6

Variables et constantes

Au sommaire de ce chapitre

- Déclarer une variable
- Types de données des variables
- Portée et durée de vie des variables
- Traitement interapplications à l'aide de variables objet

Les variables sont un élément essentiel de la programmation. Elles permettent de stock er les informations de v otre choix à tout moment de l'e xécution d'un programme et de les réexploiter à n'importe quel autre moment. Les v ariables sont un lieu de stockage que vous déterminez. Vous pouvez, par e xemple, stocker dans une v ariable une v aleur représentant le nombre de classeurs ouv erts, le nom du fichie , la v aleur ou l'adresse d'une cellule, les informations entrées par l'utilisateur dans une feuille VBA, etc.

Déclarer une variable

Pour créer une v ariable, v ous de vez la déclarer , c'est-à-dire lui af fecter un nom qu'il suffira par la suite de réutiliser pour xploiter la valeur qui y est stockée. La déclaration de variables dans un programme VBA peut être implicite ou e xplicite. Autrement dit, les programmes VBA peuv ent reconnaître l'utilisation d'une nouv elle v ariable sans que celle-ci soit préalablement créée dans une instruction de déclaration. Vous pouvez aussi paramétrer Visual Basic Editor afin d' xiger la déclaration e xplicite des v ariables avant leur utilisation.

Déclaration implicite

Si la déclaration explicite des variables n'est pas requise, le simple f ait de faire apparaître un mot non reconnu par le programme dans une instruction d'af fectation suffira pour qu ce nom soit considéré comme une variable de type `Variant` par le programme — les types de variables sont présentés plus loin dans ce chapitre. C'est le cas dans l'exemple suivant :

```
Sub DéclarImpliciteDeVariables()
  mavar = Range("D7").Value
  MsgBox "La somme totale des transactions est " & mavar, _
          vbInformation + vbOKOnly
  Instructions
End Sub
```

Le mot `MaVar` apparaît pour la première fois dans une instruction d'af fectation valide. La variable `MaVar` est donc créée et affectée à la valeur de la cellule D7. Elle est ensuite utilisée pour afficher un message à l'attention de l'utilisateur (oir Figure 6.1). L'opérateur de concaténation `&` est utilisé pour f aire apparaître la v aleur de la v ariable au cœur d'une chaîne de caractères définie

Figure 6.1
Les variables peuvent être utilisées dans des chaînes à l'aide de l'opérateur de concaténation.

Vous pouvez attribuer le nom de votre choix à une variable. Un nom de variable doit cependant respecter les règles suivantes :

- Il doit commencer par une lettre.
- Il ne peut contenir plus de 255 caractères.
- Le point, le point d'exclamation, les espaces et les caractères @, &, $ et # ne sont pas autorisés.
- Ce nom ne doit pas être un *mot réservé*, c'est-à-dire un mot reconnu comme un élément du langage Visual Basic (nom de fonction, d'objets, de propriété, d'argument nommé, etc.).

Déclaration explicite

Si les variables peuvent être déclarées implicitement, il est fortement recommandé de déclarer explicitement les variables avant de les utiliser.

Forcer la déclaration des variables avec *Option Explicit*

Visual Basic Editor permet de forcer la déclaration des variables avant leur utilisation, grâce à l'instruction `Option Explicit`. L'utilisation de cette option permet d'éviter les risques d'erreurs liées à une faute de frappe dans le nom d'une variable. Considérez la procédure suivante :

```
Sub MacroErreur()
  MaVariable = Workbooks.Count
    While MaVarable < 10
      Workbooks.Add
      MaVariable = MaVariable + 1
    Wend
End Sub
```

Cette procédure a pour but d'ouvrir des classeurs en boucle, jusqu'à ce que dix classeurs soient ouverts au total. Le nombre de classeurs ouverts (`Workbooks.Count`) est affecté à la variable `MaVariable`. Une instruction `While...Wend` — que vous découvrirez dans le prochain chapitre — est utilisée pour créer un nouveau classeur (`Workbooks.Add`) et ajouter 1 à `MaVariable` tant que la valeur de celle-ci est inférieure à 10. Cependant, le nom de la variable a été incorrectement saisi dans l'instruction suivante :

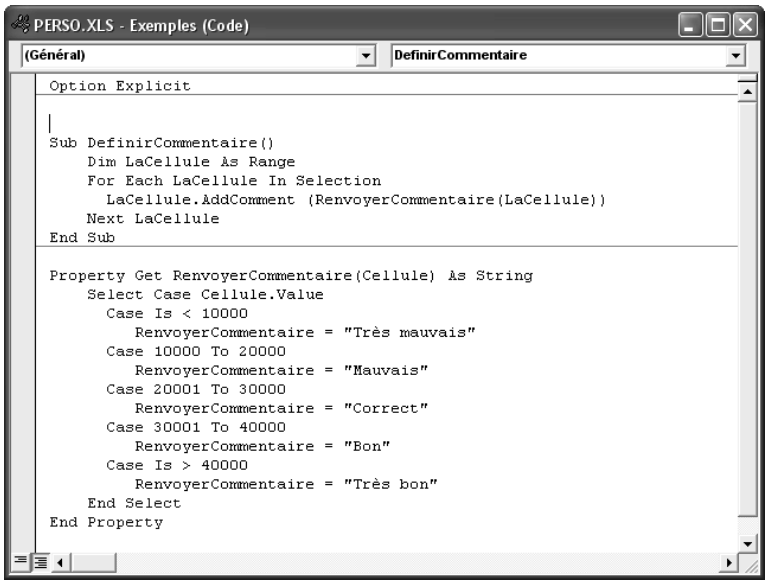
```
While MaVarable < 10
```

"`MaVarable`" n'existant pas, cette variable est créée et aucune valeur ne lui est affectée. La procédure ouvre donc des documents et incrémente `MaVariable` de 1 à l'infini, sans que la condition `MaVarable < 10` ne soit jamais respectée.

Pour interrompre une macro s'exécutant à l'infini, tapez la combinaison clavier Ctrl + Pause.

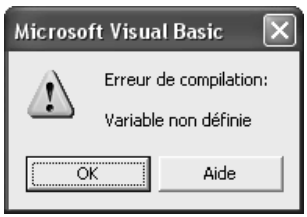
Pour éviter ce type d'erreur, forcez la déclaration explicite des variables. Pour cela, placez-vous dans la section Déclarations de la fenêtre Code du module et saisissez-y l'instruction `Option Explicit` (voir Figure 6.2).

Figure 6.2
L'instruction Option Explicit doit se trouver dans la section Déclarations de la fenêtre Code.



