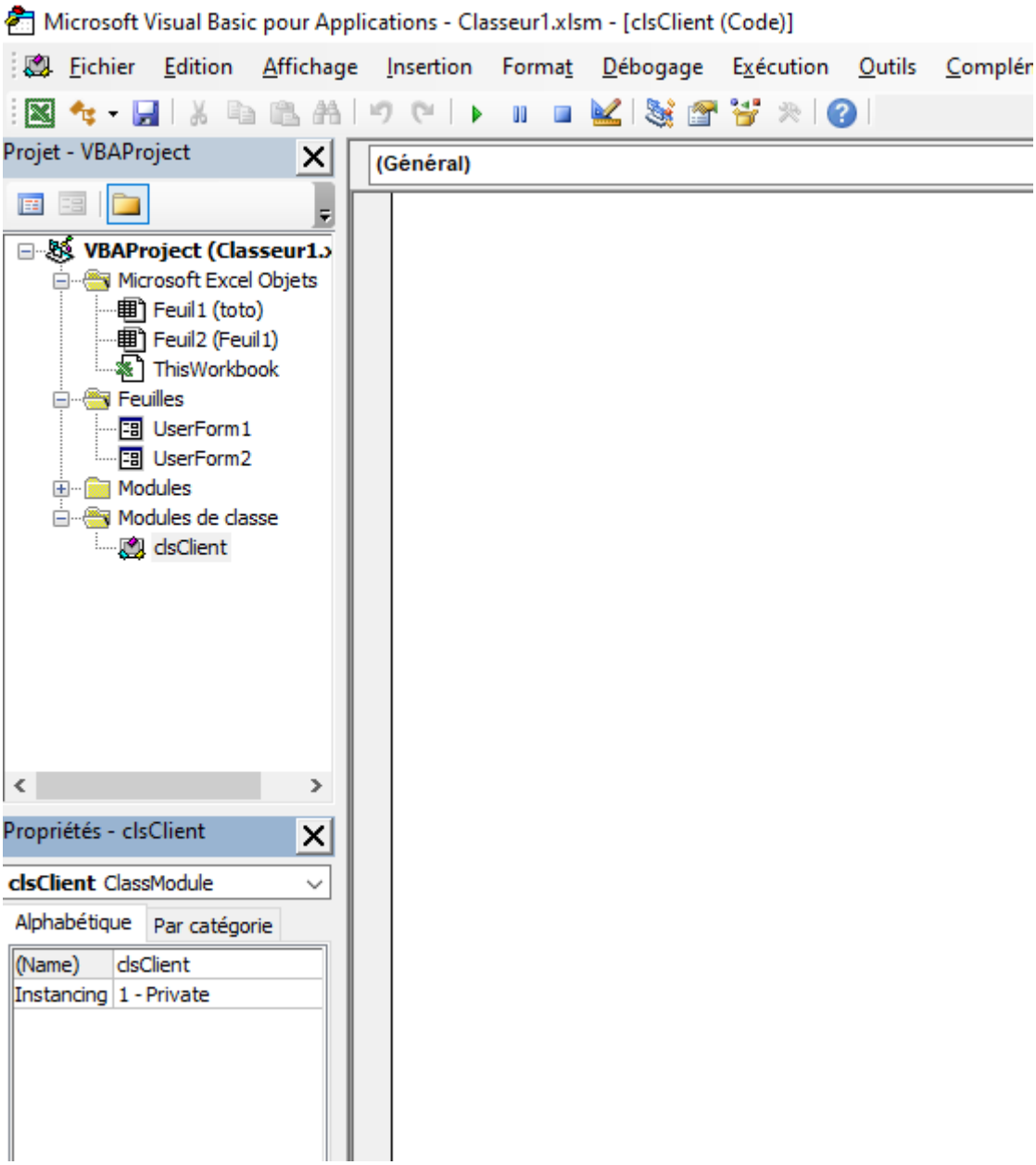

FIGURE 2.3. – Propriétés module de classe

Pour les exemples, nous utiliserons cette classe dans un module standard quelconque.