Dorénavant, l'apparition de noms de variables non préalablement déclarées à l'aide de l'instruction `Dim` dans une procédure générera une erreur. Dans notre exemple, l'utilisation de l'instruction `Option Explicit` entraînera le message d'erreur présenté à la Figure 6.3 à l'exécution du programme.

Figure 6.3
L'option Option Explicit permet de détecter immédiatement les erreurs de saisie dans les noms de variables.



Vous pouvez aussi paramétrer Visual Basic Editor pour que l'instruction `Option Explicit` soit systématiquement placée dans la section de Déclarations de tout nouveau module.

1. Sélectionnez la commande Options du menu Outils et placez-vous sur l'onglet Editeur.
2. Cochez la case Déclaration explicite des variables, puis cliquez sur OK.

   L'instruction `Option Explicit` sera désormais automatiquement placée dans la section appropriée de la fenêtre Code de chaque nouveau module.

Déclarer une variable avec *Dim*

La déclaration de variables se fait à l'aide de l'instruction `Dim`, selon la syntaxe suivante :

```
Dim NomVariable As Type
```

où *`NomVariable`* est le nom de la variable, qui sera utilisé par la suite pour y faire référence, et *`Type`* le type de données qu'accepte la variable — présentés dans la section suivante. L'argument *`Type`* est facultatif, mais la déclaration d'un type de variable peut vous faire économiser de l'espace mémoire et améliorer ainsi les performances de votre programme. Lors de la déclaration de variables, une variable de type `String` prend pour valeur une chaîne vide, une variable numérique prend la valeur 0.

Dans l'exemple suivant, les variables Message, Boutons et Titre sont déclarées à l'aide de l'instruction `Dim`, des valeurs leur sont ensuite affectées, puis sont utilisées comme arguments de `MsgBox` afin d'afficher la boîte de dialogue de la Figure 6.4

```
Sub UtiliserDim()
  Dim Message As String
  Dim Boutons As Single
  Dim Titre As String
  Message = "La procédure est terminée."
  Boutons = vbOKOnly + vbInformation
  Titre = "C'est fini"
  MsgBox Message, Boutons, Titre
End Sub
```

Figure 6.4
Des variables peuvent être utilisées comme arguments d'une fonction.

Plusieurs variables peuvent être déclarées dans une même instruction `Dim`, selon la syntaxe suivante :

```
Dim NomVar1 As Type, NomVar2 As Type, ..., NomVarn As Type
```

où *NomVar1* est le nom de la première variable déclarée, et ainsi de suite. Les trois instructions de déclaration de l'exemple suivant peuvent ainsi être ramenées à une seule instruction de déclaration :

```
Dim Message As String, Boutons As Single, Titre As String
```

Gardez à l'esprit que pour affecter un type aux variables d'une telle instruction, celui-ci doit être mentionné pour chacune des variables déclarées. L'instruction suivante déclare une variable Message de type `Variant` et une variable Titre de type `String` :

```
Dim Message, Titre As String
```

Types de données des variables

Lorsque vous créez une variable, vous pouvez déterminer son *type*. Le type d'une variable correspond au type de l'information qui peut y être stockée. Il peut s'agir d'une valeur numérique (un nombre ou une expression renvoyant un nombre), d'une chaîne de caractères, d'une date, etc. Les valeurs affectées aux variables dans des instructions d'affectation doivent être compatibles avec le type de la variable. Par exemple, déclarer une variable de type numérique et lui affecter par la suite une chaîne de caractères générera une erreur.

Vous pouvez aussi choisir de ne pas déterminer le type d'une variable. Elle sera alors de type *Variant*, et vous pourrez y stocker tous les types de données. Cette section présente les différents types de variables.

Chaînes de caractères

Les variables de type *String* — encore appelées "variables de chaîne" — permettent le stockage de données reconnues comme une chaîne de caractères. Pour déclarer une variable de type `String`, utilisez la syntaxe suivante :

```
Dim NomVariable As String
```

Une variable de chaîne peut être affectée à toute expression renvoyant une valeur de type chaîne (chaîne, propriété, fonction, etc.). Pour affecter une chaîne de caractères à une variable de type `String`, placez celle-ci entre guillemets. Si vous souhaitez placer des guillemets dans une chaîne de caractères (ou tout autre caractère), utilisez conjointement l'opérateur de concaténation `&` et la fonction `Chr` selon la syntaxe suivante :

```
"Chaîne de car." & Chr(codeANSI) & "Chaîne de car."
```

où *codeANSI* est le code ANSI du caractère que vous souhaitez insérer.

L'opérateur + peut aussi être utilisé pour concaténer des chaînes de caractères. Préférez cependant l'opérateur &, afin de distinguer aisément la concaténation de chaînes d'additions de valeurs numériques qui, elles, requièrent l'opérateur +.

Les instructions d'affectation suivantes sont toutes valides :

- `Prénom = "Luc"`
- `NomFichier = ActiveWorkbook.Name`
- `Message = "Le nom du classeur actif est " & Chr(34) & ActiveWorkbook.Name & Chr(34) & "."`

La première instruction affecte une chaîne définie à la variable Prénom. La seconde instruction affecte la valeur de la propriété `Name` du classeur actif à la variable NomFichier. La troisième conjugue l'affectation de chaînes définies dans le texte, renvoyées par une fonction et renvoyées par une propriété, en concaténant ces différentes chaînes à l'aide de l'opérateur `&`.

La macro suivante affiche la boîte de dialogue représentée à la Figure 6.5

```
Sub ConcatenerLesChaines()
  Dim Message As String, Boutons As Single, Titre As String
  Message = "Le nom du document actif est " & Chr(34) & _
            ActiveWorkbook.Name & Chr(34) & "."
  Boutons = vbInformation + vbOKOnly
  Titre = "Concaténation de chaînes"
  MsgBox Message, Boutons, Titre
End Sub
```

Figure 6.5
Le message affiché par la fonction MsgBox est toujours une chaîne de caractères.

Les variables de chaîne définies précédemment sont dites de *longueur variable* et peuvent être constituées d'environ deux milliards de caractères. Vous pouvez cependant déclarer des variables de chaîne de *longueur fixe*, autorisant de 1 à 65 400 caractères, selon la syntaxe suivante :

```
Dim NomVariable As String * longueur
```

où *longueur* est le nombre de caractères que peut recevoir la variable. La déclaration de variables de chaîne de longueur fixe peut générer une économie de mémoire — et donc un gain de performance — pour votre programme, mais doit être utilisée prudemment.

En effet, si la chaîne affectée à une variable de longueur fixe dépasse la capacité de cette dernière, la chaîne sera purement et simplement rognée. Remplacez l'instruction de déclaration des variables de l'exemple précédent par celle-ci :

```
Dim Message As String * 15, Boutons As Single, Titre As String * 5
```

Vous obtenez la boîte de dialogue présentée à la Figure 6.6.

Figure 6.6
Utilisez les variables de chaîne de longueur fixe avec prudence.

Valeurs numériques

Les variables numériques permettent le stockage de valeurs exploitables en tant que telles, c'est-à-dire sur lesquelles vous pouvez effectuer des opérations arithmétiques. Il existe plusieurs types de variables numériques. Elles se distinguent par l'échelle des valeurs qu'elles acceptent et par la place qu'elles occupent en mémoire. Le Tableau 6.1 présente les différents types de variables numériques :

Tableau 6.1 : Types de données numériques

Types de données	*Valeurs acceptées*	*Mémoire occupée*
`Byte` (octet)	Nombre entier, compris entre 0 et 255	1 octet
`Integer` (entier)	Nombre entier compris entre –32 768 et 32 767	2 octets
`Long` (entier long)	Nombre entier compris entre –2 147 483 648 et 2 147 483 647	4 octets
`Single` (simple précision)	Nombre à virgule flottante compris entre –1,401298E-45 et –3,402823E38 pour les nombres négatifs, entre 1,401298E-45 et 3,402823E38 pour les nombres positifs	4 octets
`Double` (double précision)	Nombre à virgule flottante compris entre –1,79769373486232E308 et –4,94065645841247E-324 pour les nombres négatifs, entre 4,94065645841247E-324 et 1,79769373486232E308 pour les nombres positifs	8 octets
`Currency` (monétaire)	Nombre à virgule fixe, avec quinze chiffres pour la partie entière et quatre chiffres pour la partie décimale, compris entre –922 337 203 685 477,5808 et 922 337 203 685 477,5807	8 octets

Si votre programme exploite beaucoup de variables, l'affectation du type approprié — celui qui exploite le moins d'espace mémoire — aux différentes variables en améliorera les performances. Veillez cependant à ce que les valeurs qu'une variable est susceptible de prendre soient toujours couvertes par le type de données acceptées par la variable. Par exemple, si une valeur supérieure à 255 ou inférieure à 0 est affectée à une variable de type `Byte`, une erreur sera générée (voir Figure 6.7).

Figure 6.7
Le type de données acceptées par une variable doit couvrir l'ensemble des valeurs possibles lors de l'exécution du programme.

Utilisez le point comme séparateur décimal dans le code VBA. L'utilisation de la virgule génère une erreur.

Au même titre que les variables de type `String`, une variable numérique peut être concaténée avec une chaîne de caractères à l'aide de l'opérateur `&`. Une variable numérique peut aussi être affectée à toute expression renvoyant une valeur numérique (chaîne, propriété, fonction, etc.). Vous pouvez aussi effectuer des opérations arithmétiques sur les variables numériques en utilisant les opérateurs arithmétiques présentés dans le Tableau 6.2.

Tableau 6.2 : Les opérateurs arithmétiques

Opérateur	*Description*
+	Addition.
–	Soustraction.
*	Multiplication.
/	Division.
\	Division. Seule la partie entière du résultat de la division est renvoyée (l'opération 18 \ 5 retournera la valeur 3).
^	Elévation à la puissance (2 ^ 4 renvoie 16).

A condition que le type de la variable numérique `MaVar` couvre les valeurs qui lui sont affectées, les instructions d'affectation suivantes sont toutes valides :

- `MaValeur = 58`
- `NbreClasseur = Workbooks.Count`
- `Range("D5").Value = (Range("D3").Value + Range("D4").Value) / 2`
- `SurfaceCercle = (varRayon ^ 2) * 3.14`

La première instruction affecte une valeur définie à la variable MaValeur. La deuxième instruction affecte la valeur de la propriété `Count` de l'objet (la collection) `Workbooks` (le nombre de classeurs ouverts). La troisième utilise l'opérateur arithmétique `+` pour additionner les valeurs des cellules D3 et D4, et l'opérateur arithmétique `/` pour diviser la valeur obtenue par deux. La dernière instruction élève la variable `varRayon` précédemment défini au carré à l'aide de l'opérateur `^`, puis multiplie cette valeur par 2 à l'aide de l'opérateur `*`.

Par contre, l'instruction suivante affecte une valeur de type chaîne à la variable numérique `Mavar` et génère une erreur (voir Figure 6.8).

```
Sub ErreurAffectation()
  Dim Mavar As Byte
  MaVar = ActiveWorkbook.Name
End Sub
```

Figure 6.8
Une chaîne de caractères ne peut être affectée à une variable numérique.

Rappelez-vous que les constantes sont en réalité des valeurs numériques. Une constante peut donc être affectée à une variable numérique et utilisée dans une expression arithmétique. Veillez à ne pas utiliser l'opérateur `&`, réservé à la concaténation de chaînes, dans une instruction d'affectation de valeur à une variable numérique. Par exemple, l'instruction :

```
MsgBox "Le message de la bdg", vbOKOnly + vbInformation, "Titre"
```

est valide, tandis que l'instruction :

```
MsgBox "Le message de la bdg", vbOKOnly & vbInformation, "Titre"
```

générera un message d'erreur, car l'opérateur `&` est utilisé pour additionner les valeurs affectées aux constantes `vbOKOnly` et `vbInformation`.

VBA intègre de nombreuses fonctions permettant de manipuler les valeurs numériques. Les deux fonctions suivantes en sont des exemples :

- **`ABS(nombre)`.** Renvoie la valeur absolue du *`nombre`* spécifié entre parenthèses
- **`Int(nombre)`.** Renvoie la partie entière du *`nombre`* spécifié entre parenthèses

La liste des fonctions VBA est consultable dans l'aide en ligne. Choisissez la rubrique Manuel de référence du langage. Vous y trouverez une rubrique Fonctions répertoriant les fonctions par ordre alphabétique. Vous pouvez également y choisir la rubrique Liste et index. Vous y trouverez des rubriques thématiques, telles que Résumé des mots clés financier, ou Résumé des mots clés mathématiques.

Valeurs booléennes

Les variables de type `Boolean` servent à stocker le résultat d'expressions logiques. Une variable de type `Boolean` peut renvoyer la valeur `True` ou `False` et occupe deux octets en mémoire. Une variable booléenne peut recevoir la valeur retournée par une expression renvoyant une valeur de type `Boolean`, mais peut aussi être affectée à une valeur numérique. Dans ce cas, si la valeur numérique est égale à zéro, la variable prendra la valeur `False`, `True` dans le cas contraire. Une variable booléenne utilisée comme chaîne de caractères renvoie le mot Vrai ou Faux.