2.2. Instanciation

On dit que l'on instancie une classe lorsque l'on crée un objet à partir de celle-ci. Pour cela, nous faisons appel à l'opérateur `New`.

```
1 Dim oClient1 As clsClient ' Déclaration
2 Set oClient1 = New clsClient ' Instanciation
```

À noter que nous pouvons raccourcir la déclaration et l'instanciation à une ligne de cette manière:

```
1 Dim oClient2 As New clsClient ' Déclaration avec instanciation
```

2.3. Propriétés

Notre classe est pour le moment assez vide. Nous allons commencer par définir des propriétés qui seront l'équivalent des variables de notre structure *Client* de la section précédente.

Il y a deux façons de définir des propriétés en VBA.

2.3.1. Propriétés en tant que variables membres

La première consiste à définir des propriétés en tant que variables membres.

```
1 Public sNom As String
2 Public sPrenom As String
3 Public dDateNaissance As Date
4 Private iNombreCommandes As Integer ' Inaccessible depuis
      l'extérieur
```

Pour accéder aux valeurs de l'objet, nous utilisons la même notation que pour les structures:

```
1 oClient1.sNom = "Dupont"
2 Debug.Print (oClient1.sNom) ' Dupont
3 'oClient1.iNombreCommandes = 1 ' Erreur de compilation : Membre de
      méthode ou de données introuvable
```

Comme vous pouvez le voir, la propriété `iNombreCommandes` n'est pas atteignable depuis l'extérieur, c'est normal vu que nous lui avons donné une visibilité privée!

2.3.2. Propriétés en tant que procédures Property

La seconde approche consiste à définir des propriétés en tant que procédures `Property`.

Cette seconde approche offre davantage de flexibilité pour choisir l'interface de notre classe. Contrairement aux propriétés en tant que variables membres pour lesquelles c'est tout ou rien (accessible ou non), nous pouvons choisir ce qui sera accessible en lecture ou en écriture depuis l'extérieur.

De plus cette approche permet aussi de créer des propriétés dérivées d'autres plutôt que de passer par des fonctions pour faire cela.

Cette relation entre propriété membre et propriété en tant que procédure `Property` peut être représentée de la sorte:

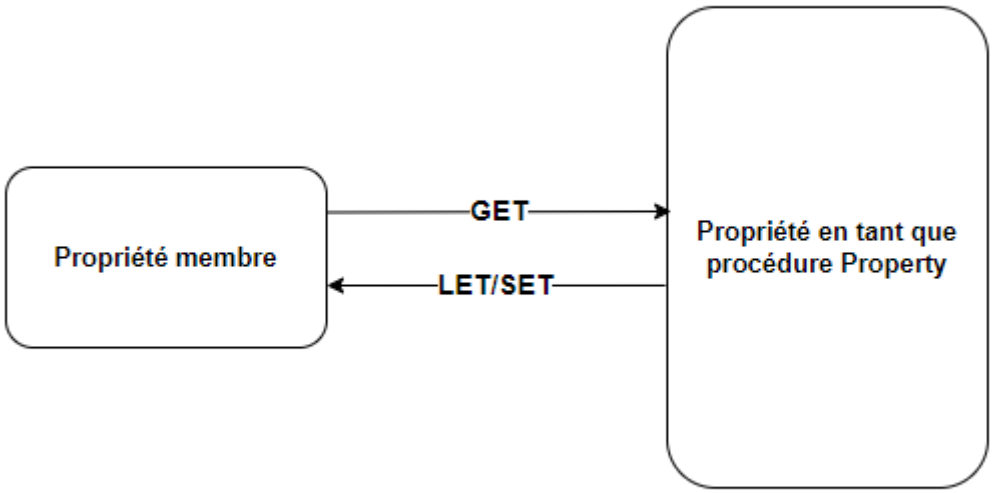

FIGURE 2.4. – Relation propriété membre et propriété procédure Property

Si la propriété membre est définie avec une visibilité publique, il n'est pas logique de définir des propriétés `Property` pour celle-ci, car nous ne pourrons pas restreindre son interface. En effet, nous pourrons toujours choisir de passer par cette propriété membre plutôt que par les propriétés de lecture ou d'écriture.
En revanche si la propriété membre est privée, nous pouvons choisir d'ouvrir son interface en lecture ou en écriture.