Dans l'exemple suivant, la variable booléenne ClasseurSauvegardé prend la valeur `False` si le classeur actif a subi des modifications depuis le dernier enregistrement ; `True` dans le cas contraire. Une boîte de dialogue affiche ensuite le message Vrai ou Faux, en fonction de la valeur de ClasseurSauvegardé.

```
Sub ClasseurSauvegardéOuNon()
  Dim ClasseurSauvegardé As Boolean
  ClasseurSauvegardé = ActiveWorkbook.Saved
  MsgBox ClasseurSauvegardé
End Sub
```

Dates

Les dates comprises entre le 1er janvier 100 et le 31 décembre 9999 peuvent être affectées à des variables de type `Date`. Ces variables sont stockées sous la forme de nombres à virgule flottante et occupent huit octets en mémoire

La partie entière du nombre représente le jour. 0 correspond au 30 décembre 1899, 1 au 31 décembre 1899, etc. Les valeurs négatives représentent des dates antérieures au 30 décembre 1899. La partie décimale du nombre représente l'heure. 0.5 correspond à 12 heures, 0.75, à 18 heures, etc. Ainsi, 36526.00001 correspond au 1er janvier 2000, à 00:00:01.

Visual Basic intègre des fonctions permettant de manipuler les dates. Par exemple, les fonctions `Date`, `Time` et `Now` renvoient respectivement la date courante, l'heure courante et la date courante suivie l'heure courante. La procédure suivante affiche la boîte de dialogue de la Figure 6.9 :

```
Sub DateEtHeure()
  Dim LaDate As Date, LHeure As Date
  LaDate = Date
  LHeure = Time
  MsgBox "Nous sommes le " & LaDate & ", il est " & LHeure & ".", _
         vbOKOnly + vbInformation, "Fonctions Date et Time"
End Sub
```

Figure 6.9
Visual Basic intègre des fonctions permettant de manipuler les dates et les heures.

Pour plus d'informations sur les fonctions de date et d'heure, reportez-vous à l'aide Visual Basic, en choisissant Résumé des mots clés de date et d'heure de la rubrique Index/Listes du Manuel de référence du langage. Vous pouvez également vous reporter au Manuel de référence pour Microsoft Excel, disponible dans le sommaire de l'aide.

L'installation par défaut d'Excel n'installe pas l'aide en ligne de VBA. Si vous n'avez pas effectué une installation personnalisée, vous devrez lancer de nouveau l'installation et installer l'aide en ligne de VBA.

Type *Variant*

Une variable, ou une variable de type `Variant`, accepte des valeurs de tout type et peut être automatiquement convertie d'un type à l'autre, tant que sa valeur est compatible avec le type vers lequel s'opère la conversion. Autrement dit, une variable de type `Variant` peut être initialement une chaîne de caractères, et être exploitée par la suite en tant que valeur numérique. Si les données qui lui sont affectées sont assimilables à une valeur numérique, la conversion se fera automatiquement, lorsque l'instruction assimilable à une opération numérique sera exécutée.

Si les variables de type `Variant` sont très pratiques, elles occupent un espace en mémoire plus important que les autres types de variables (16 octets pour les valeurs numériques et 22 octets pour les valeurs de chaîne) et peuvent donc ralentir l'exécution du programme.

Une variable, une constante ou un argument dont le type n'est pas déclaré sont assimilés à une variable de type `Variant`. Vous pouvez cependant spécifier le type `Variant` pour faciliter la lecture de votre code. Les instructions suivantes sont équivalentes :

```
Dim MaVar
Dim MaVar As Variant
```

Variables de matrice

Une variable de matrice ou de type `Array`, parfois appelée tableau, est une variable permettant de stocker plusieurs valeurs. Contrairement à une variable ordinaire ne pouvant recevoir qu'une seule valeur, vous pouvez déterminer plusieurs emplacements de stockage pour une variable de matrice. Les variables de matrice permettent de stocker des listes d'éléments de même type dans une même variable. Vous pouvez par exemple stocker dans une variable de matrice les chiffres d'affaires de tous les représentants. Pour déclarer une variable de matrice, utilisez la syntaxe suivante :

```
Dim NomVariable(NbreElements) As Type
```

où *`NomVariable`* est le nom de la variable, *`NbreElements`*, le nombre de valeurs que recevra la variable, et *`Type`*, le type de données qu'accepte la variable. Pour affecter des valeurs à une variable de matrice ou accéder à celles-ci, il suffit de spécifier la position de la valeur stockée dans la variable. La première valeur recevant l'index 0, la dernière valeur est toujours égale au nombre d'éléments contenus dans la variable moins 1.

Pour démarrer l'index d'un tableau à 1 plutôt qu'à 0, placez l'instruction `Option Base 1` *dans la section Déclarations du module.*

Une variable de matrice peut également être déclarée selon la syntaxe suivante :

```
Dim NomVariable(Début To Fin) As Type
```

où *`Début`* et *`Fin`* définissent la plage de valeurs qui sera utilisée pour stocker et accéder aux données de la variable.

Une variable de matrice peut vous servir à stocker des données de même type. Il est alors recommandé de déclarer un type approprié. Dans l'exemple suivant, la variable de matrice JoursSemaine stocke sous forme de chaînes les jours de la semaine. Une structure de contrôle `For...Next` est ensuite utilisée pour afficher les valeurs contenues par la variable dans une boîte de dialogue.

```
Sub VarMatrice()
  Dim JoursSemaine(7) As String
  JoursSemaine(0) = "Lundi"
```

```
  JoursSemaine(1) = "Mardi"
  JoursSemaine(2) = "Mercredi"
  JoursSemaine(3) = "Jeudi"
  JoursSemaine(4) = "Vendredi"
  JoursSemaine(5) = "Samedi"
  JoursSemaine(6) = "Dimanche"
  Dim compteur as Byte
    For compteur = 0 To 6
      MsgBox JoursSemaine(compteur)
    Next compteur
End Sub
```

Une variable de matrice peut aussi servir à stocker des données de types différents. Elle doit alors être de type `Variant`. La procédure suivante stocke dans une seule variable le nom, la date de naissance, l'adresse, la fonction et le salaire d'un employé. Ces données sont ensuite affichées dans une boîte de dialogue (voir Figure 6.10).

```
Sub InfosEmployé
  Dim Employé(1 To 5) As Variant
  Employé(1) = "Jean Dupont"
  Employé(2) = "25/12/71"
  Employé(3) = "14, rue des Arts"
  Employé(4) = "Chargé d'études"
  Employé(5) = 12000
  MsgBox Employé(1) & " est né le " & Employé(2) & ". Il habite " & _
  Employé(3) & " et est " & Employé(4) & ". Son salaire est de : " & _
  Employé(5) & " euros", vbOKOnly + vbInformation, "Infos employé"
End Sub
```

Figure 6.10
Toutes les informations relatives à un même sujet peuvent être stockées dans une seule variable de matrice.

Utilisez la fonction `IsArray` *pour vérifier si une variable est de type* `Array`.

Variables de matrice multidimensionnelles

Les variables de matrice peuvent être multidimensionnelles. Les données sont alors stockées horizontalement et verticalement, à la manière d'un tableau. Les variables de matrice multidimensionnelles permettent de stocker de façon cohérente les valeurs d'une feuille Excel. Les variables multidimensionnelles se déclarent selon la même syntaxe que les

variables de matrice à une dimension, en ajoutant simplement les ar guments d'index de début et de fin pour la deuxième dimension

```
Dim NomVariable(Début1 To Fin1, Début2 To Fin2) As Type
```

Considérez la feuille de classeur représentée à la Figure 6.11. Elle représente les v entes mensuelles pour l'année 2003, de quatre types de produits, soit 12 lignes sur quatre colonnes. Une v ariable de matrice multidimensionnelle permettra de stock er l'ensemble des valeurs de cette feuille logiquement.

Figure 6.11
Utilisez les variables de matrice multidimensionnelles pour stocker les valeurs d'un tableau.

	A	B	C	D	E	F
1		Livres	Vidéo	Hi-Fi	Autres	
2	Janvier	58 963,00	45 225,00	85 485,00	45 225,00	
3	Février	45 895,00	32 568,00	79 658,00	32 568,00	
4	Mars	69 785,00	46 895,00	25 689,00	46 895,00	
5	Avril	45 214,00	54 897,00	49 652,00	54 897,00	
6	Mai	45 258,00	32 568,00	36 550,00	32 568,00	
7	Juin	38 652,00	97 632,00	56 320,00	97 632,00	
8	Juillet	32 510,00	45 863,00	45 520,00	45 863,00	
9	Août	28 952,00	32 568,00	47 965,00	32 568,00	
10	Septembre	45 693,00	94 625,00	29 865,00	94 625,00	
11	Octobre	48 956,00	31 582,00	16 495,00	31 582,00	
12	Novembre	65 920,00	21 458,00	75 632,00	21 458,00	
13	Décembre	95 120,00	12 589,00	12 589,00	12 589,00	
14						

La déclaration de la variable de matrice multidimensionnelle se présentera comme suit :

```
Dim MonTableau(1 To 12, 1 To 4) As Single
```

Cette variable correspond à un tableau de données de 12 lignes sur 4 colonnes. Il suffit ensuit d'affecter logiquement les valeurs de la feuille aux espaces de stockage de la variable :

```
MonTableau(1, 1) = Cells(3, 2).Value
MonTableau(1, 2) = Cells(3, 3).Value
MonTableau(1, 3) = Cells(3, 4).Value
MonTableau(1, 4) = Cells(3, 5).Value
MonTableau(2, 1) = Cells(4, 2).Value
MonTableau(2, 2) = Cells(4, 3).Value
MonTableau(2, 3) = Cells(4, 4).Value
MonTableau(2, 4) = Cells(4, 5).Value
Etc.
```

Ainsi, pour accéder aux v entes d'un mois, il suf fira de spécifier la aleur correspondante comme premier index de la variable `MonTableau` (1 = janvier, 2 = février, etc.). Pour accéder à une catégorie de v entes, il suffira de spécifier la aleur correspondante comme second inde x de la variable `MonTableau` (1 = Livres, 2 = Vidéo, 3 = Hi-Fi, 4 = Autres). Par exemple, `MonTableau(1, 1)` renverra le chiffre des ventes du mois de janvier pour les livres ; `MonTableau(12, 2)` renverra le chiffre des ventes du mois de décembre pour la vidéo.

N'hésitez pas à utiliser les variables de matrice pour stocker les données d'une feuille Excel auxquelles un programme VBA doit accéder à de multiples reprises. La variable ainsi créée est chargée en mémoire. L'accès aux données qu'elle contient en est sensiblement plus rapide qu'un accès aux valeurs contenues dans les cellules d'une feuille de calcul.

L'utilisation d'une structure de contrôle `For...Next` permettra d'affecter l'ensemble des valeurs de ce tableau à une variable de matrice en quelques lignes de code. Vous apprendrez à utiliser cette structure au Chapitre 7.

Une variable de matrice n'est pas limitée à deux dimensions. Vous pouvez parfaitement créer une variable de matrice à trois dimensions, ou plus.

Les fonctions `LBound` *et* `UBound` *renvoient respectivement le plus petit indice et le plus grand indice disponible pour une dimension spécifiée d'un tableau et s'utilisent selon la syntaxe suivante :*

```
LBound(NomVariable, Dimension) et UBound(NomVariable, Dimension)
```

où `NomVariable` *est le nom de la variable de matrice et* `Dimension`*, la dimension dont vous souhaitez connaître le plus petit ou le plus grand indice (1 pour la première dimension, 2 pour la deuxième, etc.). Si l'argument* `Dimension` *est omis, le plus petit ou le plus grand indice de la première dimension est renvoyé.*

Variables de matrice dynamiques

Les variables de matrice dynamiques sont des variables de matrice dont vous pouvez modifier la taille. Pour déclarer une variable de matrice dynamique, ne spécifiez pas de valeur entre les parenthèses qui suivent son nom. L'instruction suivante :

```
Dim NomVariable()
```

crée une variable de matrice dynamique. Avant d'affecter des valeurs à la variable de matrice ainsi créée, vous devrez la redimensionner à l'aide de l'instruction `ReDim`, selon la syntaxe suivante :

```
ReDim NomVariable(Début To Fin)
```

Vous pouvez utiliser le mot clé `ReDim` pour redimensionner une variable de matrice autant de fois que vous le souhaitez. Les variables de matrice sont intéressantes lorsque vous ne connaissez pas la quantité de données à stocker. Supposez que, dans l'exemple précédent, la feuille de calcul des ventes ne soit pas une feuille de calcul annuelle, mais mensuelle. Le tableau s'enrichirait alors tous les mois d'une nouvelle ligne. Pour que votre programme fonctionne tout au long de l'année, vous devrez créer une variable de matrice de longueur variable. C'est ce que fait la procédure suivante :

```
1: Sub AffectationVariableArray()
2:     Dim MonTableau() As Single
3:     Dim DerniereLigne As Byte
4:     DerniereLigne = Range("A3").End(xlDown).Row
5:     Dim NbreDeLignes As Byte
6:     NbreDeLignes = DerniereLigne - 2
7:     ReDim MonTableau(NbreDeLignes, 4)
8:     Instructions d'affectation de valeurs à MonTableau
9: End Sub
```