Pour l'exemple qui va suivre, modifions nos propriétés et ajoutons une classe `clsAdresse` publique.

```
1 Private mNom As String
2 Private mPrenom As String
3 Public dDateNaissance As Date
4 Private mAdresseLivraison As clsAdresse
5 Private mNombreCommandes As Integer
```

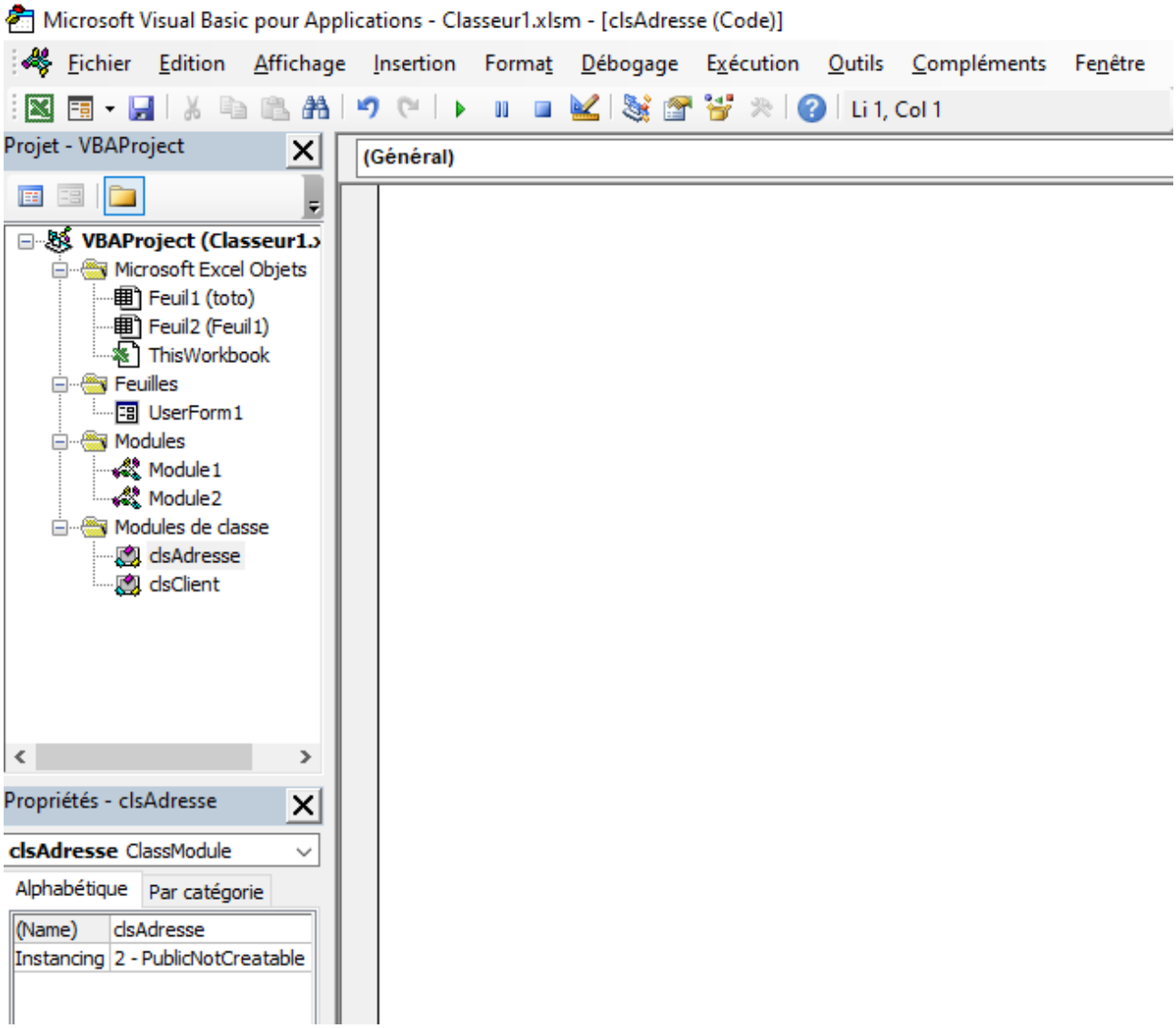

FIGURE 2.5. – Classe clsAdresse

2.3.2.1. Let/Set La `Property Let` permet d'assigner une valeur simple tandis que la `Property Set` permet d'assigner une valeur de type objet. Vous avez sans doute déjà l'habitude d'utiliser `Set` tandis que `Let` n'est pas explicitement requis.

```
1  ' Exemples ouverture interface en écriture
2
3  Public Property Let sNom(ByVal sNom As String)
4      mNom = sNom
5  End Property
6
7  Public Property Let sPrenom(ByVal sPrenom As String)
8      mPrenom = sPrenom
9  End Property
10
```

```
Public Property Set oAdresseLivraison(ByVal oAdresseLivraison As
    clsAdresse)
    Set mAdresseLivraison = oAdresseLivraison
End Property
```

2.3.2.2. Get La `Property Get` permet d'obtenir une valeur quelque soit sa nature.

```
' Exemple ouverture interface en lecture

Public Property Get iNombreCommandes() As Integer
    iNombreCommandes = mNombreCommandes
End Property

' Exemples propriétés dérivées d'autres

Public Property Get sNomComplet() As String
    sNomComplet = mPrenom & " " & UCase(mNom)
End Property

Public Property Get sTypeClient() As String
    Dim sResult As String

    If iNombreCommandes = 0 Then
        sResult = "Client timide"
    ElseIf iNombreCommandes < 10 Then
        sResult = "Client néophyte"
    ElseIf iNombreCommandes < 20 Then
        sResult = "Client régulier"
    Else
        sResult = "Super client"
    End If

    sTypeClient = sResult
End Property
```

Nous pouvons alors utiliser notre classe ainsi:

```
Dim oClient1 As clsClient ' Déclaration
Set oClient1 = New clsClient ' Instanciation

oClient1.sNom = "Dupont"
oClient1.sPrenom = "Albert"
oClient1.dDateNaissance = "15/05/1965"
Set oClient1.oAdresseLivraison = New clsAdresse
Debug.Print (oClient1.sNomComplet) ' Albert DUPONT
```

```
Debug.Print (oClient1.dDateNaissance) ' 15/05/1965
Debug.Print (oClient1.iNombreCommandes) ' 0
Debug.Print (oClient1.sTypeClient) ' Client timide
```

2.4. Événements

En VBA, les événements nous permettent de réaliser des actions en réponse à quelque chose qui se produit (l'ouverture du classeur, l'activation d'une feuille, etc…).

FIGURE 2.6. – Exemples événements feuille

2.4.1. Constructeur

Lorsqu'un objet est créé, il est d'abord construit puis initialisé.

L'événement `Class_Initialize` nous permet de définir un constructeur par défaut afin d'initialiser l'objet après sa création.

```
Private cCommandes As Collection ' Propriété ajoutée

Private Sub Class_Initialize()
    Debug.Print ("_____________________________")
    Debug.Print ("Objet créé")
    Set cCommandes = New Collection
End Sub
```

2.4.2. Destructeur

Avant qu'un objet ne soit détruit, il peut être important de réaliser certaines opérations (sauvegardes, nettoyage de la mémoire, …).

L'événement `Class_Terminate` nous permet de capturer l'objet avant sa destruction dans ce but.

```vba
Private Sub Class_Terminate()
    Set cCommandes = Nothing ' Traitements...
    Debug.Print ("Objet " & sNomComplet & " détruit")
    Debug.Print ("_____________________________")
End Sub
```

2.4.3. Événement personnalisé

En plus de ces deux événements de base, nous pouvons ajouter nos propres événements. Ce sujet ne sera pas abordé dans ce billet, sachez simplement que c'est possible.