Aux lignes 2 et 3, les variables `MonTableau` et `DerniereLigne` sont déclarées. L'instruction de la ligne 4 sert à affecter le numéro de la dernière ligne contenant des données à la variable `DerniereLigne`. La fonction `End` renvoie l'objet `Range` correspondant à la dernière cellule non vide sous (`xlDown`) la cellule A3. La propriété `Row` renvoie la valeur du numéro de ligne de cet objet. La variable NbreDeLignes est créée ligne 5. On lui affecte ensuite une valeur égale à `DerniereLigne - 2`, soit le nombre de lignes contenant des données à stocker dans la variable (les deux premières lignes ne contenant pas de données à stocker). A la ligne 7, la variable de matrice `MonTableau` est redimensionnée de façon à accueillir l'ensemble des chiffres de ventes de la feuille.

Lorsque vous redimensionnez une variable de matrice, celle-ci est réinitialisée et toutes les valeurs qui y étaient stockées sont perdues. Pour conserver les valeurs d'une variable de matrice lors de son redimensionnement, placez le mot clé `Preserve` *devant l'instruction* `ReDim`*. L'utilisation du mot clé* `Preserve` *est cependant subordonnée à certaines conditions :*

- *Vous ne pouvez redimensionner que la dernière dimension de la variable.*
- *Vous ne pouvez pas modifier le nombre de dimensions du tableau.*
- *Vous ne pouvez qu'agrandir la taille du tableau. Si vous réduisez la taille de la variable, toutes les données seront perdues.*

La fonction première d'Excel étant d'effectuer des calculs sur des données affichées sous forme de tableaux, les variables de matrice sont très utilisées dans les programmes VBA pour Excel. Elles permettent en effet de stocker les données de feuilles de calcul sous forme de variables et de travailler directement sur la variable plutôt que sur le tableau, améliorant ainsi sensiblement les performances du programme.

Variables objet

Les variables objet sont utilisées pour faire référence à un objet et occupent 4 octets en mémoire. Une fois une variable objet définie, vous pouvez interroger ou définir les propriétés de l'objet, exécuter l'une de ses méthodes en faisant simplement référence à la variable. Pour déclarer une variable objet, utilisez la syntaxe suivante :

```
Dim NomVariable As Object
```

Vous pouvez déclarer précisément le type d'objet affecté à la variable, en remplaçant `Object` par un nom d'objet reconnu par l'application. Vous pouvez, par exemple, déclarer une variable objet `Workbook` (classeur) selon la syntaxe suivante :

```
Dim MonObjetClasseur As Workbook
```

Une fois la variable déclarée, vous devez lui affecter un objet précis, utilisez pour cela le mot clé `Set`, selon la syntaxe suivante :

```
Set NomVariable = Expression
```

où *`Expression`* est une expression renvoyant un objet de l'application. Dans l'exemple suivant, la variable objet Police est déclarée en tant qu'objet `Font`, puis affectée à l'expression `Workbooks("Representant.xls").Sheets("Feuil1").Range("A1:D5").Font`, soit l'objet `Font` (police) de la plage de cellules A1:D5 de la feuille libellée "Feuil1" du classeur Representant.xls. La variable objet est ensuite utilisée pour définir la propriété `Bold` de l'objet à `True`, c'est-à-dire pour affecter l'attribut gras à la plage de cellules A1:D5 de ce classeur.

```
Sub VariablesObjet()
  Dim Police As Font
  Set Police = _
  Workbooks("Representant.xls").Sheets("Feuil1").Range("A1:D5").Font
  Police.Bold = True
End Sub
```

La fonction *GetObject*

Les variables objet vous permettent d'agir sur un objet sans que celui-ci soit ouvert. Vous pouvez ainsi interroger les valeurs d'un Tableau Excel sans que celui-ci soit ouvert. Vous pouvez également modifier l'objet auquel vous accédez ainsi. Pour accéder à un objet, stockez cet objet dans une variable objet et à l'aide de la fonction `GetObject`, selon la syntaxe suivante :

```
Set MonObjet = GetObject(pathname, class)
```

où *`pathname`* et `class` sont des arguments nommés de type chaîne, correspondant respectivement au chemin d'accès complet et au nom du fichier auquel vous souhaitez accéder et à la classe de l'objet. Si l'argument *`pathname`* est spécifié, l'argument *`class`* peut être omis.

La procédure suivante crée une variable objet de type `Workbook` et lui affecte le fichier Representant.xls, situé sur le Bureau de Windows.

```
Sub AccederObjetFerme()
  Dim ObjetClasseur As Workbook
  Set ObjetClasseur = _
  GetObject("C:\Windows\Bureau\Representant.xls")
End Sub
```

La fonction `GetObject` est particulièrement intéressante si des données entrées dans un classeur doivent être répercutées dans un ou plusieurs autres classeurs. Vous pouvez ainsi créer un programme VBA, afin que, lorsqu'une commande est e fectuée, le classeur contenant les données de stock soit mis à jour . Si nécessaire, un message pourra être af fiché afin de pré enir l'utilisateur qu'il est temps de renouv eler le stock, et ce sans même qu'il sache qu'il existe un classeur des stocks.

C'est ce que fait la procédure sui vante, en supposant que la v aleur du stock pour le produit commandé se trouve dans la cellule A13 du classeur stock.xls.

```
 1: Sub Commande()
 2:   'Instructions Dim UnitésCommandées As Integer
 3:   Dim StockRestant As Integer
 4:   Dim UnitésCommandées As Integer
 5:   UnitésCommandées = 50
 6:   StockRestant = VerifierEtMettreAJourStock(UnitésCommandées)
 7:   If StockRestant < 0 Then
 8:      MsgBox "Le stock ne permet pas d'assurer la commande. " & _
         "Le stock pour ce produit est de " & _
         (StockRestant + UnitésCommandées) & " unités."
 9:      Exit Sub
10:   Else
11:     MsgBox "Commande effectuée. Le stock restant pour ce " & _
        "produit est de " & StockRestant & " unités."
12:   End If
13:   'Suite des instructions de la commande
14: End Sub

15: Function VerifierEtMettreAJourStock(QteCommande)
16:   Dim ObjetStock As Workbook
17:   Dim StockDispo As Integer
18:   Set ObjetStock = GetObject("C:\Documents and Settings\Administrateur\
      ➥Bureau\Stock.xls")
19:   StockDispo = ObjetStock.Sheets(1).Range("A13").Value
20:   VerifierEtMettreAJourStock = StockDispo - QteCommande
21:   If VerifierEtMettreAJourStock >= 0 Then
22:   ObjetStock.Sheets(1).Range("A13").Value = _
        StockDispo - QteCommande
23:   ObjetStock.Save
24:   End If
25: End Function
```

Veillez à per sonnaliser le c hemin précisé pour la fonction `GetObject` *à la ligne 18, sinon cette macro ne fonctionnera pas.*

A la ligne 6, la procédure `Commande` appelle la procédure `VerifierEtMettreAJourStock` en lui passant la v aleur de la v ariable UnitésCommandées. La v aleur 50 a été af fectée à cette variable à la ligne 5 pour faire fonctionner le programme. Il va de soi que cette variable doit être affectée au nombre d'unités réellement commandées.

La fonction `VerifierEtMettreAJourStock` déclare la variable objet ObjetStock de type `Workbook` (ligne 16) et la variable StockDispo de type `Integer` (ligne 17). Ligne 18, la variable ObjetStock est affectée au classeur Stock.xls situé sur le Bureau de Windows. A la ligne suivante, la variable StockDispo reçoit la valeur de la cellule A13 de la première feuille de ce classeur. La fonction `VerifierEtMettreAJourStock` se voit ensuite affecter la valeur de `StockDispo - QteCommande`. Enfin, lignes 21 à 24, une structure `If...Then...Else` est utilisée pour mettre à jour la valeur du stock restant. L'instruction de la ligne 21 vérifie que le stock restant après commande est supérieur à zéro. Si c'est le cas, la valeur de la cellule A13 est mise à jour pour refléter le stock restant (ligne 22) et le classeur est ensuite sauvegardé (ligne 23).

La procédure Commande reprend alors la main. Lignes 7 à 12, une structure conditionnelle `If...Then...Else` affiche un message à l'attention de l'utilisateur, afin de l'informer sur l'état du stock. Si le stock est insuffisant pour assurer la commande (`StockRestant < 0`), l'instruction de la ligne 8 est exécutée. L'utilisateur est alors averti que la commande n'a pu être validée, et informé du stock disponible (`StockRestant + UnitésCommandées`). Si le stock permet d'assurer la commande, l'instruction de la ligne 11 affiche un message informant l'utilisateur du stock restant après commande.

La fonction *CreateObject*

La fonction `CreateObject` sert à créer une instance d'un objet, et s'utilise selon la syntaxe suivante :

```
CreateObject(class,servername)
```

où `class` et *`servername`* sont des arguments nommés de type chaîne. *`class`* correspond à la classe de l'objet dont on crée une instance. *`servername`* est facultatif et correspond au nom d'un serveur distant sur lequel est créé l'objet. Si vous omettez cet argument, l'objet est créé sur la machine sur laquelle s'exécute le programme.

Une fois créée une instance d'objet, on accède aux propriétés et aux méthodes de cet objet en utilisant le nom de la variable à laquelle il est affecté.

Dans l'exemple suivant, une instance de l'objet Excel est créée. Un nouveau classeur est créé, configuré puis enregistré dans cette instance.

```
1: Sub CreerInstancesExcel()
2:    'déclaration des variables
3:    Dim Xl As Excel.Application
4:    Dim NouvClasseur As Excel.Workbook
5:    Dim NomFichier As String
6:    'création d'une instance de l'objet Excel
7:    Set Xl = CreateObject("Excel.Application")
8:    'affichage de l'objet Xl
9:    Xl.Application.Visible = True
```

```
10:     'création d'un nouveau classeur dans l'objet Xl
11:     Set NouvClasseur = Xl.Workbooks.Add
12:     'suppression de la troisième feuille de calcul
13:     NouvClasseur.Sheets(3).Delete
14:     'affectation de noms aux feuilles 1 et 2
15:     NouvClasseur.Sheets(1).Name = "Quantites"
16:     NouvClasseur.Sheets(2).Name = "Chiffres"
17:     'définition du nom du classeur
18:     Dim compteur As Byte
19:     Dim Pos As Long
20:     NomFichier = "Ventes " & Date & ".xls"
21:     For compteur = 1 To 2
22:     Pos = InStr(NomFichier, "/")
23:     NomFichier = Left(NomFichier, Pos - 1) & "-" & _
24:     Right(NomFichier, Len(NomFichier) - Pos)
25:     Next compteur
26:     'enregistrement du classeur
27:     NouvClasseur.SaveAs "C:\Documents and Settings\Administrateur
        ➥\Bureau\Ventes\" & NomFichier
28:     NouvClasseur.Close
29:     Xl.Quit
30: End Sub
```

Veillez à personnaliser le chemin précisé pour la fonction `GetObject` *à la ligne 27, sinon cette macro ne fonctionnera pas.*

Lignes 2 à 5, les variables sont déclarées. Les variables Xl et NouvClasseur sont des variables objet de type `Excel` et `Workbook`. La variable NomFichier servira à stocker le nom d'enregistrement du classeur. Ligne 7, une instance de l'objet `Application` d'Excel est créée à l'aide de l'instruction `CreateObject`, et affectée à la variable Xl. La propriété `Visible` de l'objet `Application` est ensuite définie à True afin de faire apparaître la session Excel à l'écran.

Ligne 11, un nouveau classeur est ajouté à l'objet `Application` nouvellement créé, et affecté à la variable objet NouvClasseur. Notez que, par défaut, un nouveau classeur est créé dans la session Excel à partir de laquelle le programme est exécuté. Pour que le nouveau classeur soit créé dans la nouvelle session Excel, il est indispensable de faire référence à l'objet Xl dans l'instruction de la ligne 11 (`Xl.Workbooks.Add` et pas simplement `Workbooks.Add`). Lignes 13 à 16, la troisième feuille du classeur est supprimée et les deux autres sont renommées.