2.5. Méthodes

Les méthodes sont des fonctions ou des procédures au sein de la classe. Comme pour le reste, nous pouvons choisir leur encapsulation.

```vba
Private Function NombreCommandes() As Integer
    ' Remplace les propriétés mNombreCommandes et iNombreCommandes
      définies auparavant
    NombreCommandes = cCommandes.Count
End Function

Public Sub AjouteCommande(ByVal sCommande As String)
    cCommandes.Add sCommande
End Sub
```

Dans l'exemple ci-dessus nous définissons une méthode `NombreCommandes` et une `AjouteCommande` que nous utilisons de la sorte:

```vba
Dim i As Integer
For i = 1 To 20
    oClient1.AjouteCommande Format(i, "00000000")
Next i
```

Voici le code final:

Classe clsClient

Contenu masqué n°1

Module

Contenu masqué n°2

Pendant cette section, nous avons vu comment utiliser les classes en VBA.

Conclusion

C'est déjà la fin de ce billet.

Au cours de celui-ci, nous avons vu comment définir nos propres types de données en VBA avec les structures d'abord et les classes ensuite. À noter que le système de POO en VBA est moins poussé que dans d'autres langages tels que Java.

Pour aller plus loin, il est souvent pratique de regrouper des objets de même type au sein de structures de données telles que les collections ou encore les dictionnaires .

À bientôt!

Quelques ressources:

— La documentation concernant l'instruction Type
— Ce tutoriel sur les classes
— Ce tutoriel sur les types et les classes

Contenu masqué

Contenu masqué n°1

```
Private mNom As String
Private mPrenom As String
Public dDateNaissance As Date
Private mAdresseLivraison As clsAdresse
Private cCommandes As Collection

Public Property Let sNom(ByVal sNom As String)
    mNom = sNom
End Property

Public Property Let sPrenom(ByVal sPrenom As String)
    mPrenom = sPrenom
End Property

Public Property Get sNomComplet() As String
    sNomComplet = mPrenom & " " & UCase(mNom)
End Property
```

Contenu masqué

```
18
19  Public Property Get sTypeClient() As String
20      Dim sResult As String
21
22      Dim iNombreCommandes As Integer
23      iNombreCommandes = NombreCommandes()
24
25      If iNombreCommandes = 0 Then
26          sResult = "Client timide"
27      ElseIf iNombreCommandes < 10 Then
28          sResult = "Client néophyte"
29      ElseIf iNombreCommandes < 20 Then
30          sResult = "Client régulier"
31      Else
32          sResult = "Super client"
33      End If
34
35      sTypeClient = sResult
36  End Property
37
38  Public Property Set oAdresseLivraison(ByVal oAdresseLivraison As
        clsAdresse)
39      Set mAdresseLivraison = oAdresseLivraison
40  End Property
41
42  Private Function NombreCommandes() As Integer
43      NombreCommandes = cCommandes.Count
44  End Function
45
46  Public Sub AjouteCommande(ByVal sCommande As String)
47      cCommandes.Add sCommande
48  End Sub
49
50  Private Sub Class_Initialize()
51      Debug.Print ("_____________________________")
52      Debug.Print ("Objet créé")
53      Set cCommandes = New Collection
54  End Sub
55
56  Private Sub Class_Terminate()
57      Set cCommandes = Nothing ' Traitements...
58      Debug.Print ("Objet " & sNomComplet & " détruit")
59      Debug.Print ("_____________________________")
60  End Sub
```

Retourner au texte.

Contenu masqué n°2

```vb
Private Sub CreerClient()
    Dim oClient1 As clsClient ' Déclaration
    Set oClient1 = New clsClient ' Instanciation

    oClient1.sNom = "Dupont"
    oClient1.sPrenom = "Albert"
    oClient1.dDateNaissance = "15/05/1965"
    Set oClient1.oAdresseLivraison = New clsAdresse

    Debug.Print (oClient1.sNomComplet) ' Albert DUPONT
    Debug.Print (oClient1.dDateNaissance) ' 15/05/1965
    Debug.Print (oClient1.sTypeClient) ' Client timide

    Dim i As Integer
    For i = 1 To 20
        oClient1.AjouteCommande Format(i, "00000000")
    Next i

    Debug.Print (oClient1.sTypeClient) ' Super client
End Sub
```

Retourner au texte.

Liste des abréviations

POO Programmation Orientée Objet. 12

VBA Visual Basic for Applications. 1