Lignes 17 à 25, le nom d'enregistrement du classeur est défini. La variable NomFichier se voit tout d'abord affecter le nom Ventes suivi de la date du jour (Ventes 12/08/2007, par exemple). Ce nom contient deux fois le caractère barre oblique (/), non valide dans les noms de fichier. Lignes 21 à 25, une boucle `For...Next` est utilisée pour répéter deux fois le traitement appliqué au nom du classeur afin de substituer des traits d'union aux barres obliques (les boucles sont étudiées au prochain chapitre). On utilise pour cela les fonctions

de manipulation de chaîne `InStr`, `Left`, `Right` et `Len`. `Instr` est utilisé pour renvoyer la position du caractère / dans la chaîne `NomFichier` (ligne 22). `Len` renvoie le nombre de caractères de la chaîne `NomFichier` (ligne 24). Les fonctions `Left` et `Right` sont utilisées pour renvoyer respectivement les caractères situés à gauche et à droite des barres obliques, et un trait d'union est placé entre les deux chaînes ainsi renvoyées.

Lignes 27 et 28, le classeur est enregistré puis fermé. Ligne 29, la méthode `Quit` est appliquée à l'objet Excel, afin de fermer la session Excel créée en début de programme.

Libérer une variable objet

Pour annuler l'affectation d'un objet à une variable objet, affectez-lui la valeur `Nothing`. L'instruction suivante annule l'affectation de la variable objet Police créée précédemment :

```
Set Police = Nothing
```

Il est important d'affecter la valeur `Nothing` à une variable objet, lorsque celle-ci n'est plus utilisée par le programme. Vous libérez ainsi l'ensemble des ressources système et mémoire associées à l'objet. Dans l'exemple de programme précédent, vous devrez placer cette instruction au-dessus de la ligne 24. La variable objet ObjetStock sera ainsi libérée avant que la procédure `Commandes` ne reprenne la main.

Types de données personnalisés

Le mot clé `Type` permet de créer des types de données personnalisés, associant les types de données présentés ci-dessus. Une telle opération se révèle intéressante lorsqu'un programme doit associer de façon récurrente différents types de données. Vous pouvez alors créer un nouveau type de données auquel il suffira de faire référence chaque fois que vous souhaiterez créer une nouvelle variable regroupant ces types de données.

A l'instar des variables de matrice, le mot clé `Type` permet donc de créer des variables capables de stocker des informations multiples. Mais, contrairement aux variables de matrice, `Type` permet d'associer des types de données différents. La déclaration d'un nouveau type de données doit être placée dans la section Déclarations du module, selon la syntaxe suivante :

```
Type NomType
   Données1 As Type
   Données2 As Type
   ....
   Donnéesn As Type
End Type
```

où *Données1*, …, *Donnéesn* sont les données que contiendront les v ariables de type *NomType*. Ces noms seront emplo yés par la suite pour af fecter des v aleurs aux dif férents espaces de stockage des v ariables de type *NomType*. *Type* représente le type de données affecté à chaque élément du nouveau type de données.

Dans l'exemple suivant, un type de données `Membre` est créé, afin de pou oir intégrer dans une seule variable l'ensemble des informations concernant un membre d'une association :

```
Type Membre
  Prénom As String
  Nom As String
  Adresse As String
  CodePostal As String
  Ville As String
  Téléphone As String
  Age As Byte
End Type
```

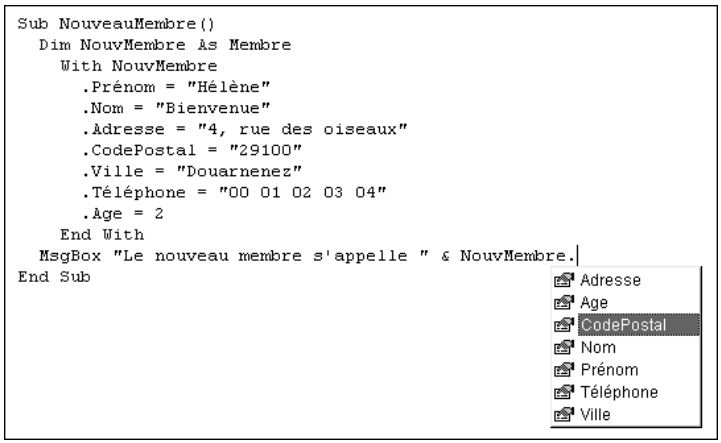
Vous pouvez maintenant créer une nouvelle variable de type `Membre`, qui sera prête à recevoir toutes les informations contenues dans ce type. Les informations contenues dans une variable peuvent ensuite être interrogées ou déf nies en faisant suivre le nom de la variable d'un point, puis du nom de la donnée. La procédure suivante crée une variable de type `Membre` et définit ses aleurs, puis af fiche une boîte de dialogue indiquant les donnée `Prénom` et `Nom` de la variable (voir Figure 6.12) :

```
Sub NouveauMembre
  Dim NouvMembre As Membre
    With NouvMembre
      .Prénom = "Hélène"
      .Nom = "Bienvenue"
      .Adresse = "4, rue des oiseaux"
      .CodePostal = "56000"
      .Ville = "Vannes"
      .Téléphone = "00 01 02 03 04"
      .Age = 2
    End With
  MsgBox "Le nouveau membre s'appelle " & NouvMembre.Prénom & " " & _
         NouvMembre.Nom, vbOKOnly + vbInformation, "Nouveau membre"
End Sub
```

Figure 6.12
Les variables de type personnalisé peuvent contenir un nombre indéterminé d'informations.

Notez que, en phase de création, un complément automatique est af fiché lorsque ous faites référence à une v ariable de type personnalisé (v oir Figure 6.13) – à condition que l'option correspondante soit validée.

Figure 6.13
Le complément automatique d'instruction est affi hé pour les types de données personnalisés.



Constantes

Les constantes permettent d'attrib uer un nom à une v aleur fi e. Il est ainsi plus aisé d'exploiter cette v aleur dans le code en f aisant référence au nom de la constante, plutôt qu'à la v aleur elle-même. P ar ailleurs, si une v aleur est susceptible d'être modifiée (un valeur de TVA, par e xemple), l'affectation de cette v aleur à une constante simplifiera le éventuelles mises à jour. Il vous suffira en e fet de modifier la aleur de la constante, plutôt que de modifier chaque occurrence de la aleur dans le code de l'ensemble de vos projets.

Une fois qu'une valeur a été af fectée à une constante , celle-ci ne peut êtr e modifiée par la suit .

Pour déclarer une constante, utilisez l'instruction `Const`, selon la syntaxe suivante :

```
Const NomConstante As Type = Valeur
```

où *`NomConstante`* est le nom de la constante, *`Type`*, le type de données de la constante —il peut s'agir de n'importe lequel des types de données présentés plus haut —, et *`Valeur`*, la valeur qui lui est af fectée. L'instruction suivante déclare la constante TVA, à laquelle la valeur 20.6 est affectée.

```
Const TVA As Single = 20.6
```